Succinct Arguments over Towers of Binary Fields

Benjamin E. Diamond
Ulvetanna
bdiamond@ulvetanna.io

Jim Posen
Ulvetanna
jposen@ulvetanna.io

Abstract

We introduce an efficient SNARK for *towers of binary fields*. Adapting Brakedown (CRYPTO '23), we construct a multilinear polynomial commitment scheme suitable for polynomials over tiny fields, including that with 2 elements. Our commitment scheme, unlike those of previous works, treats small-field polynomials with *zero embedding overhead*. We further introduce binary-field adaptations of HyperPlonk's (EUROCRYPT '23) product and permutation checks, as well as of Lasso's lookup. Our scheme's binary PLONKish variant captures standard hash functions—like Keccak-256 and Grøstl—extremely efficiently. With recourse to thorough performance benchmarks, we argue that our scheme can efficiently generate precisely those Keccak-256-proofs which critically underlie modern efforts to scale Ethereum.

1 Introduction

Succinct non-interactive arguments of knowledge, or SNARKs, have witnessed a recent surge of interest and application in blockchain protocols. Though long seen as impractical due to performance limitations, SNARKs are now a viable solution to the issue of blockchain scalability, thanks to a renewed focus on improving concrete efficiency.

Many modern SNARKs are constructed according to a framework that compiles *polynomial interactive oracle proofs* into succinct arguments of knowledge by means of a *polynomial commitment scheme*. This framework was formalized in the works of Bünz, Fisch and Szepieniec [BFS20] and Chiesa et al. [Chi+20]. The latter—along with several previous works, like Maller, Bowe, Kohlweiss, and Meiklejohn's *SONIC* [Mal+19] and Gabizon, Williamson, and Ciobotaru's *PLONK* [GWC19], in which the polynomial IOP framework is implicit—uses a polynomial commitment scheme due to Kate, Zaverucha, and Goldberg [KZG10] that relies on the hardness of the discrete logarithm problem in elliptic curve groups.

In contrast to SNARK constructions from elliptic curve cryptography, Ben-Sasson et al.'s highly influential *Fast Reed–Solomon IOP of Proximity* (FRI) [Ben+18a] has reënergized an alternative approach dating originally to Killian [Kil92]. Operating in the *interactive oracle proof* (*IOP*) model [BCS16], these schemes achieve succinct arguments with the aid of linear error-correcting codes and collision-resistant hash functions. The most popular such scheme is the *Scalable Transparent Arguments of Knowledge* (STARK) protocol of Ben-Sasson, Bentov, Horesh, and Riabzev [Ben+18b]. Subsequent expositions have reinterpreted STARKs in the polynomial IOP framework discussed above (see for example [Hab22]); in this light, we freely refer henceforth to the FRI polynomial commitment scheme, or FRI-PCS.

FRI-PCS is not the sole polynomial commitment scheme that leverages linear codes and hash functions. Golovnev et al.'s *Brakedown* polynomial commitment scheme [Gol+23], which distills ideas from Bootle, Chiesa, and Groth [BCG20] and Ames, Hazay, Ishai, and Venkitasubramaniam [Ame+23], also uses linear error-correcting codes in the IOP model. Asymptotically, Brakedown's verifier and proof size both grow on the order of the square root of the size of the polynomial being committed. Diamond and Posen [DP23] improve the concrete efficiency of Brakedown by a factor of roughly 2. While Brakedown's asymptotic verifier complexity is less favorable than that of FRI, for many practical parameter choices, the difference in concrete efficiency is minimal and the improvement in prover efficiency presents a compelling tradeoff.

The disadvantage of Brakedown's worse asymptotic complexity is further mitigated by the common practice of using *recursive proof composition* to scale verifiable computation. Instead of proving an entire statement or virtual machine execution in a single SNARK, one can often split up the statement in such

a way that allows it to be verified incrementally. Valiant shows in [Val08] that *incrementally verifiable computation* can be realized through recursive SNARK composition. A long virtual machine execution, say, could be proven using only SNARKs for circuits of bounded size. The implication is that a series of inner SNARKs with large proof sizes but fast proving times may be recursively composed with an outer SNARK with small proof size and a relatively slower proving time, producing a hybrid system with small proof size *and* fast proving time for large computations.

The class of SNARKs typified by the use of FRI-PCS or a Brakedown-style polynomial commitment scheme has three properties that lead to advantages in proving performance over SNARKs from elliptic curve group assumptions.

- **Operation over small fields.** Whereas elliptic curve groups must be on the order of 256 bits to attain a standard security level, the *ethSTARK* [Sta21] and *Plonky2* [Pol22] systems pioneer an alternative design, characterized by the use of smaller fields (specifically, of prime fields on the order of 64 bits). These systems leverage the relative efficiency of small-field arithmetic, and achieve state-of-the-art proving performance. Moreover, these protocols' use only of small-field elements moreover reduces their storage requirements, which in turn leads to better cache-efficiency on CPUs.

- **Flexibility in field selection.** Beyond just the flexibility of field size, these schemes permit the choice of fields with particular computational characteristics. *Plonky2* [Pol22], for example, highlights the benefits of the field $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$, termed the *Goldilocks field* by the authors. This prime modulus is a Solinas prime—that is, a prime of the form $\phi^2 - \phi + 1$, where here $\phi = 2^{32}$—and consequently admits an efficient procedure for modular reduction.

- **Cheaper cryptographic primitives.** Standard-issue collision-resistant hash functions are much faster than elliptic curve primitives.

As a rough comparison, committing 1 MiB of data with a FRI-based polynomial commitment scheme in the Goldilocks field using Keccak-256 is about 7-fold faster than committing with the scheme of [KZG10] over the BN254 bilinear group. We note that committing polynomials often accounts for the majority of the cost in proving a SNARK.

Given the performance gains unlocked by the use of smaller finite fields, one perspective on our work presented here is to extend this trend to its logical conclusion: SNARKS over the smallest field, $\mathbb{F}_2$.

Binary fields. Finite fields of characteristic 2, or *binary fields*, have a rich history in cryptography. The AES block cipher, and the GMAC message authentication code standardized for use alongside AES, famously use the binary fields $\mathbb{F}_2^8$ and $\mathbb{F}_2^{128}$, respectively. There is also an important line of research in cryptographically secure elliptic curves over binary fields; these curves feature uniquely efficient circuit instantiations. We recall some basic properties of binary fields which account for their applicability in cryptography. The elements of the field $\mathbb{F}_{2^k}$ can be unambiguously represented in k bits; for example, there is a bijection between bytes of data and the finite field $\mathbb{F}_{2^8}$. Field addition corresponds to the logical exclusive or operation (XOR) on these bit representations. Also, squaring elements of a binary field is significantly less expensive than multiplication of two distinct elements, thanks to the fact that $(x+y)^2 = x^2 + y^2$ for any x, y in these fields, a property sometimes referred to as the "freshman's dream."

In this work, we present a SNARK construction over the field $\mathbb{F}_2$ that competes favorably with state-of-the-art systems built with prime fields. Moreover, we argue that in the case of proving computations that depend heavily on bitwise operations, such the SHA-256 and Keccak-256 hash functions, our system outperforms the prime field-based alternatives. We must acknowledge that this is not the first work to consider SNARKs over characteristic-2 fields; it is the first we are aware of, however, to give a SNARK construction over $\mathbb{F}_2$ specifically, while avoiding *embedding overhead*, a term which we now explain. While the work of [Ben+18b] does give a STARK construction over characteristic-2 fields, naïvely applying small field techniques does not yield optimal concrete efficiency over $\mathbb{F}_2$ for a simple reason: the alphabet of Reed–Solomon codes must be at least as large as the code length. Even if not for this limitation, which comes from the FRI IOP of proximity, the ALI protocol in STARK—as well as DEEP-ALI from the successor work of Ben-Sasson, Goldberg, Kopparty, and Saraf [Ben+19]—uses fast polynomial multiplication techniques. Fast

multiplication of polynomials over $\mathbb{F}_2$ requires embedding $\mathbb{F}_2$-elements into a larger extension field, effectively limiting the field to have size on the same order as the size of the witness.

An influential line of recent works—which includes *Spartan* [Set20], *HyperPlonk* [Che+23], and *CCS* [STW23a]—holds the promise of overcoming these limitations. These works develop a toolkit for constructing SNARKs without requiring polynomial multiplication; instead they leverage the classical multivariate sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [Lun+92]. These protocols are constructed from multilinear polynomial IOPs and multilinear polynomial commitment schemes, rather than the univariate polynomial analogues. Combined with the polynomial commitment schemes of [Gol+23] and [DP23], which do not mandate the use of Reed–Solomon codes and do work for general linear codes, the multilinear regime carries the potential for efficient SNARKs over $\mathbb{F}_2$ with no embedding overhead.

While Reed–Solomon codes are far from the only choice available choice, they nonetheless remain attractive. They're efficiently encodable and maximum-distance separable, and, moreover, admit a strengthened proximity-gap result—due to Ben-Sasson, Carmon, Ishai, Kopparty, and Saraf [Ben+23]—which improves upon the best currently-available analogues proven for general linear codes.

We propose two concrete polynomial commitment schemes over $\mathbb{F}_2$, both based on Brakedown. We recall that Brakedown works at a high level by shaping the polynomial coefficients into a two-dimensional matrix, encoding it row-wise, and then randomly sampling and testing columns for proximity to the code and consistency with prover-supplied messages. One option for an $\mathbb{F}_2$-multilinear polynomial commitment is Brakedown, instantiated with a *concatenated code* constructed from a Reed–Solomon outer code and an inner code small enough to be selected by brute force. Targeting prover efficiency and implementation simplicity, we propose a second option that generalizes Brakedown but which permits the use of Reed–Solomon codes alone, using a technique we call *block-level encoding*. The idea is inspired by the concatenated code approach, but simplifies both the proving and verification procedures by omitting the step of applying an inner code. The idea is to pack elements of the encoded message into extension field elements, encode them with a Reed–Solomon code, and then randomly sample and test blocks of contiguous columns from the encoded matrix—which together form the Reed–Solomon symbols from the extension field—rather than sampling and testing individual columns. This may come at the expense of slightly larger proof size and verifier cost than the concatenated code method in certain cases, though we argue the advantages in implementation simplicity make the tradeoff worthwhile.

With our block-level encoding technique, we attain the remarkable property of zero embedding overhead in the commitment phase. That is, the cost of commiting a ν-variate multilinear polynomial $t(X_0, ..., X_{\nu-1})$ over $\mathbb{F}_2$ is nearly equivalent—aside from small data transposes—to that of committing a $\nu - \kappa$-variate polynomial $t'(X_0, \ldots, X_{\nu-\kappa})$ over the extension field $\mathbb{F}_{2^{2^\kappa}}$, which contains the same quantity of information. On the other hand, there is still an additional cost for proving evaluations of t versus t', incurred by both prover and verifier. There is also computational gap between a sumcheck over t and t' stemming from the fact that the sumcheck challenges come from a cryptographically-sized extension field, say $\mathbb{F}_{2^{128}}$. This latter issue is ameliorated by certain optimizations available in the sumcheck proving algorithm that we discuss in Section 4.2.

For this reason, we do not end our investigate of SNARKs over binary fields at $\mathbb{F}_2$. We push this approach further by utilizing a full extension tower over $\mathbb{F}_2$, of the form $\mathbb{F}_2 \subset \mathbb{F}_{2^2} \subset \mathbb{F}_{2^4} \subset \mathbb{F}_{2^8} \subset \cdots \subset \mathbb{F}_{2^{128}}$. In this manner, we introduce additional and novel flexibility at the arithmetization level to use finite fields that are appropriately sized for the data types of the high-level program.

One example of the utility of tower fields at the arithmetization level is for constraint systems that verify the *Grøstl* hash function [Gau+11]. Grøstl is a collision-resistant hash function that has undergone extensive cryptanalysis and was a finalist candidate in the SHA-3 competition. The hash function's design is based on AES and uses the same Rijndael S-box as AES does. Like AES, Grøstl can be viewed as natively defined over $\mathbb{F}_{2^8}$ and has an efficent arithmetization in constraint systems with native $\mathbb{F}_{2^8}$-operations. We believe this makes Grøstl an attractive candidate hash function in the SNARK system presented here. Accordingly, we expect low arithmetization overhead in recursively verifying SNARKs using the techniques presented here when instantiated with Grøstl. We highlight this as a notable benefit over SNARKs over prime fields, which rely on more recent, and less battle-tested, *arithmetization-optimized* hash functions, such as Poseidon [Gra+19], for efficient recursive verification.

We find several further advantages to using binary field towers beyond the arithmetization layer. In Subsection 2.3 below, we resurface an explicit, iterated construction of binary tower fields originally due

to Wiedemann [Wie88]. This tower construction has some remarkable computational properties, which were observed by Fan and Paar [FP97]; namely, multiplication and even inversion in the tower field $\mathbb{F}_{2^k}$ has asymptotic complexity $O(k^{\log_2 3})$, attained using Karatsuba techniques. Multiplication of field elements with elements residing in a subfield has even better computational complexity, which we elaborate on in Section 2.3. Chen et al. [Che+18] exploit that property of tower fields to improve the performance of polynomial multiplication in binary fields. In our performance evaluation in Section 6 below, we discuss the implications of the recently introduced Intel *Galois Field New Instructions* (GFNI) instruction set extension on software implementations targeting capable processors.

Our contributions. We summarize our contributions in this work as follows.

1. **A formal definition of *small-field polynomial commitment schemes*.** While small field technique already appear throughout various prior SNARKs—such as *Plonky2* [Pol22] and *RISC Zero* [BGT23]—the security of these schemes depends on a certain undocumented soundness property, whereby the committed polynomial's coefficients *actually reside* in the required ground field, as opposed to in the extension field from which the polynomial's evaluation query is drawn. (See Definition 3.3.)

2. **A proof that [DP23] achieves a small-field polynomial commitment scheme.** Indeed, we prove that the construction [DP23, Cons. 4.6]—with appropriate minor modifications—actually yields a small-field scheme in the strong sense outlined above, and so provides "better-than-advertised" security. (See Theorem 3.13.)

3. **A generalization of [DP23], which uses *block-level encoding*.** Our generalized construction yields an efficient small-field polynomial commitment scheme, for $\mathbb{F}_2$-polynomials, which uses Reed–Solomon codes, and which imposes zero embedding overhead during the commitment phase. (See Subsection 3.4.)

4. **An adaptation of *PLONKish* to the binary tower setting, and a SNARK for it.** We adapt the *PLONKish* arithmetization relation of HyperPlonk [Che+23, Def. 4.1] to our setting, and introduce several modifications (most importantly, a generalized constraint system, defined over a tower of fields, as opposed to just one). (See Subsection 5.1.)

5. **An adaptation of the *Lasso* lookup argument [STW23b] to the binary tower setting.** Setty, Thaler and Wahby's *Lasso* [STW23b] differs from prior lookup arguments—including that given in HyperPlonk [Che+23, § 3.7]—in that it explicitly exploits the relative cheapness of committing to small-valued elements. While the authors of [STW23b] highlight this benefit only in the setting of elliptic curve-based polynomial commitments, we show how to capture it moreover in our tower setting. (See Subsection 4.4.)

6. **An efficient shift argument for polynomials over the boolean hypercube.** That is, we define an operator, which, on input a multivariate polynomial f, maps f to the multilinear extension of that polynomial which takes the values of f on the hypercube at arbitrarily *rotated* points. This answers an open problem posed in HyperPlonk (see [Che+23, p. 52] of the full version). (See Subsection 4.3.)

7. **A performance evaluation of field arithmetic, the polynomial commitment scheme, and the sumcheck protocol, in the tower setting.** We moreover compare our software implementation to those of other state-of-the-art SNARKs based on prime fields. Our multilinear polynomial commitment scheme can commit to a 2^{28}-coefficient $\mathbb{F}_2$-polynomial about 50-fold faster than plonky2's [Pol22] Goldilocks-based implementation of the FRI-PCS can, and about 150-fold faster than Hyrax [Wah+18]. Even on 32-bit polynomials, our scheme achieves 35% and 15-fold advantages, respectively, over these latter schemes. (See Section 6.)

