SwitchFold: Code-Agnostic Succinct Polynomial Commitments via Recursive Code Switching

Mingshu Cong[1][0000−0001−5858−3411], Tsz Hon Yuen[2][0000−0002−0629−6792], and Siu-Ming Yiu[3][0000−0002−3975−8500]

[1] Shenzhen Audencia Financial Technology Institute, Shenzhen University, Shenzhen
mcong@szu.edu.cn
[2] Faculty of Information Technology, Monash University, Melbourne, Australia
john.tszhonyuen@monash.edu
[3] School of Computing and Data Science, The University of Hong Kong, Hong Kong
smyiu@cs.hku.hk

Abstract. We study hash-based polynomial commitment schemes (PCSs) over large prime fields that scale to witnesses containing billions of scalars. Such PCSs are important for emerging data-science-related applications of zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs). Our goal is to reduce prover time while preserving polylogarithmic proof size and verifier time. To this end, we seek *code-agnostic*, *succinct* PCS constructions that can be instantiated with any linear code over any target field, allowing the most efficient code for the chosen field to be selected.

Our `SwitchFold` construction achieves polylogarithmic verification by recursively applying the code-switching technique between codes of decreasing block lengths. Prior code-switching PCSs switch from an unstructured code to a foldable code; in contrast, `SwitchFold` requires no code foldability and thus decouples PCS design from code design. For a polynomial of size N and security parameter λ, instantiating `SwitchFold` with linear-time encodable codes yields $O(N)$ prover time and $O(\lambda \log^2 N)$ proof size and verifier time.

We instantiate `SwitchFold` with the Brakedown codes (Golovnev *et al.*, CRYPTO '23) and the recently proposed quasi-Abelian (QA) codes (Li *et al.*, IACR ePrint '26), yielding `BrakeFold` and `QAFold`, respectively. At $N = 2^{30}$, `QAFold` achieves the fastest prover time among all evaluated PCSs with succinct verification. Compared with TensorSwitch (Bünz *et al.*, IACR ePrint '25), the other code-agnostic PCS, `QAFold` reduces total and marginal prover time by factors of 2.2× and 2.7× under the same QA-code instantiation. With Brakedown codes, `BrakeFold` reduces the marginal prover time per additional opening by a factor of 2.9× relative to TensorSwitch.

Keywords: Hash-based polynomial commitments · Zero-knowledge proof · Code agnosticity.

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) enable an untrusted prover to convince a verifier, using a succinct proof with at most polylogarithmic proof size, that an NP statement is true. Polynomial commitment schemes (PCSs) are a central primitive underlying zkSNARKs; they allow a prover to succinctly *commit* to a polynomial and later *open* the commitment by proving polynomial evaluations at selected points. The efficiency of the PCS directly affects the efficiency of the zkSNARK protocols.

In this paper, we study hash-based, field-agnostic, large-scale PCS with the goal of minimizing the prover time while maintaining polylogarithmic proof size and verifier time. Following Brakedown [28] and BaseFold [46], we call a PCS *field-agnostic* if it can be instantiated over any sufficiently large finite field without relying on any special algebraic structure of the field. This property is particularly valuable for emerging data-science-related applications of zkSNARKs such as zero-knowledge machine learning (zkML), which produces succinct proofs for neural-network computations. zkML requires fixed-point numerical fidelity, for which certain large prime fields, such as the Mersenne field $\mathbb{F}_{2^{127}-1}$, are preferable to mitigate unintended wraparound while retaining efficient field arithmetic. Instantiating a PCS over such fields requires the field-agnostic property. Since modern neural networks may contain billions of parameters, the PCS must support commitments and openings at very *large scale*.

By definition, zkSNARKs require proof size and verifier time to grow at most polylogarithmically with the witness size. Subject to this succinctness requirement, prover time becomes the primary optimization objective. Once a PCS achieves polylogarithmic proof size and verifier time, recursive composition can compress its proofs to constant size by using a fast-prover PCS as the inner argument and a constant-size SNARK as the outer wrapper. For sufficiently large witnesses, the cost of the outer wrapper is small relative to that of generating the inner proof, making the prover time of the inner PCS the dominant bottleneck.

We work in the *hash-based* setting: the prover encodes the polynomial coefficient vector or matrix with a linear code, commits to the codeword via a Merkle tree, and answers polynomial evaluation claims by opening sampled positions with Merkle authentication paths. This setting avoids elliptic-curve and pairing assumptions, requires no trusted setup, and is plausibly post-quantum secure in the random-oracle model [33,37]. By replacing the expensive group operations required by ECC-based PCSs [15,18,27,29,32,35] with field arithmetic and hashing, hash-based schemes achieve the fastest known concrete prover times at large scale. More recently, lattice-based PCSs have offered another promising post-quantum route [13,21], but their concrete prover efficiency is not yet competitive in this regime; we therefore focus on hash-based PCSs.

Within the domain of hash-based PCSs, we introduce the notion of *code agnosticity*: a PCS is code-agnostic if it can be instantiated with any linear code while maintaining polylogarithmic proof size and verifier time, without relying on special structure of the code. In such a construction, faster encoding directly reduces prover time, while larger relative distance directly reduces proof size

and verifier time. Code agnosticity therefore decouples the design of PCSs and zkSNARKs from the design of linear codes. It implies field agnosticity, whereas the converse does not hold. We show that a code-agnostic PCS can be obtained through a careful combination of several standard techniques from the zkSNARK literature while nevertheless delivering strong concrete performance.

1.1 Our Results

In this work, we present a generic transformation, based on *recursive code switching*, from a sequence of linear codes with geometrically decreasing block lengths to a hash-based multilinear PCS. If each code admits linear-time encoding in its block length, for witness size N and security parameter λ, the resulting PCS has $O(N)$ prover time and $O(\lambda \log^2 N)$ proof size and verifier time. Our construction does not rely on any algebraic structure of the underlying linear codes and is therefore code-agnostic.

Instantiating our framework with the Brakedown code [28] and the recently proposed quasi-Abelian (QA) codes [36] yields two concrete PCSs, which we call `BrakeFold` and `QAFold`, respectively. Both schemes can be instantiated over arbitrary large prime fields. Recursive code switching reduces the proof size and verifier time of the original Brakedown- and QA-code-based PCSs from square-root to polylogarithmic complexity while retaining their efficient prover.

For $N = 2^{30}$, $\lambda = 100$, and 64 threads, the marginal prover times of `BrakeFold` and `QAFold` are 91.1 s and 71.4 s per additional opening, while their total prover times, including an unamortized Decide phase, are 754.4 s and 88.8 s, respectively. Among the evaluated PCSs with succinct verification, `QAFold` achieves the fastest prover. Relative to TensorSwitch [17], the fastest competing prover in our evaluation, `QAFold` reduces marginal and total prover time by factors of 2.7× and 2.2×, respectively, when both use the same QA code. Similarly, `BrakeFold` reduces marginal prover time by 2.9× relative to its TensorSwitch counterpart. At $N = 2^{26}$, the largest instance the widely used BaseFold [46] can process within our memory limit, `BrakeFold` and `QAFold` reduce total prover time by 2.3× and 15.4×, and proof size by 2.6× and 4.4×, respectively. The `QAFold` verifier is also 2.0× faster, while `BrakeFold` matches the BaseFold verifier time.

SwitchFold: Code-Agnostic PCS with Polylogarithmic Verification. Our main contribution is a code-agnostic PCS construction, called `SwitchFold`, based on recursive code switching. It achieves linear prover time and polylogarithmic proof size and verifier time using any sequence of codes satisfying some generic efficiency conditions.

Theorem 1 (Informal; see Theorem 6). *Let $\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}$ be a sequence of linear codes whose block lengths satisfy $k_j = k_0 t^j$ for a fixed constant $t \geq 2$, where d is chosen so that $k_d t = N$ for witness size N. Assume that each $\mathcal{C}_j$ admits $O(k_j)$-time encoding and has relative distance uniformly bounded below. Then, for security parameter λ,* `SwitchFold` *yields a multilinear PCS with $O(N)$ prover time and $O(\lambda \log^2 N)$ proof size and verifier time.*

The efficiency of `SwitchFold` is determined by the performance of the underlying code sequence: prover time is dominated by the total encoding time across code levels, while proof size and verifier time are governed by the relative distance. Consequently, faster encoding or larger relative distance directly improves the resulting PCS efficiency. In this sense, `SwitchFold` *decouples PCS design from code design*. Like other hash-based PCS constructions, it is secure in the random-oracle model and plausibly post-quantum secure.

BrakeFold and QAFold: Outperforming State-of-the-Art Succinct PCSs in Prover Time. Our second contribution provides optimized instantiations of the `SwitchFold` framework with the Brakedown [28] and QA codes [36], yielding `BrakeFold` and `QAFold`, respectively. Among the PCSs with polylogarithmic proof size and verifier time, `QAFold` achieves the fastest total prover time for large witnesses. Although `BrakeFold` is less competitive in total prover time, its marginal prover time per additional instance remains efficient for repeated PCS openings. Together, these results establish a new prover efficiency benchmark for hash-based PCSs.

Both Brakedown and QA codes admit efficient encoding but lack *foldability*, a property traditionally used to achieve polylogarithmic openings, as in FRI-based PCSs using Reed–Solomon (RS) codes [7] and BaseFold using random foldable codes (RFCs) [46]. Foldability, however, typically comes at the cost of more expensive encoding. Only recently did TensorSwitch [17] show that such succinctness can be achieved without foldable codes. Consequently, the original Brakedown and QA-code PCSs have $O(\sqrt{\lambda N})$ proof size and verifier time, limiting their applicability to large instances.

The *code-switching* framework of Ron-Zewi and Rothblum [39] traditionally combines the fast encoding of a non-foldable source code with the succinctness of a foldable target code, as demonstrated by Blaze [14] and BrakingBase [38]. These constructions, however, still rely on target-code foldability, and their code-switching cost depends on the encoding efficiency of the foldable target code. In contrast, `SwitchFold` recursively switches between non-foldable codes of decreasing block lengths, eliminating the need for an auxiliary foldable code. Moreover, `SwitchFold` uses the accumulation scheme [16] to defer the generator-matrix MLE claims introduced by code switching until finalization, substantially reducing the marginal prover cost of each additional PCS opening.

1.2 Related Work

We classify prior hash-based PCSs into five families by core technique. Table 1 summarizes their asymptotic costs, underlying codes, and field support.

FRI [7], STIR [2], WHIR [3], and Zeromorph-FRI [34]. FRI is the first hash-based PCS achieving polylogarithmic proof size and verifier complexity, and when composed via a univariate-to-multilinear transformation such as Zeromorph, yields a multilinear PCS. (Note that every multilinear PCS yields a univariate PCS, but the converse requires such a transformation.) This polylogarithmic complexity of FRI stems from the *foldability* of RS codes over FFT-

Table 1. Comparison of hash-based polynomial commitment schemes.

PCS	Commit. time	Opening time	Proof size / Verifier time	Code	Field
FRI-style schemes					
FRI [7]	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	RS	FFT-friendly
STIR [2]	$O(N \log N)$	$O(N \log N)$	$O(\lambda^2 \log \log N)$	RS	FFT-friendly
WHIR [3]	$O(N \log N)$	$O(N \log N)$	$O(\lambda \log N \log \log N)$	RS	FFT-friendly
Zeromorph-FRI [34]	$O(N \log N)$	$O(N \log N)$	$O(\lambda \log^2 N)$	RS	FFT-friendly
Interleaved-code-based schemes					
Ligero [1]	$O(N \log N)$	$O(N)$	$O(\sqrt{\lambda N})$	RS	FFT-friendly
Batch FRI [8]	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	RS	FFT-friendly
Brakedown [28]	$O(N)$	$O(N)$	$O(\sqrt{\lambda N})$	Spielman	Agnostic
Orion [45]	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	Spielman+RS	FFT-friendly
BFK24 [12]	$O(N)$	$O(N)$	$O(\sqrt{\lambda N})$	EA	Agnostic
Binary-field optimizations					
Binius [24]	$O(N \log N)$	$O(N)$	$O(\sqrt{\lambda N})$	RS	Binary
FRI-Binius [23]	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	RS	Binary
Sumcheck-coupled folding schemes					
BaseFold [46]	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	RFC	Agnostic
DeepFold [30]	$O(N \log N)$	$O(N)$	$O(\lambda \log^2 N)$	RS	FFT-friendly
Code-switching-based schemes					
Blaze [14]	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	RAA+RFC	Binary
BrakingBase [38]	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	Spielman+RFC	Agnostic
TensorSwitch [17]	$O(N)$	$O(N)$	$O(\lambda \log N)$	Agnostic	Agnostic
`SwitchFold`	$O(N)$	$O(N)$	$O(\lambda \log^2 N)$	Agnostic	Agnostic
`BrakeFold`	$26N\mathbb{F},\ 2N\mathbb{H}$	$108N\mathbb{F},\ 3N\mathbb{H}$	$33\lambda \log^2 N\,(\mathbb{F}+\mathbb{H})$	Spielman	Agnostic
`QAFold`	$2N\mathbb{F},\ 72N\mathbb{F}_+,$ $2N\mathbb{H}$	$6N\mathbb{F},\ 7N\mathbb{F}_+,$ $(N/10)\mathbb{H}$	$6\lambda \log^2 N\,(\mathbb{F}+\mathbb{H})$	QA	Prime

Notes. Here, N denotes the witness size and λ denotes the security parameter. In time bounds, $\mathbb{F}$, $\mathbb{F}_+$, and $\mathbb{H}$ denote one field multiplication, one field addition, and one hash operation, respectively; in proof-size bounds, $\mathbb{F}$ denotes one field element. The concrete efficiency for `BrakeFold` and `QAFold` use $t = 64$, rate $1/2$, and λ-bit security; their opening times include one unamortized `Decide` phase. RS, EA, RFC, RAA, Spielman, and QA denote Reed–Solomon codes, expand–accumulate codes, random foldable codes, repeat–accumulate–accumulate codes, Spielman-style codes, and quasi-Abelian codes, respectively. FFT-friendly means that the field contains a smooth multiplicative subgroup. Binary refers to binary extension fields. Agnostic means that the scheme is field-agnostic or code-agnostic.

friendly fields, whose multiplicative group contains a smooth subgroup, typically of power-of-two order. This structure ensures that pairwise *folding* an RS codeword yields another RS codeword of half the length and half the degree, while folding a non-codeword yields, with high probability, a non-codeword. Consistency of the folded codewords is enforced by a *proximity check* based on random sampling. This folding-plus-proximity-check paradigm is foundational to nearly all subsequent hash-based PCSs. STIR and WHIR refine FRI by working with a sequence of RS codes of heterogeneous relative distance, further reducing the proof size. Due to the strong distance properties of RS codes, this family enjoys low query complexity and consequently the smallest concrete proof size and verifier time among hash-based PCSs. The trade-off is that RS encoding is quasi-linear, so for large witnesses the prover is typically slower than PCSs based on linear-time encodable codes.

Ligero [1], Batch FRI [8], Brakedown [28], Orion [45], and BFK24 [12]. Ligero is the starting point of a parallel line of hash-based PCSs that builds on

interleaved codes. An interleaved code is obtained by encoding each column of a message matrix independently with a base code; the resulting matrix is a codeword of the interleaved code if and only if every column is a codeword of the base code, which can be verified by showing that a random linear combination of the columns yields a codeword of the base code. This linear combination of columns requires no specific algebraic structure of the base code. Ligero and Batch FRI instantiate the base code with Reed–Solomon codes; Brakedown and Orion replace the RS codes with Spielman-style codes that admit linear-time encoding; BFK24 uses expand-accumulate (EA) codes. However, the lack of algebraic structure in these codes prevents efficient direct verification; as a result, these PCSs have $O(\sqrt{\lambda N})$ proof size and verifier complexity. Orion recovers polylogarithmic proof size by wrapping its $O(\sqrt{\lambda N})$ proof inside an outer FRI-based SNARK, at the cost of reintroducing FFT-friendly field requirements.

Binius [24] and FRI-Binius [23]. Binius retains RS codes but operates over binary tower fields, making it particularly suitable for bit-level computations and bit-oriented witnesses, such as SHA-256 execution traces. Binius instantiates this idea in a Ligero-style protocol, while FRI-Binius combines it with FRI. Since our work targets large prime fields, this line of PCSs is not directly comparable. Nevertheless, its efficiency for hash-heavy computations and its concretely small proof size suggest that the hash-heavy verifier of many hash-based PCSs could be wrapped within an efficient outer Binius-style proof, thus reducing concrete proof size and verifier time. This supports our priority on minimizing prover time for large-scale PCSs.

BaseFold [46] and DeepFold [30]. BaseFold constructs a multilinear PCS from the random foldable code sequence, where codewords have doubling lengths, and a random linear combination of the two halves of a codeword yields a valid codeword at the next lower level. BaseFold couples this folding process with a sumcheck protocol for the MLE evaluation claim, using shared verifier challenges so that both reductions terminate at a consistent value. Instantiated with random foldable codes, it gives the first field-agnostic multilinear PCS with polylogarithmic proof size and verifier time. DeepFold employs the same sumcheck-coupled folding approach to RS codes, sacrificing field-agnosticism for better concrete proof size and verifier time due to the stronger distance of RS codes. The main limitation of this line of PCS is prover time, as all known foldable linear codes require at least $O(N \log N)$ encoding time.

Blaze [14], BrakingBase [38], and TensorSwitch [17]. This line seeks to combine the linear prover time of linear-time encodable codes, such as Brakedown and repeat-accumulate-accumulate (RAA) codes, with the polylogarithmic verification of foldable codes, such as RS codes and RFCs. The key tool is *code switching* [39]: commit using a fast source code, then switch to a foldable target code for efficient verification. Blaze instantiates code switching over binary fields via an RAA-to-RFC switch, while BrakingBase gives a field-agnostic instantiation via a Brakedown-to-RFC switch. Asymptotically, both achieve linear prover time and polylogarithmic verifier complexity. Concretely, however, the dominant

cost is opening the MLE of the source-code generator matrix under the target code. This cost is modest for RAA code generator matrices but relatively expensive in BrakingBase for the sparse, randomized Brakedown generator matrices. TensorSwitch reduces this code-switching overhead by replacing the source code with a tensor code, shrinking the generator-matrix dimensions to $O(\sqrt{N})$; the trade-off is a two-pass encoding procedure whose second pass operates on an expanded codeword, offsetting the prover-time advantage of linear-time encoding.

Comparison with `SwitchFold`. Like Blaze, BrakingBase, and TensorSwitch, `SwitchFold` uses code switching to combine fast encoding with succinct verification. Unlike prior code-switching PCSs, however, `SwitchFold` switches recursively between codes of decreasing block lengths. Each round reduces an MLE opening under one code to an opening under the next shorter code, without requiring an auxiliary foldable target code. At a high level, `SwitchFold` resembles FRI and BaseFold in recursively reducing longer codewords to shorter ones. The difference is that our reduction relies on generic code switching, rather than on foldability of the underlying codes. This makes `SwitchFold` code-agnostic.

1.3 Technical Overview

Fig. 1 illustrates the framework of `SwitchFold`, which consists of three modules:

(1) *Recursive code switching* reduces an MLE claim on a witness matrix committed under a large interleaved code to an MLE claim under interleaved codes with progressively decreasing block lengths, ultimately yielding an MLE claim on a short vector. Along the way, each code switch produces an additional MLE claim on the corresponding generator matrix.

(2) *Accumulation of generator-matrix MLE claims* accumulates and defers the opening of generator-matrix MLE claims. Because the generator matrices remain fixed across witnesses, these claims can be proved together when the multiple PCS openings are finalized. [4]

(3) *Final recursive opening of generator-matrix MLE claims* is performed only once using recursive code switching, after the generator-matrix claims arising from multiple PCS openings have been accumulated. At each level, the current generator matrix is committed to under an interleaved code derived from the code at the next lower level. Code switching then reduces its MLE opening problem to a new MLE claim on the smaller generator matrix. The recursion terminates at the base level. [5]

[4] Accumulating generator-matrix MLE claims differs from batching polynomial-commitment openings. In many zkSNARK applications, polynomials must be committed to and opened sequentially because later commitments may depend on Fiat–Shamir challenges derived from earlier prover messages, as in the sparse-matrix argument of Spartan [40]. Thus, these polynomials cannot be batched in advance.

[5] This recursive opening differs from recursive proof composition, in which an outer proof attests to the verification of an inner proof. Here, recursion occurs within a single protocol and reduces problem complexity by transforming an MLE opening on a large generator matrix into openings on progressively smaller generator matrices.

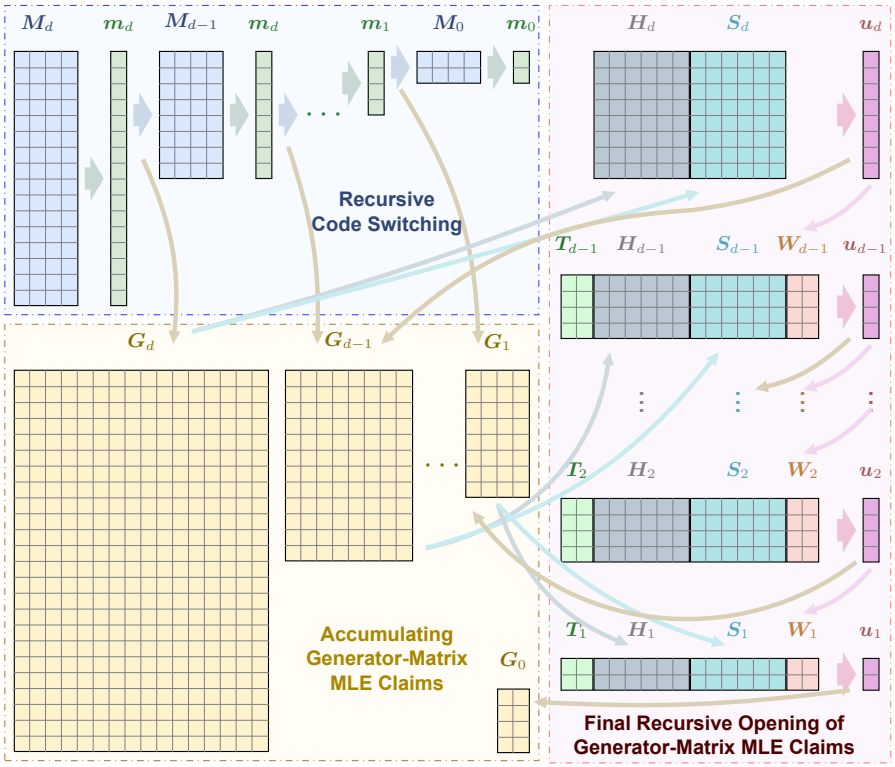

Fig. 1. Illustration of the `SwitchFold` proving workflow. By recursively compressing the messages of interleaved codes and applying code switching at each level, the prover reduces an MLE claim on a large matrix $\boldsymbol{M}_d$ to an MLE claim on a small vector $\boldsymbol{m}_0$, together with MLE claims on the generator matrices $\{\boldsymbol{G}_j\}_{j=1}^{d}$ accumulated across repeated PCS openings. In the final stage, one Merkle root jointly commits to $[\,\boldsymbol{T}_j \mid \boldsymbol{H}_j\,]$ under the derived RFC generator $\widehat{\boldsymbol{G}}_j$, while two dynamic roots commit to $\boldsymbol{S}_j$ and $\boldsymbol{W}_j$. The MLE claim on $\widehat{\boldsymbol{G}}_j$ is reduced through its RF-extension layers to a fresh $\boldsymbol{T}_{j-1}$-claim and a claim on $\boldsymbol{G}_{j-1}$. The previous compressed witness is included in the current fold, producing one new vector $\boldsymbol{u}_j$ that is reshaped into $\boldsymbol{W}_{j-1}$ for the next level. Both recursions terminate at $\boldsymbol{G}_0$.

Recursive Code Switching for Interleaved Codes. In hash-based PCSs, committing to a message $\boldsymbol{m}$ under a linear code $\mathcal{C}$ means publishing the Merkle root of the codeword $\mathsf{Enc}[\mathcal{C}](\boldsymbol{M})$, where $\mathsf{Enc}[\mathcal{C}]$ denotes the encoding function of $\mathcal{C}$. The opening protocol lets the verifier check an MLE claim on $\boldsymbol{M}$, namely the evaluation of the unique multilinear extension polynomial of $\boldsymbol{M}$ at a given point. In our formulation, this MLE claim is written as a vector–matrix–vector product $v = \boldsymbol{l}^{\top}\boldsymbol{M}\boldsymbol{r}$, and is verified using only a few queries to codeword entries.

We use a sequence of $d+1$ linear codes $\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}$, where k_j is the block length and n_j the codeword length at level j, with geometrically decreasing block lengths $k_j = tk_{j-1}$ for some constant t. We write $\boldsymbol{G}_j$ for the

generator matrix of $\mathcal{C}_j$, so $\mathsf{Enc}[\mathcal{C}_j](\boldsymbol{m}) = \boldsymbol{G}_j\boldsymbol{m}$. The interleaved code $\mathcal{C}_j^t$ encodes a $k_j \times t$ matrix $\boldsymbol{M}$ columnwise as $\mathsf{Enc}[\mathcal{C}_j^t](\boldsymbol{M}) = \boldsymbol{G}_j\boldsymbol{M}$, *i.e.*, it applies $\mathsf{Enc}[\mathcal{C}_j]$ independently to each of the t columns. Because $k_j = tk_{j-1}$, the same length-k_j message can be encoded either under $\mathcal{C}_j$ or, after reshaping into a $k_{j-1} \times t$ matrix, columnwise under $\mathcal{C}_{j-1}^t$. This equivalence allows each recursive code switch to reduce the block length, and hence the problem size, by a factor of t.

Each level uses two reductions:

(1) *Interleaved code → base code (Ligero-style):* an MLE claim under $\mathcal{C}_j^t$ reduces, via a random linear combination of columns, to an MLE claim under $\mathcal{C}_j$.

(2) *Base code → lower-level interleaved code (code switching [39]):* given the Merkle root of $\mathsf{Enc}[\mathcal{C}_j](\boldsymbol{m}_j)$, we reshape $\boldsymbol{m}_j$ into a $k_{j-1} \times t$ matrix and prove its consistency with a fresh commitment under $\mathcal{C}_{j-1}^t$ by a proximity check:

$$\boldsymbol{\Xi} \cdot \mathsf{Enc}[\mathcal{C}_j](\boldsymbol{m}_j) = \boldsymbol{\Xi}\boldsymbol{G}_j\boldsymbol{m}_j,$$

where $\boldsymbol{\Xi}$ is a random sparse boolean matrix that selects a few rows of the codeword. The left-hand side (LHS) is obtained by querying the Merkle tree; the right-hand side (RHS) is handled by a matrix-multiplication proof [44], which reduces the check to MLE claims of $\boldsymbol{m}_j$ (now reshaped into $\boldsymbol{M}_{j-1} \in \mathbb{F}^{k_{j-1} \times t}$ and committed under $\mathcal{C}_{j-1}^t$) and $\boldsymbol{G}_j$ (deferred to later).

Alternating (1) and (2) reduces the message dimension by a factor of t per round, recursing down to the base code $\mathcal{C}_0$:

$$\mathsf{Enc}[\mathcal{C}_d^t](\boldsymbol{M}_d) \overset{(1)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_d](\boldsymbol{m}_d) \overset{(2)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_{d-1}^t](\boldsymbol{M}_{d-1}) \overset{(1)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_{d-1}](\boldsymbol{m}_{d-1})$$
$$\overset{(2)}{\Longrightarrow} \cdots \overset{(1)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_1](\boldsymbol{m}_1) \overset{(2)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_0^t](\boldsymbol{M}_0) \overset{(1)}{\Longrightarrow} \mathsf{Enc}[\mathcal{C}_0](\boldsymbol{m}_0).$$

Accumulation for Generator-Matrix MLE Claims. The generator matrix $\boldsymbol{G}_j$ has dimensions $n_j \times k_j$. For codes admitting $O(k_j)$-time encoding, proper arithmetization techniques can represent $\boldsymbol{G}_j$ using $O(k_j)$ nonzero entries, yielding $O(k_j)$ commitment and opening costs.

Despite this linear cost guarantee, opening generator-matrix MLE claims can be concretely expensive for some codes. We exploit the fact that the generator matrices $\{\boldsymbol{G}_j\}_{j=0}^d$ are *fixed* across PCS instances. Rather than opening an MLE claim on each $\boldsymbol{G}_j$ after every instance, we *accumulate* these claims across openings and defer their verification. At the final stage, the claims accumulated for each $\boldsymbol{G}_j$ are combined into a single claim, requiring only one opening per level.

An MLE claim of a matrix $\boldsymbol{A}$, written in matrix form, is a vector–matrix–vector product $v = \boldsymbol{l}^\top \boldsymbol{A}\boldsymbol{r}$ for tensor-structured vectors $\boldsymbol{l}$ and $\boldsymbol{r}$. Concretely, suppose we have a previously accumulated claim $\bar{v}_j = \bar{\boldsymbol{l}}_j{}^\top \boldsymbol{G}_j \bar{\boldsymbol{r}}_j$ and a new claim $v_j = \boldsymbol{l}_j^\top \boldsymbol{G}_j \boldsymbol{r}_j$ to be accumulated. For $z_j \xleftarrow{\$} \mathbb{F}$, we batch them as:

$$\bar{v}_j + z_j v_j = \bar{\boldsymbol{l}}_j{}^\top \boldsymbol{G}_j \bar{\boldsymbol{r}}_j + z_j \boldsymbol{l}_j^\top \boldsymbol{G}_j \boldsymbol{r}_j = \langle \mathsf{vec}(\boldsymbol{G}_j),\ \bar{\boldsymbol{r}}_j \otimes \bar{\boldsymbol{l}}_j + z_j \boldsymbol{r}_j \otimes \boldsymbol{l}_j \rangle.$$

An inner-product proof, *e.g.*, via the sumcheck protocol, then reduces this to an updated MLE claim $\bar{v}'_j = \bar{\boldsymbol{l}}'^{\top}_j \boldsymbol{G}_j \bar{\boldsymbol{r}}'_j$, which becomes the new accumulator. The marginal cost of proving generator-matrix MLE claims in each additional PCS opening is thus the cost of the inner-product proof, which is relatively small.

Unlike prior applications of accumulation schemes that aggregate complete, witness-dependent proof instances, we accumulate only the generator-matrix MLE claims produced by PCS openings. Because each generator matrix is fixed across openings, these claims can be accumulated across unrelated witnesses, including witnesses of different sizes at shared code levels, while the original PCS openings are processed independently.

Final Recursive Opening of Generator-Matrix MLE Claims via Code Switching. After accumulation, the remaining generator-matrix MLE claims must be opened to finalize the proof. With suitable arithmetization, each generator matrix $\boldsymbol{G}_j$ is represented using a fixed dense matrix $\boldsymbol{H}_j$. Opening its MLE claims typically introduces a claim-dependent dense matrix $\boldsymbol{S}_j$ determined by earlier Fiat–Shamir challenges. Prior code-switching PCSs, such as Blaze and BrakingBase, commit to $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ using an auxiliary foldable code, such as the random foldable code underlying BaseFold. In contrast, `SwitchFold` reuses code switching to open the accumulated claims recursively within the original code sequence, therefore reducing the size of the opening problem at each level and eliminating the need for an auxiliary foldable code.

The idea is to commit to the arithmetized representation of $\boldsymbol{G}_j$ under the code generated by $\widehat{\boldsymbol{G}}_j$, which has the same dimensions as $\boldsymbol{G}_j$ but is derived from the next lower-level generator matrix $\boldsymbol{G}_{j-1}$.

A result underlying the RFC construction of BaseFold [46] shows that a linear code can be extended to one with twice the block length while incurring only a small loss in relative distance. Specifically, one random-foldable (RF) extension uses a random diagonal matrix $\boldsymbol{T}$ and maps a generator matrix $\boldsymbol{G}$ to:

$$\mathsf{RF}(\boldsymbol{T}, \boldsymbol{G}) := \begin{pmatrix} \boldsymbol{G} & \boldsymbol{T}\boldsymbol{G} \\ \boldsymbol{G} & -\boldsymbol{T}\boldsymbol{G} \end{pmatrix}.$$

Applying $\log_2 t$ RF extensions to $\boldsymbol{G}_{j-1}$ yields $\widehat{\boldsymbol{G}}_j$, while the corresponding diagonal data are collected into $\boldsymbol{T}_j$.

At level j, $\boldsymbol{T}_j$ contains the RF diagonal data carried from level $j+1$, $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ arise from opening the MLE claim on $\boldsymbol{G}_j$, and $\boldsymbol{W}_j$ is the witness carried from level $j+1$. These matrices are combined as:

$$\boldsymbol{U}_j := [\, \boldsymbol{T}_j \mid \boldsymbol{H}_j \mid \boldsymbol{S}_j \mid \boldsymbol{W}_j \,].$$

Together, these block commitments bind $\boldsymbol{U}_j$ under the code generated by $\widehat{\boldsymbol{G}}_j$.