8. **An arithmetization of the Keccak-f[1600] permutation.** This permutation resides at the core of the Keccak-256 hash function enshrined in the Ethereum protocol, and represents a key a bottleneck facing attempts to prove statements about the Ethereum blockchain. (See Appendix A.)

Acknowledgements. We would like to gratefully acknowledge the contributions of Justin Thaler to this work, which arose throughout the course of many fruitful discussions. This work wouldn't have been possible without the support of our colleagues at Ulvetanna; their perspectives and insights as far as computational efficiency is concerned, as well as the thorough hardware prototypes they built, guided our investigation.

2 Background and Notation

We write $\mathbb{N}$ for the set of nonnegative integers. For sets S and T, we write S^T for the set of maps $T \to S$. Below, we require that all fields be finite. We fix an arbitrary finite field K (though do focus on the case when K is of characteristic 2). For each $\nu \in \mathbb{N}$, we write $\mathcal{B}_\nu$ for the ν-dimensional *boolean hypercube* $\{0,1\}^\nu \subset K^\nu$. We occasionally identify $\mathcal{B}_\nu$ with the integer range $\{0, \dots 2^\nu - 1\}$ lexicographically; that is, we identify each $v = (v_0, \dots, v_{\nu-1})$ in $\mathcal{B}_\nu$ with the integer $\sum_{i=0}^{\nu-1} 2^i \cdot v_i$, and moreover write $\{v\}$ for this latter integer.

2.1 Polynomials

We recall certain basic facts pertaining to multivariate polynomials, referring throughout to Thaler [Tha22, § 3.5]. We recall the ring $K[X_0, \dots, X_{\nu-1}]$ of ν-variate polynomials over K. We write $K[X_0, \dots, X_{\nu-1}]^{\preceq d}$ for the set of ν-variate polynomials over K of *individual* degree at most d in each variable. *Multilinear polynomials* are multivariate polynomials of individual degree at most 1 in each variable (see [Tha22, Def. 3.4]); the set of all such polynomials is $K[X_0, \dots, X_{\nu-1}]^{\preceq 1}$. A degree-$d$ *multivariate extension* of a map $f \in K^{\mathcal{B}_\nu}$ is a polynomial $\widehat{f} \in K[X_0, \dots, X_{\nu-1}]^{\preceq d}$ for which $\widehat{f}(x) = f(x)$ holds for each $x \in \mathcal{B}_\nu$.

Each map $f \in K^{\mathcal{B}_\nu}$ admits a *unique* degree-1 multivariate extension $\widehat{f} \in K[X_0, \dots, X_{\nu-1}]^{\preceq 1}$ (see [Tha22, Fact 3.5]). We thus refer freely to *the* degree-1 multivariate extension of f; we write $\widetilde{f}$ for this polynomial and call it f's *multilinear extension* (MLE). We recall the *equality indicator function* $\mathsf{eq} : \mathcal{B}_\nu \times \mathcal{B}_\nu \to \mathcal{B}_\nu, (x, y) \mapsto x \stackrel{?}{=} y$, as well as its MLE, the *equality indicator polynomial* (see [Tha22, Lem. 3.6]):

$$\widetilde{\mathsf{eq}}(X_0, \dots, X_{\nu-1}, Y_0, \dots, Y_{\nu-1}) = \prod_{i=0}^{\nu-1} X_i \cdot Y_i + (1 - X_i) \cdot (1 - Y_i).$$

For each $f \in K^{\mathcal{B}_\nu}$, we have the following explicit representation of f's multilinear extension $\widetilde{f} \in K[X_0, \dots, X_{\nu-1}]^{\preceq 1}$:

$$\widetilde{f}(X_0, \dots, X_{\nu-1}) = \sum_{v \in \mathcal{B}_\nu} f(v) \cdot \widetilde{\mathsf{eq}}(v_0, \dots, v_{\nu-1}, X_0, \dots, X_{\nu-1}).$$

The proof that $\widetilde{f}$ is f's multilinear extension is straightforward (see [Tha22, Lem. 3.6], for example).

For each fixed $(r_0, \dots, r_{\nu-1}) \in K^\nu$, the vector $(\widetilde{\mathsf{eq}}(v_0, \dots, v_{\nu-1}, r_0, \dots, r_{\nu-1}))_{v \in \mathcal{B}_\nu}$ takes the form

$$\left(\prod_{i=0}^{\nu-1} v_i \cdot r_i + (1 - v_i) \cdot (1 - r_i) \right)_{v \in \mathcal{B}_\nu} = ((1 - r_0) \cdot \dots \cdot (1 - r_{\nu-1}), \dots, r_0 \cdot \dots \cdot r_{\nu-1}).$$

We call this vector the *tensor product expansion* of the point $(r_0, \dots, r_{\nu-1}) \in K^\nu$, and denote it by $\bigotimes_{i=0}^{\nu-1}(1 - r_i, r_i)$. We note the recursive description $\bigotimes_{i=0}^{\nu-1}(1 - r_i, r_i) = (1 - r_0) \cdot \bigotimes_{i=1}^{\nu-1}(1 - r_i, r_i) \parallel r_0 \cdot \bigotimes_{i=1}^{\nu-1}(1 - r_i, r_i)$. This description yields a $\Theta(\nu)$-time algorithm which computes $\bigotimes_{i=0}^{\nu-1}(1 - r_i, r_i)$ (see e.g. [Tha22, Lem. 3.8]).

2.2 Error-Correcting Codes

We adapt the notation of [DP23, § 2]. A *code* of block length n over the alphabet Σ is a subset of Σ^n. In Σ^n, we write d for the Hamming distance between two vectors (i.e., the number of components at which they differ). We again fix a field K. A *linear code* over K is a linear subspace of K^n. An $[n, k, d]$-*code* $C \subset K^n$ is a k-dimensional linear subspace of K^n for which each pair of unequal elements v_0 and v_1 of C satisfies $d(v_0, v_1) \geq d$.

Given a linear code $C \subset K^n$ and an integer $m \geq 1$, we have C's *m-fold interleaved code*, defined as the subset $C^m \subset (K^n)^m \cong (K^m)^n$. We understand this latter set as a length-n block code over the alphabet K^m. In particular, its elements are naturally identified with those matrices in $K^{m \times n}$ each of whose rows is a C-element. We write matrices $(u_i)_{i=0}^{m-1} \in K^{m \times n}$ row-wise. By definition of C^m, two matrices in $K^{m \times n}$ *differ* at a column if they differ at *any* of that column's components. That a matrix $(u_i)_{i=0}^{m-1} \in K^{m \times n}$ is within distance e to the code C^m—in which event we write $d^m\Big((u_i)_{i=0}^{m-1}, C^m\Big) \leq e$—thus entails precisely that there exists a subset $D \coloneqq \Delta^m\Big((u_i)_{i=0}^{m-1}, C^m\Big)$, say, of $\{0, \ldots, n-1\}$, of size at most e, for which, for each $i \in \{0, \ldots, m-1\}$, the row u_i admits a codeword $v_i \in C$ for which $u_i|_{\{0,\ldots,n-1\}\setminus D} = v_i|_{\{0,\ldots,n-1\}\setminus D}$. We emphasize that the subset $D \subset \{0, \ldots, n-1\}$ is *fixed*, and does not vary as the row-index $i \in \{0, \ldots, m-1\}$ varies. In this circumstance, following the terminology of [Ben+23], we say that the vectors $(u_i)_{i=0}^{m-1}$ feature *correlated agreement* outside of the set D, or that they feature *e-correlated agreement*. We note that the condition whereby the vectors $(u_i)_{i=0}^{m-1}$ feature e-correlated agreement with C^m implies *a fortiori* that every element in $(u_i)_{i=0}^{m-1}$'s row-span is itself within distance at most e from C.

We recall Reed–Solomon codes. For K again fixed, $S = \{s_0, \ldots, s_{n-1}\}$ a subset of K, and a message length $k \leq n$, the *Reed–Solomon code* $\mathsf{RS}_{K,S}[n,k]$ is defined as the subset $C \coloneqq \mathsf{RS}_{K,S}[n,k] = \big\{(p(s_0), \ldots, p(s_{n-1})) \,\big|\, p(X) \in K[X]^{<k}\big\}$. In words, $\mathsf{RS}_{K,S}[n,k]$ is the set of n-tuples which arise as the *evaluations*, over the n points of S, of some polynomial $p(X) \in K[X]$ of degree less than K. Here, we identify $K[X]^{<k}$ with K^k using the monomial K-basis $1, X, \ldots, X^{k-1}$ of $K[X]^{<k}$. The code $\mathsf{RS}_{K,S}[n,k]$ is of distance $d = n - k + 1$ (see e.g. Guruswami [Gur06, Def. 2.3]). Lin, Chung, and Han show in recent work [LCH14] that, for K a binary field and $S \subset K$ an appropriately chosen $\mathbb{F}_2$-affine linear subspace, the encoding function of $\mathsf{RS}_{K,S}[n,k]$—or at least of a code isomorphic to it—can be computed in $\Theta(n \cdot \log k)$ time. (The code $C \subset K^n$ of [LCH14] differs from $\mathsf{RS}_{K,S}[n,k]$ by precomposition with a K-isomorphism on K^k, and so inherits $\mathsf{RS}_{K,S}[n,k]$'s properties in full.)

2.3 Binary Towers

In this subsection, we review towers of field extensions. Throughout the rest of this subsection, we restrict to the setting of characteristic 2.

The following explicit construction of a tower over $\mathbb{F}_2$ is due to Wiedemann [Wie88], and appears also in Cohen [Coh92] and Fan and Paar [FP97], for example; we refer the reader to Blake, Gao, Mullin, Vanstone and Yaghoobian [Bla+93, § 3.4] for further historical remarks. We define a sequence of rings inductively, by setting $\mathcal{T}_0 \coloneqq \mathbb{F}_2$, $\mathcal{T}_1 \coloneqq \mathbb{F}_2[X_0]/\big(X_0^2 + X_0 + 1\big)$, and, for each $\iota > 1$, $\mathcal{T}_\iota \coloneqq \mathcal{T}_{\iota-1}[X_{\iota-1}]/\big(X_{\iota-1}^2 + X_{\iota-2} \cdot X_{\iota-1} + 1\big)$. It is shown in [Wie88, Thm. 1] that, for each $\iota > 1$, the polynomial $X_{\iota-1}^2 + X_{\iota-2} \cdot X_{\iota-1} + 1$ is irreducible in $\mathcal{T}_{\iota-1}[X_{\iota-1}]$. We conclude by induction that, for each $\iota \geq 0$, the ring $\mathcal{T}_\iota$ is a *field*, isomorphic precisely to $\mathbb{F}_{2^{2^\iota}}$.

For each $\iota > 0$, we naturally realize $\mathcal{T}_{\iota-1}$ as a subfield of $\mathcal{T}_\iota$, corresponding to (the equivalence classes of) the *constant polynomials*. Applying induction, we obtain a natural tower construction $\mathcal{T}_0 \subset \mathcal{T}_1 \subset \cdots \subset \mathcal{T}_\iota$. Moreover, for each $\iota \geq 0$, we have a straightforward identification of rings:

$$\mathcal{T}_\iota = \mathbb{F}_2[X_0, \ldots, X_{\iota-1}]/\big(X_0^2 + X_0 + 1, \ldots, X_{\iota-1}^2 + X_{\iota-2} \cdot X_{\iota-1} + 1\big).$$

This identification respects the tower structure in the obvious way; indeed, $\mathcal{T}_{\iota-1} \subset \mathcal{T}_\iota$ is precisely the subring consisting of the equivalence classes of those polynomials in which only the variables $X_0, \ldots, X_{\iota-2}$ appear.

It follows—say, from Gröbner basis considerations—that, for each $\iota \geq 0$, each equivalence class in $\mathcal{T}_\iota$ has a *unique* multilinear representative. We conclude that the set of monomials $1, X_0, X_1, X_0 \cdot X_1, \ldots, X_0 \cdots X_{\iota-1}$ gives a basis of $\mathcal{T}_\iota$ as an $\mathbb{F}_2$-vector space; we call this basis the *multilinear basis*. For each $v \in \mathcal{B}_\iota$, with boolean components $(v_0, \ldots, v_{\iota-1})$, say, we write $\beta_v \coloneqq \prod_{i=0}^{\iota-1}(v_i \cdot X_i + (1 - v_i))$; that is, β_v is that basis vector corresponding to the product of precisely those indeterminates among the list $X_0, \ldots, X_{\iota-1}$ indexed by v's components. Slightly abusing notation, we occasionally write $\beta_0, \ldots, \beta_{2^\iota-1}$ for this latter basis; in other words, we define $\beta_{\{v\}} \coloneqq \beta_v$, where we again identify $\{v\} = \sum_{i=0}^{\iota-1} v_i \cdot 2^i$.

More generally, for each pair of integers $\iota \geq 0$ and $\kappa \geq 0$, the set $1, X_\iota, X_{\iota+1}, X_\iota \cdot X_{\iota+1}, \ldots, X_\iota \cdot \cdots \cdot X_{\iota+\kappa-1}$ likewise gives a $\mathcal{T}_\iota$-basis of $\mathcal{T}_{\iota+\kappa}$; we again write $(\beta_v)_{v \in \mathcal{B}_\kappa}$ for this latter basis. That is, for each $v \in \mathcal{B}_\kappa$, we write $\beta_v \coloneqq \prod_{i=0}^{\kappa-1}(v_i \cdot X_{\iota+i} + (1 - v_i))$.

We briefly survey the efficiency of tower-field arithmetic. In practice, we represent all $\mathcal{T}_\iota$-elements in coordinates with respect to the multilinear $\mathbb{F}_2$-basis, which we moreover sort in lexicographic order. In particular, each $\mathcal{T}_\iota$-element α admits a length-2^ι coordinate vector $\alpha = (a_0, \ldots, a_{2^\iota - 1})$, with components in $\mathbb{F}_2$; we note, in light of our lexicographic basis-ordering, that this vector's 0^{th} and 1^{st} *halves* α_0 and α_1 define $\mathcal{T}_{\iota-1}$-elements for which $\alpha = \alpha_1 \cdot X_{\iota-1} + \alpha_0$ in fact holds.

Throughout, addition amounts to bitwise XOR. We multiply $\mathcal{T}_\iota$-elements in the following way. To multiply the elements $\alpha_1 \cdot X_{\iota-1} + \alpha_0$ and $\alpha_1' \cdot X_{\iota-1} + \alpha_0'$ of $\mathcal{T}_\iota$, say, we first use the Karatsuba technique—that is, we use three recursive multiplications in $\mathcal{T}_{\iota-1}$—to obtain the expression $\alpha_1 \cdot \alpha_1' \cdot X_{\iota-1}^2 + (\alpha_0 \cdot \alpha_1' + \alpha_1 \cdot \alpha_0') \cdot X_{\iota-1} + \alpha_0 \cdot \alpha_0'$. We then reduce this latter polynomial by subtracting $\alpha_1 \cdot \alpha_1' \cdot \left(X_{\iota-1}^2 + X_{\iota-2} \cdot X_{\iota-1} + 1\right)$ from it; this step itself entails computing the product $\alpha_1 \cdot \alpha_1' \cdot X_{\iota-2}$ in $\mathcal{T}_{\iota-1}$.

It is shown by Fan and Paar [FP97, § III] that, in the Wiedemann tower, each such "constant multiplication"—that is, each multiplication of a $\mathcal{T}_\iota$-element by the constant $X_{\iota-1}$—can be carried out in *linear* time $\Theta(2^\iota)$. In light of this fact, and using the "master theorem" for recurrence relations (see e.g. Cormen, Leiserson, Rivest, and Stein [Cor+22, Thm. 4.1]), we conclude that this recursive, Karatsuba-based approach features complexity $\Theta\left(2^{\log 3 \cdot \iota}\right)$ (we refer also to [FP97, § IV] for a thorough analysis).

We finally record a further key property, whereby field-elements may be multiplied by *subfield*-elements especially efficiently. In slightly more detail, the complexity of multiplying a $\mathcal{T}_\iota$-element by a $\mathcal{T}_{\iota+\kappa}$-element grows just linearly in the extension degree of $\mathcal{T}_{\iota+\kappa}$ over $\mathcal{T}_\iota$. We express this precisely as follows. For each element $\alpha \in \mathcal{T}_{\iota+\kappa}$, with coordinate representation $(a_v)_{v \in \mathcal{B}_\kappa}$ with respect to the multilinear $\mathcal{T}_\iota$-basis of $\mathcal{T}_{\iota+\kappa}$, say, and each scalar $b \in \mathcal{T}_\iota$, the representation of $b \cdot \alpha$ with respect to this basis is $(b \cdot a_v)_{v \in \mathcal{B}_\kappa}$. We conclude that the multiplication of a $\mathcal{T}_{\iota+\kappa}$-element by a $\mathcal{T}_\iota$-element can be carried out in $2^\kappa \cdot \Theta\left(2^{\log 3 \cdot \iota}\right)$ time. This property—that is, that whereby elements of differently-sized fields can be efficiently multiplied—has been noted by previous authors; we refer for example to Bernstein and Chou [BC14, § 2.4].

Comparison with classical binary fields. We contrast this work's tower-based approach with the classical, univariate treatment of binary fields. Informally, towers feature both *efficient embeddings* and *efficient small-by-large multiplications*; classical binary fields lack both of these properties. We record the details. For $f_\iota(X) \in \mathbb{F}_2[X]$ irreducible of degree 2^ι, the quotient ring $\mathbb{F}_2[X]/(f_\iota(X))$ is isomorphic to $\mathbb{F}_{2^{2^\iota}}$, and admits the $\mathbb{F}_2$-basis $1, X, \ldots, X^{2^{2^\iota}-1}$, which we call the (univariate) *monomial basis*. We again fix ι and κ in $\mathbb{N}$. Clearly, on the level of abstract fields, we have an embedding $\mathbb{F}_{2^{2^\iota}} \hookrightarrow \mathbb{F}_{2^{2^{\iota+\kappa}}}$ (in fact, we have 2^ι choices, by Galois-theoretic considerations). Identifying these objects with $\mathbb{F}_2^{2^\iota}$ and $\mathbb{F}_2^{2^{\iota+\kappa}}$, respectively—by means of their monomial bases—our embedding induces a mapping $\mathbb{F}_2^{2^\iota} \hookrightarrow \mathbb{F}_2^{2^{\iota+\kappa}}$ of $\mathbb{F}_2$-vector spaces. What is the bit-complexity of this mapping? When $\mathbb{F}_{2^{2^\iota}}$ and $\mathbb{F}_{2^{2^{\iota+\kappa}}}$ are constructed as univariate quotients, the answer is, "it's complicated". (Informally, given $a_0 + \cdots + a_{2^{2^\iota}-1} \cdot X^{2^{2^\iota}-1}$, how do we determine the coefficients of its image in $\mathbb{F}_{2^{2^{\iota+\kappa}}}$?) Obviously, using binary matrix multiplication, one can do no worse than $O(2^{2 \cdot \iota + \kappa})$ bit-operations. Given irreducible polynomials $f_\iota(X)$ and $f_{\iota+\kappa}(X)$ sufficiently carefully chosen, one may be able to do better; we refer to Bosma, Cannon and Steel [BCS97] for a thorough treatment of this issue.

In our tower setting, the embedding $\mathcal{T}_\iota \hookrightarrow \mathcal{T}_{\iota+\kappa}$ of fields again induces—via these fields' respective *multilinear* bases—a mapping $\mathbb{F}_2^{2^\iota} \hookrightarrow \mathbb{F}_2^{2^{\iota+\kappa}}$ of $\mathbb{F}_2$-vector spaces. This latter mapping, on the other hand, is free! Indeed, it amounts to a trivial zero-padding operation. In the language of [BCS97], our tower construction yields a *lattice of compatibly embedded fields* (though our "lattice" is in fact totally-ordered, since we treat only power-of-2-degree extensions).

A similar issue affects the multiplication of $\mathbb{F}_{2^{2^{\iota+\kappa}}}$-elements by $\mathbb{F}_{2^{2^\iota}}$-elements. Indeed, to multiply an element $\alpha \in \mathbb{F}_{2^{2^{\iota+\kappa}}}$ by $b \in \mathbb{F}_{2^{2^\iota}}$, say (and in fact, even to give sense to this operation), one could fix a *particular* embedding $\mathbb{F}_{2^{2^\iota}} \hookrightarrow \mathbb{F}_{2^{2^{\iota+\kappa}}}$, and multiply α by b's image under this embedding. The cost of this operation, however, would be—beyond that of embedding b—the same as a that of a standard $\mathbb{F}_{2^{2^{\iota+\kappa}}}$-multiplication; in other words, it would fail to exploit the fact that b comes from the subfield $\mathbb{F}_{2^{2^\iota}} \subset \mathbb{F}_{2^{2^{\iota+\kappa}}}$. Alternatively, we could pick an arbitrary $\mathbb{F}_{2^{2^\iota}}$-basis of $\mathbb{F}_{2^{2^{\iota+\kappa}}}$, express $\alpha = (a_0, \ldots, a_{2^\kappa - 1})$ in coordinates with respect to this basis, multiply α by b componentwise, and finally convert the result back, let's say. This approach, however, would require two conversion operations, which could each cost as many as $\Omega(2^{2 \cdot \kappa})$ (i.e., quadratically many) $\mathbb{F}_{2^{2^\iota}}$-operations in the worst case. The first insight underlying our tower approach, in fact, is that by representing $\mathbb{F}_{2^{2^{\iota+\kappa}}}$ *continually* in coordinates with respect to some $\mathbb{F}_{2^{2^\iota}}$-basis, we may avoid these conversions. This is exactly what we do; more precisely, our multilinear basis serves this purpose

simultaneously for each possible intermediate field, and never requires conversions.

3 Small-Field Polynomial Commitments

In this section, we introduce *small-field polynomial commitment schemes*, and moreover supply several instantiations based on binary tower fields. In Subsection 3.2 below, we define the basic cryptographic abstraction. We then instantiate this abstraction in two different ways. In Subsection 3.3 below, we outline a "simple" instantiation, suitable for polynomials whose coefficient field coincides with the alphabet of an available code. In Subsection 3.4 below, we introduce a further variant, designed to support the commitment of polynomials over fields *even smaller* than the alphabet of the code selected for use. Both schemes follow the Brakedown-inspired scheme of Diamond and Posen [DP23, § 4], with appropriate adaptations.

We are motivated in Subsection 3.4 by the goal of committing to polynomials over *very small fields* (like $\mathbb{F}_2$) while, simultaneously, making use of the Reed–Solomon code over *larger alphabets* (like $\mathbb{F}_{2^{16}}$). In Subsection 3.4, we attain this goal, in a way, no less, which imposes essentially no overhead beyond that inherent to, say, the commitment of an $\mathbb{F}_{2^{16}}$-polynomial of equal size in bits. In other words, we pay only for the size, in bits, of our polynomial at hand, regardless of the size of its field of definition.

3.1 The Extension Code

Before proceeding, we pause to record a certain key coding-theoretic construction, which figures prominently in what follows. Informally, given some fixed code, with symbols in a field, our construction "lifts" the code to one with symbols in a vector space over that field. The resulting object inherits many of the same properties—most essentially, the distance—of the original code.

Definition 3.1. We fix a $[n, k, d]$-code $C \subset K^n$, with generator matrix $M \in K^{n \times k}$, say, as well as a K-vector space V over K. The *extension code* $\widehat{C} \subset V^n$ of C is the image of the map $V^k \to V^n$ which sends $t \mapsto M \cdot t$.

In other words, the code $\widehat{C} \subset V^n$ simply reuses C's generator matrix; we note that the action of a K-matrix on a V-vector is well-defined.

The object $\widehat{C} \subset V^n$ isn't, strictly speaking, a linear code; indeed, its symbols take values in V, which is *not* (in general) a field. On the other hand, $\widehat{C}$ inherits C's distance, as the following theorem shows:

Theorem 3.2. *The extension code $\widehat{C} \subset V^n$ has distance d, in the sense that each pair of unequal elements u_0 and u_1 of $\widehat{C}$ satisfies $d(u_0, u_1) \geq d$.*

Proof. We write η for the dimension of V over K, and fix a K-basis $(\alpha_0, \ldots, \alpha_{\eta-1})$ of V, as well as two unequal messages t_0 and t_1 in V^k. Expressing these messages' components in coordinates with respect to the this basis, we obtain corresponding vectors $t_{0,h}$ and $t_{1,h}$, in K^k, for *each* index $h \in \{0, \ldots, \eta - 1\}$. Our hypothesis $t_0 \neq t_1$ implies that, for at least one index $h^* \in \{0, \ldots, \eta - 1\}$, the slices t_{0,h^*} and t_{1,h^*} are unequal as elements of K^k. Since $\widehat{C}$'s generator matrix consists of K-elements, the encodings $u_0 \coloneqq \mathsf{Enc}(t_0)$ and $u_1 \coloneqq \mathsf{Enc}(t_1)$ of t_0 and t_1 are themselves given, slice-wise, by the respective encodings of the slices $(t_{0,h})_{h=0}^{\eta-1}$ and $(t_{1,h})_{h=0}^{\eta-1}$. We conclude that the *slices* u_{0,h^*} and u_{1,h^*}, viewed as elements of K^n, differ at at least d positions, and thus finally that the elements u_0 and u_1 of V^n also do. We see that the distance of $\widehat{C}$ is at least d. Conversely, we may easily construct unequal codewords in V^n of distance exactly d. Indeed, given unequal messages t_0 and t_1 in K^k whose encodings differ at exactly d positions, we embed both t_0 and t_1 componentwise into V along the basis vector α_0. We see that the resulting messages' encodings u_0 and u_1 in V^n differ at exactly d positions; indeed, their discrepancies all arise from their respective 0^{th}-indexed slices, since these codewords' positive-indexed slices are all identically zero. This completes the proof. □

As V isn't necessarily itself a field, $\widehat{C}$'s "dimension" k over V is of course not well-defined in general; we note, however, that $\widehat{C}$ *is* k-dimensional over V whenever $V \,/\, K$ is a degree-η field extension.

3.2 Definition of Small-Field Polynomial Commitment Schemes

We now define small-field polynomial commitment schemes, adapting [DP23, Defs. 4.1–4.3], which themselves closely follow Setty [Set20, § 2.4]. Our adaptation requires that each multilinear polynomial $t(X_0, \ldots, X_{\ell-1})$ at hand reside in $K[X_0, \ldots, X_{\ell-1}]$, for a user-specified field K, allowed to be arbitrarily small. On the other hand, we allow each evaluation query point $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, as well as each claimed evaluation result $s \in L$, to be defined over an extension $L \,/\, K$ of K. Thus, in short, Definition 3.3 below furnishes a commitment scheme for polynomials over *small* fields, which can nonetheless be queried at points over *large* extension fields of the polynomial's field of definition.

Definition 3.3. A *small-field multilinear polynomial commitment scheme* is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Prove}, \mathsf{Verify})$, with the following syntax:

- $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, K)$. On input the security parameter λ, a size parameter ℓ, and a field K, $\Pi.\mathsf{Setup}$ samples params, which includes (possibly among other things) a field extension $L \,/\, K$.
- $(c, u) \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, $\Pi.\mathsf{Commit}$ returns a commitment c to t, together with an *opening hint* u.
- $b \leftarrow \Pi.\mathsf{Open}(\mathsf{params}, c; t, u)$. On input a commitment c, a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, and an opening hint u, $\Pi.\mathsf{Open}$ verifies the claimed decommitment t of c, using u.
- $\pi \leftarrow \Pi.\mathsf{Prove}(\mathsf{params}, c, s, (r_0, \ldots, r_{\ell-1}); t, u)$. On input a commitment c, a purported evaluation $s \in L$, an evaluation point $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, and an opening hint u, $\Pi.\mathsf{Prove}$ generates an evaluation proof π.
- $b \leftarrow \Pi.\mathsf{Verify}(\mathsf{params}, c, s, (r_0, \ldots, r_{\ell-1}), \pi)$. On input a commitment c, a purported evaluation s, an evaluation point $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and a proof π, $\Pi.\mathsf{Verify}$ outputs a success bit $b \in \{0, 1\}$.

We note that, for Π to be efficiently computable, it's necessary that $\ell = O(\log \lambda)$, as well as that the sizes $\log(|K|)$ and $\log(|L|)$ grow at most polynomially in λ. We assume as much throughout what follows.

We define the security properties *binding* and *extractability*, for small-field multilinear polynomial commitment schemes, following [DP23, Def. 4.2] and [DP23, Def. 4.3], respectively, with various minor modifications.

Definition 3.4. For each small-field multilinear polynomial commitment scheme Π, size parameter ℓ, input field K, and PPT adversary $\mathcal{A}$, we define the *binding experiment* $\mathsf{Binding}_{\mathcal{A}}^{\Pi,\ell,K}(\lambda)$ as follows:

1. The experimenter samples $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, K)$, and gives params to $\mathcal{A}$.
2. The adversary outputs $(c, t_0, t_1, u_0, u_1) \leftarrow \mathcal{A}(\mathsf{params})$, where c is a commitment, $t_0(X_0, \ldots, X_{\ell-1})$ and $t_1(X_0, \ldots, X_{\ell-1})$ are multilinear polynomials in $K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, and u_0 and u_1 are opening hints.
3. The output of the experiment is defined to be 1 if $t_0 \neq t_1$, $\Pi.\mathsf{Open}(\mathsf{params}, c; t_0, u_0)$, and $\Pi.\mathsf{Open}(\mathsf{params}, c; t_1, u_1)$ all hold; otherwise, it is defined to be 0.

The small-field multilinear polynomial commitment scheme Π is *binding* if, for each PPT adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\lambda)$ for which, for each security parameter $\lambda \in \mathbb{N}$ and each choice of ℓ and K, it holds that $\Pr\left[\mathsf{Binding}_{\mathcal{A}}^{\Pi,\ell,K}(\lambda)\right] \leq \mathsf{negl}(\lambda)$.

Definition 3.5. For each small-field multilinear polynomial commitment scheme Π, security parameter λ, values ℓ and K, PPT query sampler $\mathcal{Q}$, PPT adversary $\mathcal{A}$, expected PPT emulator $\mathcal{E}$, and PPT distinguisher $\mathcal{D}$, we define two random variables $\mathsf{Real}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}^{\Pi,\ell,K}(\lambda)$ and $\mathsf{Emul}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}^{\Pi,\ell,K}(\lambda)$, each valued in $\{0, 1\}$, as follows:

1. The experimenter samples $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, K)$, and gives params to $\mathcal{A}$, $\mathcal{Q}$ and $\mathcal{E}$.
2. The adversary outputs a commitment $c \leftarrow \mathcal{A}(\mathsf{params})$.
3. The query sampler outputs $(r_0, \ldots, r_{\ell-1}) \leftarrow \mathcal{Q}(\mathsf{params})$.
4. The experimenter proceeds in one of two separate ways:

- $\mathsf{Real}^{\Pi,\ell,K}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}(\lambda)$: Run $(s,\pi) \leftarrow \mathcal{A}(r_0,\ldots,r_{\ell-1})$. Output the single bit $\mathcal{D}(c,s,\pi)$.
- $\mathsf{Emul}^{\Pi,\ell,K}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}(\lambda)$: Run $(s,\pi;t,u) \leftarrow \mathcal{E}^{\mathcal{A}}(r_0,\ldots,r_{\ell-1})$. Output the single bit $\mathcal{D}(c,s,\pi) \wedge (\Pi.\mathsf{Verify}(\mathsf{params},c,s,(r_0,\ldots,r_{\ell-1}),\pi) \implies (\Pi.\mathsf{Open}(\mathsf{params},c;t,u) \wedge t(r_0,\ldots,r_{\ell-1}) = s))$.

The small-field multilinear polynomial commitment scheme Π is *extractable* with respect to the query sampler $\mathcal{Q}$ if, for each PPT adversary $\mathcal{A}$, there is an expected PPT emulator $\mathcal{E}$ such that, for each PPT distinguisher $\mathcal{D}$, the distributions $\left\{\mathsf{Real}^{\Pi,\ell,K}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}(\lambda)\right\}_{(\ell,K),\lambda\in\mathbb{N}}$ and $\left\{\mathsf{Emul}^{\Pi,\ell,K}_{\mathcal{Q},\mathcal{A},\mathcal{E},\mathcal{D}}(\lambda)\right\}_{(\ell,K),\lambda\in\mathbb{N}}$ are statistically close.