A code-switching protocol compresses $\boldsymbol{U}_j$ to a vector $\boldsymbol{u}_j$ and switches it from $\widehat{\mathcal{C}}_j$ to $\widehat{\mathcal{C}}^t_{j-1}$. The vector $\boldsymbol{u}_j$ is reshaped as the carried witness $\boldsymbol{W}_{j-1}$ and committed under $\widehat{\mathcal{C}}^t_{j-1}$. The code switch produces an MLE claim on $\widehat{\boldsymbol{G}}_j$, which the RF structure reduces to diagonal data forming $\boldsymbol{T}_{j-1}$ and an MLE claim on

the smaller generator matrix $\boldsymbol{G}_{j-1}$. Together with the claim on $\boldsymbol{W}_{j-1}$, these claims define an opening problem of the same form at level $j-1$. The recursion proceeds through codes with decreasing block lengths until the remaining claims on $\widehat{\boldsymbol{G}}_0 = \boldsymbol{G}_0$ are evaluated directly by the verifier.

At each recursive level, the block length, and hence the size of the problem, decreases by a factor of t. Because the reduction imposes no structural requirements on the code sequence, the final opening phase is also code-agnostic.

Instantiation with Brakedown Codes and QA Codes. In the `SwitchFold` framework, prover time is governed primarily by code encoding time, whereas proof size and verifier time depend on code relative distance. We therefore instantiate `SwitchFold` with two efficiently encodable code families: the well-known Brakedown codes [28] and the recently proposed quasi-Abelian (QA) codes [36]. Both code families can be instantiated over arbitrary prime fields, making them well suited to zkSNARK applications in data science. The resulting PCSs are called `BrakeFold` and `QAFold`, respectively.

For each code family, we develop an efficient arithmetization that represents each generator matrix $\boldsymbol{G}_j$ by a dense matrix $\boldsymbol{H}_j$ and, when necessary, a claim-dependent auxiliary matrix $\boldsymbol{S}_j$ for proving MLE evaluations. We instantiate these matrices for both code families, implement the resulting PCSs, and experimentally evaluate their prover time, proof size, and verifier time.

2 Preliminaries

2.1 Notations

We work over a finite field $\mathbb{F}$ of order q and denote its multiplicative group by $\mathbb{F}^{\times}$. Matrices are denoted by bold uppercase letters (*e.g.*, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{M}, \boldsymbol{G}$), vectors by lowercase bold letters and are column vectors by default (*e.g.*, $\boldsymbol{r}$ is a column vector and $\boldsymbol{l}^{\top}$ is its transposed row vector), and scalar field elements by lowercase letters (*e.g.*, a, b, c). Sets are denoted using blackboard bold font: $\mathbb{N}$ for the natural numbers, $2^{\mathbb{N}}$ for the powers of two, and $\mathbb{R}$ for the real numbers. We write $[n]$ for $\{1, \ldots, n\}$ and $|\cdot|$ for cardinality.

Throughout this paper, we implicitly assume that all matrix and vector dimensions, denoted by n, m, k, and t, are powers of two. If this is not the case, matrices and vectors are padded with zeros to the next power of two. Matrix and vector entries are indexed by subscripts, with indices denoted by i and j, *e.g.*, $\boldsymbol{M} = \{M_{ij}\}$ and $\boldsymbol{m} = \{m_i\}$.

We use the following matrix and vector operations:

- Inner product $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$, allowing $\langle \boldsymbol{a}^{\top}, \boldsymbol{b} \rangle$ that mixes row and column vectors and Frobenius inner product $\langle \boldsymbol{A}, \boldsymbol{B} \rangle$ between matrices;
- Addition $(\boldsymbol{A} + \boldsymbol{B})$; Multiplication $(\boldsymbol{A}\boldsymbol{B})$; Transposition $\boldsymbol{A}^{\top}$;
- Kronecker (tensor) product $(\boldsymbol{A} \otimes \boldsymbol{B})$; Concatenation $\boldsymbol{C} = (\boldsymbol{A} || \boldsymbol{B})$;
- Column-major vectorization $\mathsf{vec}(\boldsymbol{A})$ and its inverse operation $\mathsf{vec}^{-1}_{m \times n}(\boldsymbol{a})$;
- Horizontal packing of matrices $[\boldsymbol{A}_1 \mid \boldsymbol{A}_2 \mid \cdots \mid \boldsymbol{A}_t]$.

A linear code is denoted by $\mathcal{C}$, with encoding function $\mathsf{Enc}[\mathcal{C}]$ and codeword space $\mathbb{C}$. The prover and the verifier are denoted by $\mathcal{P}$ and $\mathcal{V}$, respectively. A relation is denoted by $\mathcal{R} = \{$ public input : private input $\mid$ constraint $\}$.

2.2 Reduction Proof to MLE Claims

Vectors and matrices are identified with multilinear polynomials via their multilinear extensions; see Appendix A.1 for the definitions and properties of MLEs. Throughout the paper, *we use the matrix form of MLE evaluation.* For a vector $\boldsymbol{a} \in \mathbb{F}^n$, an MLE claim is an inner product $v = \langle \boldsymbol{a}, \boldsymbol{\xi} \rangle$. For a matrix $\boldsymbol{A} \in \mathbb{F}^{m \times n}$, an MLE claim is a vector–matrix–vector product $v = \boldsymbol{l}^\top \boldsymbol{A} \boldsymbol{r}$. The vectors $\boldsymbol{\xi}$, $\boldsymbol{l}$, and $\boldsymbol{r}$ are tensor-structured. For $\boldsymbol{x} \in \mathbb{F}^{\log_2 n}$, its *tensor expansion* is defined by:

$$\mathsf{ts}(\boldsymbol{x}) \coloneqq (1 - x_1, x_1) \otimes \cdots \otimes (1 - x_{\log_2 n}, x_{\log_2 n}) \in \mathbb{F}^n.$$

When there is no ambiguity, we also refer to such $\boldsymbol{\xi}$, $\boldsymbol{l}$, and $\boldsymbol{r}$ as evaluation points. Let $\mathbb{T}(n) \coloneqq \left\{ \mathsf{ts}(\boldsymbol{x}) \;\middle|\; \boldsymbol{x} \in \mathbb{F}^{\log_2 n} \right\}$ denote the set of length-n tensor expansions.

Definition 1 (Inner-Product Proof). *Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ be n-vectors and $c \in \mathbb{F}$. An* inner-product proof *is a proof protocol between $\mathcal{P}$ and $\mathcal{V}$ for the language:*

$$\mathcal{L}_{\mathsf{ip}} = \{\ c \in \mathbb{F}, \boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n \ \mid c = \langle \boldsymbol{a}, \boldsymbol{b} \rangle\ \}$$

Inner-product proofs can be instantiated either via the sumcheck protocol [44] or via folding techniques [22]. In either case, we have the following reduction.

Fact 1 (Inner Product Reduction Proof). *There exists an inner product proof with prover time $4n + O(\log n)$ field multiplications, proof size $O(\log n)$, and verifier time $O(\log n)$. The protocol uses $\log_2 n$ challenges $\boldsymbol{x} \in \mathbb{F}^{* \log_2 n}$ to reduce $\mathcal{L}_{\mathsf{ip}}$ to proving two independent MLE claims $v_1 \stackrel{?}{=} \langle \boldsymbol{a}, \mathsf{ts}(\boldsymbol{x}) \rangle \ \wedge \ v_2 \stackrel{?}{=} \langle \boldsymbol{b}, \mathsf{ts}(\boldsymbol{x}) \rangle$, where v_1 and v_2 are proof elements sent by the prover. The soundness error and knowledge soundness error are both bounded by $2 \log_2 n / (|\mathbb{F}| - 1)$.*

Recursive applications of inner-product proofs yield protocols for a broad class of matrix expressions. An example is the matrix multiplication proof [44]; the following fact extends matrix multiplications to general matrix expressions.

Fact 2 (Matrix-Expression Reduction Proof [22, Theorem 2]). *Any matrix expression represented by a directed acyclic graph (DAG) with a constant number of matrix operations, including addition, subtraction, scalar multiplication, transpose, matrix multiplication, Frobenius inner product, Hadamard product, element-wise inversion, Kronecker product, concatenation, and truncation, admits a reduction proof with linear prover time, logarithmic proof size, and logarithmic verifier time. The reduction proof reduces verification of the matrix expression to checking MLE claims of the leaf-node matrices. Both the soundness error and the knowledge-soundness error are bounded by $O(\log n / |\mathbb{F}|)$, where n denotes the maximum dimension of the matrices involved.*

2.3 Interleaved Code

Basic definitions of linear codes and code distance are recalled in Appendix A.2. For a linear code $\mathcal{C}$, we write $\mathsf{Enc}[\mathcal{C}]$ to denote its encoding function and define the corresponding interleaved code as follows.

Definition 2 (Interleaved Code). *Let $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$ be a linear code with generator matrix $\boldsymbol{G} \in \mathbb{F}^{n \times k}$, and let $\boldsymbol{M} = [\boldsymbol{m}_1 \mid \boldsymbol{m}_2 \mid \cdots \mid \boldsymbol{m}_t] \in \mathbb{F}^{k \times t}$ be a matrix. The t-*interleaved code $\mathcal{C}^t$ *maps $\boldsymbol{M}$ to a matrix $\boldsymbol{C} \in \mathbb{F}^{n \times t}$ defined as:*

$$\boldsymbol{C} := \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}) := [\ \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}_1) \mid \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}_2) \mid \cdots \mid \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}_t)\ ] = \boldsymbol{G}\boldsymbol{M}.$$

Let $\mathbb{C} \subseteq \mathbb{F}^n$ be the codeword space of a linear code. A matrix $\boldsymbol{C} \in \mathbb{F}^{n \times t}$ is a codeword of the interleaved code $\mathcal{C}^t$ iff every column of $\boldsymbol{C}$ is a codeword of $\mathcal{C}$. A standard proximity check for such interleaved codewords samples a random vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^t$ and reduces the claim $\boldsymbol{C} \in \mathbb{C}^t$ to the compressed claim $\boldsymbol{C}\boldsymbol{r} \in \mathbb{C}$.

Lemma 1 (Random Combination of Interleaved Codewords [11, Theorem 7]). *Let $\mathcal{C} \subseteq \mathbb{F}^n$ be a linear code with relative distance δ, and let $\boldsymbol{C} \in \mathbb{F}^{n \times t}$. If some column of $\boldsymbol{C}$ is ϵ-far from $\mathcal{C}$, then for every $0 \leq \delta' < \min\{\delta/3, \epsilon\}$:*

$$\Pr_{\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^t}[\delta_H(\boldsymbol{C}\boldsymbol{r}, \mathcal{C}) < \delta'] \leq \frac{1}{(\epsilon - \delta')|\mathbb{F}|}.$$

2.4 Oracle and PCS

We formulate polynomial commitment schemes as a special class of cryptographic *oracle protocols*. Unlike an ideal oracle, an oracle protocol commits to a message and later answers queries about that message with proofs authenticating the responses. This formulation is convenient for recursive PCS constructions, in which the opening protocol of one PCS may invoke other PCSs as MLE oracles, just as it may invoke point oracles.

An oracle protocol $\Pi_{\mathcal{O}}$ commits to a message $\boldsymbol{M} \in \mathbb{M}$ as a commitment $\mathcal{O} \in \mathbb{O}$ and subsequently responds to queries $\boldsymbol{Q} \in \mathbb{Q}$ regarding this message, together with a proof attesting to the correctness of the response. Its response is specified by an *answer function* $\mathcal{F} : \mathbb{M} \times \mathbb{Q} \to \mathbb{F}$, where $\mathcal{F}(\boldsymbol{M}, \boldsymbol{Q})$ is the unique valid answer to query $\boldsymbol{Q}$ on message $\boldsymbol{M}$.

Definition 3 (Oracle Protocol). *An* oracle protocol $\Pi_{\mathcal{O}}$ *over a field $\mathbb{F}$ consists of algorithms* $(\mathsf{Setup}, \mathsf{Comm}, \mathsf{Query})$*, an answer function $\mathcal{F} : \mathbb{M} \times \mathbb{Q} \to \mathbb{F}$, and a proximity relation $\mathcal{R}_{\mathsf{prox}} \subseteq \mathbb{O} \times \mathbb{M}$, which specifies consistency between an oracle commitment and a message. The algorithms have the following syntax.*

- $\mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q}) \to \mathsf{pp}$*: Given the security parameter λ, the message set $\mathbb{M}$, and the query set $\mathbb{Q}$, this algorithm generates the public parameters* pp*.*
- $\mathsf{Comm}(\mathsf{pp}, \boldsymbol{M}) \to \mathcal{O}$*: Takes a message $\boldsymbol{M} \in \mathbb{M}$; outputs an oracle commitment $\mathcal{O} \in \mathbb{O}$.*

– $\mathsf{Query}(\mathsf{pp}, \mathcal{O}, \boldsymbol{Q}; \boldsymbol{M}) \to (v, \mathsf{bit})$*: This is an interactive protocol between a prover* $\mathcal{P}$ *and a verifier* $\mathcal{V}$*. The verifier specifies a query* $\boldsymbol{Q} \in \mathbb{Q}$*. The prover, holding private input* $\boldsymbol{M} \in \mathbb{M}$*, sends to the verifier* $v = \mathcal{F}(\boldsymbol{M}, \boldsymbol{Q})$ *and proves that this value is consistent with the oracle commitment* $\mathcal{O}$*. The verifier outputs* $\mathsf{bit} \in \{\mathsf{true}, \mathsf{false}\}$*. In PCS terminology, executing the query algorithm is also called* ***opening*** *the polynomial commitment.*

Definition 4 (Multilinear PCS as MLE Oracle). *A linear-code-based multilinear polynomial commitment scheme is an MLE oracle instantiated from a linear code:* $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$*, where each symbol in* $\mathbb{F}$ *is represented by one Merkle leaf. Let* $\mathsf{dist}_{\mathcal{C}} : \mathbb{F}^n \times \mathbb{F}^n \to [0, 1]$ *be a normalized codeword distance, and define:*

$$\delta := \min_{\boldsymbol{m} \neq \boldsymbol{m}'} \mathsf{dist}_{\mathcal{C}}(\mathsf{Enc}[\mathcal{C}](\boldsymbol{m}), \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}'))$$

be the minimum distance of $\mathcal{C}$*. Fix a proximity radius* $0 < \epsilon \leq \delta/2$*. The MLE oracle is defined as follows:*

– *The message space is* $\mathbb{M} = \mathbb{F}^k$*.*
– *The query space is* $\mathbb{Q} = \mathbb{T}(k)$*.*
– *The answer function is the MLE evaluation:*

$$\mathcal{F}(\boldsymbol{m}, \boldsymbol{\xi}) := \langle \boldsymbol{\xi}, \boldsymbol{m} \rangle, \qquad \boldsymbol{m} \in \mathbb{F}^k, \quad \boldsymbol{\xi} \in \mathbb{T}(k).$$

The commitment algorithm is $\mathsf{Comm}(\mathsf{pp}, \boldsymbol{m}) = \mathsf{Merkle}\big(\mathsf{Enc}[\mathcal{C}](\boldsymbol{m})\big)$*.*

To define the proximity relation for possibly malformed commitments, let $\mathcal{T}$ *be an authenticated partial Merkle tree under a root* $\mathcal{O}_{\mathsf{P}}$*. Let* $\mathsf{dist}_{\mathcal{C},\perp}(\mathcal{T}, \boldsymbol{c})$ *be the extension of* $\mathsf{dist}_{\mathcal{C}}$ *to authenticated partial codewords, with every unknown leaf counted as an error. The proximity relation is defined by:*

$$\mathcal{R}_{\mathsf{prox}} := \{\ \mathcal{O}_{\mathsf{P}}\ :\ \boldsymbol{m}\ \mid \exists \mathcal{T}\ \textit{s.t.}\ \mathsf{Root}(\mathcal{T}) = \mathcal{O}_{\mathsf{P}}\ \wedge\ \mathsf{dist}_{\mathcal{C},\perp}(\mathcal{T}, \mathsf{Enc}[\mathcal{C}](\boldsymbol{m})) < \epsilon\ \}.$$

We denote the resulting linear-code-based MLE oracle by $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}]$*.*

When the message is represented as a matrix $\boldsymbol{M} \in \mathbb{F}^{k_1 \times k_2}$*, we identify it with* $\mathsf{vec}(\boldsymbol{M}) \in \mathbb{F}^{k_1 k_2}$*. The query* $\boldsymbol{r} \otimes \boldsymbol{l}$ *returns* $\boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r}$*.*

An oracle protocol should satisfy binding, completeness, computational soundness, and knowledge soundness; the corresponding definitions are deferred to Appendix A.3. We also recall the general definitions of completeness, soundness, computational soundness, knowledge soundness, round-by-round computational/knowledge soundness, and special soundness for proof and argument systems, in Appendices A.4–A.6, which will be used in our security proofs.

2.5 Interactive Oracle Proof

We generalize the notion of an interactive oracle proof (IOP) [10] by replacing ideal oracles with oracle protocols, in which the prover commits to each oracle and answers oracle queries with authentication proofs.

Definition 5 (Interactive Oracle Proof). *An* interactive oracle proof *is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ in which the prover may, in some rounds, send oracle commitments instead of sending the underlying messages explicitly. After receiving such an oracle commitment $\mathcal{O}$, the verifier may issue queries $\boldsymbol{Q}$ to the committed message, and the prover responds with the corresponding answers and proofs by executing the oracle-query protocol. $\mathcal{V}$ accepts iff all oracle queries and all remaining protocol checks accept.* [6]

Lemma 2 (Transforming Interactive Protocols into IOPs). *Let Π for a relation $\mathcal{R}$ be a public-coin interactive protocol that is computationally and knowledge sound with errors $\kappa^{\mathsf{cs}}_{\Pi}$ and $\kappa^{\mathsf{ks}}_{\Pi}$, respectively. Its transcript has the form: $\pi = (\alpha_0, x_1, \alpha_1, \ldots, x_s, \alpha_s)$, where $\{\alpha_j\}_{j=0}^{s}$ are prover messages, and $\{x_j\}_{j=1}^{s}$ are verifier challenges. Assume that all verifier checks can be deferred until the full transcript has been generated without changing the acceptance predicate.*

Let $\mathcal{J} \subseteq \{0, \ldots, s\}$ be a set of indices. Suppose that, for every $j \in \mathcal{J}$, the prover message α_j is replaced by sending an oracle commitment $\mathcal{O}_j$, generated by an oracle protocol $\Pi_{\mathcal{O}_j}$ that is knowledge sound with error $\kappa^{\mathsf{ks}}_{\mathcal{O}_j}$. Let Π_{IOP} denote the protocol obtained from Π by replacing the messages $\{\alpha_j\}_{j\in\mathcal{J}}$ with the corresponding oracle commitments and replacing all verifier checks involving these messages with the associated oracle-query protocols. Then Π_{IOP} is an IOP. Moreover, it is computationally sound and knowledge sound with errors at most:

$$\kappa^{\mathsf{cs}}_{\Pi_{\mathsf{IOP}}} \leq \kappa^{\mathsf{cs}}_{\Pi} + \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}, \quad \kappa^{\mathsf{ks}}_{\Pi_{\mathsf{IOP}}} \leq \kappa^{\mathsf{ks}}_{\Pi} + \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}.$$

The proof is given in Appendix C.1.

3 Recursive Code Switching

This section and the next present the `SwitchFold` framework and prove Theorem 1 in the introduction. Section 3.1 instantiates the code-switching technique of [39] for interleaved codes, and Section 3.2 applies it recursively across a sequence of geometrically shrinking codes. This process reduces a large top-level MLE claim to a small base-level MLE claim, together with MLE claims on the generator matrices along the recursion. Section 4 then describes how these generator-matrix claims are accumulated and recursively proved.

3.1 Code Switching between Interleaved Codes

In existing hash-based PCS constructions, code switching is typically used to transform an interleaved code $\mathcal{C}^t$ with little algebraic structure to a foldable

[6] We use a slightly more general definition of an IOP to simplify the security proofs for our recursive PCS. In the original definition [10], the prover sends the ideal oracle rather than a commitment to it, and query responses contain only the answers, without accompanying proofs. The original definition is recovered by treating the ideal oracle as trivially committed and the corresponding query proofs as empty.

code $\mathcal{C}''$, such as the random foldable code in BaseFold [46]. Our use of code switching differs in that the target code $\mathcal{C}''$ is a smaller interleaved code:

$$\mathcal{C}'^t : \mathbb{F}^{k' \times t} \to \mathbb{F}^{n' \times t}, \qquad k = tk'.$$

We replace the foldability of the target code by the assumption that an efficient MLE oracle is available for $\mathcal{C}'^t$. Because the block length of $\mathcal{C}'$ is a factor t smaller than that of $\mathcal{C}$, this step reduces the construction of an MLE oracle for a large linear code to the same problem for a smaller linear code.

Given an MLE oracle protocol $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$ for $\mathcal{C}'^t$, we construct an MLE oracle protocol $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}^t]}$ for $\mathcal{C}^t$ as follows, where t denotes the source interleaving width and $k = tk'$.

Recall that $\mathcal{C}^t$ encodes a matrix $\boldsymbol{M} \in \mathbb{F}^{k \times t}$ as $\mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}) = \boldsymbol{G}\boldsymbol{M} \in \mathbb{F}^{n \times t}$. Given tensor vectors $\boldsymbol{l} \in \mathbb{T}(k)$ and $\boldsymbol{r} \in \mathbb{T}(t)$, the prover aims to prove the claimed MLE value $v = \boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r}$. To this end, the prover first sends the intermediate vector $\boldsymbol{u}^\top = \boldsymbol{l}^\top \boldsymbol{M} \in \mathbb{F}^t$. The verifier can then verify the MLE claim by checking ① $v = \langle \boldsymbol{u}, \boldsymbol{r} \rangle$ and that the claimed $\boldsymbol{u}^\top$ is indeed equal to $\boldsymbol{l}^\top \boldsymbol{M}$.

To prove $\boldsymbol{u}^\top = \boldsymbol{l}^\top \boldsymbol{M}$, we leverage Freivalds' technique [25]. The verifier samples $\tilde{\boldsymbol{r}} \xleftarrow{\$} \mathbb{F}^t$ and verify the claim $\boldsymbol{u}^\top = \boldsymbol{l}^\top \boldsymbol{M}$ by checking the scalar identity $\langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle = \boldsymbol{l}^\top \boldsymbol{M} \tilde{\boldsymbol{r}}$, whose LHS is computed directly by the verifier as ② $\tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle$. Let $\boldsymbol{m} = \boldsymbol{M}\tilde{\boldsymbol{r}} \in \mathbb{F}^k$. The remaining task is therefore to prove the two claims:

$$③ \qquad \tilde{u} = \langle \boldsymbol{l}, \boldsymbol{m} \rangle, \qquad ④ \qquad \boldsymbol{m} = \boldsymbol{M}\tilde{\boldsymbol{r}}.$$

Claim ③ is exactly an MLE claim of the vector $\boldsymbol{m}$ at the point corresponding to $\boldsymbol{l}$. Hence, the prover can commit to $\boldsymbol{m}$ using the MLE oracle under the smaller code $\mathcal{C}'^t$ and opens this commitment at $\boldsymbol{l}$.

Claim ④ is proved through checking the codeword consistency of both sides. Let $\boldsymbol{C} = \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}) = \boldsymbol{G}\boldsymbol{M} \in \mathbb{F}^{n \times t}$ be the codeword under $\mathcal{C}^t$. By linearity:

$$\mathsf{Enc}[\mathcal{C}](\boldsymbol{m}) \;=\; \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}\tilde{\boldsymbol{r}}) \;=\; \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M})\tilde{\boldsymbol{r}} \;=\; \boldsymbol{C}\tilde{\boldsymbol{r}}.$$

It suffices to prove that the $\mathcal{C}$-codeword of the newly committed vector $\boldsymbol{m}$ is consistent with the random linear combination $\boldsymbol{C}\tilde{\boldsymbol{r}}$ of the original codeword.

Consistency is ensured by a proximity check. The verifier samples ℓ rows and lets $\boldsymbol{\Xi} \in \{0,1\}^{\ell \times n}$ be the corresponding row-selection matrix. The prover proves:

$$⑤ \qquad \boldsymbol{\Xi}\boldsymbol{G}\boldsymbol{m} \;=\; \boldsymbol{\Xi} \cdot \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}) \;=\; \boldsymbol{\Xi}\boldsymbol{C}\tilde{\boldsymbol{r}}.$$

The RHS is obtained by point queries to the point-oracle commitment to $\boldsymbol{C}$. The LHS is reduced, by Fact 2, to MLE claims of $\boldsymbol{\Xi}$, $\boldsymbol{G}$, and $\boldsymbol{m}$. Since $\boldsymbol{\Xi}$ has only ℓ nonzero entries, its MLE value is computed directly by the verifier in $O(\ell \log n)$ time. The MLE claims of $\boldsymbol{G}$ and $\boldsymbol{m}$ are proved by their respective MLE oracles; in particular, $\boldsymbol{m}$ is committed using $\mathcal{C}'^t$. Consequently, the original MLE query for $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}^t]$ is reduced to an MLE query for $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]$, MLE queries to the committed generator matrix $\boldsymbol{G}$, and point queries to the original codeword $\boldsymbol{C}$.

This procedure is detailed in Algorithm 2, Appendix B.1.

For the source MLE oracle, instantiate Definition 4 with the encoder $\mathcal{C}^t$. We equip its codewords with the columnwise distance:

$$\mathsf{dist}_{\mathcal{C}^t}(\boldsymbol{A}, \boldsymbol{B}) := \max_{j \in [t]} \delta_H(\boldsymbol{a}_j, \boldsymbol{b}_j),$$

where $\boldsymbol{a}_j$ and $\boldsymbol{b}_j$ are the j-th columns of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. We instantiate the proximity relation of Definition 4 with the proximity radius $\epsilon = \delta/2$.

The following theorem summarizes its efficiency and security.

Theorem 2 (Code Switching between Interleaved Codes). *Let $\mathcal{C} : \mathbb{F}^k \mapsto \mathbb{F}^n$ and $\mathcal{C}' : \mathbb{F}^{k'} \mapsto \mathbb{F}^{n'}$ be linear codes, and let $t \in \mathbb{N}$ satisfy $k = tk'$. Let $\boldsymbol{G} \in \mathbb{F}^{n \times k}$ be the generator matrix and δ be the relative distance of $\mathcal{C}$. Assume that $n = \Theta(k)$ and that both $\boldsymbol{Ga}$ and $\boldsymbol{b}^\top \boldsymbol{G}$ can be computed in $O(k)$ time. Let $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$ be an MLE-oracle protocol for $\mathcal{C}'^t$ with prover time $\mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$, proof size $\mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$, verifier time $\mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$, computational soundness error $\kappa^{\mathsf{cs}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$, and knowledge soundness error $\kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$. Let $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$ be an MLE-oracle protocol for $\boldsymbol{G}$ with similar measures $\mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$, $\mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$, $\mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$, $\kappa^{\mathsf{cs}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$, $\kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$. Fix a proximity threshold $0 < \delta' < \delta/3$, and let $\ell = \left\lceil \frac{-\lambda}{\log_2(1-\delta')} \right\rceil$. Instantiate the source MLE-oracle relation as in Definition 4. Then the MLE-oracle protocol $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}^t]}$ whose query algorithm is given in Algorithm 2 has the following complexity and soundness bounds:*

- *Prover time:* $O(kt) + \mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$.
- *Proof size:* $O\big(\ell(t + \log k)\big) + \mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$.
- *Verifier time:* $O\big(\ell(t + \log k)\big) + \mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$.
- *Computational- and knowledge-soundness errors:* $2^{-\lambda} + O\Big(\frac{\log k + (\delta/3 - \delta')^{-1}}{|\mathbb{F}|}\Big) + \kappa_{\mathsf{Hash}} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$.

Here, κ_{Hash} is the adaptive Merkle-authentication-path error.

Proof of this theorem is provided in Appendix C.2.

3.2 Recursive Application of Code Switching

The code-switching protocol reduces the construction of an MLE oracle for an interleaved code $\mathcal{C}^t$ to the same task for a smaller interleaved code $\mathcal{C}'^t$, whose block length is smaller by a factor of t. However, one code-switching step alone does not yield polylogarithmic proof size and verifier time. To obtain polylogarithmic complexity, one would need to set the interleaving parameter to $t = O(\log N)$, where N is the original witness size, and the target code must itself admit an MLE oracle with polylogarithmic proof size and verifier time.

Blaze and BrakingBase achieve this by switching from $\mathcal{C}^t$ to a foldable target code $\mathcal{C}''$. In contrast, we apply code switching recursively through a sequence of codes with decreasing block lengths. Each step reduces the block length by a factor of t, and the target code at one level becomes the source code for the next

level. After d levels, the opening is reduced to a small base code $\mathcal{C}_0$, such as the RS code with small block length.

Specifically, we consider a sequence of linear codes:

$$\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}, \qquad k_j = tk_{j-1} \quad \text{for } j \in [d].$$

For each j, let $\boldsymbol{G}_j$ denote the generator matrix of $\mathcal{C}_j$. Assume that we are given MLE-oracle access to the generator matrices $\{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)\}_{j=1}^{d}$.

The recursive code-switching protocol is given in Algorithm 1, Appendix B.1. It recursively reduces an opening for $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}_d^t]$ to two final checks:

- a direct opening of the base-level witness $\boldsymbol{M}_0$, and
- a sequence of MLE claim checks of the generator matrices $\{\boldsymbol{G}_j\}_{j=1}^{d}$.

Consequently, the original PCS opening is decomposed into a logarithmic-depth chain of smaller openings, terminating at the base code. Here, we describe the recursive code-switching protocol with ideal MLE oracles for the generator matrices $\{\boldsymbol{G}_j\}_{j=1}^{d}$; later in Section 4, these generator-matrix claims are accumulated and then recursively proved.

Theorem 3 (Recursive Code Switching). *Let $d, k_0, n_0, t \in \mathbb{N}$, and define $k_j = k_0 t^j$, $n_j = n_0 t^j$ for $j \in [d]$. Let $\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}$ be a sequence of linear codes with relative distance at least δ. For each $j \in [d]$, let $\boldsymbol{G}_j \in \mathbb{F}^{n_j \times k_j}$ be the generator matrix of $\mathcal{C}_j$, and let $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$ be an MLE-oracle protocol for $\boldsymbol{G}_j$ with prover time $\mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$, proof size $\mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$, verifier time $\mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$, computational soundness error $\kappa^{\mathsf{cs}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$, and knowledge soundness error $\kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$. Assume that the extractor of each generator-matrix MLE-oracle protocol preserves the transcript prefix preceding that subprotocol. Let $0 < \delta' < \delta/3$ and $\ell = -\lambda/\log_2(1-\delta')$. Then the MLE-oracle protocol over $\mathcal{C}_d^t$ via recursive code switching in Algorithm 1 has the following complexity and soundness bounds:*

- *Prover time: $O(k_d t) + \sum_{j=1}^{d} \mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$.*
- *Proof size: $O\big(d\ell(t + \log k_d)\big) + \sum_{j=1}^{d} \mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$.*
- *Verifier time: $O\big(d\ell(t + \log k_d)\big) + \sum_{j=1}^{d} \mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$.*
- *Computational- and knowledge-soundness errors: $d \cdot 2^{-\lambda} + O\Big(\frac{d(\log k_d + 1/(\delta/3-\delta'))}{|\mathbb{F}|}\Big) + d \cdot \kappa_{\mathsf{Hash}} + \sum_{j=1}^{d} \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}$.*

The knowledge extractor preserves the transcript prefix at the boundaries between consecutive code-switching subprotocols; no round-by-round property is required within an individual subprotocol. Here, κ_{Hash} is the Merkle-path-consistency error of one Merkle commitment.

The proof of this theorem is given in Appendix C.3.