We note that, critically, the polynomial $t(X_0,\ldots,X_{\ell-1})$ extracted by $\mathcal{E}$ must reside in $K[X_0,\ldots,X_{\ell-1}]$, by definition of $\Pi.\mathsf{Open}$.

The following definition is analogous to [DP23, Def. 4.4]; we refer to [DP23, Rem. 4.5] for further discussion of this definition.

Definition 3.6. The query sampler $\mathcal{Q}$ is *admissible* if, for each λ, ℓ and K, and each parameter set $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda,\ell,K)$, containing $L \,/\, K$, say, the evaluation point $(r_0,\ldots,r_{\ell-1}) \leftarrow \mathcal{Q}(\mathsf{params})$ is uniform over L^ℓ.

3.3 Basic Small-Field Construction

We now give our simple small-field construction. This construction generalizes [DP23, Cons. 4.6], so as to make that scheme instantiate the small-field abstraction of Definition 3.3. In our generalization, we allow the polynomial's coefficient field and the code's alphabet to be sub-cryptographically sized, though we require that these fields be equal to *each other* (compare Subsection 3.4 below). We obtain security by the means of a cryptographically sized field extension. Our construction closely follows [DP23, Cons. 4.6], making only minor modifications throughout.

Construction 3.7 (Simple small-field polynomial commitment scheme)**.**
We define $\Pi = (\mathsf{Setup},\mathsf{Commit},\mathsf{Open},\mathsf{Prove},\mathsf{Verify})$ as follows.

- $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda,\ell,K)$. On input 1^λ, ℓ, and K, choose integers ℓ_0 and ℓ_1 for which $\ell_0+\ell_1 = \ell$, and write $m_0 \coloneqq 2^{\ell_0}$ and $m_1 \coloneqq 2^{\ell_1}$. Return an extension field $L \,/\, K$ for which $|L| \geq 2^{\omega(\log\lambda)}$, an $[n,m_1,d]$-code $C \subset K^n$ for which $n = 2^{O(\ell)}$ and $d = \Omega(n)$, and a repetition parameter $\rho = \Theta(\lambda)$.
- $(c,u) \leftarrow \Pi.\mathsf{Commit}(\mathsf{params},t)$. On input $t(X_0,\ldots,X_{\ell-1}) \in K[X_0,\ldots,X_{\ell-1}]^{\preceq 1}$, express $t = (t_0,\ldots,t_{2^\ell-1})$ in coordinates with respect to the Lagrange basis on $\{0,1\}^\ell$, collate the resulting vector into an $m_0 \times m_1$ matrix $(t_i)_{i=0}^{m_0-1}$, and encode $(t_i)_{i=0}^{m_0-1}$ row-wise, so obtaining a further matrix $(u_i)_{i=0}^{m_0-1}$. Output a Merkle commitment c to $(u_i)_{i=0}^{m_0-1}$ and the opening hint $u \coloneqq (u_i)_{i=0}^{m_0-1}$.
- $b \leftarrow \Pi.\mathsf{Open}(\mathsf{params},c;t,u)$. On input $t(X_0,\ldots,X_{\ell-1}) \in K[X_0,\ldots,X_{\ell-1}]^{\preceq 1}$, opening hint $(u_i)_{i=0}^{m_0-1}$, and commitment c, verify that $(u_i)_{i=0}^{m_0-1}$ Merkle-hashes to c. Collate t into a matrix $(t_i)_{i=0}^{m_0-1}$, encode the resulting matrix row-wise, and verify that $d^{m_0}\left((\mathsf{Enc}(t_i))_{i=0}^{m_0-1},(u_i)_{i=0}^{m_0-1}\right) \overset{?}{<} \frac{d}{3}$.

We define $\Pi.\mathsf{Prove}$ and $\Pi.\mathsf{Verify}$ by applying the Fiat–Shamir heuristic to the following interactive protocol, where $\mathcal{P}$ has $t(X_0,\ldots,X_{\ell-1})$ and $(u_i)_{i=0}^{m_0-1}$, and $\mathcal{P}$ and $\mathcal{V}$ have c, $s \in L$, and $(r_0,\ldots,r_{\ell-1}) \in L^\ell$.

- $\mathcal{P}$ sends $\mathcal{V}$ the matrix–vector product $t' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i,r_i)\cdot(t_i)_{i=0}^{m_0-1}$ in the clear.
- For each $i \in \{0,\ldots,\rho-1\}$, $\mathcal{V}$ samples $j_i \leftarrow \{0,\ldots,n-1\}$. $\mathcal{V}$ sends $\mathcal{P}$ the set $J \coloneqq \{j_0,\ldots,j_{\rho-1}\}$.
- $\mathcal{P}$ sends $\mathcal{V}$ the columns $\left\{(u_{i,j})_{i=0}^{m_0-1}\right\}_{j\in J}$, each featuring an accompanying Merkle path against c.
- $\mathcal{V}$ computes $\widehat{\mathsf{Enc}}(t')$. For each $j \in J$, $\mathcal{V}$ verifies the Merkle path attesting to $(u_{i,j})_{i=0}^{m_0-1}$, and moreover checks $\bigotimes_{i=\ell_1}^{\ell-1}(1-r_i,r_i)\cdot(u_{i,j})_{i=0}^{m_0-1} \overset{?}{=} \widehat{\mathsf{Enc}}(t')_j$. Finally, $\mathcal{V}$ requires $s \overset{?}{=} t'\cdot\bigotimes_{i=0}^{\ell_1-1}(1-r_i,r_i)$.

In the very last step, we write $\widehat{\mathsf{Enc}}$ for the encoding function of the natural extension code $\widehat{C} \subset L^n$ (see Section 2).

Though Construction 3.7 is both binding and extractable, we refrain from proving as much; instead, we defer our proofs of security to Subsection 3.4 below. The proof of security of Construction 3.7 above can be obtained by specializing that subsection's scheme's proof to the case $\kappa \coloneqq 0$.

3.4 Block-Level Encoding

In this section, we describe a further variant of the polynomial commitment scheme given in Subsection 3.3 above, suitable for polynomials over fields *smaller* than the alphabet of the linear block code selected for use. We refer throughout to Guruswami [Gur06].

The simple scheme given given in Construction 3.7 mandates the internal use of a code $C \subset K^n$ over *the same* field K as that passed into $\Pi.\mathsf{Setup}(1^\lambda, \ell, K)$. In other words, it requires that $\Pi.\mathsf{Setup}$ return a code V whose alphabet K *is identical to* the coefficient field K of the commitment scheme's message space $K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$. This restriction presents no obstacle in theory, since constant-distance, constant-rate families of codes exist even over arbitrarily small, fixed-size fields (this fact follows from the Gilbert–Varshamov bound; see [Gur06, § 2.1]). Moreover, concretely good codes over small alphabets may be obtained constructively using *concatenated codes* (see [Gur06, § 2.3]).

On the other hand, this restriction does preclude the use of "plain" Reed–Solomon codes in Construction 3.7, at least for certain ℓ and K; indeed, a Reed–Solomon $[n, k, d]$-code can exist only when $|K| \geq n$. Reed–Solomon codes remain attractive, however, for various practical reasons. They attain the Singleton bound, and so maximally favorably negotiate the tension between distance and rate. Separately, they admit efficient encoding algorithms. Specifically, each code $\mathsf{RS}_{K,S}[n, k]$'s encoding function $K^k \to K^n$ may be computed in $\Theta(n \cdot \log k)$ time, at least for certain alphabets K and evaluation sets $S \subset K$. Crucially, we may number among these favorable alphabets the fields K of characteristic 2, due to relatively recent work of Lin, Chung and Han [LCH14] (in that work, the evaluation set $S \subset K$ is an $\mathbb{F}_2$-affine linear subspace of K). We specialize from this point onwards to the binary tower setting (see Subsection 2.3).

Concatenated codes. In order to develop certain intuitions essential to our packing scheme, we first examine the effect of instantiating Construction 3.7, as written, on a concatenated code. A *concatenated code* $C \subset \mathcal{T}_\iota^n$ is defined in terms of an *outer* $[n_{\text{out}}, k_{\text{out}}, d_{\text{out}}]$-code $C_{\text{out}} \subset \mathcal{T}_{\iota+\kappa}^{n_{\text{out}}}$, say, where $\kappa \in \mathbb{N}$, and an *inner* $[n_{\text{in}}, k_{\text{in}}, d_{\text{in}}]$-code $C_{\text{in}} \subset \mathcal{T}_\iota^{n_{\text{in}}}$, where here we require $k_{\text{in}} = 2^\kappa$. The resulting concatenated code is an $[n, k, d]$-code over $C \subset \mathcal{T}_\iota^n$, where here we write $n \coloneqq n_{\text{out}} \cdot n_{\text{in}}$, $k \coloneqq k_{\text{out}} \cdot k_{\text{in}}$, and $d \coloneqq d_{\text{out}} \cdot d_{\text{in}}$ (we refer to [Gur06, § 2.3] for further details). For example, upon concatenating the *outer* $[2^{15}, 2^{14}, 2^{14}+1]$-code $\mathsf{RS}_{\mathcal{T}_4}[2^{15}, 2^{14}]$ over $\mathcal{T}_4$ with the *inner* $[2^5, 2^4, 2^3]$-code $\mathsf{RM}_{\mathcal{T}_0}[2, 5]$ over $\mathcal{T}_0$, we would obtain a $[2^{20}, 2^{18}, 2^{17}+2^3]$-code over $\mathcal{T}_0$ (here, $\mathsf{RM}_{\mathcal{T}_0}[2, 5]$ denotes a *Reed–Muller code*).

The concatenated code construction requires that the inner code's message space coincide with the outer code's alphabet. On the other hand, above, we leverage the natural identification $\mathcal{T}_\iota^{2^\kappa} \cong \mathcal{T}_{\iota+\kappa}$ of $\mathcal{T}_\iota$-vector spaces (see Subsection 2.3). In different words, we may interpret *blocks* of adjacent tower-field elements as *elements* of a larger tower field. That is, given integers ι and κ in $\mathbb{N}$, we may "pack" each block of 2^κ $\mathcal{T}_\iota$-elements into a single $\mathcal{T}_{\iota+\kappa}$-element.

We recall that the concatenated code $C \subset \mathcal{T}_\iota^n$'s encoding procedure entails the following steps:

- *pack* the initial message in $\mathcal{T}_\iota^k$ into a vector in $\mathcal{T}_{\iota+\kappa}^{k_{\text{out}}}$,
- *encode* the resulting vector using the outer code C_{out}'s encoder, so obtaining a codeword in $\mathcal{T}_{\iota+\kappa}^{n_{\text{out}}}$,
- *unpack* each individual symbol of the resulting codeword into a message, in $\mathcal{T}_\iota^{k_{\text{in}}}$, and finally
- *encode* each such message, using the inner code C_{in}, into a codeword in $\mathcal{T}_\iota^{n_{\text{in}}}$, and concatenate them.

Construction 3.7, upon being instantiated with a concatenated code $C \subset \mathcal{T}_\iota^n$, and with the extension field $\mathcal{T}_\tau \,/\, \mathcal{T}_\iota$, say, would stipulate that the verifier perform the encoding operation attached to the corresponding extension code $\widehat{C} \subset \mathcal{T}_\tau^n$. This code is clearly well-defined (we recall Subsection 3.1); on the other hand, its encoding procedure is significantly more complicated than C's is. We have already discussed above how one

might pack blocks of 2^κ $\mathcal{T}_\iota$-elements into $\mathcal{T}_{\iota+\kappa}$-elements; in contrast, the corresponding packing operation on blocks of 2^κ $\mathcal{T}_\tau$-elements is more subtle.

The subtlety arises from the interplay of the three fields $\mathcal{T}_\iota$, $\mathcal{T}_{\iota+\kappa}$, and $\mathcal{T}_\tau$. In a sense, the packing operation operates over a different "dimension" than does the field extension $\mathcal{T}_\tau \,/\, \mathcal{T}_\iota$; that is, it acts *across* $\mathcal{T}_\iota$-elements, instead of extending them. For the sake of intuition, we suggest imagining the parameterization $\iota \coloneqq 0$, $\kappa \coloneqq 4$, and $\tau \coloneqq 7$, as well as the concatenated code sketched above, throughout what follows.

Sketch of our approach. We explain the encoding procedure of a concatenated code's *extended code* in the following way. We define a certain data structure, which "packs" a number of $\mathcal{T}_\iota$-elements into a rectangular array. This data structure is depicted in Figure 1 below.

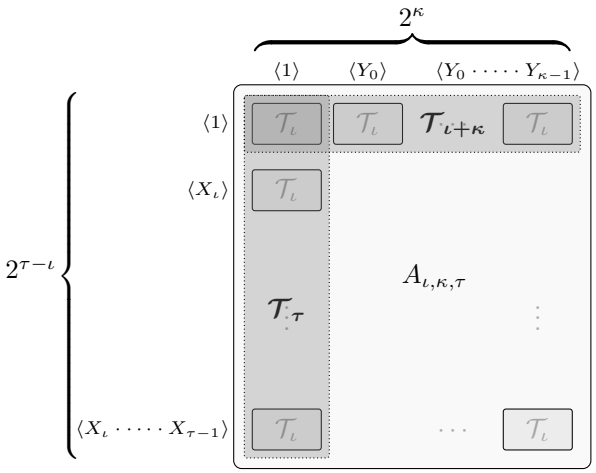

Figure 1: A depiction of our "tower algebra" data structure.

Figure 1 depicts an array of $2^{\tau-\iota}$ rows and 2^κ columns (where, again, each cell is a $\mathcal{T}_\iota$-element). The *extended* concatenated code's encoding procedure stipulates that we first pack each block of 2^κ consecutive $\mathcal{T}_\tau$-elements into exactly such an array, that we then apply the outer code—whose alphabet is $\mathcal{T}_{\iota+\kappa}$—row-wise, and that, finally, we then apply the inner code, again row-wise, to each resulting array.

In pursuit of an even simpler construction, we simply omit the inner code—or equivalently, we use the *identity* inner code—and use the Reed–Solomon outer code. Were we to apply Construction 3.7 naïvely to this latter code, we would encounter a relatively inefficient verifier; indeed, this particular concatenated code features a relative distance k_{in}-fold *worse* than the simple Reed–Solomon code's. Instead, though we *do* omit the inner code, we compensate by decreeing that the verifier test entire packed *blocks* of the prover's committed matrix, instead of testing individual columns. Crucially, we no longer view our code's encoding procedure as a $\mathcal{T}_\iota$-linear one; rather, our code's "symbols" are, now, packed vectors of $\mathcal{T}_\iota$-elements. To resuscitate our security analysis—which itself depends fundamentally on the *proximity gap* phenomenon exhibited by error-correcting codes—we must investigate in what sense the rows of our committed matrix are, in fact, codewords of *some different code*. As it turns out, the array of Figure 1 can be endowed with a certain algebraic structure—which we describe thoroughly throughout what follows—which, serving in the capacity of the alphabet of a certain extension code, makes possible our adaptation of [DP23]'s security analysis.

Interestingly, our block-level testing scheme achieves a proof size profile close to that which Construction 3.7 can attain even on a nontrivial concatenated code. (Comparing these approaches is of course difficult—and in the limit, impossible—since the latter approach mandates the selection of an ad-hoc inner code for each statement size. We opt simply to select the highest-distance known inner code for each statement size we benchmark, and to avoid asymptotic comparisons.) At the same time, it's significantly simpler, and more efficient, for the prover. These observations affirm our contention that, taken in full, this section's construction represents a compelling tradeoff. Indeed, we seek first of all to deliver a highly efficient prover; on the other hand, this measure imposes only a limited cost on the verifier. We thoroughly benchmark these

schemes' proof sizes in Table 1 below.

The tower algebra. We discuss, first informally and then precisely, two *distinct* multiplication operations, defined on $2^{\tau-\iota} \times 2^{\kappa}$-sized arrays over $\mathcal{T}_\iota$ like that in Figure 1. To multiply the entire array by a $\mathcal{T}_\iota$-element, we may simply proceed cell-wise. We may moreover coherently define multiplication operations involving elements of certain larger fields. For example, to multiply the entire array by a $\mathcal{T}_\tau$-element $r \in \mathcal{T}_\tau$, we may interpret the array's *columns* as $\mathcal{T}_\tau$-elements—respectively called $\gamma_0, \ldots, \gamma_{2^\kappa-1}$, say—and overwrite $\gamma_i \mathrel{\times}= r$ for each column index $i \in \{0, \ldots, 2^\kappa - 1\}$. On the other hand, we may moreover interpret each of the array's *rows* as a $\mathcal{T}_{\iota+\kappa}$-element. We thus further define multiplication by $\mathcal{T}_{\iota+\kappa}$-elements; that is, to multiply the entire matrix by an element $s \in \mathcal{T}_{\iota+\kappa}$, we interpret the array's *rows* as $\mathcal{T}_{\iota+\kappa}$-elements—called $(\zeta_0, \ldots, \zeta_{2^{\tau-\iota}-1})$, say—and overwrite $\zeta_i \mathrel{\times}= s$ for each $i \in \{0, \ldots, 2^{\tau-\iota} - 1\}$.

This "dual view" of the array—that is, *either* as an array of 2^κ $\mathcal{T}_\tau$-elements, with a $\mathcal{T}_\tau$-vector space structure *or* as an array of $2^{\tau-\iota}$ $\mathcal{T}_{\iota+\kappa}$-elements, with a $\mathcal{T}_{\iota+\kappa}$-vector space structure—will prove crucial throughout our exposition of the packing scheme. Essentially, our packing scheme entails packing $\mathcal{T}_\iota$-elements "horizontally", into $\mathcal{T}_{\iota+\kappa}$-elements, in order to encode them; in order to obtain cryptographic security, on the other hand, we moreover extend them "vertically", into $\mathcal{T}_\tau$-elements.

To make precise our packing scheme, we introduce a certain polynomial ring.

Definition 3.8. For parameters ι, κ, and τ in $\mathbb{N}$, where $\tau \geq \iota$, we define the *tower algebra* $A_{\iota,\kappa,\tau}$ as:

$$A_{\iota,\kappa,\tau} \coloneqq \mathcal{T}_\tau[Y_0, \ldots, Y_{\kappa-1}] / \big(Y_0^2 + X_{\iota-1} \cdot Y_0 + 1, Y_1^2 + Y_0 \cdot Y_1 + 1, \ldots, Y_{\kappa-1}^2 + Y_{\kappa-2} \cdot Y_{\kappa-1} + 1\big),$$

where we understand $X_{\iota-1}$ as a $\mathcal{T}_\tau$-element (and slightly abuse notation by letting $X_{-1} \coloneqq 1$ if necessary).

We note that $A_{\iota,\kappa,\tau}$ admits a natural description as a 2^κ-dimensional vector space over $\mathcal{T}_\tau$, via the basis $1, Y_0, Y_1, Y_0 \cdot Y_1, \ldots, Y_0 \cdot \cdots Y_{\kappa-1}$ (cf. the $\mathcal{T}_\iota$-basis $(\beta_v)_{v \in \mathcal{B}_\kappa}$ of $\mathcal{T}_{\iota+\kappa}$ from Subsection 2.3). This basis gives rise to an isomorphism $a_{\iota,\kappa,\tau} : \mathcal{T}_\tau^{2^\kappa} \to A_{\iota,\kappa,\tau}$ of $\mathcal{T}_\tau$-vector spaces, which we call the *natural embedding*. The restriction of this embedding to its domain's 0^{th} factor $\mathcal{T}_\tau \subset \mathcal{T}_\tau^{2^\kappa}$ maps $\mathcal{T}_\tau$ isomorphically to the subring $A_{\iota,0,\tau} \subset A_{\iota,\kappa,\tau}$ consisting of the *constant* polynomials in the indeterminates $Y_0, \ldots, Y_{\kappa-1}$.

We understand the tower algebra in the following way. The formal variables $Y_0, \ldots, Y_{\kappa-1}$ define "synthetic analogues" of the variables $X_\iota, \ldots, X_{\iota+\kappa-1}$, which would—upon being adjoined to $\mathcal{T}_\iota$—yield the field extension $\mathcal{T}_\iota \subset \mathcal{T}_{\iota+\kappa}$; moreover, these synthetic variables are designed to behave like their genuine analogues (by means of the relations defining $A_{\iota,\kappa,\tau}$). In fact, this design gives rise to a certain key property of the tower algebra, whereby the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$ consisting of those polynomials whose coefficients reside exclusively in the *subfield* $\mathcal{T}_\iota \subset \mathcal{T}_\tau$ is precisely $\mathcal{T}_{\iota+\kappa}$. We restate this essential property as follows:

Theorem 3.9. *The restriction $a_{\iota,\kappa,\tau}|_{\mathcal{T}_\iota^{2^\kappa}} : \mathcal{T}_\iota^{2^\kappa} \to A_{\iota,\kappa,\tau}$ of the natural embedding to the subset $\mathcal{T}_\iota^{2^\kappa} \subset \mathcal{T}_\tau^{2^\kappa}$ is an injection of $\mathcal{T}_\iota$-vector spaces, whose image, the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$, is isomorphic as a ring to $\mathcal{T}_{\iota+\kappa}$.*

Proof. Indeed, the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$ is easily seen to be identical to $\mathcal{T}_{\iota+\kappa}$, albeit with the variables $X_\iota, \ldots, X_{\iota+\kappa-1}$ respectively renamed to $Y_0, \ldots, Y_{\kappa-1}$. □

We implicitly, and unambiguously, understand $A_{\iota,\kappa,\tau}$ as a $\mathcal{T}_\iota$-vector space in the first part of the statement of Theorem 3.9; indeed, this action arises from the subring $\mathcal{T}_\iota \subset A_{\iota,\kappa,\tau}$ consisting of those *constant* polynomials in the indeterminates $Y_0, \ldots, Y_{\kappa-1}$ whose constant—i.e., only—term resides in the subfield $\mathcal{T}_\iota \subset \mathcal{T}_\tau$.

On the other hand, Theorem 3.9 shows that, over certain fields strictly larger than $\mathcal{T}_\iota$, the ring $A_{\iota,\kappa,\tau}$ admits multiple—and incompatible—vector space structures, a fact which we now take pains to explain carefully. Of course, $A_{\iota,\kappa,\tau}$ has an obvious $\mathcal{T}_\tau$-action—already noted above—coming from the subring $\mathcal{T}_\tau \cong A_{\iota,0,\tau} \subset A_{\iota,\kappa,\tau}$ consisting of *constant* polynomials in the indeterminates $Y_0, \ldots, Y_{\kappa-1}$. To distinguish this subring from Theorem 3.9's, we call it the *constant subring* throughout what follows. On the other hand, Theorem 3.9 further realizes the field $\mathcal{T}_{\iota+\kappa} \cong A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$ as the subring consisting of those *arbitrary-degree* polynomials in the indeterminates $Y_0, \ldots, Y_{\kappa-1}$ whose coefficients, on the other hand, reside in $\mathcal{T}_\iota \subset \mathcal{T}_\tau$. We refer to Theorem 3.9's subring, throughout what follows, as the *synthetic subring*. We take care below, whenever we understand $A_{\iota,\kappa,\tau}$ as an algebra or as a vector space, to carefully specify the particular field, and the particular vector space structure, that we intend. As a rule, whenever we speak of $A_{\iota,\kappa,\tau}$ as a *$\mathcal{T}_\tau$-algebra*, we understand the *constant subring*; whenever we speak of it as a $\mathcal{T}_{\iota+\kappa}$-algebra, we understand

the *synthetic subring*. (The constant and synthetic subrings appear in Figure 1 as the vertical and horizontal shaded regions, respectively.)

We write $(\beta_v)_{v \in \mathcal{B}_{\tau-\iota}}$ for the multilinear $\mathcal{T}_\iota$-basis of $\mathcal{T}_\tau$ (i.e., for the basis $1, X_\iota, X_{\iota+1}, X_\iota \cdot X_{\iota+1}, \ldots, X_\iota \cdot \cdots \cdot X_{\tau-1}$; we refer again to Subsection 2.3). We finally note that $(\beta_v)_{v \in \mathcal{B}_{\tau-\iota}}$ *simultaneously* yields a $\mathcal{T}_{\iota+\kappa}$-basis of $A_{\iota,\kappa,\tau}$—where we of course endow the latter ring with the synthetic $\mathcal{T}_{\iota+\kappa}$-vector space structure—provided that we identify each $\beta_v \in \mathcal{T}_\tau$ with the *constant* polynomial β_v in the indeterminates $Y_0, \ldots, Y_{\kappa-1}$.

For each ι, κ, and τ in $\mathbb{N}$, each tower algebra $A_{\iota,\kappa,\tau}$, and each standard $[n,k,d]$-code $C \subset \mathcal{T}_{\iota+\kappa}^n$ over the alphabet $\mathcal{T}_{\iota+\kappa}$, we recall the *extension code* construction of Definition 3.1. That is, in view of the *synthetic* $\mathcal{T}_{\iota+\kappa}$-vector space structure—i.e., that of Theorem 3.9—on $A_{\iota,\kappa,\tau}$, C's generator matrix induces a map $\widehat{\mathsf{Enc}} : A_{\iota,\kappa,\tau}^k \to A_{\iota,\kappa,\tau}^n$ of $\mathcal{T}_{\iota+\kappa}$-vector spaces; we write $\widehat{C} \subset A_{\iota,\kappa,\tau}^n$ for this map's image. (Equivalently, we may simply embed C's generator matrix entry-wise along the subring $\mathcal{T}_{\iota+\kappa} \subset A_{\iota,\kappa,\tau}$ of Theorem 3.9, and view it as an $A_{\iota,\kappa,\tau}$-matrix.) It is shown in Theorem 3.2 above that $\widehat{C} \subset A_{\iota,\kappa,\tau}^n$ has distance d. Importantly, we note that $\widehat{\mathsf{Enc}}$ is *simultaneously* $\mathcal{T}_\tau$-linear, where now we understand both $A_{\iota,\kappa,\tau}^k$ and $A_{\iota,\kappa,\tau}^n$ as $\mathcal{T}_\tau$-vector spaces (via the *constant embedding* on each factor). To show this, we observe first that $\widehat{\mathsf{Enc}}$ amounts to a matrix–vector product over the ring $A_{\iota,\kappa,\tau}$ (where we again synthetically embed $\mathcal{T}_{\iota+\kappa} \subset A_{\iota,\kappa,\tau}$). On the other hand, any $\mathcal{T}_\tau$-linear combination of $A_{\iota,\kappa,\tau}^k$-vectors can itself be expressed as a scalar–vector combination over the ring $A_{\iota,\kappa,\tau}$ (where we now embed $\mathcal{T}_\tau \subset A_{\iota,\kappa,\tau}$). The $\mathcal{T}_\tau$-linearity of $\widehat{\mathsf{Enc}}$ thus amounts to a distributive matrix identity over $A_{\iota,\kappa,\tau}$; on the other hand, matrix multiplication is certainly distributive for arbitrary commutative rings.

We finally prepare the ground for our packing construction by recording a *proximity gap* result—that is, an analogue of [DP23, Thm. 3.1]—for tower algebras. In the below theorem, we give meaning to the row-combination $\bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i) \cdot (u_i)_{i=0}^{m_0-1}$ by means of the *constant* $\mathcal{T}_\tau$-vector space structure on $A_{\iota,\kappa,\tau}$. The key difference between [DP23, Thm. 3.1] and Theorem 3.10 below, then, is that the code at hand has symbols in the $\mathcal{T}_\tau$-*vector space* $A_{\iota,\kappa,\tau}$, though the combination vector $\bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i)$ nonetheless still has entries in the ground field $\mathcal{T}_\tau$.

Theorem 3.10 (Diamond–Posen [DP23, Thm. 3.1])**.** *Fix an arbitrary $[n,k,d]$-code $C \subset \mathcal{T}_{\iota+\kappa}^n$, with extended code $\widehat{C} \subset A_{\iota,\kappa,\tau}^n$, and a proximity parameter $e \in \{1, \ldots, \lfloor \frac{d-1}{3} \rfloor\}$. If elements $u_0, \ldots, u_{m_0-1}$ of $A_{\iota,\kappa,\tau}^n$ satisfy*

$$\Pr_{(r_{\ell_1},\ldots,r_{\ell-1}) \in \mathcal{T}_\tau^{\ell_0}} \left[d\left(\begin{bmatrix} & \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i) & \end{bmatrix} \cdot \begin{bmatrix} — & u_0 & — \\ & \cdots & \\ — & u_{m_0-1} & — \end{bmatrix}, \widehat{C} \right) \leq e \right] > 2 \cdot \log m_0 \cdot \frac{e}{|\mathcal{T}_\tau|},$$

then $d^{m_0}\left((u_i)_{i=0}^{m-1}, \widehat{C}^{m_0}\right) \leq e$.

Proof. The proof goes through almost exactly as does that of [DP23, Thm. 3.1], with select modifications. Indeed, we require only a substitute for the Schwartz–Zippel-based argument given in [DP23, Lem. 3.4]. In our setting, each locus $C_{b,j} \subset \mathcal{T}_\tau^{\ell_0-1}$ is, now, the vanishing locus in $\mathcal{T}_\tau^{\ell_0-1}$ of a certain polynomial expression in the variables $(r_{\ell_1}, \ldots, r_{\ell-1})$, whose *coefficients*, on the other hand, *reside in* $A_{\iota,\kappa,\tau}$ (and moreover are not all zero). Decomposing each such coefficient into a 2^κ-tuple of $\mathcal{T}_\tau$-elements, using the natural $\mathcal{T}_\tau$-basis $1, Y_0, Y_1, \ldots, Y_0 \cdot \cdots Y_{\kappa-1}$ of $A_{\iota,\kappa,\tau}$, we see that the vanishing locus $C_{b,j}$ is the *intersection* in $\mathcal{T}_\tau^{\ell_0-1}$ of 2^κ vanishing loci, each itself the vanishing locus of a certain combination of the $\ell_0 - 1$-variate, multilinear Lagrange basis polynomials in the standard polynomial ring $\mathcal{T}_\tau[R_{\ell_1}, \ldots, R_{\ell-2}]$. Moreover, *at least one* among these latter combinations features a nonzero combination vector. Applying Schwartz–Zippel to all 2^κ loci, then, we see that at least one among these loci is bounded from above in mass by $(\ell_0 - 1) \cdot \frac{e}{|\mathcal{T}_\tau|}$, so that their intersection also is. This completes the argument that $\mu(C_{b,j}) \leq \frac{\ell_0-1}{|\mathcal{T}_\tau|}$. We note that a identical adaptation, in the univariate setting, must also be made to the proof of [DP23, Thm. 2.1]. Up to these adjustments, the proof of [DP23, Thm. 3.1] otherwise goes through in our setting without change. □

Our construction. We now define our packing-based construction, which adapts and extends Construction 3.7 above. Slightly restricting that construction's signature, we require that K take the form $\mathcal{T}_\iota$, for some ι (and that $\Pi.\mathsf{Setup}$ directly accept the parameter ι, instead of K).

Construction 3.11 (Block-level encoding-based polynomial commitment scheme)**.**
We define $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Prove}, \mathsf{Verify})$ as follows.