Algorithm 1: Recursive Code-Switching Protocol

Public input: $d, k_0, n_0, t \in \mathbb{N}$, $\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}$ with generator matrices $\{\boldsymbol{G}_j\}_{j=0}^{d}$, point $\boldsymbol{l} \in \mathbb{T}(k_d), \boldsymbol{r} \in \mathbb{T}(t)$, relative-distance bound δ and proximity threshold $0 < \delta' < \delta/3$

Private input: witness matrix $\boldsymbol{M} \in \mathbb{F}^{k_d \times t}$

Oracle access: $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}_d^t](\boldsymbol{M}))$, $\{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)\}_{j=1}^{d}$

Output: claimed MLE value v, true or false

1 $\mathcal{P} \to \mathcal{V}$: $v \leftarrow \boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r} \in \mathbb{F}$;

2 Initialize: $v_d \leftarrow v$; $\boldsymbol{l}_d \leftarrow \boldsymbol{l}$; $\boldsymbol{r}_d \leftarrow \boldsymbol{r}$; $\boldsymbol{M}_d \leftarrow \boldsymbol{M}$; $\ell \leftarrow \lceil -\lambda / \log_2(1-\delta') \rceil$;

3 **for** j from d down to 1 **do**

4 $\quad \mathcal{P}$ computes $\boldsymbol{u}_j^\top \leftarrow \boldsymbol{l}_j^\top \boldsymbol{M}_j \in \mathbb{F}^t$;

5 $\quad \mathcal{P} \to \mathcal{V}$: $\boldsymbol{u}_j \in \mathbb{F}^t$;

6 $\quad \mathcal{V}$ checks $v_j \stackrel{?}{=} \langle \boldsymbol{u}_j, \boldsymbol{r}_j \rangle$; if not, **return** false;

7 $\quad \mathcal{V} \to \mathcal{P}$: sample $\tilde{\boldsymbol{r}}_j \xleftarrow{\$} \mathbb{F}^t$;

8 $\quad \mathcal{P}$ and $\mathcal{V}$ compute $\tilde{u}_j \leftarrow \langle \boldsymbol{u}_j, \tilde{\boldsymbol{r}}_j \rangle \in \mathbb{F}$;

9 $\quad \mathcal{P}$ computes $\boldsymbol{m}_j \leftarrow \boldsymbol{M}_j \tilde{\boldsymbol{r}}_j \in \mathbb{F}^{k_j}$;

10 $\quad \mathcal{P}$ reshapes $\boldsymbol{M}_{j-1} \leftarrow \mathsf{vec}^{-1}_{k_{j-1} \times t}(\boldsymbol{m}_j) \in \mathbb{F}^{k_{j-1} \times t}$;

11 $\quad \mathcal{P} \to \mathcal{V}$: $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}_{j-1}^t](\boldsymbol{M}_{j-1})) \leftarrow \mathsf{Merkle}(\mathsf{Enc}[\mathcal{C}_{j-1}^t](\boldsymbol{M}_{j-1}))$;

12 $\quad \mathcal{V}$ samples ℓ indices independently from $[n_j]$, denoted by $\mathcal{I}_j \in [n_j]^\ell$;

13 $\quad \mathcal{V} \to \mathcal{P}$: $\mathcal{I}_j$;

14 $\quad \mathcal{P} \to \mathcal{V}$: $\boldsymbol{\iota}_j \in \mathbb{F}^\ell$, the restriction of $\mathsf{Enc}[\mathcal{C}_j](\boldsymbol{m}_j)$ to the positions in $\mathcal{I}_j$;

15 $\quad \mathcal{P}$ and $\mathcal{V}$ derive the selection matrix $\boldsymbol{\Xi}_j \in \{0,1\}^{\ell \times n_j}$ from $\mathcal{I}_j$;

16 $\quad \mathcal{P}$ and $\mathcal{V}$ verify $\boldsymbol{\iota}_j$ by querying $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}_j^t](\boldsymbol{M}_j))$ at the rows indexed by $\mathcal{I}_j$, and computing their corresponding inner products with $\tilde{\boldsymbol{r}}_j$;

17 $\quad \mathcal{P}$ and $\mathcal{V}$ run matrix multiplication proof protocols to prove: $\boldsymbol{\iota}_j = \boldsymbol{\Xi}_j \boldsymbol{G}_j \boldsymbol{m}_j \wedge \tilde{u}_j = \langle \boldsymbol{l}_j, \boldsymbol{m}_j \rangle$, reducing to checking MLE claims of $\boldsymbol{\iota}_j$, $\boldsymbol{\Xi}_j$, $\boldsymbol{G}_j$, and $\boldsymbol{m}_j$. For these claims, $\boldsymbol{G}_j$ evaluated at $(\boldsymbol{l}_{Gj}, \boldsymbol{r}_{Gj})$ gives v_{Gj}, and $\boldsymbol{m}_j$ evaluated at $\boldsymbol{\xi}_j$ gives v_{j-1}, where $\boldsymbol{\xi}_j = \boldsymbol{r}_{j-1} \otimes \boldsymbol{l}_{j-1}$, $\boldsymbol{r}_{j-1} \in \mathbb{T}(t)$, $\boldsymbol{l}_{j-1} \in \mathbb{T}(k_{j-1})$;

18 $\quad \mathcal{V}$ checks the MLE claims of $\boldsymbol{\iota}_j$ and $\boldsymbol{\Xi}_j$ directly;

19 $\mathcal{P} \to \mathcal{V}$: $\boldsymbol{M}_0$;

20 $\mathcal{V}$ checks $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}_0^t](\boldsymbol{M}_0)) \stackrel{?}{=} \mathsf{Merkle}(\boldsymbol{G}_0 \boldsymbol{M}_0)$ and directly verifies $v_0 = \boldsymbol{l}_0^\top \boldsymbol{M}_0 \boldsymbol{r}_0$;

21 $\mathcal{P}$ and $\mathcal{V}$ invoke $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}.\mathsf{Query}$ to prove and verify the MLE claims $\{v_{Gj} = \boldsymbol{l}_{Gj}^\top \boldsymbol{G}_j \boldsymbol{r}_{Gj}\}_{j=1}^{d}$;

22 **return** true if all checks and subprotocols accept; otherwise false;

4 Proving Generator-Matrix MLE Claims

4.1 Accumulating Generator-Matrix MLE Claims

Recursive code switching reduces MLE claims under one code to MLE claims under a smaller code, while introducing additional MLE claims on the generator matrix of the source code. In particular, Line 21 of Algorithm 1 requires proving MLE claims for all generator matrices in the code sequence. Thus, the claimed

linear prover time relies on the assumption that each generator-matrix MLE claim can be proved in time linear in the corresponding block length.

This assumption of linear prover time is reasonable from an asymptotic perspective. If the encoding algorithm of a code with generator matrix $\boldsymbol{G}$ runs in time linear in the block length, then the generator-matrix MLE evaluation $v = \boldsymbol{l}^\top \boldsymbol{G}\boldsymbol{r} = \langle \boldsymbol{l}, \mathsf{Enc}[\mathcal{C}](\boldsymbol{r})\rangle$ can also be computed in linear time. When the encoder admits a linear-size arithmetic-circuit representation, techniques such as R1CS [9] or Plonk [26] can express this computation as a constraint system. However, this generic approach may incur large concrete constants. As a result, generator-matrix MLE openings may dominate the prover cost in Algorithm 1, offsetting the gains from recursive code switching.

We mitigate this cost by exploiting the fact that the generator matrices are fixed across repeated PCS openings. Suppose the same code sequence is used to commit to multiple witness matrices $\boldsymbol{M}^{(1)}, \ldots, \boldsymbol{M}^{(T)}$. Although the witnesses differ, their openings generate MLE claims to the same generator-matrix oracles $\{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)\}_{j=1}^{d}$. Instead of proving these generator-matrix MLE claims independently for each witness, we accumulate them across witnesses and query the corresponding MLE oracle only once per code level during finalization.

This accumulation model is well suited to many zkSNARK applications, particularly blockchain Layer-2 zk-rollups. Proofs for multiple computations within a block may use the same code sequence, allowing the sequencer to accumulate their generator-matrix MLE claims and prove them at block finalization with only one opening per code level.

At a high level, an accumulator acc maintains an aggregate of previous generator-matrix MLE claims. An Accum protocol updates this state after each PCS opening, and a Decide protocol proves the accumulated claims at finalization. When a new PCS opening arrives at time i, Algorithm 1 produces a fresh set of generator-matrix MLE claims, denoted by $u^{(i)}$. The accumulation step folds these new claims into the previous accumulator $\mathsf{acc}^{(i-1)}$, producing an updated accumulator $\mathsf{acc}^{(i)}$, as illustrated below.

$$\boxed{\begin{array}{l}\mathsf{acc}^{(i-1)} = \\ \{\bar{v}_j^{(i-1)}, \bar{\boldsymbol{l}}_j^{(i-1)}, \bar{\boldsymbol{r}}_j^{(i-1)}\}_{j=1}^{d}\end{array}} + \boxed{\begin{array}{l}u^{(i)} = \\ \{v_j^{(i)}, \boldsymbol{l}_j^{(i)}, \boldsymbol{r}_j^{(i)}\}_{j=1}^{d}\end{array}} \Longrightarrow \boxed{\begin{array}{l}\mathsf{acc}^{(i)} = \\ \{\bar{v}_j^{(i)}, \bar{\boldsymbol{l}}_j^{(i)}, \bar{\boldsymbol{r}}_j^{(i)}\}_{j=1}^{d}\end{array}}$$

A formal definition of an accumulation scheme, which comprises an Accum protocol and a Decide protocol, is given in Appendix A.7. Sections 4.2 and 4.3 instantiate the accumulation scheme for generator-matrix MLE claims.

4.2 Updating the Generator-Matrix Accumulator

We first instantiate the Accum protocol for generator-matrix MLE claims. At step $i \in [T]$, for each $j \in [d]$, the protocol accumulates the previous generator-matrix MLE claims aggregated in the accumulator together with the fresh generator-matrix MLE claims generated by the i-th PCS opening:

$$\bar{v}_j^{(i-1)} = \bar{\boldsymbol{l}}_j^{(i-1)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i-1)} \quad \wedge \quad v_j^{(i)} = \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \boldsymbol{r}_j^{(i)}. \tag{1}$$

The goal is to fold these two claims into a single updated claim:

$$\bar{v}_j^{(i)} = \bar{\boldsymbol{l}}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)},$$

which is then stored in the updated accumulator. Thus, each accumulation step batches two MLE openings into one, which is a standard technique in zkSNARK design; see, *e.g.*, [20, Section 3.8].

However, a direct application of the standard technique represents each matrix MLE claim as an inner product against $\boldsymbol{l}_j^{(i)} \otimes \boldsymbol{r}_j^{(i)}$. This vector has dimension $n_j k_j = O(k_j^2)$, resulting in prohibitive quadratic prover work. We therefore exploit the fact that both $\boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j$ and $\boldsymbol{G}_j \boldsymbol{r}_j^{(i)}$ can be computed in $O(k_j)$ time to reduce the accumulation cost to $O(k_j)$. Appendix B.2 describes the efficient method for batching generator-matrix MLE claims, while Algorithm 3 in Appendix B.1 gives the full Accum protocol.

Theorem 4 (Security of Generator-Matrix MLE Claim Accumulation). *Let* $\{\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}\}_{j=1}^d$ *be a sequence of MLE-oracle protocols for the generator matrices* $\{\boldsymbol{G}_j\}_{j=1}^d$. *Let* Accum *be the protocol in Algorithm 3, and let* Decide *verify the final accumulator* $\mathsf{acc}^{(T)}$ *by invoking* $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G}_j)}.\mathsf{Query}$ *for every* $j \in [d]$ *on the MLE claim stored in* $\mathsf{acc}^{(T)}$. *Then the resulting accumulation scheme is complete, computationally sound, and knowledge sound. Moreover, each additional accumulated instance increases both the computational-soundness and knowledge-soundness errors by at most* $O\Big(\frac{d \log k_d}{|\mathbb{F}|}\Big)$, *in addition to the soundness errors of proving the final generator-matrix MLE claims performed by* Decide.

The proof of Theorem 4 is provided in Appendix C.4.

4.3 Recursive Reduction of Generator-Matrix MLE Claims

It remains to design the final Decide protocol for the accumulated generator-matrix MLE claims. After all accumulation steps, we omit the superscript (T) and denote the final accumulated claims by $\{v_j, \boldsymbol{l}_j, \boldsymbol{r}_j\}_{j=1}^d$. The goal of Decide is to produce a proof for $v_j = \boldsymbol{l}_j^\top \boldsymbol{G}_j \boldsymbol{r}_j,\ \forall j \in [d]$.

As discussed in Section 4.1, if the code $\mathcal{C}_j$ admits a linear-time encoder, then the MLE value $\boldsymbol{l}^\top \boldsymbol{G}_j \boldsymbol{r} = \langle \boldsymbol{l}, \mathsf{Enc}[\mathcal{C}_j](\boldsymbol{r}) \rangle$ can be evaluated in linear time. A generic strategy is to arithmetize the encoder as a constraint system. For concrete code families, however, code-specific arithmetizations can provide better efficiency.

For each $j \in [d]$, we assume that the generator matrix $\boldsymbol{G}_j$ admits an arithmetization represented by a fixed dense matrix $\boldsymbol{H}_j \in \mathbb{F}^{k_j \times h_j}$. We further assume a reduction proof protocol Π_{G_j} that can reduce an MLE claim on $\boldsymbol{G}_j$ into MLE claims on $\boldsymbol{H}_j$ and a claim-dependent auxiliary matrix $\boldsymbol{S}_j \in \mathbb{F}^{k_j \times s_j}$. For example, when Π_{G_j} is derived from a constraint system, $\boldsymbol{H}_j$ contains the fixed constraint matrices, while $\boldsymbol{S}_j$ contains the corresponding witness assignment. Code-specific arithmetization may enable more efficient reductions. Section 5 will present optimized instantiations for the Brakedown and QA codes.

Random-foldable extension. At level j, the matrices $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ are committed under an auxiliary code $\widehat{\mathcal{C}}_j$, whose generator matrix $\widehat{\boldsymbol{G}}_j$ is derived from $\boldsymbol{G}_{j-1}$. Code switching reduces this opening to one under the lower-level auxiliary interleaved code $\widehat{\mathcal{C}}^t_{j-1}$, while introducing an MLE claim on $\widehat{\boldsymbol{G}}_j$. By construction, this claim reduces to an MLE claim on $\boldsymbol{G}_{j-1}$. These claims form the same opening problem at level $j-1$, allowing the reduction to recurse to the base level.

We construct $\widehat{\boldsymbol{G}}_j$ from $\boldsymbol{G}_{j-1}$ using the random-foldable (RF) extension underlying the RFC construction of BaseFold [46]. Given a generator matrix $\boldsymbol{G} \in \mathbb{F}^{n\times k}$ and a random diagonal matrix $\boldsymbol{T} \in \mathbb{F}^{n\times n}$, one RF extension is defined as follows:

$$\mathsf{RF}(\boldsymbol{T}, \boldsymbol{G}) := \begin{pmatrix} \boldsymbol{G} & \boldsymbol{TG} \\ \boldsymbol{G} & -\boldsymbol{TG} \end{pmatrix} \in \mathbb{F}^{2n\times 2k}.$$

Fact 3 (Distance of RF Extension [46, Theorem 2])**.** *Fix a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq 2^{10}$ and a security parameter $\lambda \in \mathbb{N}$. Let $\boldsymbol{G} \in \mathbb{F}^{n\times k}$ generate a linear code of relative distance δ, and sample the diagonal entries of $\boldsymbol{T} \in \mathbb{F}^{n\times n}$ independently and uniformly from $\mathbb{F}^*$. Then, except with probability at most $2^{-\lambda}$, the code generated by $\mathsf{RF}(\boldsymbol{T}, \boldsymbol{G})$ has relative distance greater than:*

$$\delta - \frac{1}{n} - \frac{\lambda + 2.002\big((1-\delta)n + 1\big) + 1.2n}{2n\big(\log|\mathbb{F}| - 1.001\big)}.$$

Write $r := n_j/k_j = n_0/k_0$, and assume that r is an integer. Leveraging the RF extension, we construct $\widehat{\boldsymbol{G}}_j$ from $\boldsymbol{G}_{j-1}$ by applying $\log_2 t$ RF extensions parameterized by the public random diagonal matrices:

$$\boldsymbol{T}_{j,1} \in \mathbb{F}^{n_j/2\times n_j/2}, \boldsymbol{T}_{j,2} \in \mathbb{F}^{n_j/2^2\times n_j/2^2}, \ldots, \boldsymbol{T}_{j,\log_2 t} \in \mathbb{F}^{n_{j-1}\times n_{j-1}}.$$

Specifically, we define:

$$\widehat{\boldsymbol{G}}_j^{(0)} := \boldsymbol{G}_{j-1}, \qquad \widehat{\boldsymbol{G}}_j^{(i)} := \mathsf{RF}\Big(\boldsymbol{T}_{j,\log_2 t - i + 1}, \widehat{\boldsymbol{G}}_j^{(i-1)}\Big), \qquad \widehat{\boldsymbol{G}}_j := \widehat{\boldsymbol{G}}_j^{(\log_2 t)}. \quad (2)$$

Thus, $\widehat{\boldsymbol{G}}_j \in \mathbb{F}^{n_j\times k_j}$. At the base level, we set $\widehat{\boldsymbol{G}}_0 := \boldsymbol{G}_0$.

This construction supports a recursive reduction of MLE claims. Consider one RF extension, and write $\boldsymbol{l} = [\,(1-x_{l1})\boldsymbol{l}' \mid x_{l1}\boldsymbol{l}'\,]$, $\boldsymbol{r} = [\,(1-x_{r1})\boldsymbol{r}' \mid x_{r1}\boldsymbol{r}'\,]$. A direct expansion of the MLE claim gives:

$$\boldsymbol{l}^\top \mathsf{RF}(\boldsymbol{T}, \boldsymbol{G})\boldsymbol{r} = [(1-x_{r1})\boldsymbol{l}'^\top + (1-2x_{l1})x_{r1}\boldsymbol{l}'^\top \boldsymbol{T}]\boldsymbol{G}\boldsymbol{r}'. \quad (3)$$

The LHS is an MLE claim on $\mathsf{RF}(\boldsymbol{T}, \boldsymbol{G})$. The RHS is a matrix expression that Fact 2 reduces to MLE claims on $\boldsymbol{T}$ and $\boldsymbol{G}$. Applying this reduction successively through all $\log_2 t$ RF extensions reduces an MLE claim on $\widehat{\boldsymbol{G}}_j$ to claims on $\boldsymbol{T}_{j,1}, \ldots, \boldsymbol{T}_{j,\log_2 t}$ and an MLE claim on $\boldsymbol{G}_{j-1}$. The latter, batched with the MLE claim of $\boldsymbol{G}_j$ from the accumulation stage, are proved recursively as follows.

Commitments During the Setup and Proving Phases. Because each $\boldsymbol{T}_{j,i}$ is diagonal, an MLE claim on $\boldsymbol{T}_{j,i}$ reduces to an MLE claim on its diagonal

vector. For each $j \in \{0, \ldots, d-1\}$, we concatenate the diagonal entries of $\boldsymbol{T}_{j+1,1}, \ldots, \boldsymbol{T}_{j+1,\log_2 t}$. There are $\sum_{i=1}^{\log_2 t} \frac{n_{j+1}}{2^i} = n_j(t-1) = k_j r(t-1)$ entries in total, which we reshape as $\boldsymbol{T}_j \in \mathbb{F}^{k_j \times r(t-1)}$, $j \in \{0, \ldots, d-1\}$. At the outermost level, we set $\boldsymbol{T}_d \in \mathbb{F}^{k_d \times 0}$ to be an empty dummy matrix.

Because $\boldsymbol{T}_j$ and $\boldsymbol{H}_j$ are fixed independently of the MLE claim being opened, the transparent setup algorithm encodes them jointly under the code generated by $\widehat{\boldsymbol{G}}_j$ and includes the resulting point-oracle commitment in the public parameters. The encoded matrix underlying this commitment is:

$$\widehat{\boldsymbol{C}}_j^{\mathsf{setup}} := \widehat{\boldsymbol{G}}_j \left[\boldsymbol{T}_j \mid \boldsymbol{H}_j \right], \qquad j \in [d]. \tag{4}$$

The reduction protocol Π_{G_j} may additionally produce a claim-dependent matrix $\boldsymbol{S}_j$. We use $\boldsymbol{S}_j$ and $\widehat{\boldsymbol{C}}_j^{\mathsf{aux}}$ for the auxiliary matrix and its codewords:

$$\widehat{\boldsymbol{C}}_j^{\mathsf{aux}} := \widehat{\boldsymbol{G}}_j \boldsymbol{S}_j, \qquad j \in [d]. \tag{5}$$

For $j < d$, the protocol also receives a carried witness $\boldsymbol{W}_j \in \mathbb{F}^{k_j \times t}$ produced by the code switch at level $j+1$. As part of that code switch, the prover commits to $\boldsymbol{W}_j$ under the code generated by $\widehat{\boldsymbol{G}}_j$:

$$\widehat{\boldsymbol{C}}_j^{\mathsf{carry}} := \widehat{\boldsymbol{G}}_j \boldsymbol{W}_j, \qquad j \in [d-1]. \tag{6}$$

At the outermost level, there is no incoming carried witness, so $\boldsymbol{W}_d \in \mathbb{F}^{k_d \times 0}$ and $\widehat{\boldsymbol{C}}_d^{\mathsf{carry}} \in \mathbb{F}^{n_d \times 0}$ are empty matrices.

Recursive Code Switching in the Decide Protocol. At level j, the protocol Π_{G_j} reduces the accumulated MLE claim on $\boldsymbol{G}_j$ to claims on $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$. These claims are combined with the incoming MLE claims on $\boldsymbol{T}_j$ and $\boldsymbol{W}_j$ from level $j+1$, yielding a single MLE claim on:

$$\boldsymbol{U}_j := \left[\boldsymbol{T}_j \mid \boldsymbol{H}_j \mid \boldsymbol{S}_j \mid \boldsymbol{W}_j \right].$$

For $j < d$, $\boldsymbol{U}_j \in \mathbb{F}^{k_j \times (r(t-1)+t+h_j+s_j)}$.

The setup, auxiliary, and carry commitments jointly encode $\boldsymbol{U}_j$ under $\widehat{\mathcal{C}}_j$:

$$\widehat{\boldsymbol{C}}_j := \left[\widehat{\boldsymbol{C}}_j^{\mathsf{setup}} \mid \widehat{\boldsymbol{C}}_j^{\mathsf{aux}} \mid \widehat{\boldsymbol{C}}_j^{\mathsf{carry}} \right] = \widehat{\boldsymbol{G}}_j \boldsymbol{U}_j.$$

We apply the code-switching protocol of Section 3.1 to $\boldsymbol{U}_j$, reducing this opening under $\widehat{\mathcal{C}}_j$ to one under $\widehat{\mathcal{C}}_{j-1}^t$. The code switch reduces the original opening problem to an opening of $\boldsymbol{W}_{j-1}$ under the lower-level interleaved code $\widehat{\mathcal{C}}_{j-1}^t$, while introducing an additional MLE claim on the source code generator matrix $\widehat{\boldsymbol{G}}_j$.

Expanding the RF construction of $\widehat{\boldsymbol{G}}_j$ reduces its MLE claim to MLE claims on $\boldsymbol{T}_{j-1}$ and $\boldsymbol{G}_{j-1}$. The claim on $\boldsymbol{G}_{j-1}$ is accumulated with the accumulator produced by Accum at level $j-1$. Applying $\Pi_{G_{j-1}}$ to the resulting claim yields MLE claims on $\boldsymbol{H}_{j-1}$ and $\boldsymbol{S}_{j-1}$. Together with the claims on $\boldsymbol{T}_{j-1}$ and $\boldsymbol{W}_{j-1}$, these claims form an opening problem of the same form at level $j-1$.

The recursion continues until level 0, where the verifier evaluates the claims on the public matrices $\boldsymbol{T}_0$ and $\boldsymbol{G}_0$ directly and checks the opening of $\boldsymbol{W}_0$ against its carry commitment. The full protocol is given in Algorithm 4, Appendix B.1.

Theorem 5 (Recursive Opening of Generator-Matrix MLE Claims). *For each $j \in [d]$, let $\boldsymbol{G}_j \in \mathbb{F}^{n_j \times k_j}$ be the generator matrix of $\mathcal{C}_j$, where $k_j = tk_{j-1}$ and $n_j = tn_{j-1}$, and write $r := n_j/k_j = n_0/k_0 \in \mathbb{N}$. Assume that $k_0, n_0 = O(1)$. Assume that there exists a reduction protocol Π_{G_j} that reduces MLE claims to $\boldsymbol{G}_j$ to MLE claims to a dense representation matrix $\boldsymbol{H}_j \in \mathbb{F}^{k_j \times h_j}$ and an auxiliary matrix $\boldsymbol{S}_j \in \mathbb{F}^{k_j \times s_j}$. Assume further that Π_{G_j} is a computationally sound reduction and has prover time $O(k_j)$, proof size $O(\log k_j)$, verifier time $O(\log k_j)$, and computational soundness error $O\left(\frac{\log k_j}{|\mathbb{F}|}\right)$. Suppose that $h_j \le h$ and $s_j \le s$ for all $j \in [d]$. For each j, let $\widehat{\boldsymbol{G}}_j$ be constructed from $\boldsymbol{G}_{j-1}$ and $\boldsymbol{T}_{j,1}, \ldots, \boldsymbol{T}_{j,\log_2 t}$ as in Eq. (2), and let $\widehat{\delta}_j$ be the relative distance of its code. For each $j \in [d]$, fix a proximity threshold $0 < \widehat{\delta}'_j < \widehat{\delta}_j/3$. Let $\ell_j := \left\lceil \frac{-\lambda}{\log_2\left(1-\widehat{\delta}'_j\right)} \right\rceil$. Then Algorithm 4 has the following complexity and soundness bounds:*

- *Prover time: $O(\log t \cdot k_d(h + s + rt))$.*
- *Proof size and verifier time: $O\left(\sum_{j=1}^{d} \ell_j \left(h + s + rt + \log k_j\right)\right)$.*
- *Computational- and knowledge-soundness errors:*

$$d \cdot 2^{-\lambda} + O\left(\frac{1}{|\mathbb{F}|} \sum_{j=1}^{d} \left(h + s + rt + \log k_j + \left(\widehat{\delta}_j/3 - \widehat{\delta}'_j\right)^{-1}\right)\right) + 3d \cdot \kappa_{\mathsf{Hash}}.$$

Here, κ_{Hash} is the adaptive Merkle-path-consistency error of one commitment.

The proof of Theorem 5 is provided in Appendix C.5

4.4 Putting Everything Together

We now combine the three components developed above. Recursive code switching (Algorithm 1) reduces each opening claim to generator-matrix MLE claims. The Accum protocol (Algorithm 3, Appendix B.1) accumulates these generator-matrix claims across multiple openings. Finally, the Decide protocol (Algorithm 4, Appendix B.1) proves the accumulated generator-matrix claims. Together, these components form the `SwitchFold` framework.

The commitment algorithm Comm of our `SwitchFold` PCS commits to a witness matrix $\boldsymbol{M} \in \mathbb{F}^{k_d \times t}$ by committing to the Merkle tree of its encoded codeword $\mathsf{Enc}[\mathcal{C}_d^t](\boldsymbol{M})$. The opening algorithm is given in Algorithm 6. The following theorem summarizes its overall efficiency and security.

Theorem 6 (`SwitchFold`). *Let $k_0, n_0, t \in \mathbb{N}$ be constants with $t \ge 2$ a power of two, and let $r := n_0/k_0 \in \mathbb{N}$. Let $\{\mathcal{C}_j : \mathbb{F}^{k_j} \to \mathbb{F}^{n_j}\}_{j=0}^{d}$ be a sequence of linear-time encodable codes with $k_j = k_0 t^j$, $n_j = n_0 t^j$, and a uniform lower bound of relative distance $\delta > 0$. Let $\boldsymbol{G}_j$ denote the generator matrix of $\mathcal{C}_j$, and assume that for every $\boldsymbol{r} \in \mathbb{F}^{k_j}$ and $\boldsymbol{l} \in \mathbb{F}^{n_j}$, both products $\boldsymbol{G}_j\boldsymbol{r}$ and $\boldsymbol{l}^\top \boldsymbol{G}_j$ can be computed in $O(k_j)$ time, with constants independent of j. For every j, let $\widehat{\delta}_j$ denote the relative distance of the code generated by $\widehat{\boldsymbol{G}}_j$, and assume that*

$\widehat{\delta}_j \geq \widehat{\delta} > 0$. *Choose proximity thresholds* $0 < \delta' < \delta/3$ *and* $0 < \widehat{\delta}'_j < \widehat{\delta}_j/3$ *so that* $\delta' = \Theta(\delta)$ *and* $\delta/3 - \delta' = \Theta(\delta)$ *and* $\widehat{\delta}'_j = \Theta(\widehat{\delta}_j)$ *and* $\widehat{\delta}_j/3 - \widehat{\delta}'_j = \Theta(\widehat{\delta}_j)$ *for every* j. *Let* $N = k_d t$ *be the witness size of the PCS. Then, for any polynomially bounded* $T \in \mathbb{N}$ *with* $T \geq 1$, *Algorithm 6 is a multilinear PCS supporting* T *opening instances with the following efficiency and soundness bounds:*

- *Prover time:* $O(TN)$.
- *Proof size and verifier time:* $O\big(T\lambda \log^2 N\big)$.
- *Computational- and knowledge-soundness errors:* $O\Big(T \cdot \frac{\log^2 N}{|\mathbb{F}|} + T \log N \cdot \big(2^{-\lambda} + \kappa_{\mathsf{Hash}}\big)\Big)$.

The proof of Theorem 6 is provided in Appendix C.6.

5 Instantiation with Concrete Linear Codes

5.1 Choosing the Right Codes

Code agnosticity gives `SwitchFold` the flexibility to use arbitrary linear codes. The code sequence may even combine codes from different code families across levels. The concrete performance of `SwitchFold`, however, depends directly on the properties of the selected codes. Choosing an appropriate code sequence is therefore essential for concrete efficiency.

The top-level code has the greatest effect on prover time. To commit to a witness matrix $\boldsymbol{M} \in \mathbb{F}^{k_d \times t}$, the prover computes $\mathsf{Enc}[\mathcal{C}_d^t](\boldsymbol{M})$ and Merkle-commits to the resulting codeword. Fast encoding of the top-level code directly reduces this dominant cost. For a fixed block length, a higher rate also produces a shorter codeword and reduces the number of hashes required by the commitment.

The priorities change at lower levels. Recursive opening commits to matrices whose dimensions decrease geometrically, so the contribution of lower-level encoding to total prover time is smaller. Relative distance, however, determines the number of proximity queries at every level: a larger relative distance reduces both proof size and verifier time. It can therefore be advantageous to use a fast-encoding code at the top level and codes with better distance at lower levels, even if their encoders are moderately slower.

Moreover, the final Decide protocol depends on the arithmetization of $\boldsymbol{G}_j$. The reduction Π_{G_j} represents its MLE claims using the dense matrix $\boldsymbol{H}_j$ and, when necessary, the claim-dependent auxiliary matrix $\boldsymbol{S}_j$. Thus, the dimensions of these matrices and the prover and verifier costs of Π_{G_j} determine the code-specific cost of the final recursive opening.

These observations give the following principles for selecting a code sequence:

- prioritize fast encoding and a favorable rate at the top level;
- seek a favorable rate–distance tradeoff at every level, placing greater emphasis on relative distance at lower levels; and
- prefer efficient generator-matrix arithmetizations, with small $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ and an efficient reduction Π_{G_j}.

Designing new linear codes is orthogonal to this work, so we instantiate `SwitchFold` with two existing code families that admit efficient encoding over large prime fields. The first is the widely used Brakedown code [28], a Spielman-style code with linear-time encoding and strong concrete performance. The second is the recently revisited family of quasi-Abelian codes [36]. Although QA encoding is not strictly linear, it is concretely fast and provides substantially larger relative distance than the Brakedown code.

5.2 Instantiation with the Brakedown Code

We first instantiate `SwitchFold` with the Brakedown code [28], obtaining a PCS that we call `BrakeFold`. The Brakedown code family is parameterized by constants $\alpha, \beta, r \in \mathbb{R}$ satisfying $0 < \alpha < 1$, $0 < \beta < \alpha/1.28$, $r > (1+2\beta)/(1-\alpha) > 1$. The code rate is $1/r$ and the relative distance is $\delta = \beta/r$. We assume that $1/\alpha$, $1/\beta$, and r are integers. For $j \in [d]$, we set $k_j = k_0/\alpha^j$, and $n_j = rk_j$.

The code construction samples sparse matrices:

$$\left\{ \boldsymbol{A}_j \in \mathbb{F}^{k_{j-1} \times k_j},\ \boldsymbol{B}_j \in \mathbb{F}^{(n_j - k_j - n_{j-1}) \times n_{j-1}} \right\}_{j=1}^{d}.$$

Each column of $\boldsymbol{A}_j$ contains c_j uniformly random nonzero entries at random positions, and each column of $\boldsymbol{B}_j$ contains e_j uniformly random nonzero entries at random positions. $c_j, e_j \in \mathbb{N}$ are computed as functions of $\alpha, \beta, r, |\mathbb{F}|$, and n_j as per [28, Eqs. (7)–(8)]. These parameters are denoted by c_j and d_j in [28]; we write e_j for the second parameter to avoid conflict with the recursion depth d.

The base code $\mathcal{C}_0$ is a rate-$1/r$ Reed–Solomon code $\mathcal{C}_{\mathsf{RS}}$ with the encoding function $\mathsf{Enc}_{\mathsf{RS}}$. For $j \in [d]$, $\mathcal{C}_j$ encodes $\boldsymbol{m} \in \mathbb{F}^{k_j}$ recursively as:

$$\mathsf{Enc}_0(\boldsymbol{m}) = \mathsf{Enc}_{\mathsf{RS}}(\boldsymbol{m}), \quad \mathsf{Enc}_j(\boldsymbol{m}) = \begin{pmatrix} \boldsymbol{m} \\ \mathsf{Enc}_{j-1}(\boldsymbol{A}_j \boldsymbol{m}) \\ \boldsymbol{B}_j \mathsf{Enc}_{j-1}(\boldsymbol{A}_j \boldsymbol{m}) \end{pmatrix}, \quad j \in [d]. \tag{7}$$

This recursive structure gives a concretely efficient linear-time encoder, as multiplication with sparse matrices $\boldsymbol{A}_j$ and $\boldsymbol{B}_j$ can be efficiently computed, and the recursive call is made on a message of geometrically shrinking length $k_{j-1} = \alpha k_j$.