- $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, \iota)$. On input 1^λ, ℓ, and ι, choose integers ℓ_0 and ℓ_1 for which $\ell_0 + \ell_1 = \ell$, and write $m_0 \coloneqq 2^{\ell_0}$ and $m_1 \coloneqq 2^{\ell_1}$. Return an integer $\kappa \geq 0$, a tower height $\tau \geq \log(\omega(\log\lambda))$, an $\left[n, \frac{m_1}{2^\kappa}, d\right]$-code $C \subset \mathcal{T}_{\iota+\kappa}^n$ for which $n = 2^{O(\ell)}$ and $d = \Omega(n)$, and a repetition parameter $\rho = \Theta(\lambda)$.
- $(c, u) \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input $t(X_0, \ldots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, express $t = (t_0, \ldots, t_{2^\ell-1})$ in coordinates with respect to the multilinear Lagrange basis and collate the result row-wise into an $m_0 \times m_1$ matrix $(t_i)_{i=0}^{m_0-1}$. By grouping the column indices $\{0, \ldots, m_1 - 1\}$ into 2^κ-sized chunks and, for each row, applying the natural embedding chunk-wise, realize $(t_i)_{i=0}^{m_0-1}$ as an $m_0 \times \frac{m_1}{2^\kappa}$ matrix, with entries in $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$. Apply $\widehat{C}$'s encoding function row-wise to each of $(t_i)_{i=0}^{m_0-1}$'s rows, so obtaining a further, $m_0 \times n$ matrix $(u_i)_{i=0}^{m_0-1}$, again with entries in $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$. Output a Merkle commitment c to $(u_i)_{i=0}^{m_0-1}$ and the opening hint $u \coloneqq (u_i)_{i=0}^{m_0-1}$.
- $b \leftarrow \Pi.\mathsf{Open}(\mathsf{params}, c; t, u)$. On input $t(X_0, \ldots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, opening hint $(u_i)_{i=0}^{m_0-1}$, and commitment c, verify that $(u_i)_{i=0}^{m_0-1}$ Merkle-hashes to c. Collate t into a matrix $(t_i)_{i=0}^{m_0-1}$, encode the resulting matrix as above, and verify that $d^{m_0}\left((\mathsf{Enc}(t_i))_{i=0}^{m_0-1}, (u_i)_{i=0}^{m_0-1}\right) \overset{?}{<} \frac{d}{3}$.

We define $\Pi.\mathsf{Prove}$ and $\Pi.\mathsf{Verify}$ by applying the Fiat–Shamir heuristic to the following interactive protocol, where $\mathcal{P}$ has $t(X_0, \ldots, X_{\ell-1})$ and $(u_i)_{i=0}^{m_0-1}$, and $\mathcal{P}$ and $\mathcal{V}$ have c, s, and $(r_0, \ldots, r_{\ell-1}) \in \mathcal{T}_\tau^\ell$.

- $\mathcal{P}$ computes the matrix–vector product $t' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1}$, here interpreting the matrix $(t_i)_{i=0}^{m_0-1}$ as an unpacked, $m_0 \times m_1$ matrix with entries in $\mathcal{T}_\iota$. $\mathcal{P}$ sends $\mathcal{V}$ t' in the clear.
- For each $i \in \{0, \ldots, \rho - 1\}$, $\mathcal{V}$ samples $j_i \leftarrow \{0, \ldots, n - 1\}$. $\mathcal{V}$ sends $\mathcal{P}$ the set $J \coloneqq \{j_0, \ldots, j_{\rho-1}\}$.
- For each $j \in J$, $\mathcal{P}$ sends $\mathcal{V}$ the column $(u_{i,j})_{i=0}^{m_0-1}$, interpreted as a vector with entries in the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$, as well as an accompanying Merkle authentication path against c.
- First, $\mathcal{V}$ requires $s \overset{?}{=} t' \cdot \bigotimes_{i=0}^{\ell_1-1}(1 - r_i, r_i)$ (i.e., a simple dot-product over $\mathcal{T}_\tau$). $\mathcal{V}$ then applies the *natural embedding* to the $\mathcal{T}_\tau$-vector t', chunk-wise, so realizing it as a length-$\frac{m_1}{2^\kappa}$ vector with entries in $A_{\iota,\kappa,\tau}$, and finally encodes this latter vector, writing $u' \coloneqq \widehat{\mathsf{Enc}}(t')$, say. For each $j \in J$, $\mathcal{V}$ verifies the Merkle path attesting to $(u_{i,j})_{i=0}^{m_0-1}$, and moreover checks $\bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (u_{i,j})_{i=0}^{m_0-1} \overset{?}{=} u'_j$, where we use the *constant* $\mathcal{T}_\tau$-action on $A_{\iota,\kappa,\tau}$ on the left, and the equality is one of $A_{\iota,\kappa,\tau}$-elements.

We again require that $\iota \in O(\log(\lambda))$, lest the scheme fail to be efficiently computable; we moreover assume that $\tau \geq \iota$, so that the tower algebra $A_{\iota,\kappa,\tau}$ is well-defined. We note that the growth requirement $\tau \geq \log(\omega(\log\lambda))$ captures precisely the condition whereby $\frac{1}{|\mathcal{T}_\tau|}$ is negligible in λ. Indeed, while requiring $\tau \geq \Omega(\log\lambda)$, say, would more-than-guarantee our scheme's asymptotic security, the more delicate allowance $\tau \geq \log(\omega(\log\lambda))$ in fact suffices, and moreover figures centrally in our sharp asymptotic *efficiency* analysis below (see Theorem 3.14).

We emphasize that Construction 3.11's setup routine $\Pi.\mathsf{Setup}$ returns a code C over the alphabet $\mathcal{T}_{\iota+\kappa}$, which—in general—is *larger than* the coefficient field $\mathcal{T}_\iota$ at hand. On the other hand, the efficiency of Construction 3.11 is identical to that which Construction 3.7 above *would* feature if it were run on a $\mathcal{T}_{\iota+\kappa}$-matrix of size $m_0 \times \frac{m_1}{2^\kappa}$. In other words, Construction 3.11 makes possible the use of a code over an alphabet 2^κ-fold larger, say, than $\mathcal{T}_\iota$, and yet simultaneously "compensates" for this expense by shrinking the prover's matrix.

Construction 3.11's completeness amounts to the "commutativity" of a certain sequence of actions on the $\mathcal{T}_\iota$-matrix $(t_i)_{i=0}^{m_0-1}$; that is, $(t_i)_{i=0}^{m_0-1}$ either is *combined*, *packed*, and then *encoded*, or else is *packed*, *encoded*, and then *combined*. Since the natural embedding is $\mathcal{T}_\tau$-linear, the first pathway's *combination* and *packing* operations can be interchanged. On the other hand, the interchangability of the *combination* and *encoding* operations entails exactly the $\mathcal{T}_\tau$-linearity of $\widehat{\mathsf{Enc}}$, which we have already established above.

We note that the security results below draw significantly from [DP23, § 4], and repeat certain swathes of that work verbatim.

Theorem 3.12. *The scheme of Construction 3.11 is binding.*

Proof. Deferred to Appendix B. □

Theorem 3.13. *If the query sampler $\mathcal{Q}$ is admissible, then the scheme of Construction 3.11 is extractable.*

Proof. Deferred to Appendix B. □

3.5 Efficiency

In this subsection, we discuss the efficiency of Construction 3.11, with a view towards attaining certain *concrete soundness* thresholds. We note that a somewhat more rudimentary treatment of this section's material appears in [DP23].

Verifier cost. Departing slightly from standard efficiency analyses, we analyze both *proof size* and *verifier runtime* under one banner; indeed, we view both metrics as disparate aspects of a unified *verifier cost*. (This approach comports well with the cost structure of Ethereum, say, in which each transaction's *calldata size* and *verification complexity* contribute jointly to its gas cost.) We define the relevant variables as follows:

- $\mathfrak{b}$: The cost, to the verifier, of each bit transmitted to it.
- $\mathfrak{T}_{\iota}$: The cost, to the verifier, of multiplying two $\mathcal{T}_{\iota}$-elements.
- $\mathfrak{T}_{\tau}$: The cost, to the verifier, of multiplying two $\mathcal{T}_{\tau}$-elements.
- $\mathfrak{Enc}$: The cost, to the verifier, of encoding a message in $\mathcal{T}_{\iota+\kappa}^{m_1/2^{\kappa}}$.
- $\mathfrak{Hash}$. The cost, to the verifier, of hashing a single $\mathcal{T}_{\iota+\kappa}$-element.

We recall that a $\mathcal{T}_{\tau}$-element and a $\mathcal{T}_{\iota}$-element can be multiplied together with cost $2^{\tau-\iota} \cdot \mathfrak{T}_{\iota}$. Finally, we ignore throughout the cost of addition (which amounts to bitwise XOR).

We reckon the verifier's costs as follows. The prover must transmit to the verifier the message t', which consists of m_1 $\mathcal{T}_{\tau}$-elements, as well as the ρ m_0-element columns $(u_{i,j})_{i=0}^{m_0-1}$, for $j \in J$, each valued in $\mathcal{T}_{\iota+\kappa}$. The total proof size is thus $2^{\tau} \cdot m_1 + 2^{\iota+\kappa} \cdot m_0 \cdot \rho$ bits. Computationally, the verifier must first compute the tensor-expansions $\bigotimes_{i=0}^{\ell_1-1}(1-r_i, r_i)$ and $\bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i)$. Using the algorithm [Tha22, Lem. 3.8], the verifier can compute these using m_1 and m_0 $\mathcal{T}_{\tau}$-multiplications, respectively. To encode the message t', the verifier must perform $C \subset \mathcal{T}_{\iota+\kappa}^{n}$'s encoding operation $2^{\tau-\iota}$ times. In addition, the verifier must perform $\rho \cdot 2^{\kappa}$ $\mathcal{T}_{\tau}$-by-$\mathcal{T}_{\iota}$ dot products, each of length m_0. The total cost of these latter dot-products equals that of $m_0 \cdot \rho \cdot 2^{\kappa} \cdot 2^{\tau-\iota}$ $\mathcal{T}_{\iota}$-multiplications. Finally, the verifier must perform ρ Merkle-path verifications. Each such verification entails hashing a column of m_0 $\mathcal{T}_{\iota+\kappa}$-elements (as well as performing l_1 further hash evaluations, which we ignore).

Adding all of these components, we obtain the following total verifier costs:

- $\mathfrak{b}$: $2^{\tau} \cdot m_1 + 2^{\iota+\kappa} \cdot m_0 \cdot \rho$.
- $\mathfrak{T}_{\iota}$: $m_0 \cdot \rho \cdot 2^{\tau-\iota+\kappa}$.
- $\mathfrak{T}_{\tau}$: $m_0 + m_1$.
- $\mathfrak{Enc}$: $2^{\tau-\iota}$.
- $\mathfrak{Hash}$. $\rho \cdot m_0$.

We pause to record to the following fundamental asymptotic guarantee:

Theorem 3.14. *For each fixed $\iota \in \mathbb{N}$, and arbitrary ℓ and λ in $\mathbb{N}$, Construction 3.11 can be instantiated in such a way as to impose verifier cost $\widetilde{O}\left(\sqrt{\lambda \cdot 2^{\ell}}\right)$, counting both bits transferred and bit-operations performed.*

Proof. Deferred to Appendix B. □

We note that the analyses of both Brakedown [Gol+23, Thm. 1] and of Diamond and Posen [DP23, § 4] measure only *field-elements transferred* and *field-operations*. Theorem 3.14 performs a sharper asymptotic analysis; it shows that—provided that it chooses τ sufficiently carefully—Construction 3.11 in fact attains square-root verifier efficiency, in both in the security parameter and the polynomial's size, even at the level of bits.

Concrete soundness. We identify and discuss, in concrete terms, the various sources of soundness error which arise throughout Theorem 3.13. We refer throughout to the parameters d, n, ρ, ι, κ, τ, m_0 and m_1, recalling their roles in $\Pi.\mathsf{Setup}$.

- **Tensor batching error Ξ_B.** This is the probability, taken over the query sampler's choice of $(r_0, \ldots, r_{\ell-1}) \leftarrow \mathcal{T}_\tau^\ell$, that, though $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) \geq \frac{d}{3}$, we nonetheless have $d\Big(u', \widehat{C}\Big) < \frac{d}{3}$, where we write $u' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i) \cdot (u_i)_{i=0}^{m_0-1}$. By Theorem 3.10 (see also Lemma B.1), $\Xi_B \leq 2 \cdot \ell_0 \cdot \frac{d}{3 \cdot |\mathcal{T}_\tau|}$.
- **Non-proximal per-query error Ξ_N.** This is the probability, taken over the verifier's choice of a single index $j \leftarrow \{0, \ldots, n-1\}$, that, though $d\Big(u', \widehat{C}\Big) \geq \frac{d}{3}$, nonetheless $u'_j = \mathsf{Enc}(t')_j$ holds. The analysis of Lemma B.1 shows that $\Xi_N \leq 1 - \frac{d}{3 \cdot n}$.
- **Proximal per-query error Ξ_P.** This is the probability, taken over the verifier's choice of a single index $j \leftarrow \{0, \ldots, n-1\}$, that, in the case $d\Big(u', \widehat{C}\Big) < \frac{d}{3}$ but the message $t' \neq \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1}$ is wrong, nonetheless $u'_j = \mathsf{Enc}(t')_j$ holds. The analysis of Lemma B.2 shows that $\Xi_P \leq 1 - \frac{2 \cdot d}{3 \cdot n}$.

Putting these three sources of error together, and following the analyses of Lemmas B.1 and B.2, we define the protocol's *total soundness error* as follows:

$$\Xi \coloneqq \Xi(d, n, \rho, \tau, \ell_0, \ell_1) = \max(\Xi_B + \Xi_N^\rho, \Xi_P^\rho).$$

We justify this definition in the following way (in fact, this is a very rough summary of the proof of Theorem 3.13). We note that either the prover's committed matrix $(u_i)_{i=0}^{m_0-1}$ satisfies $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) < \frac{d}{3}$ or it doesn't. If it doesn't, then the analysis of Lemma B.1 bounds the verifier's acceptance probability from above by $\Xi_B + \Xi_N^\rho$. If it does, then the message list $(t_i)_{i=0}^{m_0-1}$ is well-defined, so that t' is either correct or it's not; in the latter case, Lemma B.2 bounds the verifier's probability of acceptance by Ξ_P^ρ. Barring all of these failure events, we indeed have that $s = t(r_0, \ldots, r_{\ell-1})$. We note that we slightly simplify our treatment here by analyzing Construction 3.11 as an *IOP*, and ignoring the runtime of the emulator $\mathcal{E}$, as well as the probability that $\mathcal{E}$ aborts on a successful proof (say, because it fails to extract $(u_i)_{i=0}^{m_0-1}$). This simplification, in the setting of concrete analysis, is justified in Brakedown [Gol+23, p. 211], for example.

We define the *bits of security* obtained by Construction 3.11 as $\lambda \coloneqq -\log(\Xi)$.

Case studies. In order to concretely assess the performance characteristics of Construction 3.11, we study various instantiations of that scheme. For comparison, we also explore various approaches based on the use of concatenated codes in Construction 3.7. In each the following examples, we set $\iota \coloneqq 0$ (that is, we commit to $\mathbb{F}_2$-polynomials), as well as $\ell \coloneqq 32$, so that the total size of the polynomial at hand is 512 MiB. Throughout each example, we attain 100 bits of security. To standardize the case studies' respective prover complexities, we consider only codes with the fixed relative rate $\gamma \coloneqq \frac{1}{4}$.

Example 3.15 (Reed–Solomon code with block-level testing)**.** We begin with the efficiency of Construction 3.11. We first remark that the alphabet size parameter $\kappa \coloneqq 4$ makes available *only* those width parameters ℓ_1 at most 18; indeed, the Reed–Solomon requirement $|K| \geq n$ demands that $|\mathcal{T}_\kappa| = 2^{2^\kappa} \geq \frac{1}{\gamma} \cdot \frac{2^{\ell_1}}{2^\kappa}$, so that $2^\kappa \geq 2 + \ell_1 - \kappa$. In fact, we set $\kappa \coloneqq 4$, $\ell_0 \coloneqq 14$ and $\ell_1 \coloneqq 18$ (these choices yield the smallest possible proofs). We thus have $m_1 = 2^{18}$, $k = 2^{14}$, and $n = 2^{16}$. Setting $\tau \coloneqq 7$—and using $d = n - k + 1 = 2^{16} - 2^{14} + 1 = 49{,}153$—we compute $\Xi_B \leq 2 \cdot 14 \cdot \frac{d}{3 \cdot 2^{128}} \approx 2^{-109.193}$. Moreover, we compute the non-proximal per-query error

$\Xi_N \leq 1 - \frac{d}{3 \cdot n} \approx 0.75$ and the proximal per-query error $\Xi_P \leq 1 - \frac{2 \cdot d}{3 \cdot n} \approx 0.5$. Using a direct computation, we see that the total soundness error Ξ of equation (3.5) drops below 2^{-100} just when the number of queries ρ becomes 241 or greater. Using the expression for $\mathfrak{b}$ given above, we compute directly the proof size of 11.531 MiB, or about $2^{26.527}$ bits.