The Brakedown paper shows that, for $n_d \leq 2^{30}$ and $|\mathbb{F}| \geq 2^{127}$, except with probability at most 2^{-100} over the sampled matrices, every code in the sequence has relative distance $\delta_j \geq \beta/r$ [28, Claim 2].

Our choice of Brakedown parameters. In our instantiation we set $\alpha = 1/4$, $\beta = 1/8$, and $r = 2$. The code rate is then $1/r = 1/2$ and the relative distance is $\delta = \beta/r = 1/16$. For these parameters, all relevant dimensions are powers of two whenever k_0 is chosen to be a power of two. The resulting parameter values are summarized below.

k_d	$\lvert\mathbb{F}\rvert$	δ	rate	α	β	r	c_j	e_j
$\leq 2^{30}$	$\geq 2^{127}$	$1/16$	$1/2$	$1/4$	$1/8$	2	≤ 11	≤ 16

With this choice, since $1/\alpha = 4$ and $r = 2$ are powers of two, choosing k_0 to be a power of two ensures that all dimensions $\{k_j, n_j\}_{j=0}^{d}$ are powers of two. Moreover, for these parameters, $n_j - k_j - n_{j-1} = n_{j-1}$, so the second and third blocks in Eq. (7) have equal size, and all block dimensions remain powers of two.

Vector–Generator-Matrix Products and Generator-Matrix Reduction. The recursive form of the Brakedown generator matrix in Eq. (9) enables efficient linear-time computation of both $\boldsymbol{G}_j \boldsymbol{r}_j$ and $\boldsymbol{l}_j^\top \boldsymbol{G}_j$. The former is exactly the Brakedown encoding of $\boldsymbol{r}_j$, while the latter can be computed recursively; see Appendix B.3. Moreover, the generator matrix can be arithmetized through the sparse matrices $\{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^{d}$, yielding the reduction protocol Π_{G_d} via lookup arguments. Details are provided in Appendices B.4–B.6.

Reed–Solomon Codes for Lower Levels. The Brakedown code offers concretely efficient linear-time encoding but has modest relative distance: under our parameters, its relative distance is only 1/16. Sacrificing relative distance for faster encoding is advantageous at the top level, where encoding dominates prover time, but less attractive at lower levels, where relative distance largely determines the number of proximity queries. We therefore use the Brakedown code only for the top-level generator matrix $\boldsymbol{G}_d$ and use RS codes of the same rate at all lower levels $j < d$. We select the RS codes solely for their superior distance, not for foldability: transitions between adjacent levels still use the generic code-switching procedure of `SwitchFold` rather than relying on the foldability.

5.3 `QAFold`: Instantiation with the Quasi-Abelian Code

We also instantiate `SwitchFold` with the recently proposed QA codes [36], yielding a PCS that we call `QAFold`. Although QA encoding requires $O(N \log N)$ asymptotic complexity, the quasilinear term consists entirely of field additions, while the number of field multiplications remains $O(N)$. Consequently, QA codes offer concrete encoding performance comparable to Brakedown codes while achieving substantially larger relative distance.

QA codes are constructed using the Walsh–Hadamard transform.

Definition 6 (Walsh–Hadamard transform). *Let n be a power of two that is invertible in $\mathbb{F}$. Define the $n \times n$ Hadamard matrix recursively by*

$$\boldsymbol{\Phi}_1 := (1), \qquad \boldsymbol{\Phi}_n := \begin{pmatrix} \boldsymbol{\Phi}_{n/2} & \boldsymbol{\Phi}_{n/2} \\ \boldsymbol{\Phi}_{n/2} & -\boldsymbol{\Phi}_{n/2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\otimes \log_2 n}.$$

For $\boldsymbol{a} \in \mathbb{F}^n$, the Walsh–Hadamard transform of $\boldsymbol{a}$ is defined as $\mathsf{WHT}_n(\boldsymbol{a}) := \boldsymbol{\Phi}_n \boldsymbol{a}$.

Definition 7 (Quasi-Abelian code [36, Theorem 4.3]). *A QA code of rate $1/r$ and message length k_j is specified by r vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_r \in \mathbb{F}^{k_j}$. For each $i \in [r]$, define $\boldsymbol{\alpha}_i := k_j^{-1} \mathsf{WHT}_{k_j}(\boldsymbol{a}_i)$. The encoding map $\mathsf{Enc} : \mathbb{F}^{k_j} \to \mathbb{F}^{r k_j}$ and the generator matrix $\boldsymbol{G}_j$ are given by:*

$$\mathsf{Enc}(\boldsymbol{m}) := \begin{pmatrix} \mathsf{WHT}_{k_j}\big(\mathsf{WHT}_{k_j}(\boldsymbol{m}) \circ \boldsymbol{\alpha}_1\big) \\ \vdots \\ \mathsf{WHT}_{k_j}\big(\mathsf{WHT}_{k_j}(\boldsymbol{m}) \circ \boldsymbol{\alpha}_r\big) \end{pmatrix}, \; \boldsymbol{G}_j := \begin{pmatrix} \boldsymbol{\Phi}_{k_j} \mathsf{diag}(\boldsymbol{\alpha}_1) \boldsymbol{\Phi}_{k_j} \\ \vdots \\ \boldsymbol{\Phi}_{k_j} \mathsf{diag}(\boldsymbol{\alpha}_r) \boldsymbol{\Phi}_{k_j} \end{pmatrix} \in \mathbb{F}^{r k_j \times k_j}.$$

where $\boldsymbol{m} \in \mathbb{F}^{k_j}$, $\circ$ *denotes element-wise multiplication, and* $\mathsf{diag}(\cdot)$ *denotes the diagonal matrix transformation.*

Relative Distance. QA codes provide strong concrete distance guarantees. For rate 1/2, $r = 2$, and a prime field of size approximately 2^{128}, applying the distance analysis of [36, Corollary 3.21] gives:

Block length k_j	2^2	2^6	2^{10}	2^{14}	2^{18}	2^{22}	2^{26}	2^{30}
Relative distance δ_j	≥ 0.48	≥ 0.46	≥ 0.45	≥ 0.43	≥ 0.42	≥ 0.40	≥ 0.39	≥ 0.37

Each bound holds with probability at least $1 - 2^{-127}$ over random QA codes.

Generator-Matrix Products and Reduction for QA Codes. At each level j, `QAFold` instantiates $\mathcal{C}_j$ with a QA code over $\mathbb{F}$ with message length k_j, using the WHT-based encoding map above as Enc_j. The generator-matrix–vector product $\boldsymbol{G}_j\boldsymbol{r}$ is precisely the QA encoding of $\boldsymbol{r}$. To compute $\boldsymbol{l}^\top \boldsymbol{G}_j$, we partition $\boldsymbol{l}$ into r subvectors corresponding to the r symmetric blocks of $\boldsymbol{G}_j$, apply the same WHT-based computation to each subvector, and sum the results. Hence, both $\boldsymbol{G}_j\boldsymbol{r}$ and $\boldsymbol{l}^\top \boldsymbol{G}_j$ can be computed efficiently using the QA encoding algorithm.

Because the Hadamard matrices are public, $\boldsymbol{G}_j$ is completely determined by the $k_j \times r$ dense matrix $\boldsymbol{H}_j := [\,\boldsymbol{\alpha}_1 \mid \ldots \mid \boldsymbol{\alpha}_r\,] \in \mathbb{F}^{k_j \times r}$. Using Fact 2, an MLE claim on $\boldsymbol{G}_j$ reduces to MLE claims on $\mathsf{diag}(\boldsymbol{\alpha}_i)$, for $i \in [r]$, and on $\boldsymbol{\Phi}_{k_j}$. The diagonal-matrix claims reduce to claims on the vectors $\boldsymbol{\alpha}_i$ and can be accumulated into an MLE claim on $\boldsymbol{H}_j$. The claims on $\boldsymbol{\Phi}_{k_j}$ are evaluated directly in logarithmic time using the following Fact 4.

Fact 4 (MLE Evaluation of the Walsh–Hadamard Matrix)**.** *Let* $\boldsymbol{l} = \mathsf{ts}(\boldsymbol{x}_l) \in \mathbb{F}^n$ *and* $\boldsymbol{r} = \mathsf{ts}(\boldsymbol{x}_r) \in \mathbb{F}^n$*. Then,* $\boldsymbol{l}^\top \boldsymbol{\Phi}_n \boldsymbol{r} = \prod_{i=1}^{\log_2 n} (1 - 2x_{l,i}x_{r,i})$.

The proof is provided in Appendix C.7.

Therefore, in the notation of Section 4.3, the setup commitment at level j binds to $[\,\boldsymbol{T}_j \mid \boldsymbol{H}_j\,]$, while $\boldsymbol{S}_j$ is empty and no auxiliary commitment is required. The recursive code-switching procedure of `SwitchFold` then applies unchanged.

6 Experimental Evaluation

We implement in Rust both `BrakeFold` and `QAFold`[7] and compare them with Brakedown,[8] BaseFold, Blaze, Zeromorph-FRI,[9] Binius,[10] BrakingBase,[11] WHIR,[12] and DeepFold.[13] We also implement a Ligero-style PCS instantiated with QA

[7] Our code is available at: `https://github.com/mirandaprivate/brakefold`.
[8] `https://github.com/arkworks-rs/poly-commit`.
[9] `https://github.com/hadasz/plonkish_basefold`.
[10] `https://github.com/binius-zk/binius64`.
[11] `https://github.com/arithmic/BrakingBase`.
[12] `https://github.com/worldfnd/whir`.
[13] `https://github.com/paulguoyanpei/DeepFold-Hyperplonk`.

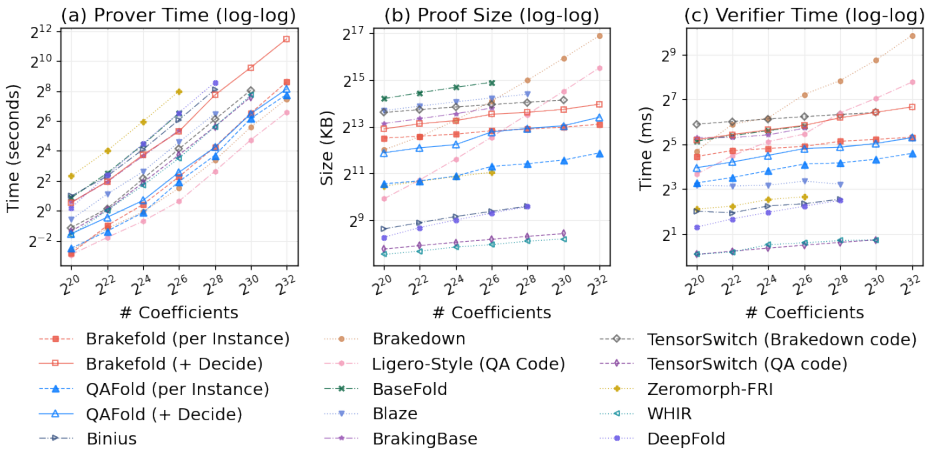

Fig. 2. Performance comparison of `BrakeFold`, `QAFold`, and other hash-based multilinear PCSs. Witness sizes range from 2^{20} to 2^{32} field elements; out-of-RAM results are omitted. Prover time is the sum of commitment and opening time. Dashed `BrakeFold` and `QAFold` curves show the marginal cost of an additional opening, while solid curves include the unamortized Decide phase.

codes and the TensorSwitch PCS instantiated with both Brakedown and QA codes. All experiments were conducted on a server equipped with two Intel Xeon Gold 6342 CPUs at 2.80 GHz, totalling 48 physical cores (96 threads) and 377 GiB of RAM. We used 64 threads and set $\lambda = 100$. We report the prover time, proof size, and verifier time of each PCS in Fig. 2. We combine the commitment and opening phases into a single prover-time metric because, in typical zkSNARK pipelines, the prover commits to the witness and subsequently opens the commitment. Consequently, both phases contribute to the overall end-to-end prover cost in zkSNARK protocols. We instantiate `BrakeFold`, `QAFold`, Brakedown, the Ligero-style QA code baseline, BaseFold, BrakingBase, and TensorSwitch over the Mersenne prime field $\mathbb{F}_{2^{127}-1}$. Other schemes use 128-bit fields required by their constructions.

For our schemes, we report both marginal and unamortized costs. BrakeFold (per Instance) and QAFold (per Instance) measure the per instance cost of each additional opening. BrakeFold (+ Decide) and QAFold (+ Decide) include the Decide phase and report the total cost when the fixed overhead is charged.

Prover Time: `QAFold` Outperforms Prior Succinct-Verifier PCSs. At $N = 2^{30}$, `QAFold` takes in prover time 71.4 s per additional opening and 88.8 s in total including the Decide phase. By comparison, its non-succinct counterpart, the Ligero-style PCS instantiated with QA codes, takes 26.4 s. The marginal and total prover overhead of achieving succinct verification are therefore 2.7× and 3.4×, respectively. `QAFold` delivers the fastest total prover time among all evaluated PCSs with polylogarithmic proof size and verifier time.

For `BrakeFold`, the fixed Decide cost makes the unamortized total prover time less competitive, but its marginal prover time remains lower than that of all evaluated succinct-verifier PCSs. At $N = 2^{30}$, each additional opening takes 91.1 s, compared with 47.7 s for the non-succinct Brakedown baseline, yielding a 1.9× marginal overhead for succinct verification. The Decide phase adds 663.3 s, bringing the unamortized total to 754.4 s. Thus, `BrakeFold` is best suited to recurring openings over which this fixed Decide cost can be amortized.

Both `QAFold` and `BrakeFold` exhibit strong scalability, completing instances with $N = 2^{32}$ on our machine. They are the only evaluated succinct-verifier PCSs to reach this scale. By comparison, the widely used field-agnostic BaseFold PCS completes instances only up to $N = 2^{26}$ in our benchmarks. At this common witness size, the total prover times of `QAFold` and `BrakeFold` are 15.4× and 2.3× faster than that of BaseFold, respectively.

Proof Size and Verifier Time: Larger but Compressible. Binius, WHIR, Zeromorph-FRI, DeepFold, and QA-based TensorSwitch produce smaller proofs and verify faster than `BrakeFold` and `QAFold`. However, Binius operates over binary extension fields, while WHIR, Zeromorph-FRI, and DeepFold require FFT-friendly fields. TensorSwitch is therefore the most direct comparison, as it is also field-agnostic and supports large prime fields.

At $N = 2^{30}$, the total `QAFold` proof including Decide is 24.5× larger and its verifier is 19.8× slower than those of QA-based TensorSwitch. In return, its marginal and total prover times are 2.7× and 2.2× faster, respectively. TensorSwitch is more sensitive to code distance. With the lower-distance Brakedown code, its verifier advantage disappears: at $N = 2^{30}$, the complete `BrakeFold` proof is 1.3× smaller and its verifier time matches that of TensorSwitch.

Although the verifier performance of `QAFold` and `BrakeFold` trails the state of the art, recursive proof composition can mitigate this limitation. Their proofs can be compressed by a Binius outer proof, which efficiently proves the bit-oriented Fiat–Shamir hash computations.

Nevertheless, both `QAFold` and `BrakeFold` outperform the widely used BaseFold. At $N = 2^{26}$, they reduce total prover time by factors of 15.4× and 2.3×, and proof size by factors of 4.4× and 2.6×, respectively. The `QAFold` verifier is 2.0× faster, while the `BrakeFold` verifier time matches that of BaseFold.

7 Conclusion

We proposed `SwitchFold`, a new hash-based PCS construction based on recursive code switching. Our construction transforms a sequence of codes with geometrically increasing block lengths, linear-time encoding, and the required efficient reductions for their generator matrices into a PCS with polylogarithmic proof size and verifier time. It relies only on generic coding-theoretic properties, rather than on foldability of the underlying codes. We instantiated `SwitchFold` with the Brakedown and QA code sequences, obtaining `BrakeFold` and `QAFold`. At $N = 2^{30}$, `QAFold` requires 88.8 s of total prover time, the lowest among all evaluated PCSs with polylogarithmic proof size and verifier time.

Acknowledgment

The authors acknowledge the use of generative AI tools for coding assistance, grammar and spell checking, and basic language editing during manuscript preparation. No scientific content in this paper was generated using AI.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: CCS. p. 2087–2104 (2017)
2. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: STIR: reed-solomon proximity testing with fewer queries. In: CRYPTO. pp. 380–413 (2024)
3. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: WHIR: reed-solomon proximity testing with super-fast verification. In: EUROCRYPT. pp. 214–243 (2025)
4. Arun, A., Setty, S.T.V., Thaler, J.: Jolt: Snarks for virtual machines via lookups. In: EUROCRYPT. pp. 3–33 (2024)
5. Attema, T., Cramer, R., Kohl, L.: A compressed Σ-protocol theory for lattices. In: CRYPTO. pp. 549–579 (2021)
6. Attema, T., Fehr, S., Resch, N.: Generalized special-sound interactive proofs and their knowledge soundness. In: TCC. pp. 424–454 (2023)
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. In: ICALP. pp. 14:1–14:17 (2018)
8. Ben-Sasson, E., Carmon, D., Ishai, Y., Kopparty, S., Saraf, S.: Proximity gaps for reed-solomon codes. In: FOCS. pp. 900–909 (2020)
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for C: verifying program executions succinctly and in zero knowledge. In: CRYPTO. pp. 90–108 (2013)
10. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. pp. 31–60 (2016)
11. Ben-Sasson, E., Kopparty, S., Saraf, S.: Worst-case to average case reductions for the distance to a code. In: CCC. pp. 24:1–24:23 (2018)
12. Block, A.R., Fang, Z., Katz, J., Thaler, J., Waldner, H., Zhang, Y.: Field-agnostic SNARKs from expand-accumulate codes. In: CRYPTO. p. 276–307 (2024)
13. Boneh, D., Chen, B.: Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. In: ASIACRYPT. pp. 330–362 (2025)
14. Brehm, M., Chen, B., Fisch, B., Resch, N., Rothblum, R.D., Zeilberger, H.: Blaze: Fast snarks from interleaved RAA codes. In: EUROCRYPT. pp. 123–152 (2025)
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: S&P. pp. 315–334 (2018)
16. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC. p. 1–18 (2020)
17. Bünz, B., Fenzi, G., Rothblum, R.D., Wang, W.: Tensorswitch: Nearly optimal polynomial commitments from tensor codes. IACR Cryptol. ePrint Arch. 2025/2065 (2025)
18. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: ASIACRYPT. pp. 65–97 (2021)
19. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D.: Fiat-shamir from simpler assumptions. IACR Cryptol. ePrint Arch. 2018/1004 (2018)

20. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT. pp. 499–530 (2023)
21. Cini, V., Malavolta, G., Nguyen, N.K., Wee, H.: Polynomial commitments from lattices: Post-quantum security, fast verification and transparent setup. In: CRYPTO. pp. 207–242 (2024)
22. Cong, M., Chow, S.S.M., Yiu, S.M., Yuen, T.H.: Scalable zksnarks for matrix computations: A generic framework for verifiable deep learning. In: ASIACRYPT. pp. 363–395 (2025)
23. Diamond, B.E., Posen, J.: Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, 2024/504 (2024)
24. Diamond, B.E., Posen, J.: Succinct arguments over towers of binary fields. In: EUROCRYPT. p. 93–122 (2025)
25. Freivalds, R.: Probabilistic machines can use less running time. In: IFIP. pp. 839–842 (1977)
26. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. 2019/953 (2019)
27. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: EUROCRYPT. pp. 626–645 (2013)
28. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: CRYPTO. pp. 193–226 (2023)
29. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT. pp. 305–326 (2016)
30. Guo, Y., Liu, X., Huang, K., Qu, W., Tao, T., Zhang, J.: Deepfold: efficient multilinear polynomial commitment from Reed-Solomon code and its application to zero-knowledge proofs. In: USENIX Security (2025)
31. Haböck, U.: Multivariate lookups based on logarithmic derivatives. IACR Cryptol. ePrint Arch. 2022/1530 (2022)
32. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT. pp. 177–194 (2010)
33. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: STOC. pp. 723–732 (1992)
34. Kohrita, T., Towa, P.: Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, 2023/917 (2023)
35. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: TCC. pp. 1–34 (2021)
36. 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, 2026/939 (2026)
37. Micali, S.: Computationally sound proofs. SIAM J. Comput. **30**(4), 1253–1298 (2000)
38. Nair, V., Sharma, A., Thankey, B.: BrakingBase: A linear prover, poly-logarithmic verifier, field-agnostic polynomial commitment scheme. In: ASIACRYPT. pp. 427–455 (2025)
39. Ron-Zewi, N., Rothblum, R.: Local proofs approaching the witness length. J. ACM **71**(3), 18 (2024)
40. Setty, S.T.V.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO. pp. 704–737 (2020)
41. Setty, S.T.V., Thaler, J.: Twist and shout: Faster memory checking arguments via one-hot addressing and increments. IACR Cryptol. ePrint Arch. 2025/105 (2025)

42. Setty, S.T.V., Thaler, J., Wahby, R.S.: Unlocking the lookup singularity with lasso. In: EUROCRYPT. pp. 180–209 (2024)
43. Sun, H., Li, J., Zhang, H.: zkLLM: Zero knowledge proofs for large language models. In: CCS (2024)
44. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: CRYPTO. pp. 71–89 (2013)
45. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO. pp. 299–328 (2022)
46. Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: CRYPTO. pp. 138–169 (2024)

A Deferred Definitions

A.1 Multilinear Extension

Vectors and matrices are in one-to-one correspondence with multilinear polynomials via their multilinear extension (MLE), defined as follows.

Definition 8 (Multilinear Extension [44]). *The MLE of a matrix $\boldsymbol{A} \in \mathbb{F}^{m\times n}$ is the unique multilinear polynomial, denoted by $\widehat{\boldsymbol{A}}(\boldsymbol{x}) : \mathbb{F}^{\log_2 m} \times \mathbb{F}^{\log_2 n} \to \mathbb{F}$ such that:*

$$\widehat{\boldsymbol{A}}(\boldsymbol{i}, \boldsymbol{j}) = A_{ij}, \quad \forall\, \boldsymbol{i} \in \{0,1\}^{\log_2 m}, \boldsymbol{j} \in \{0,1\}^{\log_2 n},$$

where $\boldsymbol{i}$ and $\boldsymbol{j}$ range over the boolean hypercube and correspond to the little-endian bit decomposition of the index i and j, respectively.

An MLE evaluation can be represented as a vector–matrix–vector product between the witness matrix and tensor-structured evaluation vectors. In this paper, we use the following matrix-based representation of MLE evaluations.

Fact 5 (MLE Evaluation in Matrix Form). *Let $\boldsymbol{A} \in \mathbb{F}^{m\times n}$ be a matrix, and let $\widehat{\boldsymbol{A}}$ denote its MLE. Then, for $\boldsymbol{x} = (\boldsymbol{x}_l, \boldsymbol{x}_r)$, we have:*

$$\widehat{\boldsymbol{A}}(\boldsymbol{x}) = \boldsymbol{l}^\top \boldsymbol{A} \boldsymbol{r}, \qquad \text{where} \quad \boldsymbol{l} = \mathsf{ts}(\boldsymbol{x}_l), \quad \boldsymbol{r} = \mathsf{ts}(\boldsymbol{x}_r).$$

Here, $\mathsf{ts}(\cdot)$ is defined below.

Definition 9 (Tensor Expansion). *Let $n \in 2^{\mathbb{N}}$ and let $\boldsymbol{x} = (x_1, \ldots, x_{\log_2 n}) \in \mathbb{F}^{\log_2 n}$. The* tensor expansion *of $\boldsymbol{x}$ is the vector in $\mathbb{F}^n$ defined as follows:*

$$\mathsf{ts}(\boldsymbol{x}) \coloneqq (1 - x_1, x_1) \otimes (1 - x_2, x_2) \otimes \cdots \otimes (1 - x_{\log_2 n}, x_{\log_2 n}) \in \mathbb{F}^n.$$

We denote by $\mathbb{T}(n) \subseteq \mathbb{F}^n$ the set of all such tensor expansions of $\boldsymbol{x} \in \mathbb{F}^{\log_2 n}$:

$$\mathbb{T}(n) \coloneqq \left\{ \mathsf{ts}(\boldsymbol{x}) \;\middle|\; \boldsymbol{x} \in \mathbb{F}^{\log_2 n} \right\}.$$

Fact 6 (Efficient Inner Product of Tensor Expansions). *Let $\boldsymbol{a} = \mathsf{ts}(\boldsymbol{x})$ and $\boldsymbol{b} = \mathsf{ts}(\boldsymbol{y})$ be tensor-structured vectors in $\mathbb{T}(n)$. Then their inner product can be computed using $O(\log n)$ field multiplications as follows:*

$$\langle \mathsf{ts}(\boldsymbol{x}), \mathsf{ts}(\boldsymbol{y}) \rangle \;=\; \prod_{j=1}^{\log_2 n} \Big((1 - x_j)(1 - y_j) + x_j y_j \Big).$$

Lemma 3 (Schwartz-Zippel-DeMillo-Lipton Lemma). *Let $\phi(\boldsymbol{x}) : \mathbb{F}^\mu \to \mathbb{F}$ be a nonzero μ-variate polynomial such that the degree of ϕ in each variable is at most d. Let $\boldsymbol{x} \xleftarrow{\$} S^\mu$ for some finite set $S \subseteq \mathbb{F}$. Then:*

$$\Pr_{\boldsymbol{x} \xleftarrow{\$} S^\mu}[\phi(\boldsymbol{x}) = 0] \leq \frac{\mu \cdot d}{|S|}.$$

A.2 Linear Code

Definition 10 (Hamming Distance). *The* Hamming distance *between vectors $\boldsymbol{a} = \{a_i\}_{i=1}^n$ and $\boldsymbol{b} = \{b_i\}_{i=1}^n$ in $\mathbb{F}^n$ is defined as:*

$$\Delta_H(\boldsymbol{a}, \boldsymbol{b}) \;=\; |\{\, i \in [n] \mid a_i \neq b_i \,\}| \in \mathbb{N}.$$

The relative (Hamming) distance *is $\delta_H(\boldsymbol{a}, \boldsymbol{b}) \;:=\; \Delta_H(\boldsymbol{a}, \boldsymbol{b})/n \in \mathbb{R}$. We say that $\boldsymbol{a}$ is δ-far from $\boldsymbol{b}$ if $\delta_H(\boldsymbol{a}, \boldsymbol{b}) \geq \delta$.*

Definition 11 (Linear Code). *A* linear error-correcting code $\mathcal{C}$ *over a finite field $\mathbb{F}$ with message length k and codeword length n is an injective linear map $\mathcal{C} : \mathbb{F}^k \to \mathbb{F}^n$, whose image is a k-dimensional linear subspace $\mathbb{C} \subseteq \mathbb{F}^n$. $\rho := k/n \in \mathbb{R}$ is called the rate of the code. The code $\mathcal{C}$ can be specified by a* generator matrix *$\boldsymbol{G} \in \mathbb{F}^{n \times k}$, whose columns form a basis of $\mathbb{C}$. The encoding of a message vector $\boldsymbol{m} \in \mathbb{F}^k$ is given by $\boldsymbol{c} := \mathsf{Enc}[\mathcal{C}](\boldsymbol{m}) := \boldsymbol{G}\boldsymbol{m} \in \mathbb{C}$.*

The minimum distance of the code, denoted by Δ, is defined as:

$$\Delta := \min_{\boldsymbol{c}_1 \neq \boldsymbol{c}_2 \in \mathbb{C}} \Delta_H(\boldsymbol{c}_1, \boldsymbol{c}_2),$$

where Δ_H denotes the Hamming distance. Equivalently, $\Delta = \min_{\boldsymbol{c} \in \mathbb{C} \setminus \{\boldsymbol{0}\}} \Delta_H(\boldsymbol{c}, \boldsymbol{0})$.

If $\mathcal{C}$ has minimum distance Δ, we say that $\mathcal{C}$ is an $[n, k, \Delta]$ code. The relative distance *of $\mathcal{C}$ is defined as $\delta := \Delta/n$.* [14]

A.3 Security Properties of Oracle Protocols

An oracle protocol is required to satisfy the following security properties.

Definition 12 (Binding). *An oracle is* binding *with binding error κ_{bind} if, for every PPT adversary $\mathcal{A}$:*

$$\Pr\left(\begin{array}{l} \boldsymbol{M}_1 \neq \boldsymbol{M}_2 \\ \wedge\, (\mathcal{O}, \boldsymbol{M}_1) \in \mathcal{R}_{\mathsf{prox}} \wedge (\mathcal{O}, \boldsymbol{M}_2) \in \mathcal{R}_{\mathsf{prox}} \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q}); \\ \mathcal{O}, \boldsymbol{M}_1, \boldsymbol{M}_2 \leftarrow \mathcal{A}(\mathsf{pp}) \end{array} \right) \leq \kappa_{\mathsf{bind}},$$

where the probability is over the randomness of Setup *and $\mathcal{A}$. The oracle is binding if κ_{bind} is negligible in the security parameter λ.*

[14] We use column vectors by default, so encoding is $\boldsymbol{c} = \boldsymbol{G}\boldsymbol{m}$ with $\boldsymbol{G} \in \mathbb{F}^{n \times k}$. Some literature uses row vectors and writes $\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}$ with $\boldsymbol{G} \in \mathbb{F}^{k \times n}$.

Definition 13 (Completeness). *An oracle satisfies* completeness *if, for any non-uniform PPT adversary $\mathcal{A}$ and honest $\mathcal{P}$ and $\mathcal{V}$:*

$$\Pr\left(\begin{array}{l|l} \mathsf{Query}(\mathsf{pp}, \mathcal{O}, \boldsymbol{Q}; \boldsymbol{M})\langle \mathcal{P}, \mathcal{V}\rangle & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q});\ \ \boldsymbol{M}, \boldsymbol{Q} \leftarrow \mathcal{A}(\mathsf{pp}); \\ \quad = (v, \mathsf{true}) & \mathcal{O} \leftarrow \mathsf{Comm}(\mathsf{pp}, \boldsymbol{M});\ \ v \leftarrow \mathcal{F}(\boldsymbol{M}, \boldsymbol{Q}) \end{array}\right) = 1.$$

Definition 14 (Computational Soundness). *An oracle satisfies* computational soundness with error κ^{cs} *if the protocol* Query *is computationally sound. That is, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and any PPT (potentially dishonest) prover $\mathcal{P}^*$, $\mathcal{V}$ accepts with negligible probability:*

$$\Pr\left(\begin{array}{l|l} \mathsf{Query}(\mathsf{pp}, \mathcal{O}, \boldsymbol{Q}; \boldsymbol{M}^*)\langle \mathcal{P}^*, \mathcal{V}\rangle & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q}); \\ \quad = (v, \mathsf{true}) & \mathcal{O}, \boldsymbol{Q}, v, \boldsymbol{M}^* \leftarrow \mathcal{A}(\mathsf{pp}); \\ & \forall \boldsymbol{M},\ (\mathcal{O}, \boldsymbol{M}) \notin \mathcal{R}_{\mathsf{prox}} \vee v \neq \mathcal{F}(\boldsymbol{M}, \boldsymbol{Q}) \end{array}\right) \leq \kappa^{\mathsf{cs}},$$

Definition 15 (Knowledge Soundness). *An oracle is* knowledge sound with error κ^{ks} *if for any PPT adversary $\mathcal{A}$ and any PPT (potentially dishonest) prover $\mathcal{P}^*$ (with access to shared randomness* rand*), there exists an extractor $\mathcal{E}$. Given black-box oracle access to $\mathcal{P}^*$, the extractor $\mathcal{E}$ runs in an expected number of steps that is polynomial in the message size (where queries to $\mathcal{P}^*$ count as a single step) and outputs an accepting witness $\boldsymbol{M}$ with probability:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q}); \\ (\mathcal{O}, v, \boldsymbol{Q}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{rand}); \\ \boldsymbol{M} \leftarrow \mathcal{E}(\mathsf{pp}, \mathcal{P}^*, \mathsf{rand}); \\ [(\mathcal{O}, \boldsymbol{M}) \in \mathcal{R}_{\mathsf{prox}} \\ \quad \wedge\, v = \mathcal{F}(\boldsymbol{M}, \boldsymbol{Q})\ ] \end{array}\right] \geq \Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{M}, \mathbb{Q}); \\ (\mathcal{O}, v, \boldsymbol{Q}, \boldsymbol{M}^*) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{rand}); \\ \mathsf{Query}(\mathsf{pp}, \mathcal{O}, \boldsymbol{Q}; \boldsymbol{M}^*)\langle \mathcal{P}^*(\mathsf{rand}), \mathcal{V}\rangle \\ \qquad\qquad = (v, \mathsf{true}) \end{array}\right] - \kappa^{\mathsf{ks}}.$$

Lemma 4 (Knowledge Soundness Implies Computational Soundness). *Any oracle that is knowledge sound with error κ^{ks} is also computationally sound with error $\kappa^{\mathsf{cs}} \leq \kappa^{\mathsf{ks}}$.*

A.4 Concepts of Soundness

Below, we provide general definitions of completeness, computational soundness, and knowledge soundness for proof systems, argument systems, and arguments of knowledge, not limited to polynomial commitment schemes.

Definition 16 (Proof/Argument/Argument of Knowledge). *A proof, an argument, and an argument of knowledge, defined by the $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ tuple below, should satisfy soundness, computational soundness, and knowledge soundness, respectively. They all possess perfect completeness.*

- Setup*: Given a security parameter 1^λ, outputs public parameters* pp*.*
- *Prover $\mathcal{P}$: Given the public parameters* pp*, the instance u, outputs a proof transcript π for a language $\mathcal{L}$. Alternatively, given* pp*, an instance u, and a private witness w, the prover outputs a transcript π for a relation $\mathcal{R}$.*

– *Verifier* $\mathcal{V}$*: Given* pp*, instance* u*, and transcript* π*, outputs* true *or* false*.*

Let $(\mathcal{P} \rightleftharpoons \mathcal{V})$ *denote the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$*.*

Definition 17 (Perfect Completeness). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for a language* $\mathcal{L}$ */ a relation* $\mathcal{R}$ *has perfect completeness if for any non-uniform PPT adversary* $\mathcal{A}$*:*

$$\Pr\left(\begin{array}{l}\{u \in \mathcal{L} \ / \ \mathcal{R}\} \\ \ \vee \mathcal{V}(\mathsf{pp}, u, \pi) = \mathsf{true}\end{array}\middle|\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ u \ / \ (u, w) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \pi \leftarrow \mathcal{P}(\mathsf{pp}, u) \ / \ \mathcal{P}(\mathsf{pp}, u, w)\end{array}\right) = 1,$$

where $\lambda, \mathsf{pp}, u, w,$ *and* π *stand for the security parameter, public structured random strings, the instance, the witness, and the transcript, respectively.*

Here, we write $u \in \mathcal{R}$ to mean $\exists w$ s.t. $(u, w) \in \mathcal{R}$.

Definition 18 (Soundness [19]). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for a language* $\mathcal{L}$ */ a relation* $\mathcal{R}$ *is sound with a soundness error* $\kappa^{\mathsf{s}}(u) \colon \mathbb{N} \mapsto [0, 1]$ *if for any unbounded interactive* $\mathcal{P}^*$*, the probability below is bounded by negligible* $\kappa^{\mathsf{s}}(u)$*:*

$$\Pr\left(\mathcal{V}(\mathsf{pp}, u, \pi) = \mathsf{true} \,\middle|\, u \notin \mathcal{L} \ / \ \mathcal{R};\ \pi \leftarrow (\mathcal{P}^* \rightleftharpoons \mathcal{V})(\mathsf{pp}, u)\right) \leq \kappa^{\mathsf{s}}(u).$$

An argument satisfies computational soundness, weaker than soundness:

Definition 19 (Computational Soundness). *A protocol* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for the language* $\mathcal{L}$ */ a relation* $\mathcal{R}$ *has a computational soundness error* $\kappa^{\mathsf{cs}}(u) \colon \mathbb{N} \mapsto [0, 1]$ *if for any PPT adversary* $\mathcal{P}^*$*, the probability below is bounded by* $\kappa^{\mathsf{cs}}(u)$*. If* $\kappa^{\mathsf{cs}}(u)$ *is negligible, the protocol is computationally sound.*

$$\Pr\left(\mathcal{V}(\mathsf{pp}, u, \pi) = \mathsf{true} \,\middle|\, u \notin \mathcal{L} \ / \ \mathcal{R};\ \pi \leftarrow (\mathcal{P}^* \rightleftharpoons \mathcal{V})(\mathsf{pp}, u)\right) \leq \kappa^{\mathsf{cs}}(u).$$

In interactive proofs, by definition, $\kappa^{\mathsf{cs}}(u) \leq \kappa^{\mathsf{s}}(u)$.

An argument of knowledge requires knowledge soundness [5]:

Definition 20 (Knowledge Soundness). *A protocol* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for a relation* $\mathcal{R}$ *is knowledge sound with a knowledge soundness error* $\kappa^{\mathsf{ks}}(u)$ *if there exists an algorithm* $\mathcal{E}$*, called a knowledge extractor, with the following characteristics: Given input* u *and black-box oracle access to a (potentially dishonest) prover* $\mathcal{P}^*$*, the extractor* $\mathcal{E}$ *runs in an expected number of steps that is polynomial in* $|u|$ *(where queries to* $\mathcal{P}^*$ *count as a single step) and outputs a witness* $(u, w) \in \mathcal{R}$ *with probability:*

$$\Pr((u, \mathcal{E}(\mathcal{P}^*, u)) \in \mathcal{R}) \geq \epsilon(\mathcal{P}^*, u) - \kappa^{\mathsf{ks}}(u),$$

where $\epsilon(\mathcal{P}^*, u)$ *represents the probability that* $\mathcal{P}^*$ *generates an accepting transcript. The protocol is considered knowledge sound if* $\kappa^{\mathsf{ks}}(u)$ *is negligible.*

If a protocol is knowledge sound with knowledge-soundness error $\kappa^{\mathsf{ks}}(u)$, then it must also be computationally sound with computational soundness error $\kappa^{\mathsf{cs}}(u) \leq \kappa^{\mathsf{ks}}(u)$. Otherwise, for some input $u \notin \mathcal{R}$ and a PPT adversary $\mathcal{P}^*$, we would have:

$$\Pr\left(\mathcal{V}(\mathsf{pp}, u, \pi) = \mathsf{true} \,\middle|\, u \notin \mathcal{R};\ \pi \leftarrow (\mathcal{P}^* \rightleftharpoons \mathcal{V})(\mathsf{pp}, u)\right) > \kappa^{\mathsf{ks}}(u).$$

In this case, for such an input u, the extractor $\mathcal{E}$ must be able to extract a witness w with positive probability such that $u \in \mathcal{R}$. This contradicts $u \notin \mathcal{R}$.

A standard way to obtain knowledge soundness is to combine a computationally sound argument system with a knowledge-sound oracle, as captured by the following lemma.

Lemma 5 (Computationally Sound Protocol + Knowledge Sound Oracle = Knowledge Sound Protocol). *Let Π be a protocol that is computationally sound with soundness error $\kappa^{\mathsf{cs}}_{\Pi}$ for the relation:*

$$\mathcal{R}_{\Pi} = \{\ \mathsf{public\ input}\ :\ \boldsymbol{M}\ \mid \mathsf{constraint}\ \}.$$

Assume that the verifier's final check in Π corresponds to the answer function of a knowledge-sound oracle $\mathcal{O}$ for $\boldsymbol{M}$, which has knowledge-soundness error $\kappa^{\mathsf{ks}}_{\mathcal{O}}$. Then the following three-step protocol Π' is knowledge sound with knowledge-soundness error at most $\kappa^{\mathsf{cs}}_{\Pi} + \kappa^{\mathsf{ks}}_{\mathcal{O}}$:

- Commit: *the prover sends the oracle commitment $\mathcal{O}(\boldsymbol{M})$ to the verifier;*
- Reduce: *the prover and verifier execute Π;*
- Query: *the prover and verifier execute $\Pi_{\mathcal{O}}.\mathsf{Query}$ and verify the query answer.*

Proof. Completeness. Completeness follows directly from the completeness of Π and of the oracle query protocol $\Pi_{\mathcal{O}}.\mathsf{Query}$.

Knowledge soundness. Let $\mathcal{P}^*$ cause the verifier to accept with probability ϱ. By the knowledge soundness of $\Pi_{\mathcal{O}}$, an extractor outputs a witness $\boldsymbol{M}$ with probability at least $\varrho - \kappa^{\mathsf{ks}}_{\mathcal{O}}$ such that Π accepts on input $(\mathsf{public\ input}, \boldsymbol{M})$. By the computational soundness of Π, the probability that Π accepts while $(\mathsf{public\ input}, \boldsymbol{M}) \notin \mathcal{R}_{\Pi}$ is at most $\kappa^{\mathsf{cs}}_{\Pi}$. Therefore, the extractor outputs $\boldsymbol{M}$ satisfying $(\mathsf{public\ input}, \boldsymbol{M}) \in \mathcal{R}_{\Pi}$ with probability at least $\varrho - \kappa^{\mathsf{ks}}_{\mathcal{O}} - \kappa^{\mathsf{cs}}_{\Pi}$. □

A.5 Round-by-Round Soundness

Definition 21 (Round-by-Round Soundness [19]). *Let $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a $(2\nu+1)$-message public coin interactive proof system for a relation $\mathcal{R}$. For any instance $u \in \{0,1\}^*$, and any prefix π_j of a protocol transcript. Let $\mathsf{H}(u, \pi_j)$ denote the distribution of the next message of $\mathcal{V}$ when the transcript so far is $\pi_j = (\alpha_0, x_1, \alpha_1, \ldots, \alpha_{j-1}, x_j, \alpha_j), j \in [\nu]$ that contains the challenges and responses of the in the initial $2j+1$ moves. Π has round-by-round soundness error $\{\kappa^{\mathsf{rbr}}_j\}_{j=1}^{\nu}$ if there exists a series of deterministic (not necessarily efficiently computable) functions $\{\mathsf{State}_j\}_{j=1}^{\nu}$ that takes as input an instance u and a transcript prefix π_j and outputs either* accept *or* reject *such that the following properties hold:*

– *If* $u \notin \mathcal{L}$*, then* $\mathsf{State}_0(u, \emptyset) = \mathsf{reject}$*, where* $\emptyset$ *denotes the empty transcript.*
– *If* $\mathsf{State}_{j-1}(u, \pi_{j-1}) = \mathsf{reject}$ *for a transcript prefix* π_{j-1}*, then for every potential prover's message* α_j *and verifier's challenge* x_j*, it holds that:*

$$\Pr\left(\mathsf{State}_j(u, \pi_j) = \mathsf{accept} \;\middle|\; \begin{array}{l} \mathsf{State}_{j-1}(u, \pi_{j-1}) = \mathsf{reject}; \\ x_j \leftarrow \mathsf{H}(u, \pi_{j-1}); \pi_j \leftarrow (\pi_{j-1} \| x_j \| \alpha_j) \end{array}\right) \leq \kappa_j^{\mathsf{rbr}}.$$

– *For any full transcript* π *for which the verifier halts, if* $\mathsf{State}_\nu(u, \pi) = \mathsf{reject}$ *then* $\mathcal{V}(u, \pi) = \mathsf{false}$*.*

The lemma below connects the round-by-round and overall soundness errors.

Lemma 6 (Round-by-Round Soundness Implies Soundness [19, Proposition 5.4]). *Let* Π *be a* $(2\nu + 1)$*-move interactive proof with round-by-round soundness error* $\kappa_j^{\mathsf{rbr}}, j \in [\nu]$*. Then,* Π *has a soundness error* $\sum_{j=1}^{\nu} \kappa_j^{\mathsf{rbr}}$*.*

Similarly, we can define round-by-round computational soundness and show that the overall computational soundness error is bounded by the sum of the round-by-round computational soundness errors.

For knowledge soundness, we use the following definition.

Definition 22 (Round-by-Round Knowledge Soundness). *Let* Π *be a* $(2\nu+1)$*-message public-coin interactive proof system for a relation* $\mathcal{R}$*, using the notation of Definition 21. We say that* Π *is* round-by-round knowledge sound *with errors* $\{\kappa_j^{\mathsf{rbrks}}\}_{j=1}^{\nu}$ *if the following holds. There exist deterministic state functions* $\{\mathsf{State}_j\}_{j=1}^{\nu}$ *and expected polynomial-time extractors* $\{\mathcal{E}_j\}_{j=1}^{\nu}$ *such that, for any round* $j \in [\nu]$*, any transcript prefix* π_{j-1}*, and any adversarial prover* $\mathcal{P}_j^*$ *producing an accepting pair* (x_j, α_j) *with probability* ϱ_j*, the extractor* $\mathcal{E}_j$ *outputs a witness* w_{j-1} *satisfying* State_{j-1} *with probability:*

$$\Pr\left(\begin{array}{l} w_{j-1} \leftarrow \mathcal{E}(u, \pi_{j-1}, \mathcal{P}_j^*) \\ \mathsf{State}_{j-1}(u, \pi_{j-1} : w_{j-1}) = \mathsf{accept} \end{array} \;\middle|\; \begin{array}{l} x_j \leftarrow \mathsf{H}(u, \pi_{j-1}); \\ \alpha_j, w_j \leftarrow \mathcal{P}_j^*(\pi_{j-1} \| x_j); \\ \mathsf{State}_j(u, \pi_{j-1} \| x_j \| \alpha_j, w_j) = \mathsf{accept} \end{array}\right) \geq \varrho_j - \kappa_j^{\mathsf{rbrks}}.$$

Lemma 7 (Round-by-Round Knowledge Soundness Implies Knowledge Soundness). *Let* Π *be a* $(2\nu + 1)$*-move interactive proof with round-by-round knowledge soundness error* $\{\kappa_j^{\mathsf{rbrks}}\}_{j=1}^{\nu}$*. Then,* Π *has a knowledge soundness error* $\sum_{j=1}^{\nu} \kappa_j^{\mathsf{rbrks}}$*.*

We use this argument at the granularity of subprotocols. The state is recorded at each subprotocol boundary, and the extractor for an accepting subprotocol maps a valid output state to a valid input state while preserving the preceding transcript prefix. Applying these subprotocol extractors gives the additive error bound; no round-by-round property is required within an individual subprotocol.

A.6 Special Soundness

Attema *et al.* [5] showed that special soundness of interactive protocols implies knowledge soundness.

Definition 23 (Public-Coin Interactive Proof). *A $(2\nu+1)$-move protocol $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is a public-coin interactive protocol where in the $2i$-th move, $\mathcal{V}$ sends the ith challenge $x_i \xleftarrow{\$} \mathbb{X}_i$ from the randomness space $\mathbb{X}_i$. All challenges $x_1, \ldots, x_\nu$ sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages.*

Definition 24 (Tree of Transcripts [5]). *An $(t_1, \ldots, t_\nu)$-tree of transcripts for a $(2\nu+1)$-move public-coin protocol is a set of $T = \Pi_{i=1}^{\nu} t_i$ transcripts arranged in a tree. Nodes in this tree represent the prover's messages, and edges correspond to the verifier's challenges. Each node at depth i has precisely t_i children, corresponding to t_i pairwise distinct challenges. Every transcript corresponds to a unique path from the root node to a leaf node.*

Definition 25 ($(t_1, \ldots, t_\nu)$-Special Soundness [5]). *A $(2\nu+1)$-move public-coin protocol is $(t_1, \ldots, t_\nu)$-special sound if there exists a PPT algorithm $\mathcal{E}^{\mathrm{s}}$ that on input an $(t_1, \ldots, t_\nu)$-tree of accepting transcripts outputs a witness w for a relation $\mathcal{R}$.*

Special soundness implies standard knowledge soundness, known as the general forking lemma [5, Theorem 1].

Lemma 8 (General Forking Lemma [5]). *Let $\nu, t_1, \ldots, t_\nu \in \mathbb{N}$ be such that $T = \Pi_{i=1}^{\nu} t_i$ can be upper bounded by a polynomial of $|u|$. Let $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a $(t_1, \ldots, t_\nu)$-special sound $(2\nu+1)$-move interactive protocol for the relation $\mathcal{R}$, where $\mathcal{V}$ samples each challenge uniformly at random from a challenge set of size $p = O(2^\lambda) \geq \max_i(t_i)$. Then $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is knowledge sound with knowledge soundness error:*

$$\kappa_\lambda^{\mathsf{ks}}(|u|) = \frac{p^\nu - \Pi_{i=1}^{\nu}(p - t_i + 1)}{p^\nu} \leq \frac{\sum_{i=1}^{\nu}(t_i - 1)}{p}.$$

Here, $\kappa_\lambda^{\mathsf{ks}}(|u|)$ is negligible in λ.

For interactive proofs of proximity (IOPPs), the standard notion of special soundness is insufficient. To address this, Attema *et al.* [6] introduced a generalized notion of special soundness, defined with respect to a monotone structure.

Definition 26 (Monotone Structure). *Let $\mathbb{J}$ be a nonempty finite set and let $\mathcal{J} \subseteq 2^{\mathbb{J}}$ be a family of subsets. The pair $(\mathcal{J}, \mathbb{J})$ is called a* monotone structure *if $\mathcal{J}$ is closed under supersets, i.e., if $I_1 \in \mathcal{J}$ and $I_1 \subseteq I_2 \subseteq \mathbb{J}$, then $I_2 \in \mathcal{J}$.*

Definition 27 ($\mathcal{J}$-out-of-$\mathbb{J}$ Generalized Special Soundness [6]). *Let $(\mathcal{J}, \mathbb{J})$ be a monotone structure. A 3-round public-coin interactive protocol $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$, with challenge space $\mathbb{J}$, is said to be* $\mathcal{J}$-out-of-$\mathbb{J}$ special sound

if there exists a polynomial-time algorithm that, given an instance u and accepting transcripts $(\alpha, x_1, \alpha_1), \ldots, (\alpha, x_t, \alpha_t)$ sharing the same first message α, and satisfying $\{x_1, \ldots, x_t\} \in \mathcal{J}$, outputs a witness w such that $(u, w) \in \mathcal{R}$. We also say that $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is $\mathcal{J}$-special sound.

Lemma 9 (Generalized Special Soundness Implies Knowledge Soundness). [6, Theorem 1] *Let $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a $\mathcal{J}$-out-of-$\mathbb{J}$ special-sound interactive protocol. Define $t_{\mathcal{J}}$ to be the maximum length of a sequence $\{x_1, \ldots, x_t\} \subseteq \mathbb{J}$ such that for each $i \in [t]$, there exists a set $J \in \mathcal{J}$ with $\{x_1, \ldots, x_{i-1}\} \subseteq J$ but $J \setminus \{x_i\} \notin \mathcal{J}$. If $t_{\mathcal{J}}$ is polynomial in the statement size and, for all $i \in [t_{\mathcal{J}}]$, a useful x_i can be sampled in polynomial time given $\{x_1, \ldots, x_{i-1}\}$, then $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is knowledge sound with knowledge soundness error:*

$$\kappa^{\mathsf{ks}} = \max_{J \notin \mathcal{J}} \frac{|J|}{|\mathbb{J}|}.$$

Corollary 1 (Soundness of Proximity Checks). *Let $\boldsymbol{c}, \boldsymbol{c}' \in \mathbb{F}^n$ be fixed before the verifier samples ℓ positions independently and uniformly at random. If $\delta_H(\boldsymbol{c}, \boldsymbol{c}') \geq \epsilon$, the probability that the two vectors agree at every sampled position is at most:*

$$(1 - \epsilon)^{\ell}.$$

Proof. The vectors disagree on at least an ϵ fraction of their positions. Each independently sampled position therefore misses every disagreement with probability at most $1 - \epsilon$. The claim follows by independence. □

A.7 Accumulation Scheme

Accumulation schemes were formulated in the context of incrementally verifiable computation (IVC). We use the following simplified definition.

Definition 28 (Accumulation scheme [16, Section 4.1], simplified). *Let $T = T(\lambda)$ be polynomially bounded, and let $\mathcal{R}^{(i)} \subseteq \mathbb{U}^{(i)} \times \mathbb{W}^{(i)}$, $i \in [T]$ be efficiently decidable relations. A sequential accumulation scheme for $\{\mathcal{R}^{(i)}\}_{i=1}^{T}$ is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Accum}, \mathsf{Decide})$, with the following syntax.*

- $\mathsf{Setup}(1^{\lambda}, T) \to (\mathsf{pp}, \mathsf{acc}^{(0)})$*: on input the security parameter and the accumulation length, outputs public parameters* pp *and an initial accumulator* $\mathsf{acc}^{(0)}$.
- $\mathsf{Accum}(\mathsf{pp}, i, u^{(i)}, \mathsf{acc}^{(i-1)}, w^{(i)}) \to (\mathsf{acc}^{(i)}, \mathsf{bit})$*: an interactive protocol that, on input the public parameters, an index $i \in [T]$, an instance $u^{(i)} \in \mathbb{U}^{(i)}$, the prover's private witness $w^{(i)} \in \mathbb{W}^{(i)}$, and the previous accumulator* $\mathsf{acc}^{(i-1)}$*, outputs an updated accumulator* $\mathsf{acc}^{(i)}$ *and an acceptance bit indicating whether the update is accepted by the verifier.*
- $\mathsf{Decide}(\mathsf{pp}, \mathsf{acc}^{(T)}, \{w^{(i)}\}_{i=1}^{T}) \to \mathsf{bit}$*: an interactive protocol that, on input the public parameters, the final accumulator* $\mathsf{acc}^{(T)}$*, and prover's witnesses $\{w^{(i)}\}_{i=1}^{T}$, outputs a bit indicating whether* $\mathsf{acc}^{(T)}$ *is a valid accumulator.*

Given an accumulation scheme, define the accumulated relation:

$$\mathcal{R}_{\mathsf{ACC},T} := \left\{ \left((u^{(1)}, \ldots, u^{(T)}) : (w^{(1)}, \ldots, w^{(T)})\right) \;\middle|\; \bigwedge_{i=1}^{T} (u^{(i)}, w^{(i)}) \in \mathcal{R}^{(i)} \right\}.$$

The accumulation scheme ACC induces a proof protocol $\Pi_{\mathsf{ACC},T}$ for the relation $\mathcal{R}_{\mathsf{ACC},T}$ as follows: it runs Accum sequentially on each pair $(u^{(i)}, w^{(i)})$, for $i \in [T]$, and then runs Decide on the final accumulator. We say that the accumulation scheme is secure if, for every polynomially bounded $T = T(\lambda)$, the induced protocol $\Pi_{\mathsf{ACC},T}$ is a complete, computationally sound, and knowledge-sound argument system for $\mathcal{R}_{\mathsf{ACC},T}$. By the round-by-round soundness framework defined in Appendix A.5, it suffices to establish security properties for each Accum step and for the final Decide protocol.

B Deferred Technical Details

B.1 Protocol Pseudocode

We present the deferred pseudocode for Sections 3–5 in Algorithms 2–6.

Algorithm 2: $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}^t]}.\mathsf{Query}$ via Code Switching from $\mathcal{C}^t$ to $\mathcal{C}'^t$

Public input: $k, n, t \in \mathbb{N}$, $\mathcal{C} : \mathbb{F}^k \mapsto \mathbb{F}^n$, $\mathcal{C}' : \mathbb{F}^{k'} \mapsto \mathbb{F}^{n'}$,
evaluation point $\boldsymbol{l} \in \mathbb{T}(k), \boldsymbol{r} \in \mathbb{T}(t)$,
relative-distance bound δ and proximity threshold $0 < \delta' < \delta/3$
Private input: witness matrix $\boldsymbol{M} \in \mathbb{F}^{k \times t}$
Oracle access: $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}))$, $\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})$, $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$
Output: claimed MLE value v, true or false

$\mathcal{P} \to \mathcal{V}$: $v \leftarrow \boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r} \in \mathbb{F}$;
$\mathcal{P}$ computes $\boldsymbol{u}^\top \leftarrow \boldsymbol{l}^\top \boldsymbol{M} \in \mathbb{F}^t$;
$\mathcal{P} \to \mathcal{V}$: $\boldsymbol{u} \in \mathbb{F}^t$;
$\mathcal{V}$ checks $v \stackrel{?}{=} \langle \boldsymbol{u}, \boldsymbol{r} \rangle$; if not, **return** false;
$\mathcal{V} \to \mathcal{P}$: sample $\tilde{\boldsymbol{r}} \xleftarrow{\$} \mathbb{F}^t$;
$\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{u} \leftarrow \langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle \in \mathbb{F}$;
$\mathcal{P}$ computes:
    $\boldsymbol{m} \leftarrow \boldsymbol{M} \tilde{\boldsymbol{r}} \in \mathbb{F}^k$; Reshape $\boldsymbol{M}' \leftarrow \mathsf{vec}^{-1}_{k' \times t}(\boldsymbol{m}) \in \mathbb{F}^{k' \times t}$;
    $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t](\boldsymbol{M}') \leftarrow \Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}.\mathsf{Comm}(\boldsymbol{M}')$;
$\mathcal{P} \to \mathcal{V}$: $\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t](\boldsymbol{M}')$;
Set $\ell \leftarrow \left\lceil \frac{-\lambda}{\log_2(1-\delta')} \right\rceil$, and let $\mathcal{V}$ sample ℓ indices independently from $[n]$, denoted by $\mathcal{I} \in [n]^\ell$;
$\mathcal{V} \to \mathcal{P}$: $\mathcal{I}$;
$\mathcal{P} \to \mathcal{V}$: $\boldsymbol{\iota} \in \mathbb{F}^\ell$, the restriction of $\mathsf{Enc}[\mathcal{C}](\boldsymbol{m})$ to the positions in $\mathcal{I}$;
$\mathcal{P}$ and $\mathcal{V}$ derive the selection matrix $\boldsymbol{\Xi} \in \{0,1\}^{\ell \times n}$ from $\mathcal{I}$;
$\mathcal{P}$ and $\mathcal{V}$ run the matrix multiplication protocols to prove $\boldsymbol{\iota} = \boldsymbol{\Xi} \boldsymbol{G} \boldsymbol{m}$ and $\tilde{u} = \langle \boldsymbol{l}, \boldsymbol{m} \rangle$, reducing their verification to MLE claims on $\boldsymbol{\iota}$, $\boldsymbol{\Xi}$, $\boldsymbol{G}$, and $\boldsymbol{m}$;
$\mathcal{V}$ checks the MLE evaluations of $\boldsymbol{\iota}$ and $\boldsymbol{\Xi}$ directly;
$\mathcal{P}$ and $\mathcal{V}$ verify $\boldsymbol{\iota}$ by querying $\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}))$ at the rows indexed by $\mathcal{I}$, compute their inner products with $\tilde{\boldsymbol{r}}$, and checks their equivalence to $\boldsymbol{\iota}$;
$\mathcal{P}$ and $\mathcal{V}$ verify the MLE claims of $\boldsymbol{G}$ using $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.\mathsf{Query}$;
$\mathcal{P}$ and $\mathcal{V}$ verify the MLE claims of $\boldsymbol{m}$ using $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t](\boldsymbol{M}')}.\mathsf{Query}$;
return true if all checks and subprotocols accept; otherwise false;

Algorithm 3: Protocol of Accum for Generator-Matrix MLE Claims

Public input: previous accumulator $\mathsf{acc}^{(i-1)} = \{\bar{v}_j^{(i-1)}, \bar{\boldsymbol{l}}_j^{(i-1)}, \bar{\boldsymbol{r}}_j^{(i-1)}\}_{j=1}^{d}$, fresh MLE claims $\{v_j^{(i)}, \boldsymbol{l}_j^{(i)}, \boldsymbol{r}_j^{(i)}\}_{j=1}^{d}$

Private input: generator matrices $\{\boldsymbol{G}_j \in \mathbb{F}^{n_j \times k_j}\}_{j=1}^{d}$

Output: updated accumulator $\mathsf{acc}^{(i)} = \{\bar{v}_j^{(i)}, \bar{\boldsymbol{l}}_j^{(i)}, \bar{\boldsymbol{r}}_j^{(i)}\}_{j=1}^{d}$, true or false

foreach $j \in [d]$ **do**

$\mathcal{P}$ computes: $\bar{\boldsymbol{g}}_j^{(i-1)\top} \leftarrow \bar{\boldsymbol{l}}_j^{(i-1)\top} \boldsymbol{G}_j \in \mathbb{F}^{k_j}$; $\boldsymbol{g}_j^{(i)\top} \leftarrow \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \in \mathbb{F}^{k_j}$;

$\mathcal{P}$ and $\mathcal{V}$ run in parallel the two inner-product protocols for:

$$\bar{v}_j^{(i-1)} = \langle \bar{\boldsymbol{r}}_j^{(i-1)}, \bar{\boldsymbol{g}}_j^{(i-1)} \rangle \qquad \text{and} \qquad v_j^{(i)} = \langle \boldsymbol{r}_j^{(i)}, \boldsymbol{g}_j^{(i)} \rangle,$$

using the same random challenges. Let the tensor-expanded challenges be $\bar{\boldsymbol{r}}_j^{(i)} \in \mathbb{T}(k_j)$. The two checks are reduced to:

$$\tilde{v}_j^{(i-1)} = \langle \bar{\boldsymbol{r}}_j^{(i)}, \bar{\boldsymbol{g}}_j^{(i-1)} \rangle, \qquad \tilde{v}_j^{(i)} = \langle \bar{\boldsymbol{r}}_j^{(i)}, \boldsymbol{g}_j^{(i)} \rangle;$$

$\mathcal{V} \to \mathcal{P}$: sample $z_j \xleftarrow{\$} \mathbb{F}$;

$\mathcal{P}$ computes:

$\tilde{\boldsymbol{l}}_j \leftarrow \bar{\boldsymbol{l}}_j^{(i-1)} + z_j \boldsymbol{l}_j^{(i)} \in \mathbb{F}^{n_j}$; $\tilde{v}_j \leftarrow \tilde{v}_j^{(i-1)} + z_j \tilde{v}_j^{(i)} \in \mathbb{F}$;

$\tilde{\boldsymbol{h}}_j \leftarrow \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)} \in \mathbb{F}^{n_j}$;

$\mathcal{P}$ and $\mathcal{V}$ run an inner-product protocol for $\tilde{v}_j = \langle \tilde{\boldsymbol{l}}_j, \tilde{\boldsymbol{h}}_j \rangle$, using fresh random challenges whose tensor expansion gives $\bar{\boldsymbol{l}}_j^{(i)} \in \mathbb{T}(n_j)$. This reduces the check to the new accumulator claim:

$$\bar{v}_j^{(i)} \stackrel{?}{=} \langle \bar{\boldsymbol{l}}_j^{(i)}, \tilde{\boldsymbol{h}}_j \rangle = \bar{\boldsymbol{l}}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)};$$

$\mathsf{acc}^{(i)} \leftarrow \{\bar{v}_j^{(i)}, \bar{\boldsymbol{l}}_j^{(i)}, \bar{\boldsymbol{r}}_j^{(i)}\}_{j=1}^{d}$;

return $(\mathsf{acc}^{(i)}, \mathsf{true})$ if all inner-product protocols accept; otherwise $(\perp, \mathsf{false})$;

Algorithm 4: Recursive Opening of Generator-Matrix MLE Claims

Public input: $d, k_0, n_0, t \in \mathbb{N}$, $\boldsymbol{G}_0 \in \mathbb{F}^{n_0 \times k_0}$, MLE claims $\{v_j, \boldsymbol{l}_j, \boldsymbol{r}_j\}_{j=1}^{d}$, relative distances $\{\widehat{\delta}_j\}_{j=1}^{d}$, and proximity thresholds $\{\widehat{\delta}'_j\}_{j=1}^{d}$ satisfying $0 < \widehat{\delta}'_j < \widehat{\delta}_j/3$ for every j

Private input: generator matrices $\{\boldsymbol{G}_j \in \mathbb{F}^{n_j \times k_j}\}_{j=1}^{d}$, dense representation matrices $\{\boldsymbol{H}_j \in \mathbb{F}^{k_j \times h_j}\}_{j=1}^{d}$, and RF-extension diagonal matrices $\{\boldsymbol{T}_{j,i}\}_{j\in[d],\, i\in[\log_2 t]}$ defining $\{\widehat{\boldsymbol{G}}_j\}_{j=1}^{d}$

Oracle access: for every $j \in [d]$, the point oracle in pp committing to $[\,\boldsymbol{T}_j \mid \boldsymbol{H}_j\,]$ under $\widehat{\mathcal{C}}_j$

Output: true or false

Each displayed $\widehat{\boldsymbol{C}}$ below denotes the encoded matrix underlying the corresponding point-oracle commitment. Set $r \leftarrow n_0/k_0$, and form $\{\boldsymbol{T}_j\}_{j=0}^{d-1}$, with $\boldsymbol{T}_j \in \mathbb{F}^{k_j \times r(t-1)}$, by concatenating and reshaping the diagonal entries of $\boldsymbol{T}_{j+1,1}, \ldots, \boldsymbol{T}_{j+1,\log_2 t}$;

Set $\boldsymbol{T}_d \leftarrow \varnothing, \boldsymbol{W}_d \leftarrow \varnothing, \widehat{\boldsymbol{C}}_d^{\mathsf{carry}} \leftarrow \varnothing$, and store the accumulated input MLE claim on each $\boldsymbol{G}_j$;

for j from d down to 1 **do**

  if $j < d$ **then**

    Accumulate the input claim on $\boldsymbol{G}_j$ with the RF-extension-induced claim at level $j+1$, using the batching protocol of Algorithm 3;

  Run Π_{G_j} on the resulting MLE claim, following its internal challenge order. Commit each claim-dependent block before sampling the next challenge on which later blocks depend. After at most two such blocks have been fixed, let $\boldsymbol{S}_j \in \mathbb{F}^{k_j \times s_j}$ be their concatenation and let the concatenation of their auxiliary codewords be

$$\widehat{\boldsymbol{C}}_j^{\mathsf{aux}} := \widehat{\boldsymbol{G}}_j \boldsymbol{S}_j.$$

  Combine the claims on $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ with the incoming claims on $\boldsymbol{T}_j$ and $\boldsymbol{W}_j$ into one MLE claim on:

$$\boldsymbol{U}_j := [\,\boldsymbol{T}_j \mid \boldsymbol{H}_j \mid \boldsymbol{S}_j \mid \boldsymbol{W}_j\,],$$

  whose codeword is:

$$\widehat{\boldsymbol{C}}_j := \left[\, \widehat{\boldsymbol{C}}_j^{\mathsf{setup}} \mid \widehat{\boldsymbol{C}}_j^{\mathsf{aux}} \mid \widehat{\boldsymbol{C}}_j^{\mathsf{carry}} \,\right] = \widehat{\boldsymbol{G}}_j \boldsymbol{U}_j.$$