Example 3.16 (Concatenated code with trivial inner code)**.** For reference, we compare Example 3.15 to the construction whereby a trivial concatenated code—i.e., with Reed–Solomon outer code and identity inner code—is used in Construction 3.7 (i.e., *without* block-level testing). We again set $\kappa := 4$, $\ell_0 := 15$ and $\ell_1 := 21$. In this setting, the resulting *binary* code has distance $d = 49{,}153$ identical to the code of the above construction; on the other hand, its message length $k = 2^{18}$ and block length $n = 2^{20}$ are both 2^κ-fold higher. We thus obtain the identical batching error $\Xi_B \approx 2^{-109.193}$; our non-proximal and proximal per-query errors, on the other hand, are $\Xi_N = 1 - \frac{d}{3 \cdot n} \approx 0.984$ and $\Xi_P = 1 - \frac{2 \cdot d}{3 \cdot n} \approx 0.969$. Again calculating directly, we see that 4,402 queries are required to obtain 100 bits of soundness. This scheme's queries, however, are each 16-fold cheaper than Example 3.15's are; we obtain a total proof size of 12.598 MiB, or about $2^{26.655}$ bits.

Example 3.17 (Nontrivial concatenated code)**.** We finally examine the efficiency of Construction 3.7's instantiation on a *nontrivial* concatenated code (i.e., with nonidentity inner code). In order to run an apples-to-apples comparison—i.e., between schemes whose prover costs are comparable—we set both our inner code and outer code's rates to be $\frac{1}{2}$, so that our concatenated code has rate $\frac{1}{4}$. Specifically, we set $\kappa := 4$, and set $C_{\text{out}} \subset \mathcal{T}_4^{n_{\text{out}}}$ to be the Reed–Solomon code $\mathsf{RS}_{\mathcal{T}_4}[2^{15}, 2^{14}]$; for $C_{\text{in}} \subset \mathcal{T}_0^{k_{\text{in}}}$, we use the Reed–Muller $[32, 16, 8]$-code $\mathsf{RM}_{\mathcal{T}_0}[2, 5]$. (We note that 8 is actually the *best possible* distance that a binary $[32, 16]$-code can attain; we refer to the database of Grassl [Gra].) We see that our concatenated code satisfies $k = 2^{18}$ and $n = 2^{20}$, and has distance $d = 8 \cdot (2^{14} + 1) = 131{,}080$. We accordingly compute $\Xi_B \leq 2 \cdot 14 \cdot \frac{d}{3 \cdot 2^{128}} \approx 2^{-107.778}$, as well as $\Xi_N = 1 - \frac{d}{3 \cdot n} \approx 0.958$ and $\Xi_P = 1 - \frac{2 \cdot d}{3 \cdot n} \approx 0.917$. We calculate that 1,629 queries suffice to deliver 100 bits of soundness, and obtain a proof size of 7.182 MiB, or $2^{25.844}$ bits.

Remark 3.18. We find it plausible that, in the setting of Example 3.15, the stronger proximity gap result of Ben-Sasson, Carmon, Ishai, Kopparty, and Saraf [Ben+23, Thm. 1.4] could be brought to bear. Indeed, that result guarantees that, in the Reed–Solomon setting, even for those proximity parameters $e \in \{0, \ldots, \lfloor \frac{d-1}{2} \rfloor\}$ allowed to range as high as the unique decoding radius, we nonetheless obtain a proximity gap, albeit with the false witness probability $\frac{n}{\mathcal{T}_\tau}$ slightly worse than that of $\frac{e+1}{|\mathcal{T}_\tau|}$ guaranteed by [DP23, Thm. 2.1] (we refer to [DP23, Rem. 3.7] for further comparison of these results). Of course, to apply that result to Example 3.15, we would need an analogue of Theorem 3.10 above; that is, we would need a result in the *algebra* setting which adapts [Ben+23, Thm. 1.4], precisely as Theorem 3.10 adapts [DP23, Thm. 3.1]. While we feel confident that such an adaptation should be possible, we have not undertaken it. *If* that adaptation were available, then, in Example 3.15, we would obtain the rather better proof size of 50.5 MiB, or $2^{28.658}$ bits.

We record selected proof size benchmarks in the below table. We record the benchmarks derived above, which pertain to the case $\ell = 32$ (so that the total data size is 512 MiB), as well as benchmarks for the further case $\ell = 36$ (corresponding to 8 GiB of total data).

Construction Used	Num. Variables ℓ	Parameters (ℓ_0, ℓ_1, κ)	Proof Size (MiB)
Reed–Solomon with block-level testing.	32	(14, 18, 4)	11.531
(See Example 3.15.)	36	(15, 21, 5)	66.527
Reed–Solomon, assum. prox-gap $\lfloor \frac{d-1}{2} \rfloor$.	32	(14, 18, 4)	8.625
(See Remark 3.18.)	36	(15, 21, 5)	50.500
Concatenated code w/ ad-hoc inner code.	32	(14, 18, 4)	7.182
(See Example 3.17.)	36	(16, 20, 5)	33.063

Table 1: Proof size benchmarks.

In the final benchmark—that describing a concatenated code with $\kappa := 5$—we use the ad-hoc inner $[64, 32, 12]$-code of Grassl [Gra+19] (this code is a subcode of an *extended BCH code*). As Grassl's database

indicates, we are able neither to construct, nor to rule out the existence of, a binary $[64, 32, 16]$-code. The existence of just such a code would further improve the benchmark given in the last row.

We present comprehensive benchmarks in Section 6 below.

4 Polynomial IOPs for Binary Tower Fields

In this section, we review and develop several interactive protocols and polynomial IOPs, which we moreover specialize to the setting of binary tower fields. Certain among these protocols adapt already-known techniques, but surface further performance improvements made possible by the tower setting. We refer throughout to Chen, Bünz, Boneh and Zhang's *HyperPlonk* [Che+23, Def. 4.1], though we modify rather significantly that work's formalisms.

4.1 Definitions and Notions

We fix throughout what follows a maximal tower height $\tau \in \mathbb{N}$; we understand $\tau := \tau(\lambda)$ as depending on an available security parameter.

Definition 4.1. A *polynomial IOP* $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is an interactive protocol in which the parties may freely use a certain *multilinear polynomial oracle*, which operates as follows, on the security parameter $\lambda \in \mathbb{N}$:

FUNCTIONALITY 4.2 (polynomial oracle)**.**
A tower height $\tau := \tau(\lambda)$ and a binary tower $\mathcal{T}_0 \subset \mathcal{T}_1 \subset \cdots \subset \mathcal{T}_\tau$ are fixed.

- On input $(\texttt{submit}, \iota, \nu, t)$ from $\mathcal{I}$ or $\mathcal{P}$, where $\iota \in \{0, \ldots, \tau\}$, $\nu \in \mathbb{N}$, and $t \in \mathcal{T}_\iota[X_0, \ldots, X_{\nu-1}]^{\preceq 1}$, output $(\texttt{receipt}, \iota, \nu, [t])$ to $\mathcal{I}$, $\mathcal{P}$ and $\mathcal{V}$, where $[t]$ is some unique handle onto the polynomial t.
- On input $(\texttt{query}, [t], r)$ from $\mathcal{V}$, where $r \in \mathcal{T}_\tau^\nu$, send $\mathcal{V}$ $(\texttt{evaluation}, t(r_0, \ldots, r_{\nu-1}))$.

Definition 4.3. The polynomial IOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for the indexed relation R is *secure* if, for each PPT adversary $\mathcal{A}$, there exists an expected PPT emulator $\mathcal{E}$ and a negligible function negl, such that, for each security parameter $\lambda \in \mathbb{N}$ and each pair $(\mathbb{i}, \mathbb{x})$, provided that the protocol is run on the security parameter λ, writing $\mathsf{vp} := \mathcal{I}(\mathbb{i})$ and $\mathbb{w} \leftarrow \mathcal{E}^{\mathcal{A}}(\mathbb{i}, \mathbb{x})$, we have $|\Pr[\langle \mathcal{A}(\mathbb{i}, \mathbb{x}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1] - \Pr[R(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 1]| \leq \mathsf{negl}(\lambda)$.

We note that we grant $\mathcal{E}$ full internal access to $\mathcal{A}$. In particular, $\mathcal{E}$ may intercept all outbound messages sent by $\mathcal{A}$, *including* those messages $(\texttt{submit}, \iota, \nu, t)$ $\mathcal{A}$ sends directly to the polynomial oracle, as well as, of course, those it sends to $\mathcal{V}$. We note that, in practice, our emulator $\mathcal{E}$ will be *straight-line* (i.e., non-rewinding) and strict polynomial-time, though these latter properties aren't required by Definition 4.3.

It is shown in [BFS20, § E] that, by inlining an extractable polynomial commitment scheme (in the sense of Definition 3.5) into a secure polynomial IOP (in the sense of Definition 4.3), one obtains a secure argument of knowledge for the relation R.

Definition 4.4. For parameters ι, ν, and μ in $\mathbb{N}$, ν-variate, μ-ary *polynomial predicate* over $\mathcal{T}_\iota$ is a boolean-valued function $\Phi_{\iota,\nu} : \mathcal{T}_\iota[X_0, \ldots, X_{\nu-1}]^\mu \to \{0, 1\}$.

Example 4.5. We record certain key polynomial predicates, roughly following HyperPlonk [Che+23].

1. **Query.** On parameters ι and ν in $\mathbb{N}$, $s \in \mathcal{T}_\tau$, and $r \in \mathcal{T}_\tau^\nu$, sends $\mathsf{Query}(r, s)_{\iota,\nu} : T \mapsto T(r_0, \ldots, r_{\nu-1}) = s$.
2. **Sum.** On parameters ι and ν in $\mathbb{N}$ and $e \in \mathcal{T}_\iota$, sends $\mathsf{Sum}(e)_{\iota,\nu} : T \mapsto \sum_{v \in \mathcal{B}_\nu} T(v) = e$.
3. **Zero.** On parameters ι and ν in $\mathbb{N}$, sends $\mathsf{Zero}_{\iota,\nu} : T \mapsto \bigwedge_{v \in \mathcal{B}_\nu} T(v) = 0$.
4. **Product.** On parameters ι and ν in $\mathbb{N}$, the binary *product predicate* sends $\mathsf{Product}_{\iota,\nu} : (T, U) \mapsto \prod_{v \in \mathcal{B}_\nu} T(v) = \prod_{v \in \mathcal{B}_\nu} U(v) \wedge \bigwedge_{v \in \mathcal{B}_\nu} (T(v) = 0 \iff U(v) = 0)$.
5. **Multiset.** On parameters ι, ν, and μ in $\mathbb{N}$, the $2 \cdot \mu$-ary *multiset predicate* sends $\mathsf{Multiset}(\mu)_{\iota,\nu} : (T_0, \ldots, T_{\mu-1}, U_0, \ldots, U_{\mu-1}) \mapsto \{(T_0(v), \ldots, T_{\mu-1}(v)) \mid v \in \mathcal{B}_\nu\} = \{(U_0(v), \ldots, U_{\mu-1}(v)) \mid v \in \mathcal{B}_\nu\}$, where we understand both objects on the right-hand side as *multisets* (counted with multiplicity).

6. **Permutation.** On parameters ι, ν, and μ in $\mathbb{N}$, and a bijection $\sigma : \{0, \dots, \mu - 1\} \times \mathcal{B}_\nu \to \{0, \dots, \mu - 1\} \times \mathcal{B}_\nu$, the μ-ary *permutation predicate* sends $\mathsf{Permutation}(\sigma)_{\iota,\nu} : (T_0, \dots, T_{\mu-1}) \mapsto \bigwedge_{(i,v)\in\{0,\dots,\mu-1\}\times\mathcal{B}_\nu} T_{i'}(v') = T_i(v)$, where we write $(i', v') \coloneqq \sigma(i, v)$ for each $(i, v) \in \{0, \dots, \mu-1\}\times\mathcal{B}_\nu$.

7. **Lookup.** On parameters ι and ν in $\mathbb{N}$, sends $\mathsf{Lookup}_{\iota,\nu} : (T, U) \mapsto \bigwedge_{v\in\mathcal{B}_\nu} \exists v' \in \mathcal{B}_\nu : U(v) = T(v')$.

We note that each predicate $\mathsf{Query}(r, s)_{\iota,\nu}$ can be evaluated directly by the verifier, on any handle $[t]$, by means of a single query to the polynomial oracle.

Our product predicate diverges from HyperPlonk's [Che+23, § 3.3] in various respects. Their predicate requires that the "denominator" U be everywhere-nonzero on the cube, as well as that the product $\prod_{v\in\mathcal{B}_\nu} \frac{T(v)}{U(v)}$ equal a prescribed value. We simplify that predicate by specializing this prescribed value to 1; on the other hand, we also more correctly handle the case of "division by zero" (their protocol actually *fails* to assert that U is everywhere-nonvanishing on the cube). Though our product predicate indeed admits a generalization which more closely follows [Che+23, § 3.3]—i.e., where the product $\prod_{v\in\mathcal{B}_\nu} \frac{T(v)}{U(v)}$ can be arbitrary, *and* where we moreover correctly handle those denominators U which vanish—this generalization is complicated, and we have opted not to present it. We discuss this matter further in Remark 4.18 below. Finally, we present a permutation predicate slightly more sophisticated than HyperPlonk's [Che+23, § 3.5]; specifically, ours supports permutations which act across multiple "columns".

The following notational abstraction figures extensively in what follows.

Definition 4.6. A ν-variate *virtual polynomial* over $\mathcal{T}_\iota$ is a list of handles $[t_0], \dots, [t_{\mu-1}]$, each representing a polynomial defined over $\mathcal{T}_\iota$, together with an arithmetic circuit, whose leaves are either indeterminates in the list $X_0, \dots X_{\nu-1}$ or else constants in $\mathcal{T}_\iota$, and in which we permit not just the binary gates $+$ and $\times$, but moreover, for each $i \in \{0, \dots, \mu - 1\}$, the ν_i-ary gate $t_i(X_0, \dots, X_{\nu_i-1})$ (assuming that t_i is ν_i-variate). We write $T \in \mathcal{T}_\iota[X_0, \dots, X_{\nu-1}]$ for the polynomial represented by the circuit, and $[T]$ for the virtual polynomial.

We note that each virtual polynomial $[T]$ may be evaluated at any input $(x_0, \dots, x_{\nu-1}) \in \mathcal{T}_\iota^\nu$—albeit in general, not efficiently—by any machine which can query the handles $[t_0], \dots, [t_{\mu-1}]$. We now treat *efficient* protocols for virtual polynomials.

Definition 4.7. A *virtual polynomial protocol* for the μ-ary polynomial predicate $\Phi_{\iota,\nu}$ is an interactive protocol $\Sigma = (\mathcal{P}, \mathcal{V})$ which takes as common input a list $([T_0], \dots, [T_{\mu-1}])$ of ν-variate virtual polynomials. The protocol Σ is *secure* with respect to $\Phi_{\iota,\nu}$ if, for each PPT adversary $\mathcal{A}$, there is a negligible function negl for which, for each $\lambda \in \mathbb{N}$ and each input list $([T_0], \dots, [T_{\mu-1}])$, if the protocol is run on the security parameter λ, then we have $\Pr[\langle \mathcal{A}(T_0, \dots, T_{\mu-1}), \mathcal{V}([T_0], \dots, [T_{\mu-1}])\rangle = 1 \wedge \Phi_{\iota,\nu}(T_0, \dots, T_{\mu-1}) = 0] \leq \mathsf{negl}(\lambda)$.

We note that, in the cases we treat below, a *single* negligible function negl suffices for all adversaries $\mathcal{A}$, though this state of affairs is not mandated by Definition 4.7.