  Apply the code-switching protocol in Algorithm 5 on this claim. It returns:

  (i) $\boldsymbol{W}_{j-1} = \mathsf{vec}^{-1}_{k_{j-1}\times t}(\boldsymbol{u}_j)$ and its carry commitment $\widehat{\boldsymbol{C}}_{j-1}^{\mathsf{carry}} = \widehat{\boldsymbol{G}}_{j-1}\boldsymbol{W}_{j-1}$;

  (ii) an MLE claim on $\boldsymbol{T}_{j-1}$; and

  (iii) one RF-induced MLE claim on $\boldsymbol{G}_{j-1}$.

  if $j = 1$ **then**

    The prover sends $\boldsymbol{W}_0$. Verify $\widehat{\boldsymbol{C}}_0^{\mathsf{carry}} = \boldsymbol{G}_0\boldsymbol{W}_0$, evaluate the incoming MLE claims on the public matrix $\boldsymbol{T}_0$ and the revealed matrix $\boldsymbol{W}_0$, and evaluate the remaining MLE claims on $\widehat{\boldsymbol{G}}_0 = \boldsymbol{G}_0$ directly;

  else

    Continue with $\boldsymbol{T}_{j-1}$, $\boldsymbol{W}_{j-1}$, and the RF-induced claim on $\boldsymbol{G}_{j-1}$;

return true if all checks and subprotocols accept; otherwise false;

Algorithm 5: Code Switching from $\widehat{\mathcal{C}}_j$ to $\widehat{\mathcal{C}}^t_{j-1}$ in the Decide Protocol

Public input: level j, an MLE claim $v^*_j = \boldsymbol{l}^{*\top}_j \boldsymbol{U}_j \boldsymbol{r}^*_j$, where $\boldsymbol{l}^*_j \in \mathbb{T}(k_j)$ and $\boldsymbol{r}^*_j$ is the tensor expansion whose length equals the number of columns of $\boldsymbol{U}_j$, the relative-distance $\widehat{\delta}_j$, and a proximity threshold $0 < \widehat{\delta}'_j < \widehat{\delta}_j/3$

Private input: $\boldsymbol{U}_j = [\,\boldsymbol{T}_j \mid \boldsymbol{H}_j \mid \boldsymbol{S}_j \mid \boldsymbol{W}_j\,]$

Oracle access: for every $j \in [d]$, the MLE oracles committing to $[\,\boldsymbol{T}_j \mid \boldsymbol{H}_j\,]$, $\boldsymbol{S}_j$, and $\boldsymbol{W}_j$ under $\widehat{\mathcal{C}}_j$

Output: MLE oracle commitments to $\boldsymbol{S}_{j-1}$ and $\boldsymbol{W}_{j-1}$ under $\widehat{\mathcal{C}}_{j-1}$, an MLE claim on $\boldsymbol{U}_{j-1}$, and true or false

1 $\mathcal{P}$ sends $\boldsymbol{a}^\top_j := \boldsymbol{l}^{*\top}_j \boldsymbol{U}_j$, and $\mathcal{V}$ checks $v^*_j = \langle \boldsymbol{a}_j, \boldsymbol{r}^*_j \rangle$; if not, **return** false;

2 $\mathcal{V}$ samples a vector $\widetilde{\boldsymbol{r}}_j$ whose length equals the number of columns of $\boldsymbol{U}_j$. $\mathcal{P}$ sets:

$$\boldsymbol{u}_j := \boldsymbol{U}_j \widetilde{\boldsymbol{r}}_j \in \mathbb{F}^{k_j}$$

and $\mathcal{P}$ and $\mathcal{V}$ set $\widetilde{u}_j := \langle \boldsymbol{a}_j, \widetilde{\boldsymbol{r}}_j \rangle$;

3 Set: $\boldsymbol{W}_{j-1} := \mathsf{vec}^{-1}_{k_{j-1} \times t}(\boldsymbol{u}_j)$. $\mathcal{P}$ sends the carry oracle commitment:

$$\widehat{\boldsymbol{C}}^{\mathsf{carry}}_{j-1} := \widehat{\boldsymbol{G}}_{j-1} \boldsymbol{W}_{j-1},$$

thereby switching from $\widehat{\mathcal{C}}_j$ to $\widehat{\mathcal{C}}^t_{j-1}$;

4 Set: $\ell_j := \left\lceil \frac{-\lambda}{\log_2(1-\widehat{\delta}'_j)} \right\rceil$, sample ℓ_j rows independently, and verify:

$$\widehat{\boldsymbol{G}}_j \boldsymbol{u}_j = \widehat{\boldsymbol{C}}_j \widetilde{\boldsymbol{r}}_j$$

by opening the corresponding rows of the setup, auxiliary, and carry commitments;

5 Apply the matrix-expression reduction to prove $\widetilde{u}_j = \boldsymbol{l}^{*\top}_j \boldsymbol{u}_j$ and to obtain MLE claims on $\boldsymbol{u}_j$ and $\widehat{\boldsymbol{G}}_j$. Interpret the former as the incoming MLE claim on $\boldsymbol{W}_{j-1}$;

6 Recursively expand the MLE claim on $\widehat{\boldsymbol{G}}_j$ through its $\log_2 t$ RF-extension layers. At each layer, accumulate all claims reaching the same intermediate generator before continuing. This produces MLE claims on $\boldsymbol{T}_{j,1}, \ldots, \boldsymbol{T}_{j,\log_2 t}$ and one RF-extension-induced MLE claim on $\boldsymbol{G}_{j-1}$. Convert the diagonal-matrix claims into the corresponding MLE claim on $\boldsymbol{T}_{j-1}$;

7 $\mathcal{P}$ computes the auxiliary matrix $\boldsymbol{S}_{j-1}$ from the MLE claim on $\boldsymbol{G}_{j-1}$ and sends the auxiliary oracle commitment:

$$\widehat{\boldsymbol{C}}^{\mathsf{aux}}_{j-1} := \widehat{\boldsymbol{G}}_{j-1} \boldsymbol{S}_{j-1},$$

8 Run the reduction protocol for $\boldsymbol{G}_{j-1}$, obtaining MLE claims on $\boldsymbol{H}_{j-1}$ and an auxiliary matrix $\boldsymbol{S}_{j-1}$;

9 Batch the MLE claims on $\boldsymbol{T}_{j-1}$, $\boldsymbol{H}_{j-1}$, $\boldsymbol{S}_{j-1}$, and $\boldsymbol{W}_{j-1}$ into a single MLE claim on the concatenated vector $\boldsymbol{U}_{j-1} = [\,\boldsymbol{T}_{j-1} \mid \boldsymbol{H}_{j-1} \mid \boldsymbol{S}_{j-1} \mid \boldsymbol{W}_{j-1}\,]$;

10 **return** the carry commitment to $\boldsymbol{S}_{j-1}$ and $\boldsymbol{W}_{j-1}$, the MLE claim on $\boldsymbol{U}_{j-1}$, and the verifier decision;

Algorithm 6: Opening Algorithm of the `SwitchFold` PCS

Public input: $d, k_0, n_0, t \in \mathbb{N}$, base-level generator matrix $\boldsymbol{G}_0 \in \mathbb{F}^{n_0 \times k_0}$, evaluation points $\{(\boldsymbol{l}^{(i)}, \boldsymbol{r}^{(i)})\}_{i=1}^{T}$, where $\boldsymbol{l}^{(i)} \in \mathbb{T}(k_d)$ and $\boldsymbol{r}^{(i)} \in \mathbb{T}(t)$, relative-distance bounds δ and $\{\widehat{\delta}_j\}_{j=1}^{d}$, proximity thresholds δ' and $\{\widehat{\delta}'_j\}_{j=1}^{d}$ satisfying $0 < \delta' < \delta/3$ and $0 < \widehat{\delta}'_j < \widehat{\delta}_j/3$ for every j

Private input: witness matrices $\{\boldsymbol{M}^{(i)} \in \mathbb{F}^{k_d \times t}\}_{i=1}^{T}$, generator matrices $\{\boldsymbol{G}_j \in \mathbb{F}^{n_j \times k_j}\}_{j=1}^{d}$, dense representation matrices $\{\boldsymbol{H}_j \in \mathbb{F}^{k_j \times h_j}\}_{j=1}^{d}$, RF-extension parameters $\{\boldsymbol{T}_{j,i}\}_{j \in [d],\, i \in [\log_2 t]}$

Oracle access: point oracles $\{\mathcal{O}_{\mathsf{P}}(\mathsf{Enc}[\mathcal{C}_d^t](\boldsymbol{M}^{(i)}))\}_{i=1}^{T}$, transparent-setup point oracles $\{\mathcal{O}_{\mathsf{P}}(\widehat{\boldsymbol{G}}_j\,[\,\boldsymbol{T}_j \mid \boldsymbol{H}_j\,])\}_{j=1}^{d}$, generator-matrix reduction protocols $\{\Pi_{G_j}\}_{j=1}^{d}$

Output: claimed MLE values $\{v^{(i)}\}_{i=1}^{T}$, true or false

1 $\mathsf{acc}^{(0)} \leftarrow \{(0, \mathbf{0}, \mathbf{0}) \in \mathbb{F} \times \mathbb{F}^{n_j} \times \mathbb{F}^{k_j}\}_{j=1}^{d}$;

2 **foreach** $i \in [T]$ **do**

3 $\quad \mathcal{P}$ computes $v^{(i)} \leftarrow (\boldsymbol{l}^{(i)})^{\top} \boldsymbol{M}^{(i)} \boldsymbol{r}^{(i)} \in \mathbb{F}$;

4 $\quad \mathcal{P} \to \mathcal{V}$: $v^{(i)}$;

5 $\quad \mathcal{P}$ and $\mathcal{V}$ run the recursive code-switching protocol of Algorithm 1 (Lines 1– 20) on input $(\boldsymbol{l}^{(i)}, \boldsymbol{r}^{(i)})$ and witness $\boldsymbol{M}^{(i)}$, obtaining generator-matrix MLE claims:

$$\{v_j^{(i)} = (\boldsymbol{l}_j^{(i)})^{\top} \boldsymbol{G}_j \boldsymbol{r}_j^{(i)}\}_{j=1}^{d};$$

If any check in this truncated recursive code-switching execution fails, **return** false;

6 $\quad \mathcal{P}$ and $\mathcal{V}$ run Algorithm 3 with previous accumulator $\mathsf{acc}^{(i-1)}$ and fresh claims $\{(v_j^{(i)}, \boldsymbol{l}_j^{(i)}, \boldsymbol{r}_j^{(i)})\}_{j=1}^{d}$, obtaining $\mathsf{acc}^{(i)}$. If the Accum protocol returns false, **return** false;

7 $\mathcal{P}$ and $\mathcal{V}$ run Algorithm 4 on the final accumulator:

$$\mathsf{acc}^{(T)} = \{(\bar{v}_j^{(T)}, \bar{\boldsymbol{l}}_j^{(T)}, \bar{\boldsymbol{r}}_j^{(T)})\}_{j=1}^{d};$$

8 **return** true if all checks and subprotocols accept; otherwise false;

B.2 Batching Generator-Matrix MLE Claims

Two generator-matrix MLE claims can be batched by expressing them as inner products with the vectorized generator matrix:

$$\bar{\boldsymbol{l}}_j^{(i-1)\top}\boldsymbol{G}_j\bar{\boldsymbol{r}}_j^{(i-1)} = \left\langle \mathsf{vec}(\boldsymbol{G}_j), \bar{\boldsymbol{r}}_j^{(i-1)} \otimes \bar{\boldsymbol{l}}_j^{(i-1)} \right\rangle,\ \boldsymbol{l}_j^{(i)\top}\boldsymbol{G}_j\boldsymbol{r}_j^{(i)} = \left\langle \mathsf{vec}(\boldsymbol{G}_j), \boldsymbol{r}_j^{(i)} \otimes \boldsymbol{l}_j^{(i)} \right\rangle.$$

Then, by sampling $z_j \xleftarrow{\$} \mathbb{F}$, one reduces Eq. (1) to a single inner-product claim:

$$\bar{v}_j^{(i-1)} + z_j v_j^{(i)} = \left\langle \mathsf{vec}(\boldsymbol{G}_j), \bar{\boldsymbol{r}}_j^{(i-1)} \otimes \bar{\boldsymbol{l}}_j^{(i-1)} + z_j \boldsymbol{r}_j^{(i)} \otimes \boldsymbol{l}_j^{(i)} \right\rangle.$$

Such an inner-product claim can be proved using an inner-product proof (Fact 1).

However, this direct approach could be inefficient in our setting. The vector $\mathsf{vec}(\boldsymbol{G}_j)$ has length $n_j k_j$, so even when $\boldsymbol{G}_j$ is sparse, the resulting prover cost can be large. We instead exploit the fact that $\boldsymbol{G}_j$ is the generator matrix of a code with linear-time encoding. Specifically, for every $\boldsymbol{r} \in \mathbb{F}^{k_j}$, $\boldsymbol{G}_j\boldsymbol{r} = \mathsf{Enc}[\mathcal{C}_j](\boldsymbol{r})$ can be computed in $O(k_j)$ time. We also assume that the left row-vector projection $\boldsymbol{l}^\top\boldsymbol{G}_j$ can be computed in $O(n_j)$ time for every $\boldsymbol{l} \in \mathbb{F}^{n_j}$. This allows us to avoid an inner-product proof over the full $n_j k_j$-dimensional vector $\mathsf{vec}(\boldsymbol{G}_j)$.

Our accumulation proceeds in two stages. First, write the two claims as:

$$\bar{\boldsymbol{l}}_j^{(i-1)\top}\boldsymbol{G}_j\bar{\boldsymbol{r}}_j^{(i-1)} = \left\langle \bar{\boldsymbol{l}}_j^{(i-1)\top}\boldsymbol{G}_j, \bar{\boldsymbol{r}}_j^{(i-1)} \right\rangle,\ \boldsymbol{l}_j^{(i)\top}\boldsymbol{G}_j\boldsymbol{r}_j^{(i)} = \left\langle \boldsymbol{l}_j^{(i)\top}\boldsymbol{G}_j, \boldsymbol{r}_j^{(i)} \right\rangle.$$

Running the two inner-product proofs in parallel with the same verifier challenges replaces the RHS vectors in these checks with a common tensor vector $\bar{\boldsymbol{r}}_j^{(i)}$.

Second, using a fresh random scalar $z_j \xleftarrow{\$} \mathbb{F}$, the two resulting LHS claims are folded into a single claim of the form:

$$\left\langle \bar{\boldsymbol{l}}_j^{(i-1)} + z_j \boldsymbol{l}_j^{(i)}, \boldsymbol{G}_j\bar{\boldsymbol{r}}_j^{(i)} \right\rangle.$$

A final inner-product proof then reduces this claim to checking:

$$\bar{v}_j^{(i)} = \left\langle \bar{\boldsymbol{l}}_j^{(i)}, \boldsymbol{G}_j\bar{\boldsymbol{r}}_j^{(i)} \right\rangle = \bar{\boldsymbol{l}}_j^{(i)\top}\boldsymbol{G}_j\bar{\boldsymbol{r}}_j^{(i)}.$$

This is the MLE claim of the updated accumulator corresponding to the generator matrix $\boldsymbol{G}_j$. Algorithm 3 in Appendix B.1 gives the full Accum protocol.

B.3 Efficient Products for Brakedown Generator Matrices

The recursive Brakedown generator structure permits efficient computation of both $\boldsymbol{G}_j\boldsymbol{r}_j$ and $\boldsymbol{l}_j^\top\boldsymbol{G}_j$. As discussed in Section 5.1, these operations are important to the concrete efficiency of `SwitchFold`.

The right multiplication $\boldsymbol{G}_j\boldsymbol{r}_j = \mathsf{Enc}[\mathcal{C}_j](\boldsymbol{r}_j)$ is exactly the Brakedown encoding algorithm and can therefore be computed recursively using Eq. (7). We

now show that the left multiplication $\boldsymbol{l}_j^\top \boldsymbol{G}_j$ can also be computed recursively in linear time.

Let $\boldsymbol{l}_j \in \mathbb{F}^{n_j}$. We decompose $\boldsymbol{l}_j$ according to the three row blocks of $\boldsymbol{G}_j$:

$$\boldsymbol{l}_j = [\, \boldsymbol{l}_{j,\mathsf{L}} \mid \boldsymbol{l}_{j,\mathsf{M}} \mid \boldsymbol{l}_{j,\mathsf{R}} \,], \text{ where } \boldsymbol{l}_{j,\mathsf{L}} \in \mathbb{F}^{k_j},\ \boldsymbol{l}_{j,\mathsf{M}} \in \mathbb{F}^{n_{j-1}},\ \boldsymbol{l}_{j,\mathsf{R}} \in \mathbb{F}^{n_j - k_j - n_{j-1}}.$$

Using the block-matrix form of $\boldsymbol{G}_j$ from Eq. (9), we obtain:

$$\boldsymbol{l}_j^\top \boldsymbol{G}_j = \boldsymbol{l}_{j,\mathsf{L}}^\top + \left(\boldsymbol{l}_{j,\mathsf{M}}^\top + \boldsymbol{l}_{j,\mathsf{R}}^\top \boldsymbol{B}_j\right) \boldsymbol{G}_{j-1} \boldsymbol{A}_j = \boldsymbol{l}_{j,\mathsf{L}}^\top + \boldsymbol{l}_{j-1}^\top \boldsymbol{G}_{j-1} \boldsymbol{A}_j,$$

$$\text{where} \qquad \boldsymbol{l}_{j-1}^\top := \boldsymbol{l}_{j,\mathsf{M}}^\top + \boldsymbol{l}_{j,\mathsf{R}}^\top \boldsymbol{B}_j \in \mathbb{F}^{n_{j-1}}.$$

We now bound the cost of this recursion. Assume inductively that, for every $\boldsymbol{l}_{j-1} \in \mathbb{F}^{n_{j-1}}$, the product $\boldsymbol{l}_{j-1}^\top \boldsymbol{G}_{j-1}$ can be computed using at most τk_{j-1} field multiplications. Since each column of $\boldsymbol{B}_j$ has at most e_j nonzero entries, computing $\boldsymbol{l}_{j,\mathsf{R}}^\top \boldsymbol{B}_j$ costs at most $e_j n_{j-1}$ field multiplications. By the induction hypothesis, computing $\boldsymbol{l}_{j-1}^\top \boldsymbol{G}_{j-1}$ costs at most τk_{j-1} field multiplications. Finally, since each column of $\boldsymbol{A}_j$ has at most c_j nonzero entries, multiplying by $\boldsymbol{A}_j$ costs at most $c_j k_j$ field multiplications.

Using $k_j = k_{j-1}/\alpha$ and $n_{j-1} = r k_{j-1}$, the total cost is bounded by:

$$e_j n_{j-1} + \tau k_{j-1} + c_j k_j = (r e_j + \tau + c_j/\alpha) k_{j-1}.$$

Since we require there exist uniform bounds $c, e \in \mathbb{N}$ such that $c_j \le c$ and $e_j \le e$ for all j, it suffices to choose $\tau \ge (c/\alpha + re)/(1/\alpha - 1)$. Then:

$$(c/\alpha + re + \tau) k_{j-1} \le \tau k_j.$$

Therefore, $\boldsymbol{l}_j^\top \boldsymbol{G}_j$ can be computed using at most τk_j field multiplications. By induction, this efficiency bound holds at every level j.

B.4 Lookup Proof

Lookup proofs allow a prover to show that a collection of queried values, such as the entries of a matrix $\boldsymbol{A}$, are contained in a prescribed table $\boldsymbol{t} \in \mathbb{F}^m$. They are a useful primitive in many proof systems: *e.g.*, they support sparse matrix arithmetization as in Spartan [40], memory and address consistency in zkVM systems such as Jolt [4], and the verification of nonlinear functions in zero-knowledge machine learning [43]. Motivated by these applications, many efficient lookup arguments have been proposed, including log-derivative lookup (LogUp) [31], Lasso [42], and Twist and Shout [41]. We choose LogUp because it scales well for large lookup instances.

Lemma 10 (Log-Derivative Lookup [31]). *Let $\boldsymbol{a} \in \mathbb{F}^N$ and let $\boldsymbol{t} \in \mathbb{F}^m$ be a lookup table. Then $\{a_i\}_{i=1}^N \subseteq \{t_j\}_{j=1}^m$ as multisets iff there exists a multiplicity vector $\boldsymbol{\nu} \in \mathbb{F}^m$ (with ν_j counting the occurrences of t_j in $\boldsymbol{a}$) such that the rational-function identity:*

$$\sum_{i=1}^{N} \frac{1}{x + a_i} \;=\; \sum_{j=1}^{m} \frac{\nu_j}{x + t_j},$$

holds in $\mathbb{F}(x)$. By the Schwartz–Zippel–DeMillo–Lipton lemma, evaluating both sides at a uniformly random $z \xleftarrow{\$} \mathbb{F}$ verifies the identity except with probability at most $(N+m)/|\mathbb{F}|$.

Corollary 2. *Let $\boldsymbol{E} \in \mathbb{F}^{n\times k}$ be a matrix, let $\boldsymbol{t} \in \mathbb{F}^m$ be a lookup table, and let $\boldsymbol{J} \in [m]^{n\times k}$ be an index matrix. Let $\boldsymbol{m} = (0, 1, \ldots, m-1)^\top$. If $\boldsymbol{E}$ is looked up from $\boldsymbol{t}$ at the positions specified by $\boldsymbol{J}$, then there exists $\boldsymbol{\nu} \in \mathbb{F}^m$ such that, for all $\eta, z \in \mathbb{F}$ for which the denominators are nonzero:*

$$\left\langle \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{E} + \eta \boldsymbol{J})}, \mathbf{1} \right\rangle = \left\langle \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{t} + \eta \boldsymbol{m})}, \boldsymbol{\nu} \right\rangle,$$

where division is entrywise. Conversely, if the indexed lookup relation is false and independent $\eta, z \xleftarrow{\$} \mathbb{F}$ are sampled after $\boldsymbol{E}$ is fixed, the identity holds with probability at most $O((nk+m)/|\mathbb{F}|)$.

Proof. The matrix $\boldsymbol{E}$ is looked up from $\boldsymbol{t}$ at the positions specified by $\boldsymbol{J}$ exactly when every value-index pair $(E_{i,\ell}, J_{i,\ell})$ belongs to the table of pairs $\{(t_j, j)\}_{j=1}^m$, with multiplicities recorded by $\boldsymbol{\nu}$. The challenge η compresses these pairs as:

$$E_{i,\ell} + \eta J_{i,\ell} \qquad \text{and} \qquad t_j + \eta j.$$

If the pair multisets differ, equality of the compressed multisets is a nonzero polynomial condition in η of degree at most nk. Except with probability $O(nk/|\mathbb{F}|)$, the compressed multisets therefore still differ. Conditioned on this event, Lemma 10, evaluated independently at z, accepts with probability at most $(nk+m)/|\mathbb{F}|$. The union bound gives the stated error. □

B.5 Sparse Matrix Arithmetization via Lookup Proof.

A useful application of lookup proofs is sparse matrix commitment: instead of committing to the full matrix, the prover commits only to its nonzero entries and proves their consistency using lookup proofs. We consider a regular sparse matrix $\boldsymbol{A} \in \mathbb{F}^{m\times k}$, where each column contains exactly c nonzero entries. We represent $\boldsymbol{A}$ by two dense matrices:

$$\boldsymbol{E}, \boldsymbol{J} \in \mathbb{F}^{c\times k},$$

where $E_{i,j}$ stores the value of the i-th nonzero entry in column j, and $J_{i,j}$ stores its row index. Given $\boldsymbol{l} \in \mathbb{F}^m$, define $\boldsymbol{L} \in \mathbb{F}^{c\times k}$ by:

$$L_{i,j} := l_{J_{i,j}}.$$

Then, for any $\boldsymbol{r} \in \mathbb{F}^k$, we have:

$$\boldsymbol{l}^\top \boldsymbol{A} \boldsymbol{r} = \mathbf{1}^\top (\boldsymbol{E} \circ \boldsymbol{L}) \boldsymbol{r}.$$

Therefore, after representing the sparse matrix by $(\boldsymbol{E}, \boldsymbol{J})$, it remains to prove that each entry of $\boldsymbol{L}$ is correctly looked up from $\boldsymbol{l}$ at the index specified by $\boldsymbol{J}$.

Let $\boldsymbol{m} = (0, 1, \ldots, m-1)^\top$ be the vector of table indices, and let $\boldsymbol{\nu} \in \mathbb{F}^m$ be the multiplicity vector induced by $\boldsymbol{J}$, namely ν_j equals the number of occurrences of the index j in $\boldsymbol{J}$. The lookup consistency of $\boldsymbol{L}$ with respect to $(\boldsymbol{l}, \boldsymbol{J})$ is enforced by the LogUp identity:

$$\left\langle \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{L} + \eta \boldsymbol{J})}, \mathbf{1} \right\rangle = \left\langle \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{l} + \eta \boldsymbol{m})}, \boldsymbol{\nu} \right\rangle.$$

Here the pair $(L_{i,\ell}, J_{i,\ell})$ is matched against the table pair (l_j, j), and $l_j + \eta j$ compresses each value-index pair into a single field element. The index matrix $\boldsymbol{J}$ and the multiplicity vector $\boldsymbol{\nu}$ are fixed and committed in the setup phase.

During the lookup proof, the prover first commits to $\boldsymbol{L} \in \mathbb{F}^{c \times k}$. Only then does the verifier sample independent challenges $\eta, z \xleftarrow{\$} \mathbb{F}$. If a denominator vanishes, the verifier resamples z; for fixed η, at most $ck + m$ values are excluded. The prover then commits to:

$$\boldsymbol{P} := \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{L} + \eta \boldsymbol{J})} \in \mathbb{F}^{c \times k}, \qquad \boldsymbol{q} := \frac{\mathbf{1}}{\mathbf{1} - z(\boldsymbol{l} + \eta \boldsymbol{m})} \in \mathbb{F}^m.$$

It then proves the inverse relations:

$$\begin{aligned} \boldsymbol{P} \circ \big(\mathbf{1} - z(\boldsymbol{L} + \eta \boldsymbol{J})\big) &= \mathbf{1}, \\ \boldsymbol{q} \circ \big(\mathbf{1} - z(\boldsymbol{l} + \eta \boldsymbol{m})\big) &= \mathbf{1}, \end{aligned}$$

together with the log-derivative equality:

$$\langle \boldsymbol{P}, \mathbf{1} \rangle = \langle \boldsymbol{q}, \boldsymbol{\nu} \rangle.$$

These relations can be batched into two matrix/vector inner-product checks. Let $\tilde{\boldsymbol{l}} \in \mathbb{F}^{*c}$, $\tilde{\boldsymbol{r}} \in \mathbb{F}^{*k}$, and $\tilde{\boldsymbol{\xi}} \in \mathbb{F}^{*m}$ be tensor-structured random vectors, and let $\rho \in \mathbb{F}^*$ be an additional batching challenge. The prover sends:

$$u := \langle \boldsymbol{P}, \mathbf{1} \rangle.$$

The verifier then checks:

$$\begin{aligned} \left\langle \boldsymbol{P} \circ (\tilde{\boldsymbol{l}} \tilde{\boldsymbol{r}}^\top), \mathbf{1} - z(\boldsymbol{L} + \eta \boldsymbol{J}) + \rho\big(\tilde{\boldsymbol{l}}^{-1} \tilde{\boldsymbol{r}}^{-1\top}\big) \right\rangle &= \langle \tilde{\boldsymbol{l}}, \mathbf{1} \rangle \cdot \langle \tilde{\boldsymbol{r}}, \mathbf{1} \rangle + \rho u, \\ \left\langle \boldsymbol{q} \circ \tilde{\boldsymbol{\xi}}, \mathbf{1} - z(\boldsymbol{l} + \eta \boldsymbol{m}) + \rho\big(\boldsymbol{\nu} \circ \tilde{\boldsymbol{\xi}}^{-1}\big) \right\rangle &= \langle \tilde{\boldsymbol{\xi}}, \mathbf{1} \rangle + \rho u. \end{aligned} \tag{8}$$

The first equation batches the inverse relation for $\boldsymbol{P}$ with the claim $u = \langle \boldsymbol{P}, \mathbf{1} \rangle$, while the second batches the inverse relation for $\boldsymbol{q}$ with the claim $u = \langle \boldsymbol{q}, \boldsymbol{\nu} \rangle$.

These equations are compatible with matrix-expression reductions. In particular, the Hadamard products appearing in the equations do not require additional proof protocols. Instead, an MLE claim on $\boldsymbol{P} \circ (\tilde{\boldsymbol{l}} \tilde{\boldsymbol{r}}^\top)$ at the point $(\boldsymbol{l}, \boldsymbol{r})$ can be reduced directly to an MLE claim on $\boldsymbol{P}$ at the point $(\boldsymbol{l} \circ \tilde{\boldsymbol{l}}, \boldsymbol{r} \circ \tilde{\boldsymbol{r}})$:

$$\boldsymbol{l}^\top \big[\boldsymbol{P} \circ (\tilde{\boldsymbol{l}} \tilde{\boldsymbol{r}}^\top)\big] \boldsymbol{r} = (\boldsymbol{l} \circ \tilde{\boldsymbol{l}})^\top \boldsymbol{P} (\boldsymbol{r} \circ \tilde{\boldsymbol{r}}).$$

Similarly, an MLE claim for $\boldsymbol{q} \circ \tilde{\boldsymbol{\xi}}$ at the point $\boldsymbol{\xi}$ can be reduced to an MLE claim for $\boldsymbol{q}$ at the point $\tilde{\boldsymbol{\xi}} \circ \boldsymbol{\xi}$:

$$\langle \boldsymbol{q} \circ \tilde{\boldsymbol{\xi}}, \boldsymbol{\xi} \rangle = \langle \boldsymbol{q}, \tilde{\boldsymbol{\xi}} \circ \boldsymbol{\xi} \rangle.$$

The same observation applies to the term $\boldsymbol{\nu} \circ \tilde{\boldsymbol{\xi}}^{-1}$. Therefore, Eq. (8) can be efficiently reduced to vector and matrix MLE claims using inner-product proofs.

Before the lookup proof, the prover commits to the index matrix $\boldsymbol{J}$ and the multiplicity vector $\boldsymbol{\nu}$, which contain $ck + m$ field elements in total. During the lookup proof, the prover commits to $\boldsymbol{L}$ before η, z and to $\boldsymbol{P}, \boldsymbol{q}$ afterward, contributing $2ck + m$ field elements. Therefore, the sparse MLE opening involves committed vectors of total length $3ck + 2m$. The lookup reduction has computational soundness error $O((ck + m)/|\mathbb{F}|)$, in addition to the errors of its inner-product reductions.

The public index vector $\boldsymbol{m} = (0, 1, \ldots, m-1)$ need not be explicitly opened. Suppose:

$$m = 2^{\mu}, \qquad \boldsymbol{\xi} = \mathsf{ts}(\boldsymbol{x}) \quad \text{for} \quad \boldsymbol{x} = (x_1, \ldots, x_{\mu}) \in \mathbb{F}^{\mu}.$$

Then: $v_{\mu} \coloneqq \langle \boldsymbol{m}, \boldsymbol{\xi} \rangle$ can be computed recursively.

By Definition 9, $\boldsymbol{\xi}$ factors as:

$$\boldsymbol{\xi} = (1 - x_1,\, x_1) \otimes \boldsymbol{\xi}', \qquad \boldsymbol{\xi}' = \mathsf{ts}((x_2, \ldots, x_{\mu})) \in \mathbb{F}^{2^{\mu-1}}.$$

The first half of $\boldsymbol{\xi}$ equals $(1 - x_1) \cdot \boldsymbol{\xi}'$ and the second half equals $x_1 \cdot \boldsymbol{\xi}'$. Splitting $\boldsymbol{m}$ correspondingly, the two halves are:

$$\boldsymbol{m}' = (0, 1, \ldots, 2^{\mu-1} - 1), \qquad \boldsymbol{m}' + 2^{\mu-1} \cdot \boldsymbol{1}.$$

Letting $v_{\mu-1} \coloneqq \langle \boldsymbol{m}', \boldsymbol{\xi}' \rangle$ denote the corresponding MLE value on the first $2^{\mu-1}$ indices, we have:

$$\begin{aligned} \langle \boldsymbol{m}, \boldsymbol{\xi} \rangle &= (1 - x_1)\, \langle \boldsymbol{m}', \boldsymbol{\xi}' \rangle + x_1\, \langle \boldsymbol{m}' + 2^{\mu-1} \cdot \boldsymbol{1},\, \boldsymbol{\xi}' \rangle \\ &= \big((1 - x_1) + x_1\big)\, v_{\mu-1} + x_1 \cdot 2^{\mu-1}\, \langle \boldsymbol{1}, \boldsymbol{\xi}' \rangle. \end{aligned}$$

Using $\langle \boldsymbol{1}, \boldsymbol{\xi}' \rangle = \prod_{k=2}^{\mu} \big((1 - x_k) + x_k\big) = 1$, the recursion simplifies to:

$$v_{\mu} \;=\; v_{\mu-1} + 2^{\mu-1}\, x_1, \qquad v_1 = x_1,$$

where $v_{\mu-1}$ on the right-hand side is the MLE on indices $\{0, \ldots, 2^{\mu-1} - 1\}$ evaluated at $(x_2, \ldots, x_{\mu})$. Unrolling the recurrence gives the closed form $\langle \boldsymbol{m}, \boldsymbol{\xi} \rangle = \sum_{k=1}^{\mu} 2^{\mu-k}\, x_k$. Hence, the verifier can evaluate $\langle \boldsymbol{m}, \boldsymbol{\xi} \rangle$ directly using $O(\log m)$ field operations, without invoking an additional MLE query protocol.

B.6 Arithmetization of Brakedown Generator Matrices

We now describe the code-specific arithmetization used for Brakedown generator matrices. Our goal is to specify a dense representation matrix $\boldsymbol{H}_j$ for $\boldsymbol{G}_j$, together with a reduction protocol Π_{G_j} that reduces an MLE claim on $\boldsymbol{G}_j$ to MLE claims on $\boldsymbol{H}_j$ and an auxiliary witness matrix $\boldsymbol{S}_j$.

From the recursive encoding in Eq. (7), the generator matrices $\{\boldsymbol{G}_j\}_{j=0}^{d}$ admit the following recursive construction:

$$\boldsymbol{G}_0 = \begin{pmatrix} 1 & \omega_1 & \cdots & \omega_1^{k_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_{n_0} & \cdots & \omega_{n_0}^{k_0-1} \end{pmatrix} \in \mathbb{F}^{n_0 \times k_0}, \ \boldsymbol{G}_j = \begin{pmatrix} \boldsymbol{I}_{k_j} \\ \boldsymbol{G}_{j-1}\boldsymbol{A}_j \\ \boldsymbol{B}_j\boldsymbol{G}_{j-1}\boldsymbol{A}_j \end{pmatrix} \in \mathbb{F}^{n_j \times k_j}, \ j \in [d], \tag{9}$$

where $\omega_1, \ldots, \omega_{n_0} \in \mathbb{F}$ are the distinct evaluation points of the RS code, and $\boldsymbol{I}_{k_j}$ is the identity matrix. The three row blocks of $\boldsymbol{G}_j$ have heights k_j, n_{j-1}, and $n_j - k_j - n_{j-1}$, respectively, and these sum to n_j.

Expanding Eq. (9) reduces the generator-matrix claims to claims on the sparse matrices $\{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^{d}$ and the base generator $\boldsymbol{G}_0$. We arithmetize each pair $(\boldsymbol{A}_i, \boldsymbol{B}_i)$ and matrix-pack all resulting fixed tables and lookup witnesses into one fixed matrix $\boldsymbol{H}_d$ and one claim-dependent matrix $\boldsymbol{S}_d$.

We use the sparse-matrix arithmetization described in Appendix B.5. In general, a sparse matrix $\boldsymbol{A}$ can be represented by dense matrices storing its nonzero values and their row indices. For an MLE claim of the form $\boldsymbol{l}^\top \boldsymbol{A}\boldsymbol{r}$, lookup proofs (Appendix B.4) are then used to prove the entries of $\boldsymbol{l}$ and $\boldsymbol{r}$ selected by each nonzero entry of $\boldsymbol{A}$.

The Brakedown sparse matrices have additional regularity: each column contains the same constant number of nonzero entries. We call such a sparse matrix *regular*. Let $\boldsymbol{A} \in \mathbb{F}^{m \times k}$ be regular with exactly c nonzero entries per column. Such a regular sparse matrix $\boldsymbol{A}$ can be represented by two dense matrices $\boldsymbol{E}, \boldsymbol{J} \in \mathbb{F}^{c \times k}$ and a multiplicity vector $\boldsymbol{\nu} \in \mathbb{F}^m$. Here $E_{i,j}$ stores the value of the i-th nonzero entry in column j, $J_{i,j}$ stores its row index, and ν_i counts the number of occurrences of row index i in $\boldsymbol{J}$. The lookup proof additionally uses auxiliary matrices $\boldsymbol{L}, \boldsymbol{P} \in \mathbb{F}^{c \times k}, \boldsymbol{q} \in \mathbb{F}^m$, whose construction is detailed in Appendix B.5.

For every $i \in [d]$, we apply this arithmetization to:

$$\boldsymbol{A}_i \in \mathbb{F}^{k_{i-1} \times k_i}, \qquad \boldsymbol{B}_i \in \mathbb{F}^{(n_i - k_i - n_{i-1}) \times n_{i-1}}.$$

Set:

$$k_{d+1} := 2k_d, \qquad n_{d+1} := 2n_d.$$

Let:

$$c := \max_{i \in [d]} c_i, \qquad e := \max_{i \in [d]} e_i.$$

We pad the blocks to their designated widths c and e, place the arithmetization matrices for each level horizontally, and stack the levels vertically in decreasing order. Blank cells in the following display are zero; the multiplicity vectors occupy unused cells in their corresponding A or B blocks. The remaining rows are also zero-padded so that the packed matrix has $2k_d$ rows:

$$\boldsymbol{H}_{d+1} := \begin{pmatrix} \boldsymbol{E}^{A_d} & \boldsymbol{J}^{A_d} & \boldsymbol{\nu}^{A_d} & \boldsymbol{E}^{B_d} & \boldsymbol{J}^{B_d} & \boldsymbol{\nu}^{B_d} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{E}^{A_1} & \boldsymbol{J}^{A_1} & \boldsymbol{\nu}^{A_1} & \boldsymbol{E}^{B_1} & \boldsymbol{J}^{B_1} & \boldsymbol{\nu}^{B_1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \in \mathbb{F}^{2k_d \times (2c + 2e\alpha r + 2)},$$

and the lookup witnesses are packed in exactly the same layout:

$$\boldsymbol{S}_{d+1} := \begin{pmatrix} \boldsymbol{L}^{A_d} & \boldsymbol{P}^{A_d} & \boldsymbol{q}^{A_d} & \boldsymbol{L}^{B_d} & \boldsymbol{P}^{B_d} & \boldsymbol{q}^{B_d} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{L}^{A_1} & \boldsymbol{P}^{A_1} & \boldsymbol{q}^{A_1} & \boldsymbol{L}^{B_1} & \boldsymbol{P}^{B_1} & \boldsymbol{q}^{B_1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \in \mathbb{F}^{2k_d \times (2c+2e\alpha r+2)}.$$

Here $\boldsymbol{S}_{d+1}$ denotes only the column layout. We have:

$$h_{d+1} = s_{d+1} = 2c + 2e\alpha r + 2.$$

For $\alpha = 1/4$, $r = 2$, $c_i \leq 11$, and $e_i \leq 16$, this gives $h_{d+1} = s_{d+1} \leq 40$. Across all packed sparse matrices, the total number of lookup entries is $O(k_d)$. The resulting reduction has prover time $O(k_d)$, polylogarithmic proof size and verifier time, and computational soundness error $O(k_d/|\mathbb{F}|)$.

Adjustment to the Generic Recursion. This instantiation makes a small adjustment to the generic protocol of Section 4.3. There, $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$ are committed and opened separately at each level j. Here, we instead matrix-pack all fixed tables and lookup witnesses directly into $\boldsymbol{H}_d$ and $\boldsymbol{S}_d$, each with $k_{d+1} = 2k_d$ rows. To commit to these packed matrices, we define $\widehat{\boldsymbol{G}}_d$ by applying $\log_2 t + 1$ RF extensions to $\boldsymbol{G}_{d-1}$, rather than the $\log_2 t$ extensions used at the other levels. Thus, the first code switch is from $\widehat{\mathcal{C}}_d$ to $\widehat{\mathcal{C}}_{d-1}^{2t}$, producing a carry matrix $\boldsymbol{W}_{d-1}$ with $2t$ columns. The diagonal entries of the $\log_2 t+1$ RF extensions form $\boldsymbol{T}_{d-1} \in \mathbb{F}^{k_{d-1} \times r(2t-1)}$. Thus, at this first transition the definitions of the carry and RF-data blocks use the dimension ratio $2t$ in place of t. For every subsequent level, $\widehat{\boldsymbol{G}}_j$ is defined using $\log_2 t$ RF extensions, and the switched matrices have t columns. This first code switch is an instance of Theorem 2 with dimension ratio $2t$; all remaining transitions use ratio t. Because both ratios are constant, the proof and asymptotic bounds of the generic recursion are unchanged.

MLE Claims on RS Generator Matrices. To improve verifier efficiency, `BrakeFold` instantiates its lower levels with RS codes. An MLE claim on an RS generator matrix can be reduced to a claim-dependent auxiliary matrix.

At level j, choose a public element $\omega_j \in \mathbb{F}^*$ such that $1, \omega_j, \ldots, \omega_j^{n_j-1}$ are distinct, and define:

$$\boldsymbol{\omega}_j := \left(1, \omega_j, \ldots, \omega_j^{n_j-1}\right)^\top \in \mathbb{F}^{n_j}.$$

The RS generator matrix is the Vandermonde matrix:

$$\boldsymbol{G}_j = \left(\omega_j^{(i-1)q}\right)_{i \in [n_j],\, q \in \{0,\ldots,k_j-1\}} \in \mathbb{F}^{n_j \times k_j}.$$

Consider an MLE claim:

$$v = \boldsymbol{l}^\top \boldsymbol{G}_j \boldsymbol{r}, \qquad \boldsymbol{l} = \mathsf{ts}(\boldsymbol{x}_l) \in \mathbb{F}^{n_j}, \qquad \boldsymbol{r} = \mathsf{ts}(\boldsymbol{x}_r) \in \mathbb{F}^{k_j}.$$

Let $m_j := \log_2 k_j$. Expanding the tensor structure of $\boldsymbol{r}$ gives:

$$\boldsymbol{G}_j \boldsymbol{r} = \prod_{i=1}^{m_j} \left((1 - x_{r,i})\mathbf{1} + x_{r,i} \boldsymbol{\omega}_j^{2^{m_j - i}} \right),$$

where vector powers and products are element-wise. Consequently:

$$v = \left\langle \boldsymbol{l}, \prod_{i=1}^{m_j} \left((1 - x_{r,i})\mathbf{1} + x_{r,i} \boldsymbol{\omega}_j^{2^{m_j - i}} \right) \right\rangle.$$

Define the public power matrix:

$$\boldsymbol{H}'_j := \left[\boldsymbol{\omega}_j^{2^{m_j - 1}} \mid \ldots \mid \boldsymbol{\omega}_j \right] \in \mathbb{F}^{n_j \times m_j}.$$

Writing $r = n_j / k_j$, we reshape this fixed matrix as:

$$\boldsymbol{H}_j := \mathsf{vec}^{-1}_{k_j \times r m_j}\big(\mathsf{vec}(\boldsymbol{H}'_j)\big) \in \mathbb{F}^{k_j \times r m_j}.$$

Because $\boldsymbol{H}_j$ is independent of the opening claim, it is committed during the setup phase as the fixed representation matrix at level j.

The prover forms the prefix-product matrix $\boldsymbol{S}'_j \in \mathbb{F}^{n_j \times m_j}$, whose q-th column is:

$$\boldsymbol{S}'_{j,[:,q]} := \prod_{i=1}^{q} \left((1 - x_{r,i})\mathbf{1} + x_{r,i} \boldsymbol{\omega}_j^{2^{m_j - i}} \right), \qquad q \in [m_j].$$

The same data are reshaped as

$$\boldsymbol{S}_j := \mathsf{vec}^{-1}_{k_j \times r m_j}\big(\mathsf{vec}(\boldsymbol{S}'_j)\big) \in \mathbb{F}^{k_j \times r m_j}$$

and committed as the claim-dependent auxiliary matrix at level j.

The prover then proves the matrix relations:

$$\left[\mathbf{1} \mid \boldsymbol{S}'_{j,[:,1:m_j - 1]} \right] \circ \left(\mathbf{1}(\mathbf{1} - \boldsymbol{x}_r)^\top + \boldsymbol{H}'_j \mathsf{diag}(\boldsymbol{x}_r) \right) = \boldsymbol{S}'_j, \qquad v = \boldsymbol{l}^\top \boldsymbol{S}'_j \boldsymbol{e}_{m_j},$$

where $\boldsymbol{e}_{m_j} \in \mathbb{F}^{m_j}$ selects the last column. By Fact 2, these relations reduce to MLE claims on $\boldsymbol{S}'_j$ and $\boldsymbol{H}'_j$. These claims are authenticated by the commitments to $\boldsymbol{S}_j$ and $\boldsymbol{H}_j$, respectively. The surrounding recursive code-switching procedure is unchanged.

C Security Proofs

C.1 Proof of Lemma 2

Proof. Without loss of generality we assume $\mathcal{J} = \{0, 1, \ldots, s\}$, *i.e.*, every prover message is replaced by an oracle: a non-replaced message α_j can always be viewed as a trivial oracle whose query simply returns α_j and whose knowledge-soundness error is 0, so including it in $\mathcal{J}$ does not affect the sum $\sum_{j \in \mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}$.

Completeness. Completeness of Π_{IOP} follows from the completeness of Π together with the completeness of each oracle $\Pi_{\mathcal{O}_j}$.

Computational soundness. Let $\mathcal{P}^*$ make the verifier of Π_{IOP} accept a false statement with probability ϱ. For each oracle commitment $\mathcal{O}_j$ produced in the transcript, run the knowledge extractor of $\Pi_{\mathcal{O}_j}$ to obtain a candidate message α_j. By the knowledge soundness of $\Pi_{\mathcal{O}_j}$ and a union bound over $j \in \mathcal{J}$, with probability at least $\varrho - \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}$ every extracted α_j correctly answers the oracle queries of $\Pi_{\mathcal{O}_j}$. Replacing each $\mathcal{O}_j$ by the extracted α_j then yields an accepting transcript for the original protocol Π. Since Π is computationally sound on a false statement, this can happen with probability at most $\kappa^{\mathsf{cs}}_{\Pi}$. That is:

$$\varrho - \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j} \le \kappa^{\mathsf{cs}}_{\Pi}, \qquad \text{hence} \qquad \varrho \le \kappa^{\mathsf{cs}}_{\Pi} + \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}.$$

Knowledge soundness. Consider any adversarial prover $\mathcal{P}^*$ that makes the verifier of Π_{IOP} accept with probability ϱ. By the same union bound as above, a PPT extractor obtains values $\{\alpha_j\}_{j\in\mathcal{J}}$ such that the reconstructed transcript $\pi = (\alpha_0, x_1, \alpha_1, \ldots, x_s, \alpha_s)$ is accepting for Π with probability at least $\varrho - \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}$. Invoking the knowledge extractor of Π on this transcript, a valid witness for the target relation is obtained with probability at least $\varrho - \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j} - \kappa^{\mathsf{ks}}_{\Pi}$. Hence Π_{IOP} is knowledge sound with error at most $\kappa^{\mathsf{ks}}_{\Pi} + \sum_{j\in\mathcal{J}} \kappa^{\mathsf{ks}}_{\mathcal{O}_j}$. □

C.2 Proof of Theorem 2

Proof. For an interleaved codeword $\boldsymbol{C} \in \mathbb{F}^{n\times t}$, each row $\boldsymbol{c}_i^\top \in \mathbb{F}^t$ is one Merkle leaf. Unauthenticated rows are treated as unknown and counted as errors, giving the row-wise instantiation of Definition 4.

This proximity relation is binding. Suppose two messages $\boldsymbol{M}$ and $\boldsymbol{M'}$ are both within partial-tree distance less than $\delta/2$ of the same root. Under Merkle-path consistency, their encoded codewords agree outside the union of their error sets. Hence:

$$\delta_H\big(\mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}), \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M'})\big) < \delta,$$

contradicting the relative distance of $\mathcal{C}^t$ unless $\boldsymbol{M} = \boldsymbol{M'}$. Thus, the committed message is unique except with the Merkle-path-consistency error.

Completeness. An honest prover sends $v = \boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r}$. Then, it sends $\boldsymbol{u}^\top = \boldsymbol{l}^\top \boldsymbol{M}$. Hence:

$$v = \boldsymbol{l}^\top \boldsymbol{M} \boldsymbol{r} = \langle \boldsymbol{u}, \boldsymbol{r} \rangle,$$

so the verifier's first check accepts.

For the verifier's random vector $\tilde{\boldsymbol{r}}$, the honest prover sets:

$$\boldsymbol{m} = \boldsymbol{M}\tilde{\boldsymbol{r}}, \qquad \tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle.$$

Since $\boldsymbol{u}^\top = \boldsymbol{l}^\top \boldsymbol{M}$, we have:

$$\tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle = \boldsymbol{l}^\top \boldsymbol{M} \tilde{\boldsymbol{r}} = \langle \boldsymbol{l}, \boldsymbol{m} \rangle.$$

Moreover, if:

$$\boldsymbol{C} = \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}) = \boldsymbol{G}\boldsymbol{M},$$

then:

$$\boldsymbol{G}\boldsymbol{m} = \boldsymbol{G}\boldsymbol{M}\tilde{\boldsymbol{r}} = \boldsymbol{C}\tilde{\boldsymbol{r}}.$$

Therefore, the sampled consistency checks satisfy:

$$\boldsymbol{\Xi}\boldsymbol{G}\boldsymbol{m} = \boldsymbol{\Xi}\boldsymbol{C}\tilde{\boldsymbol{r}}.$$

By the completeness of the matrix multiplication proof, the subprotocol at Line 15 accepts. By correctness of the Merkle authentication paths and completeness of $\Pi_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$ and $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$, the oracle-query subprotocols at Lines 17–19 also accept. Thus the whole protocol accepts.

Computational soundness. Fix the source root and the prover state immediately before $\tilde{\boldsymbol{r}}$ is sampled. By adaptive Merkle-path consistency, except with error κ_{Hash}, there is a single maximal authenticated partial tree $\mathcal{T}$, fixed by this prefix, that contains every row that the saved prover state can successfully authenticate in any subsequent public-coin continuation. Every accepted authentication path in such a continuation agrees with the corresponding row of $\mathcal{T}$. Condition on this event. Let $K(\mathcal{T}) \subseteq [n]$ be its known rows, let $\mathcal{U} := [n] \setminus K(\mathcal{T})$, and write:

$$\mu := \frac{|\mathcal{U}|}{n}.$$

Indeed, two accepted paths under the same root that assign inconsistent values to a common leaf or internal node yield a hash collision by following the paths upward from their first inconsistency. The probability of such an inconsistency is κ_{Hash}. Since $\mathcal{T}$ is determined by the saved prover state, it is independent of $\tilde{\boldsymbol{r}}$ and the sampled row indices. Fix an arbitrary completion $\boldsymbol{C} \in \mathbb{F}^{n\times t}$ of its known rows. Every accepted point query agrees with $\boldsymbol{C}$; a row outside $K(\mathcal{T})$ cannot be successfully authenticated from the fixed state and is therefore treated as unknown.

For both soundness arguments, we first analyze an idealized execution with direct access to $\boldsymbol{G}$ and the target message $\boldsymbol{m}$, and then lift these accesses to the corresponding MLE-oracle protocols.

Define:

$$\delta_H^t(\boldsymbol{A}, \boldsymbol{B}) := \max_{j\in[t]} \delta_H(\boldsymbol{a}_j, \boldsymbol{b}_j),$$

where $\boldsymbol{a}_j$ and $\boldsymbol{b}_j$ denote the j-th columns of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. For a row-wise partial tree $\mathcal{T}$, define $\delta_{H,\perp}^t(\mathcal{T}, \boldsymbol{A})$ analogously as the maximum of the columnwise partial distances, counting every unknown row as an error in every column. We define the distance from the partial tree to the interleaved code by:

$$\delta_{H,\perp}^t(\mathcal{T}, \mathcal{C}^t) := \min_{\boldsymbol{M}'\in\mathbb{F}^{k\times t}} \delta_{H,\perp}^t\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}')\big),$$

and fix a deterministic nearest-message rule:

$$\mathsf{Dec}(\mathcal{T}) \in \mathsf{argmin}_{\boldsymbol{M}'\in\mathbb{F}^{k\times t}} \delta_{H,\perp}^t\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}')\big).$$

For the computational soundness analysis, write $\overline{\boldsymbol{M}} := \mathsf{Dec}(\mathcal{T})$. In particular:

$$\delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\overline{\boldsymbol{M}})\big) = \delta^t_{H,\perp}(\mathcal{T}, \mathcal{C}^t).$$

Whenever this distance is less than $\delta/2$, the nearest message $\overline{\boldsymbol{M}}$ is unique because $\mathcal{C}^t$ has relative distance δ: every unknown row is counted in the distance to both candidate codewords. The vector $\boldsymbol{m}$ below always denotes the prover response authenticated by the target MLE oracle.

We will use the following partial-tree form of Lemma 1. For a partial vector $\boldsymbol{a}$ whose entries are known on $K(\mathcal{T})$, define:

$$\delta_{H,\perp}(\boldsymbol{a}, \boldsymbol{c}) := \mu + \frac{1}{n}\,|\{\, i \in K(\mathcal{T}) : a_i \neq c_i \,\}|\,, \qquad \delta_{H,\perp}(\boldsymbol{a}, \mathcal{C}) := \min_{\boldsymbol{c}\in\mathcal{C}} \delta_{H,\perp}(\boldsymbol{a}, \boldsymbol{c}).$$

For $\boldsymbol{r} \in \mathbb{F}^t$, we write $\mathcal{T}\boldsymbol{r}$ for the partial vector whose i-th known entry is $\boldsymbol{C}_i\boldsymbol{r}$, leaving the rows in $\mathcal{U}$ unknown. Let $\bar{\boldsymbol{C}}$ be any partial matrix with this set of unknown rows. If some column of $\bar{\boldsymbol{C}}$ is ϵ-far from $\mathcal{C}$ in this distance, then, we claim that for every $0 \le \rho < \min\{\delta/3, \epsilon\}$,

$$\Pr_{\boldsymbol{r}\xleftarrow{\$}\mathbb{F}^t}\big[\delta_{H,\perp}(\bar{\boldsymbol{C}}\boldsymbol{r}, \mathcal{C}) < \rho\big] \le \frac{1}{(\epsilon-\rho)|\mathbb{F}|}. \tag{10}$$

To prove Eq. (10), first observe that the event is impossible when $\mu \ge \rho$, because unknown rows alone contribute distance μ. Suppose therefore that $\mu < \rho$, and puncture $\mathcal{C}$ and $\boldsymbol{C}$ to the known rows. After normalizing by the remaining length, the punctured code has relative distance at least $(\delta-\mu)/(1-\mu)$. Moreover, the corresponding column is at least $(\epsilon-\mu)/(1-\mu)$-far from the punctured code, while the target radius becomes $(\rho-\mu)/(1-\mu)$. Applying Lemma 1 to the punctured code therefore gives:

$$\frac{1-\mu}{(\epsilon-\rho)|\mathbb{F}|} \le \frac{1}{(\epsilon-\rho)|\mathbb{F}|},$$

proving Eq. (10).

We consider the following sequence of states:

$$\mathsf{State}_0 = \big\{v = \boldsymbol{l}^\top\overline{\boldsymbol{M}}\boldsymbol{r} \ \wedge\ \delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\overline{\boldsymbol{M}})\big) < \delta/2\big\},$$

$$\mathsf{State}_1 = \big\{v = \langle \boldsymbol{u}, \boldsymbol{r}\rangle \ \wedge\ \boldsymbol{u}^\top = \boldsymbol{l}^\top\overline{\boldsymbol{M}} \ \wedge\ \delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\overline{\boldsymbol{M}})\big) < \delta/2\big\},$$

$$\mathsf{State}_2 = \left\{\begin{array}{l} v = \langle \boldsymbol{u}, \boldsymbol{r}\rangle \ \wedge\ \tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}}\rangle \ \wedge\ \tilde{u} = \langle \boldsymbol{l}, \boldsymbol{m}\rangle \\ \wedge\ \delta_{H,\perp}\big(\mathcal{T}\tilde{\boldsymbol{r}}, \boldsymbol{G}\boldsymbol{m}\big) < \delta' \end{array}\right\},$$

$$\mathsf{State}_3 = \left\{\begin{array}{l} v = \langle \boldsymbol{u}, \boldsymbol{r}\rangle \ \wedge\ \tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}}\rangle \ \wedge\ \tilde{u} = \langle \boldsymbol{l}, \boldsymbol{m}\rangle \\ \wedge\ \mathcal{I} \subseteq K(\mathcal{T}) \\ \wedge\ \boldsymbol{\iota} = \boldsymbol{\Xi}\boldsymbol{G}\boldsymbol{m} \ \wedge\ \boldsymbol{\iota} = \boldsymbol{\Xi}\boldsymbol{C}\tilde{\boldsymbol{r}} \end{array}\right\},$$

$$\mathsf{State}_4 = \left\{\begin{array}{l} v = \langle \boldsymbol{u}, \boldsymbol{r}\rangle \ \wedge\ \tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}}\rangle \\ \wedge\ \text{the matrix multiplication proof accepts} \\ \wedge\ \text{all point-oracle queries are accepted} \\ \wedge\ \text{all reduced MLE claims are accepted} \end{array}\right\}.$$

$\mathsf{State}_0 \Rightarrow \mathsf{State}_1$. If $\delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\overline{\boldsymbol{M}})\big) \geq \delta/2$, then both states are false because they contain the same proximity condition. If $v \neq \boldsymbol{l}^\top \overline{\boldsymbol{M}} \boldsymbol{r}$, then no vector $\boldsymbol{u}$ can satisfy both:

$$v = \langle \boldsymbol{u}, \boldsymbol{r} \rangle \quad \text{and} \quad \boldsymbol{u}^\top = \boldsymbol{l}^\top \overline{\boldsymbol{M}}.$$

Thus this transition introduces no soundness error.

$\mathsf{State}_1 \Rightarrow \mathsf{State}_2$. Let $\mathbf{1} \in \mathbb{F}^k$ be the all-one vector and define the kernel subcode:

$$\mathcal{C}_{\boldsymbol{l}} := \{\, \boldsymbol{G}\boldsymbol{z} : \boldsymbol{l}^\top \boldsymbol{z} = 0 \,\} \subseteq \mathcal{C}.$$

Because $\boldsymbol{l} = \bigotimes_i (1 - x_{l,i}, x_{l,i})$, $\boldsymbol{l}^\top \mathbf{1} = \prod_i ((1 - x_{l,i}) + x_{l,i}) = 1$. Moreover, $\mathcal{C}_{\boldsymbol{l}}$ is a linear subcode of $\mathcal{C}$, so its relative distance is at least δ. Define:

$$\boldsymbol{D} := \boldsymbol{C} - (\boldsymbol{G}\mathbf{1})\boldsymbol{u}^\top.$$

We regard $\boldsymbol{D}$ as a partial matrix with the same unknown rows as $\mathcal{T}$. If State_1 is false after the verifier has accepted $v = \langle \boldsymbol{u}, \boldsymbol{r} \rangle$, some column of the partial matrix $\boldsymbol{D}$ is at least $\delta/2$-far from $\mathcal{C}_{\boldsymbol{l}}$ in partial-tree distance. Otherwise, for every $j \in [t]$ there would exist $\boldsymbol{z}_j$ satisfying $\boldsymbol{l}^\top \boldsymbol{z}_j = 0$ and $\delta_{H,\perp}(\boldsymbol{d}_j, \boldsymbol{G}\boldsymbol{z}_j) < \delta/2$. Letting the j-th column of $\boldsymbol{M}'$ be $\boldsymbol{z}_j + u_j \mathbf{1}$ would give:

$$\delta^t_{H,\perp}\big(\mathcal{T}, \boldsymbol{G}\boldsymbol{M}'\big) < \delta/2, \qquad \boldsymbol{l}^\top \boldsymbol{M}' = \boldsymbol{u}^\top.$$

If the proximity condition in State_1 were false, this would contradict the minimality of $\overline{\boldsymbol{M}}$. Otherwise, uniqueness within partial-tree distance $\delta/2$ would imply $\boldsymbol{M}' = \overline{\boldsymbol{M}}$, establishing State_1, again a contradiction.

Applying Eq. (10) to $\boldsymbol{D}$ and $\mathcal{C}_{\boldsymbol{l}}$ therefore gives:

$$\Pr_{\tilde{\boldsymbol{r}} \xleftarrow{\$} \mathbb{F}^t} [\delta_{H,\perp}(\boldsymbol{D}\tilde{\boldsymbol{r}}, \mathcal{C}_{\boldsymbol{l}}) < \delta'] \;\leq\; \frac{1}{(\delta/2 - \delta')|\mathbb{F}|} \leq \frac{1}{(\delta/3 - \delta')|\mathbb{F}|}.$$

In State_2, let $\boldsymbol{z} := \boldsymbol{m} - \tilde{u}\mathbf{1}$. The equality $\tilde{u} = \boldsymbol{l}^\top \boldsymbol{m}$ implies $\boldsymbol{l}^\top \boldsymbol{z} = 0$, and hence $\boldsymbol{G}\boldsymbol{z} \in \mathcal{C}_{\boldsymbol{l}}$. Furthermore, using $\tilde{u} = \langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle$, we have:

$$\boldsymbol{D}\tilde{\boldsymbol{r}} - \boldsymbol{G}\boldsymbol{z} = \boldsymbol{C}\tilde{\boldsymbol{r}} - \boldsymbol{G}\boldsymbol{m}.$$

Thus, the proximity condition in State_2 implies $\delta_{H,\perp}(\boldsymbol{D}\tilde{\boldsymbol{r}}, \mathcal{C}_{\boldsymbol{l}}) < \delta'$, which occurs with probability at most $1/((\delta/3 - \delta')|\mathbb{F}|)$ when State_1 is false.

$\mathsf{State}_2 \Rightarrow \mathsf{State}_3$. Define the set of rows on which the point-oracle check cannot accept by:

$$\mathcal{B} := \mathcal{U} \cup \{ i \in K(\mathcal{T}) : (\boldsymbol{C}\tilde{\boldsymbol{r}})_i \neq (\boldsymbol{G}\boldsymbol{m})_i \}.$$

The normalized size $|\mathcal{B}|/n$ is precisely $\delta_{H,\perp}(\mathcal{T}\tilde{\boldsymbol{r}}, \boldsymbol{G}\boldsymbol{m})$. Thus, if the proximity condition in State_2 is false, then $|\mathcal{B}|/n \geq \delta'$. Querying a row in $\mathcal{B}$ either encounters an unknown leaf, for which no authentication path is available, or detects an inconsistent value. Thus, whenever State_2 is false but the preceding scalar equalities hold, a random position detects the failure with probability at least

δ'. Since the verifier samples ℓ positions independently, the probability that all sampled positions miss $\mathcal{B}$ is at most $(1-\delta')^\ell$. By the choice:

$$\ell = \left\lceil \frac{-\lambda}{\log_2(1-\delta')} \right\rceil,$$

this is at most $2^{-\lambda}$.

$\mathsf{State}_3 \Rightarrow \mathsf{State}_4$. Conditioned on Merkle-path consistency, acceptance of the point-oracle queries enforces $\mathcal{I} \subseteq K(\mathcal{T})$ and $\boldsymbol{\iota} = \boldsymbol{\Xi C \tilde{r}}$. By the soundness of the matrix multiplication protocol, the probability that an incorrectly reduced multiplication claim is accepted is:

$$O\left(\frac{\log n + \log k}{|\mathbb{F}|}\right).$$

Since n is a constant multiple of k, this term is $O(\log k/|\mathbb{F}|)$.

Combining the above bounds and applying Lemma 6 in Appendix A.5, the protocol conditioned on adaptive Merkle-path consistency has computational soundness error:

$$2^{-\lambda} + O\left(\frac{\log k + (\delta/3 - \delta')^{-1}}{|\mathbb{F}|}\right).$$

Finally, we account for the oracle lift. As per Lemma 2, knowledge soundness of the two MLE-oracle protocols contributes $\kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]}$ and $\kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}$. The Merkle-authentication-path error of the source commitment contributes κ_{Hash}. The computational soundness error of $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}^t]}$ is therefore at most:

$$2^{-\lambda} + O\left(\frac{\log k + (\delta/3 - \delta')^{-1}}{|\mathbb{F}|}\right) + \kappa_{\mathsf{Hash}} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

Knowledge soundness. Conditioned on adaptive Merkle-path consistency, the partial tree $\mathcal{T}$ and the completion $\boldsymbol{C}$ fixed above are independent of the verifier challenges. By the knowledge soundness of the matrix multiplication protocol, except with error:

$$O\left(\frac{\log n + \log k}{|\mathbb{F}|}\right) = O\left(\frac{\log k}{|\mathbb{F}|}\right),$$

the extracted responses satisfy:

$$\tilde{u} = \boldsymbol{l}^\top \boldsymbol{m}, \qquad \boldsymbol{\iota} = \boldsymbol{\Xi G m}.$$

Once $\boldsymbol{m}$ has been fixed by the binding target commitment, both $\boldsymbol{C\tilde{r}}$ and $\boldsymbol{Gm}$ are fixed before the sampled-row consistency check. Let $\mathcal{B}$ be the union of the unknown rows and the known rows on which these two vectors disagree, as above. A query to any row in $\mathcal{B}$ rejects. Therefore, except with probability:

$$(1-\delta')^\ell \leq 2^{-\lambda},$$

an accepting transcript satisfies $|\mathcal{B}|/n < \delta'$. Equivalently:

$$\delta_{H,\perp}\big(\mathcal{T}\tilde{\boldsymbol{r}}, \boldsymbol{G}\boldsymbol{m}\big) < \delta'. \tag{11}$$

We next apply the partial-tree form of Lemma 1 to random linear combinations of the columns authenticated by $\mathcal{T}$. Define:

$$\epsilon := \max_{\boldsymbol{a}\in\mathbb{F}^t} \delta_{H,\perp}(\mathcal{T}\boldsymbol{a}, \mathcal{C}).$$

If $\epsilon \geq \delta/3$, choose $\boldsymbol{a}$ attaining this maximum. Extend $\boldsymbol{a}$ to the first column of an invertible matrix $\boldsymbol{A} \in \mathbb{F}^{t\times t}$. Because $\boldsymbol{A}\boldsymbol{z}$ is uniform in $\mathbb{F}^t$ for uniform $\boldsymbol{z}$, applying Eq. (10) to the partial matrix represented by $\mathcal{T}\boldsymbol{A}$ gives:

$$\Pr_{\tilde{\boldsymbol{r}}\xleftarrow{\$}\mathbb{F}^t}\left[\delta_{H,\perp}(\mathcal{T}\tilde{\boldsymbol{r}}, \mathcal{C}) < \delta'\right] \leq \frac{1}{(\epsilon-\delta')|\mathbb{F}|} \leq \frac{1}{(\delta/3-\delta')|\mathbb{F}|}.$$

We charge this probability to the knowledge-soundness error. In the remaining case, $\mathcal{T}\boldsymbol{a}$ is strictly within partial-tree distance $\delta/3$ of $\mathcal{C}$ for every $\boldsymbol{a} \in \mathbb{F}^t$.

Thus, for every $\boldsymbol{a} \in \mathbb{F}^t$, there exists a codeword $\boldsymbol{c}^\star(\boldsymbol{a}) \in \mathcal{C}$ such that:

$$\delta_{H,\perp}\big(\mathcal{T}\boldsymbol{a}, \boldsymbol{c}^\star(\boldsymbol{a})\big) < \delta/3.$$

This codeword is unique. Indeed, on the known rows, two such codewords differ on fewer than $2(\delta/3-\mu)n$ positions, and they can differ on at most μn unknown rows. Their relative distance is therefore less than $2\delta/3 - \mu < \delta$, contradicting the relative distance of $\mathcal{C}$.

The resulting map $\boldsymbol{c}^\star : \mathbb{F}^t \to \mathcal{C}$ is linear. For any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^t$, the three relevant error sets contain fewer than $(\delta/3-\mu)n$ known rows each. Hence:

$$\delta_H(\boldsymbol{c}^\star(\boldsymbol{a}+\boldsymbol{b}), \boldsymbol{c}^\star(\boldsymbol{a}) + \boldsymbol{c}^\star(\boldsymbol{b})) < \mu + 3(\delta/3-\mu) \leq \delta,$$

where the term μ accounts once for all unknown rows. Since both $\boldsymbol{c}^\star(\boldsymbol{a}+\boldsymbol{b})$ and $\boldsymbol{c}^\star(\boldsymbol{a}) + \boldsymbol{c}^\star(\boldsymbol{b})$ are codewords, they must be equal. Hence:

$$\boldsymbol{c}^\star(\boldsymbol{a}+\boldsymbol{b}) = \boldsymbol{c}^\star(\boldsymbol{a}) + \boldsymbol{c}^\star(\boldsymbol{b}).$$

For $\alpha \neq 0$, the codeword $\alpha\boldsymbol{c}^\star(\boldsymbol{a})$ is within partial-tree distance $\delta/3$ of $\mathcal{T}(\alpha\boldsymbol{a})$, so uniqueness gives $\boldsymbol{c}^\star(\alpha\boldsymbol{a}) = \alpha\boldsymbol{c}^\star(\boldsymbol{a})$. For $\alpha = 0$, the same identity follows from the fact that $\mathcal{T}\boldsymbol{0}$ has zero known entries and from uniqueness. Hence $\boldsymbol{c}^\star$ is linear.

Let $\boldsymbol{e}_j = (0,\ldots,0,1,0,\ldots,0)^\top \in \mathbb{F}^t$ be the j-th standard basis vector, so that $\mathcal{T}\boldsymbol{e}_j$ is the j-th partial column authenticated by $\mathcal{T}$. Define $\boldsymbol{m}_j^\star$ and $\boldsymbol{M}^\star$ by:

$$\boldsymbol{c}^\star(\boldsymbol{e}_j) = \boldsymbol{G}\boldsymbol{m}_j^\star, \qquad \boldsymbol{M}^\star := [\,\boldsymbol{m}_1^\star \mid \ldots \mid \boldsymbol{m}_t^\star\,].$$

For any $\boldsymbol{a} = \sum_{j=1}^t a_j\boldsymbol{e}_j$, linearity of $\boldsymbol{c}^\star$ gives:

$$\boldsymbol{c}^\star(\boldsymbol{a}) = \sum_{j=1}^t a_j\boldsymbol{c}^\star(\boldsymbol{e}_j) = \boldsymbol{G}\boldsymbol{M}^\star\boldsymbol{a}.$$

Applying the proximity guarantee to the standard basis vectors yields

$$\delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M}^\star)\big) < \delta/3. \tag{12}$$

Thus, there exists a matrix $\boldsymbol{M}^\star$ whose encoding is close to the partial tree $\mathcal{T}$. We next show how to extract $\boldsymbol{M}^\star$.

Fix the transcript prefix immediately before $\tilde{\boldsymbol{r}}$ is sampled. The extractor rewinds only from complete subprotocol boundaries: it saves the state before each target, matrix-multiplication, or generator-matrix subprotocol, applies the corresponding extractor to that suffix, and then restores the common prefix.

Call an accepting continuation *good* if the extracted target and generator-matrix messages satisfy their oracle relations, the extracted matrix-multiplication equalities hold, and Eq. (11) holds. This predicate is used only in the analysis and need not be evaluated by the extractor.

The challenge space is $\mathbb{J} := \mathbb{F}^t$. Let $\mathcal{J}$ consist of the subsets of $\mathbb{J}$ that span $\mathbb{F}^t$. A qualifying generalized transcript tree requires at most t good continuations whose first challenges span $\mathbb{F}^t$.

Consider t good accepting continuations with linearly independent challenges $\tilde{\boldsymbol{r}}_1, \ldots, \tilde{\boldsymbol{r}}_t$ and extracted target messages $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_t$. For every $i \in [t]$, the codewords:

$$\boldsymbol{G}\boldsymbol{m}_i \qquad \text{and} \qquad \boldsymbol{c}^\star(\tilde{\boldsymbol{r}}_i) = \boldsymbol{G}\boldsymbol{M}^\star\tilde{\boldsymbol{r}}_i$$

are within partial-tree distances δ' and $\delta/3$, respectively, of $\mathcal{T}\tilde{\boldsymbol{r}}_i$. Their relative distance is therefore less than

$$\delta' + \delta/3 - \mu < \delta.$$

By the relative distance of $\mathcal{C}$, the two codewords are equal. Since $\boldsymbol{G}$ has full column rank:

$$\boldsymbol{m}_i = \boldsymbol{M}^\star\tilde{\boldsymbol{r}}_i, \qquad i \in [t]. \tag{13}$$

Define:

$$\boldsymbol{R} := [\,\tilde{\boldsymbol{r}}_1 \mid \ldots \mid \tilde{\boldsymbol{r}}_t\,], \qquad \boldsymbol{Y} := [\,\boldsymbol{m}_1 \mid \ldots \mid \boldsymbol{m}_t\,].$$

Because $\boldsymbol{R}$ is invertible, the extractor outputs:

$$\boldsymbol{M} := \boldsymbol{Y}\boldsymbol{R}^{-1}.$$

Eq. (13) implies $\boldsymbol{M} = \boldsymbol{M}^\star$. Moreover, the scalar checks in the t continuations give:

$$\boldsymbol{l}^\top\boldsymbol{Y} = \boldsymbol{u}^\top\boldsymbol{R},$$

and hence: $\boldsymbol{l}^\top\boldsymbol{M} = \boldsymbol{u}^\top$. Together with: $v = \langle \boldsymbol{u}, \boldsymbol{r}\rangle$, this gives:

$$v = \boldsymbol{l}^\top\boldsymbol{M}\boldsymbol{r}.$$

Finally, Eq. (12) gives

$$\delta^t_{H,\perp}\big(\mathcal{T}, \mathsf{Enc}[\mathcal{C}^t](\boldsymbol{M})\big) < \delta/3 < \delta/2.$$

Thus, $\boldsymbol{M}$ is a valid witness for the source MLE-oracle relation.

Every nonspanning subset of $\mathbb{F}^t$ is contained in a proper linear subspace and therefore contains at most $|\mathbb{F}|^{t-1}$ challenges. Consequently, the first-layer access structure contributes knowledge error at most $1/|\mathbb{F}|$.

For a fixed prefix, both $\mathcal{T}$ and ϵ are fixed. If $\epsilon \geq \delta/3$, an accepting continuation either violates Eq. (11), which occurs with probability at most $2^{-\lambda}$, or satisfies it, which by Eq. (10) occurs with probability at most:

$$\frac{1}{(\delta/3-\delta')|\mathbb{F}|}.$$

We distinguish two cases.

If $\epsilon \geq \delta/3$, an accepting continuation either violates Eq. (11), which occurs with probability at most $2^{-\lambda}$, or satisfies that equation. In the latter case, Eq. (10) bounds its probability by:

$$\frac{1}{(\delta/3-\delta')|\mathbb{F}|}.$$

Thus, the entire accepting probability of such prefixes is charged to these two error terms.

Suppose instead that $\epsilon < \delta/3$. In this case, the preceding argument defines a fixed matrix $\boldsymbol{M}^\star$. An accepting continuation is good unless the sampled-row check fails, the matrix-multiplication extractor fails, or one of the two MLE-oracle extractors fails. The corresponding error is at most:

$$2^{-\lambda} + O\left(\frac{\log k}{|\mathbb{F}|}\right) + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

Averaging over all pre-challenge prefixes and combining the two cases, the accepting probability not covered by good continuations is at most:

$$2^{-\lambda} + \frac{1}{(\delta/3-\delta')|\mathbb{F}|} + O\left(\frac{\log k}{|\mathbb{F}|}\right) + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

Each complete-subprotocol error is measured over its prefix distribution and is therefore charged once, before rewinding constructs the continuation tree.

Consider now the remaining good continuations. If their first challenges do not span $\mathbb{F}^t$, they lie in a proper linear subspace of $\mathbb{F}^t$, which contains at most $|\mathbb{F}|^{t-1}$ challenges. This contributes knowledge error at most $1/|\mathbb{F}|$. Otherwise, the tree-finding extractor obtains t good continuations with linearly independent first challenges, from which the preceding algebra extracts $\boldsymbol{M} = \boldsymbol{M}^\star$.

Adaptive Merkle-path inconsistency contributes an additional κ_{Hash}. Thus, the total knowledge-soundness error is:

$$2^{-\lambda} + \frac{1}{(\delta/3-\delta')|\mathbb{F}|} + O\left(\frac{\log k}{|\mathbb{F}|}\right) + \kappa_{\mathsf{Hash}} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \kappa^{\mathsf{ks}}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

All rewinding starts after the fixed transcript prefix and therefore preserves it. The qualifying tree has at most t leaves, and each constituent extractor runs

in expected polynomial time. Hence the complete extractor also runs in expected polynomial time.

Efficiency. The prover computes $\boldsymbol{m} = \boldsymbol{M}\tilde{\boldsymbol{r}}$, evaluates the inner product $\langle \boldsymbol{u}, \tilde{\boldsymbol{r}} \rangle$, and performs $O(kt)$ field multiplications required by Thaler's matrix multiplication protocol. Thus, beyond the invoked MLE-oracle protocols, the additional prover time is $O(kt)$, and the total prover time is:

$$O(kt) + \mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{TP}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

The proof contains the vector $\boldsymbol{u} \in \mathbb{F}^t$, the sampled values for ℓ positions and their authentication paths, the reduced MLE-query data, and the two invoked MLE-oracle proofs. This gives proof size:

$$O\big(\log k + \ell(t + \log k)\big) + \mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{S}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

The verifier performs the corresponding inner-product proof verifications, Merkle-path verifications, sampled-row consistency verifications, and the two invoked MLE-oracle verifications. Hence the verifier time is:

$$O\big(\log k + \ell(t + \log k)\big) + \mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}'^t]} + \mathsf{TV}_{\mathcal{O}_{\mathsf{MLE}}(\boldsymbol{G})}.$$

This proves the theorem. □

C.3 Proof of Theorem 3

Proof. We prove the theorem by induction on d.

For the base case $d = 1$, Algorithm 1 coincides with the one-step code-switching protocol of Algorithm 2. Hence the claim follows directly from Theorem 2.

Assume that the theorem holds for $d = d^*$. Then Algorithm 1 applied to the code sequence $\{\mathcal{C}_j\}_{j=1}^{d^*}$ instantiates an MLE-oracle protocol $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}_{d^*}^t]}$ with the stated efficiency and soundness bounds.

Now consider $d = d^* + 1$. The first iteration of the loop in Algorithm 1 performs a one-step code switch from $\mathcal{C}_{d^*+1}^t$ to $\mathcal{C}_{d^*}^t$. Equivalently, it is an instance of Algorithm 2 with:

$$\mathcal{C} = \mathcal{C}_{d^*+1}, \quad \mathcal{C}' = \mathcal{C}_{d^*},$$

and with the established MLE oracle $\Pi_{\mathcal{O}_{\mathsf{MLE}}[\mathcal{C}_{d^*}^t]}$ implemented by the recursive protocol of depth d^*.

By Theorem 2, this yields an MLE-oracle protocol for $\mathcal{C}_{d^*+1}^t$ whose efficiency and soundness errors are obtained, according to Lemmas 6 and 7, by adding the one-step code-switching costs and errors to those of the depth-d^* recursive oracle. These additions give exactly the bounds stated in Theorem 3 for $d = d^* + 1$.

Therefore, by induction, the theorem holds for all $d \geq 1$. □

C.4 Proof of Theorem 4

Proof. By Lemma 5 in Appendix A.4, a computationally sound protocol followed by queries to knowledge-sound oracles yields a knowledge-sound protocol. Since the Decide algorithm invokes the knowledge-sound MLE oracles for $\{\boldsymbol{G}_j\}_{j=1}^d$, it suffices to prove that the sequential execution of Accum is computationally sound.

We use the round-by-round soundness framework of Definition 21 in Appendix A.5. The fresh claims $\{v_j^{(i)}, \boldsymbol{l}_j^{(i)}, \boldsymbol{r}_j^{(i)}\}_{j=1}^d$ are the output of the preceding recursive protocol. In the analysis of Accum, denote the event that these fresh claims are correct by:

$$\mathsf{Fresh}_i \;=\; \left\{ \bigwedge_{j=1}^{d} v_j^{(i)} = \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \boldsymbol{r}_j^{(i)} \right\}.$$

Whether Fresh_i holds is determined by the preceding recursive protocol. The role of Accum is to prevent a false Fresh_i from being transformed into a valid updated accumulator while accepting. The state tracked by Accum is the validity of the current accumulator. For each accumulation step i, define:

$$\mathsf{State}_i \;=\; \left\{ \bigwedge_{j=1}^{d} \bar{v}_j^{(i)} = \bar{\boldsymbol{l}}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)} \right\}.$$

The initial accumulator is $\mathsf{acc}^{(0)} = \{(0, \mathbf{0}, \mathbf{0})\}_{j=1}^d$, so State_0 holds. The one-step computational soundness is the condition: if $\mathsf{State}_{i-1} \wedge \mathsf{Fresh}_i$ is false, then except with probability $O(d \log k_d / |\mathbb{F}|)$, either the updated accumulator does not satisfy State_i or Accum rejects. Equivalently, the event:

$$\neg(\mathsf{State}_{i-1} \wedge \mathsf{Fresh}_i) \;\wedge\; \mathsf{State}_i \;\wedge\; \{\mathsf{Accum} \text{ accepts}\}$$

has probability $O(d \log k_d / |\mathbb{F}|)$.

Security of per-level protocol. Fix any level $j \in [d]$, and consider the update of the j-th accumulator claim within step i. We introduce four sub-states tracking the soundness within this update:

- $\mathsf{Sub\text{-}state}_0^{(j)}$: the input claims are valid, *i.e.* $\bar{v}_j^{(i-1)} = \bar{\boldsymbol{l}}_j^{(i-1)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i-1)}$ and $v_j^{(i)} = \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \boldsymbol{r}_j^{(i)}$.
- $\mathsf{Sub\text{-}state}_1^{(j)}$: after the two parallel inner-product proofs accept with shared tensor challenge $\bar{\boldsymbol{r}}_j^{(i)} \in \mathbb{T}(k_j)$, the reduced scalar claims $\tilde{v}_j^{(i-1)} = \bar{\boldsymbol{l}}_j^{(i-1)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$ and $\tilde{v}_j^{(i)} = \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$ are correct.
- $\mathsf{Sub\text{-}state}_2^{(j)}$: after sampling $z_j \xleftarrow{\$} \mathbb{F}$ and forming $\tilde{v}_j = \tilde{v}_j^{(i-1)} + z_j \tilde{v}_j^{(i)}$, the batched relation $\tilde{v}_j = \langle \tilde{\boldsymbol{l}}_j, \tilde{\boldsymbol{h}}_j \rangle$, $\tilde{\boldsymbol{l}}_j = \bar{\boldsymbol{l}}_j^{(i-1)} + z_j \boldsymbol{l}_j^{(i)}$, and $\tilde{\boldsymbol{h}}_j = \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$ holds.
- $\mathsf{Sub\text{-}state}_3^{(j)}$: after the second inner-product proof with tensor challenge $\bar{\boldsymbol{l}}_j^{(i)} \in \mathbb{T}(n_j)$, the new accumulator claim $\bar{v}_j^{(i)} = \langle \bar{\boldsymbol{l}}_j^{(i)}, \tilde{\boldsymbol{h}}_j \rangle = \bar{\boldsymbol{l}}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$ holds.

By construction, $\mathsf{Sub\text{-}state}_3^{(j)}$ is exactly the level-j component of State_i, while $\mathsf{Sub\text{-}state}_0^{(j)}$ is the level-j component of State_{i-1} and the level-j component of Fresh_i. Hence, if $\mathsf{State}_{i-1} \wedge \mathsf{Fresh}_i$ is false while State_i holds, then there exists some $j \in [d]$ for which $\mathsf{Sub\text{-}state}_0^{(j)}$ is false but $\mathsf{Sub\text{-}state}_3^{(j)}$ is true.

Sub-state transitions.

- $\mathsf{Sub\text{-}state}_0^{(j)} \Rightarrow \mathsf{Sub\text{-}state}_1^{(j)}$: Two inner-product protocols are run in parallel with shared verifier challenges $\boldsymbol{x} \in \mathbb{F}^{\log_2 k_j}$. Under the event that both inner-product protocols accept, Fact 1 guarantees, except with computational soundness error $O(\log k_j/|\mathbb{F}|)$ in a single run, that the two input inner-product claims are consistently reduced to the claims with respect to the tensor expansion $\bar{\boldsymbol{r}}_j^{(i)} = \mathsf{ts}(\boldsymbol{x})$. Hence, conditioned on acceptance of these two inner-product protocols, $\mathsf{Sub\text{-}state}_0^{(j)}$ implies $\mathsf{Sub\text{-}state}_1^{(j)}$ except with error $O(\log k_j/|\mathbb{F}|)$.
- $\mathsf{Sub\text{-}state}_1^{(j)} \Rightarrow \mathsf{Sub\text{-}state}_2^{(j)}$: If $\mathsf{Sub\text{-}state}_1^{(j)}$ holds, then the two reduced scalar equalities give:

$$\tilde{v}_j \;=\; \tilde{v}_j^{(i-1)} + z_j \tilde{v}_j^{(i)} \;=\; (\bar{\boldsymbol{l}}_j^{(i-1)} + z_j \boldsymbol{l}_j^{(i)})^\top \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)} \;=\; \langle \tilde{\boldsymbol{l}}_j, \tilde{\boldsymbol{h}}_j \rangle,$$

  which is exactly $\mathsf{Sub\text{-}state}_2^{(j)}$. If $\mathsf{Sub\text{-}state}_1^{(j)}$ is false, define the two errors: $a_j = \tilde{v}_j^{(i-1)} - \bar{\boldsymbol{l}}_j^{(i-1)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$, and $b_j = \tilde{v}_j^{(i)} - \boldsymbol{l}_j^{(i)\top} \boldsymbol{G}_j \bar{\boldsymbol{r}}_j^{(i)}$. Then $\mathsf{Sub\text{-}state}_1^{(j)}$ is false exactly when $(a_j, b_j) \neq (0, 0)$. Using the definitions of $\tilde{\boldsymbol{l}}_j$ and $\tilde{\boldsymbol{h}}_j$, the batched identity in $\mathsf{Sub\text{-}state}_2^{(j)}$ is equivalent to: $a_j + z_j b_j = 0$. This is a non-zero polynomial of degree at most one in z_j. Since $z_j \xleftarrow{\$} \mathbb{F}$, it vanishes with probability at most $1/|\mathbb{F}|$. Hence, if $\mathsf{Sub\text{-}state}_1^{(j)}$ is false, the probability that $\mathsf{Sub\text{-}state}_2^{(j)}$ is true is at most $1/|\mathbb{F}|$.
- $\mathsf{Sub\text{-}state}_2^{(j)} \Rightarrow \mathsf{Sub\text{-}state}_3^{(j)}$: The final inner-product proof reduces the claim $\tilde{v}_j = \langle \tilde{\boldsymbol{l}}_j, \tilde{\boldsymbol{h}}_j \rangle$ on vectors of length n_j to the updated accumulator claim $\bar{v}_j^{(i)} = \langle \bar{\boldsymbol{l}}_j^{(i)}, \tilde{\boldsymbol{h}}_j \rangle$. By Fact 1, the probability that $\mathsf{Sub\text{-}state}_2^{(j)}$ is false, $\mathsf{Sub\text{-}state}_3^{(j)}$ is true, and the final inner-product protocol accepts is at most $O(\log k_j/|\mathbb{F}|)$.

Per-instance computational soundness. If $\mathsf{Sub\text{-}state}_0^{(j)}$ is false but $\mathsf{Sub\text{-}state}_3^{(j)}$ is true and all subprotocols accept, then at least one of the three local transitions above must fail. Summing the errors above, the probability of this bad event for the j-th level update is at most $O(\log k_j/|\mathbb{F}|)$. Taking a union bound over $j \in [d]$, one Accum step incurs computational soundness error at most:

$$\sum_{j=1}^{d} O\left(\frac{\log k_j}{|\mathbb{F}|}\right) \;\leq\; O\left(\frac{d \log k_d}{|\mathbb{F}|}\right).$$

Composition over T instances. By Lemma 6 in Appendix A.5, the sequential composition of the Accum algorithm over T instances is computationally sound

with accumulated error $O(Td \log k_d/|\mathbb{F}|)$. Combining this bound with the knowledge soundness of the final MLE oracles in Decide, via Lemma 5, yields the claimed computational soundness and knowledge soundness errors of the accumulation scheme. □

C.5 Proof of Theorem 5

Proof. We analyze one recursive level and then compose the reductions from level d to level 0.

Security. At level j, the setup, auxiliary, and carry commitments jointly authenticate:

$$\boldsymbol{U}_j := [\, \boldsymbol{T}_j \mid \boldsymbol{H}_j \mid \boldsymbol{S}_j \mid \boldsymbol{W}_j \,]$$

under $\widehat{\boldsymbol{G}}_j$. The setup commitment is fixed in the public parameters, while the auxiliary and carry commitments are fixed during proving.

Conditioned on Merkle-path consistency, the common authenticated rows of these commitments define a partial row oracle for $\widehat{\boldsymbol{G}}_j \boldsymbol{U}_j$. Each source-row query opens the same row under every nonempty constituent commitment. Their roots can therefore be treated as a single virtual row oracle, to which the partial-tree and extraction arguments of Theorem 2 apply. The consistency errors of the constituent roots are accounted for by a union bound.

The protocol Π_{G_j} reduces the accumulated MLE claim on $\boldsymbol{G}_j$ to claims on $\boldsymbol{H}_j$ and $\boldsymbol{S}_j$. Once all commitments have been fixed, these claims are combined with those on $\boldsymbol{T}_j$ and $\boldsymbol{W}_j$ into one MLE claim on $\boldsymbol{U}_j$. For $j < d$, the number of columns of $\boldsymbol{U}_j$ is at most:

$$h + s + r(t-1) + t = O(h + s + rt);$$

the outermost level contains fewer columns because $\boldsymbol{T}_d$ and $\boldsymbol{W}_d$ are empty. The column-compression challenge uses this dimension, and the knowledge-error analysis of Theorem 2 remains unchanged.

Code switching reduces the opening under $\widehat{\mathcal{C}}_j$ to an opening of $\boldsymbol{W}_{j-1}$ under $\widehat{\mathcal{C}}^t_{j-1}$, together with an MLE claim on $\widehat{\boldsymbol{G}}_j$. Expanding its $\log_2 t$ RF layers reduces the latter to claims on $\boldsymbol{T}_{j-1}$ and $\boldsymbol{G}_{j-1}$. The claim on $\boldsymbol{G}_{j-1}$ is accumulated with the other claims at level $j-1$, and applying $\Pi_{G_{j-1}}$ reproduces the same opening problem. The recursion terminates at level 0, where the verifier directly checks the revealed $\boldsymbol{W}_0$ and evaluates the remaining claims. This costs $O(1)$ because $k_0, n_0 = O(1)$.

At level j, the reduction Π_{G_j}, the accumulation and matrix-expression proofs, and the interleaved-code reduction contribute total soundness error:

$$O\left(\frac{h + s + rt + \log k_j + (\widehat{\delta}_j/3 - \widehat{\delta}'_j)^{-1}}{|\mathbb{F}|}\right).$$

The proximity test uses:

$$\ell_j = \left\lceil \frac{-\lambda}{\log_2(1 - \widehat{\delta}'_j)} \right\rceil$$

queries and therefore fails with probability at most $2^{-\lambda}$. Hence the soundness error of the level-j reduction is:

$$2^{-\lambda} + O\left(\frac{h + s + rt + \log k_j + (\widehat{\delta}_j/3 - \widehat{\delta}'_j)^{-1}}{|\mathbb{F}|}\right),$$

excluding Merkle-path inconsistency.

There are three nonempty point-oracle commitments per level. A union bound over these commitments and the d recursive levels gives total error:

$$d \cdot 2^{-\lambda} + O\left(\frac{1}{|\mathbb{F}|}\sum_{j=1}^{d}\left(h + s + rt + \log k_j + (\widehat{\delta}_j/3 - \widehat{\delta}'_j)^{-1}\right)\right) + 3d \cdot \kappa_{\mathsf{Hash}}.$$

This proves computational soundness. For knowledge soundness, apply the extractor of Theorem 2 to the virtual row oracle at each level. It extracts $\boldsymbol{W}_{j-1}$ while charging the sampled-row, matrix-reduction, and constituent-oracle errors once at that level. Each level is prefix preserving, so Lemma 7 composes the extractors from level d to level 0, yielding the same error bound.

Efficiency. At level j, running Π_{G_j}, encoding the columns of $\boldsymbol{U}_j$, and reducing the $\log_2 t$ RF layers costs:

$$O(\log t \cdot k_j(h + s + rt)).$$

Because $k_j = tk_{j-1}$, $\sum_{j=1}^{d} k_j = O(k_d)$. The total prover time is therefore:

$$O(\log t \cdot k_d(h + s + rt)).$$

At level j, the proof size and verifier time are:

$$O(\ell_j(h + s + rt + \log k_j)),$$

which also absorbs the $O(\log k_j)$ cost of Π_{G_j}. Summing over all levels gives:

$$O\left(\sum_{j=1}^{d} \ell_j(h + s + rt + \log k_j)\right),$$

for both proof size and verifier time. □

C.6 Proof of Theorem 6

Proof. The recursive protocol is truncated before opening the generator-matrix MLE claims, which are instead accumulated across the T opening instances and verified once by Decide. The computational- and knowledge-soundness guarantees therefore follow by composing Theorems 3, 4, and 5. Lemmas 6 and 7 apply at the subprotocol boundaries, allowing their soundness errors to be added.

Combining the three theorems gives the following bounds for T opening instances:

– Prover time:

$$O(k_d tT + \log t \cdot k_d(h + s + rt)) .$$

– Proof size and verifier time:

$$O\left(d\ell(t + \log k_d)T + \sum_{j=1}^{d} \ell_j \left(h + s + rt + \log k_j\right) \right) .$$

– Computational- and knowledge-soundness errors:

$$\begin{aligned} & d(T+1)2^{-\lambda} + O\left(\frac{dT\left(\log k_d + (\delta/3 - \delta')^{-1}\right)}{|\mathbb{F}|} \right) \\ & \quad + O\left(\frac{1}{|\mathbb{F}|} \sum_{j=1}^{d} \left(h + s + rt + \log k_j + (\widehat{\delta}_j/3 - \widehat{\delta}'_j)^{-1} \right) \right) \\ & \quad + \big(T(d+1) + 3d\big)\kappa_{\mathsf{Hash}}. \end{aligned}$$

Because each $\mathcal{C}_j$ has a linear-time encoder, its encoding computation admits a linear-size arithmetic-circuit arithmetization, such as an R1CS representation. The generic construction of Section 4.3 then yields a generator-matrix reduction satisfying the interface of Theorem 5. This generic reduction achieves the required asymptotic complexity, although code-specific reductions may be more efficient in practice. In particular, its fixed and auxiliary matrices can be represented using $h, s = O(1)$ columns.

Since t, r, h, s, and k_0 are constants and $N = k_d t$, we have:

$$k_d = \Theta(N), \qquad \log k_d = \Theta(\log N), \qquad d = \log_t(k_d/k_0) = \Theta(\log N).$$

Choose δ' as a fixed constant fraction of δ below $\delta/3$, and choose each $\widehat{\delta}'_j$ as a fixed constant fraction of $\widehat{\delta}_j$ below $\widehat{\delta}_j/3$. Because:

$$\delta = \Omega(1) \qquad \text{and} \qquad \widehat{\delta}_j \geq \widehat{\delta} = \Omega(1),$$

we obtain:

$$\ell = O(\lambda) \qquad \text{and} \qquad \ell_j = O(\lambda) \quad \text{for every } j \in [d].$$

Prover time. Since $T \geq 1$:

$$O(k_d tT + \log t \cdot k_d(h + s + rt)) = O(NT + N) \quad = O(TN).$$

Proof size and verifier time. The T recursive openings contribute:

$$O(d\ell(t + \log k_d)T) = O\big(T\lambda \log^2 N\big) .$$

The one-time Decide phase contributes:

$$O\left(\sum_{j=1}^{d} \ell_j(h+s+rt+\log k_j)\right) = O\left(\lambda \sum_{j=1}^{d}(1+\log k_j)\right) \quad = O\big(\lambda \log^2 N\big)\,.$$

Therefore, the total proof size and verifier time are:

$$O\big(T\lambda \log^2 N\big)\,.$$

Computational- and knowledge-soundness errors. The first term satisfies:

$$d(T+1)2^{-\lambda} = O\big(T \log N \cdot 2^{-\lambda}\big)\,.$$

The second and third terms satisfy:

$$O\left(\frac{dT\left(\log k_d + (\delta/3-\delta')^{-1}\right)}{|\mathbb{F}|}\right)$$
$$+\, O\left(\frac{1}{|\mathbb{F}|}\sum_{j=1}^{d}\left(h+s+rt+\log k_j + (\widehat{\delta}_j/3-\widehat{\delta}'_j)^{-1}\right)\right) = O\left(T\cdot\frac{\log^2 N}{|\mathbb{F}|}\right).$$

The last term satisfies:

$$\big(T(d+1)+4d\big)\kappa_{\mathsf{Hash}} = O(T\log N\cdot\kappa_{\mathsf{Hash}})\,.$$

Combining these bounds gives:

$$O\left(T\cdot\frac{\log^2 N}{|\mathbb{F}|} + T\log N\cdot\left(2^{-\lambda}+\kappa_{\mathsf{Hash}}\right)\right).$$

This proves the theorem. □

C.7 Proof of Fact 4

Proof. By the tensor-product definitions of $\boldsymbol{\Phi}_n$, $\boldsymbol{l}$, and $\boldsymbol{r}$, we have:

$$\boldsymbol{l}^\top \boldsymbol{\Phi}_n \boldsymbol{r} = \prod_{i=1}^{\log_2 n} \left(1-x_{l,i}\;\; x_{l,i}\right)\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1-x_{r,i}\\ x_{r,i}\end{pmatrix}.$$

For each i, the corresponding factor is:

$$(1-x_{l,i}) + x_{l,i}(1-2x_{r,i}) = 1-2x_{l,i}x_{r,i}.$$

Multiplying these factors over all $i \in [\log_2 n]$ proves the claim. □