We highlight, in particular, the special case of Definition 4.7 in which $\Phi_{\iota,\nu}$ takes the form $\mathsf{Query}(r, s)_{\iota,\nu}$.

Definition 4.8. An *evaluation protocol* for the ν-variate virtual polynomial $[T]$ over $\mathcal{T}_\iota$ is a family of virtual polynomial protocols, parameterized by $r \in \mathcal{T}_\tau^\nu$ and $s \in \mathcal{T}_\tau$, for the predicates $\mathsf{Query}(r, s)_{\iota,\nu}$ on the input $[T]$.

In practice, we often attach to each virtual polynomial T an appropriate evaluation protocol, and refer to the resulting bundle as an *evaluable* virtual polynomial.

Example 4.9 (Compositions)**.** A certain particularly simple sort of virtual polynomial consists of a list of ν-variate handles $[t_0], \dots, [t_{\mu-1}]$, together with a μ-variate *composition polynomial* $g \in \mathcal{T}_\iota[X_0, \dots, X_{\mu-1}]$, and represents the composition $T \coloneqq g(t_0(X_0, \dots, X_{\nu-1}), \dots, t_{\mu-1}(X_0, \dots, X_{\nu-1}))$. We note that $[T]$ admits an efficient evaluation protocol, at least if g is succinct; indeed, to decide the predicate $\mathsf{Query}(r, s)_{\iota,\nu}$, $\mathcal{V}$ may directly query each of the handles at r, evaluate g *itself* on the results, and finally compare the result to s.

Example 4.10 (Piecewise multilinears)**.** A further sort of virtual polynomial arises in the following way. For integers ι, ν, and μ in $\mathbb{N}$, where $\mu = 2^\alpha$ is a power of 2, say, and ν-variate handles $[t_0], \dots, [t_{\mu-1}]$ over $\mathcal{T}_\iota$, we introduce a piecewise function $T \in \mathcal{T}_\iota^{\mathcal{B}_{\nu+\alpha}}$, defined so that, for each $v \in \mathcal{B}_\nu$ and $u \in \mathcal{B}_\alpha$, $T(u \parallel v) = t_{\{u\}}(v)$ holds (we recall the identification $\{u\} \coloneqq \sum_{i=0}^{\alpha-1} 2^i \cdot u_i$). We finally identify T with its multilinear extension $T(X_0, \dots, X_{\nu+\alpha-1}) \in \mathcal{T}_\iota[X_0, \dots, X_{\nu+\alpha-1}]^{\preceq 1}$. We note that T defines a valid virtual polynomial in the

handles $[t_0], \ldots, [t_{\mu-1}]$; moreover, T is evaluable, provided μ is small. Indeed, to decide $\mathsf{Query}(r, s)_{\iota,\nu+\alpha}$, say, $\mathcal{V}$ may destructure $(r_0, \ldots, r_{\nu+\alpha-1}) \coloneqq r$, query the polynomials $[t_0], \ldots, [t_{\mu-1}]$ at $(r_\alpha, \ldots, r_{\nu+\alpha-1})$, obtaining the results $s_0, \ldots, s_{\mu-1}$, say, and finally output $s \stackrel{?}{=} \bigotimes_{i=0}^{\alpha-1}(1-r_i, r_i)\cdot(s_i)_{i=0}^{\mu-1}$ (here, $\bigotimes_{i=0}^{\alpha-1}(1-r_i, r_i)$ is a tensor product expansion in the sense of Subsection 2.2, and can be computed in $\Theta(\mu)$ time). The correctness of this procedure is essentially [Tha22, Lem. 3.6]. We write $[T] \coloneqq \mathtt{merge}([t_0], \ldots, [t_{\mu-1}])$ for this construction.

We finally note that virtual polynomials can be composed. Indeed, upon replacing some among the handles $[t_0], \ldots, [t_{\mu-1}]$ of some virtual polynomial $[T]$ with *further* virtual polynomials, we may nonetheless "unroll" the resulting object into a proper virtual polynomial $[T']$ in its own right. Finally, if $[T]$ *and* all of the sub-virtual polynomials are efficiently evaluable, then the composed virtual polynomial $[T']$ also is.

4.2 Prior Virtual Polynomial Protocols

By way of background, we briefly recall various well-known virtual protocols, for use below. We refer primarily to Thaler [Tha22] and HyperPlonk [Che+23, § 3].

Sumcheck. The *sumcheck* protocol is a virtual polynomial protocol for the predicate $\mathsf{Sum}(e)_{\iota,\nu} : T \mapsto \sum_{v\in\mathcal{B}_\nu} T(v) = e$. Internally, on input an evaluable, ν-variate virtual polynomial $[T]$, sumcheck invokes $[T]$'s implicit $\mathsf{Query}(r, s)_{\iota,\nu}$ protocol, on parameters $r \in \mathcal{T}_\tau^\nu$ and $s \in \mathcal{T}_\tau$ derived during the course of the sumcheck. The definition of the sumcheck protocol, as well as a proof that it securely evaluates $\mathsf{Sum}(e)_{\iota,\nu}$ in the sense of Definition 4.7, appear in Thaler [Tha22, § 4.1]. The protocol's soundness error is at most $\frac{\nu\cdot d}{|\mathcal{T}_\tau|}$, where d is the maximum *individual* degree exhibited by any of T's variables (plus the error inherent to $[T]$'s evaluation protocol). We emphasize that the known, highly-efficient algorithms for the sumcheck protocol's prover *require* that $[T]$ take the particular form given in Example 4.9 (i.e., that $[T]$ be a composition of multilinears); we refer to [Tha22, Lem. 4.5] for a discussion of these algorithms.

Zerocheck. We recall the predicate $\mathsf{Zero}_{\iota,\nu} : T \mapsto \bigwedge_{v\in\mathcal{B}_\nu} T(v) = 0$, as well as the *zerocheck* protocol of HyperPlonk [Che+23, § 3.2] (see also *Spartan* [Set20]).

PROTOCOL 4.11 (Zerocheck)**.**
Parameters ι, ν, and τ in $\mathbb{N}$ and a ν-variate virtual polynomials $[T]$ over $\mathcal{T}_\iota$ is fixed.

- $\mathcal{V}$ samples $r \leftarrow \mathcal{T}_\nu^\tau$, and sends r to $\mathcal{P}$.
- $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck protocol, with statement 0, on the virtual polynomial $[T'] \coloneqq \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\nu-1}, X_0, \ldots, X_{\nu-1}) \cdot T(X_0, \ldots, X_{\nu-1})$.

We note first of all that $[T']$ is a valid virtual polynomial, which moreover admits its own evaluation protocol. Indeed, to decide $\mathsf{Query}(r', s')_{\tau,\nu}(T')$, say, $\mathcal{V}$ may, after first *locally* evaluating $a \coloneqq \widetilde{\mathsf{eq}}(r, r')$—which takes $O(\nu)$ work—immediately return $s' \stackrel{?}{=} 0$ in case $a = 0$, and otherwise proceed with the appropriate protocol (i.e., that attached to $[T]$) deciding $\mathsf{Query}\left(r', \frac{s'}{a}\right)_{\iota,\nu}(T)$.

Theorem 4.12. *Protocol 4.11 securely decides the predicate* $\mathsf{Zero}_{\iota,\nu}$ *on* T*.*

Proof. Assuming that $\mathsf{Zero}_{\iota,\nu}(T) = 0$, we show that $\mathsf{Sum}(0)_{\tau,\nu}(T') = 1$ holds with negligible probability over $\mathcal{V}$'s random coins. Our hypothesis implies precisely that T's MLE $\widetilde{T} = \sum_{v\in\mathcal{B}_\nu} T(v) \cdot \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\nu-1}, X_0, \ldots, X_{\nu-1})$ is not identically zero. By the Schwartz–Zippel lemma, the probability, over $\mathcal{V}$'s choice of $r \leftarrow \mathcal{T}_\nu^\tau$, that $\widetilde{T}(r) = 0$ is thus at most $\frac{\nu}{|\mathcal{T}_\tau|}$. On the other hand, if $\widetilde{T}(r) \neq 0$, then $\sum_{v\in\mathcal{B}_\nu} T(v) \cdot \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\nu-1}, r_0, \ldots, r_{\nu-1}) \neq 0$, so that $\mathsf{Sum}(0)_{\tau,\nu}(T') = 0$, as required. □

The soundness error of the zerocheck protocol thus $\frac{\nu}{|\mathcal{T}_\tau|} + \frac{(d+1)\cdot\nu}{|\mathcal{T}_\tau|}$, where d, again, is the maximum individual degree exhibited by any of T's variables (*plus*, again, the error inherent to $[T]$'s implicit evaluation protocol). The first of these two terms is a zerocheck-specific soundness error; the term $\frac{(d+1)\cdot\nu}{|\mathcal{T}_\tau|}$ arises from zerocheck's internal use of the sumcheck on $[T']$.

Product check. We now record a protocol for the product predicate $\mathsf{Product}_{\iota,\nu} : (T, U) \mapsto \prod_{v\in\mathcal{B}_\nu} T(v) = \prod_{v\in\mathcal{B}_\nu} U(v) \wedge \bigwedge_{v\in\mathcal{B}_\nu}(T(v) = 0 \iff U(v) = 0)$ above, roughly following Setty and Lee's *Quarks* [SL20, § 5] and HyperPlonk [Che+23, § 3.3].

PROTOCOL 4.13 (Product check)**.**
Parameters ι and ν in $\mathbb{N}$, and ν-variate virtual polynomials $[T]$ and $[U]$ over $\mathcal{T}_\iota$, are fixed.

- $\mathcal{P}$ defines the function $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$ as follows. For each $v \in \mathcal{B}_\nu$ $\mathcal{P}$ sets $f(v) \coloneqq \frac{T(v)}{U(v)}$ if $U(v) \neq 0$, and $f(v) \coloneqq 1$ otherwise. $\mathcal{P}$ submits $(\texttt{submit}, \iota, \nu + 1, f')$ to the oracle, where $f' \in \mathcal{T}_\iota[X_0, \ldots, X_\nu]^{\preceq 1}$ is such that, for each $v \in \mathcal{B}_{\nu+1}$, both $f'(v \parallel 0) = f(v)$ and $f'(v \parallel 1) = f'(0 \parallel v) \cdot f'(1 \parallel v)$ hold.
- Upon receiving $(\texttt{receipt}, \iota, \nu + 1, [f'])$ from the oracle, $\mathcal{V}$ submits $(\texttt{query}, [f'], (0, 1, \ldots, 1))$ to the oracle; $\mathcal{V}$ requires that the response $(\texttt{evaluation}, f'(0, 1, \ldots, 1))$ satisfy $f'(0, 1, \ldots, 1) \stackrel{?}{=} 1$.
- $\mathcal{P}$ and $\mathcal{V}$ define a $\nu + 1$-variate virtual polynomial $[T']$ as follows:

$$[T'] \coloneqq \texttt{merge}([T], [f'](\cdot \parallel 1)) - \texttt{merge}([U], [f'](0 \parallel \cdot)) \cdot \texttt{merge}([f'](\cdot \parallel 0), [f'](1 \parallel \cdot)).$$

  $\mathcal{P}$ and $\mathcal{V}$ run a zerocheck on the virtual polynomial $[T']$.

Above, the expression $[f'](\cdot \parallel 0)$ denotes the ν-variate virtual polynomial which one obtains from the $\nu + 1$-variate handle $[f']$ upon fixing that function's last argument to be 0; its variants are analogous.

We modify the protocol given in [Che+23, § 3.3] in two distinct ways. On the one hand, our prover constructs the auxiliary function f in such a way as to appropriately handle the vanishing of the "denominator" U within the cube; we discuss this issue further in Remark 4.18. Separately, we define the virtual polynomial $[T']$ above—that is, the target of the zerocheck reduction—differently than does [Che+23, § 3.3], as we presently explain. The work [Che+23, § 3.3] sets (adapting their notation to ours) $[T'] \coloneqq \texttt{merge}([T] - [U] \cdot [f'](\cdot \parallel 0), [f'](\cdot \parallel 1) - [f'](0 \parallel \cdot) \cdot [f'](1 \parallel \cdot))$. While this construction is correct—and in fact agrees with our $[T']$ identically on $\mathcal{B}_{\nu+1}$—it suffers from the defect whereby $[T']$ is *not* a *composition of multilinears* in the sense of Example 4.9, *even if* T and U are themselves multilinear, and so fails to admit an (obvious) efficient sumcheck. Our construction remedies this issue, in that our $[T']$ *is* a composition of multilinears, at least if T and U are themselves multilinear. We emphasize that our protocol is correct and secure regardless of T and U; on the other hand, the efficiency of its implementation may require that T and U be multilinear (as they will be in our applications below).

Theorem 4.14. *Protocol 4.13 securely decides the predicate* $\mathsf{Product}_{\iota,\nu}$ *on* $[T]$ *and* $[U]$.

Proof. Assuming that $\mathcal{V}$ accepts and that $\mathsf{Zero}_{\iota,\nu+1}(T') = 1$, where T' is the virtual polynomial constructed during Protocol 4.13, we show that $\mathsf{Product}_{\iota,\nu}(T, U) = 1$ holds with probability 1. It follows directly from the definition of $\texttt{merge}$ that, under our hypothesis $\mathsf{Zero}_{\iota,\nu+1}(T') = 1$ we have, for each $v \in \mathcal{B}_\nu$, that both $T(v) = U(v) \cdot f'(v \parallel 0)$ and $f'(v \parallel 1) = f'(0 \parallel v) \cdot f'(1 \parallel v)$ hold. This latter equality, in light of [SL20, Lem 5.1], implies that $\prod_{v\in\mathcal{B}_\nu} f'(v \parallel 0) = f'(0, 1, \ldots, 1)$, which itself equals 1 whenever $\mathcal{V}$ accepts. Taking the product of the former equality over all $v \in \mathcal{B}_\nu$, we thus conclude immediately that $\prod_{v\in\mathcal{B}_\nu} T(v) = \prod_{v\in\mathcal{B}_\nu} U(v)$. Separately, from the relation $\prod_{v\in\mathcal{B}_\nu} f'(v \parallel 0) = 1$, we conclude that, for each $v \in \mathcal{B}_\nu$, $f'(v \parallel 0)$ is individually nonzero, so that the guarantee $T(v) = U(v) \cdot f'(v \parallel 0)$ in particular implies $T(v) = 0 \iff U(v) = 0$. □

The product check protocol—once fully unrolled—makes just one query each to $[T]$ and $[U]$. Its soundness error is thus that of the zerocheck protocol, when run on $[T']$, together with whatever error arises from $[T]$ $[U]$'s respective implicit query protocols.

Remark 4.15. Were we to remove the conjunct $\bigwedge_{v\in\mathcal{B}_\nu}(T(v) = 0 \iff U(v) = 0)$ from the predicate $\mathsf{Product}_{\iota,\nu}$ above, Protocol 4.13 would cease to be complete. Indeed, upon initiating Protocol 4.13 on polynomials T and U for which, at the point $v^* \in \mathcal{B}_\nu$ let's say, $U(v^*) \neq 0$ and $T(v^*) = 0$ both held—and for which $\prod_{v\in\mathcal{B}_\nu} T(v) = \prod_{v\in\mathcal{B}_\nu} U(v)$ moreover held, let's say (so that $U(v) = 0$ for some $v \in \mathcal{B}_\nu \setminus \{v^*\}$)—$\mathcal{P}$ would find itself unable to generate a passing proof. Indeed, to pass, $\mathcal{P}$ would have to set $f'(v^* \parallel 0) = 0$; this would necessitate, in turn, that $\prod_{v\in\mathcal{B}_\nu} f'(v \parallel 0) = f'(0, 1, \ldots, 1) = 0$. Separately, assuming $U(v^*) = 0$ and $T(v^*) \neq 0$, $\mathcal{P}$ would become unable to select $f'(v^* \parallel 0)$ so as to cause $T(v^*) = U(v^*) \cdot f'(v^* \parallel 0)$ to hold.

Multiset check. We recall the $2 \cdot \mu$-ary *multiset predicate* $\mathsf{Multiset}(\mu)_{\iota,\nu} : (T_0, \ldots, T_{\mu-1}, U_0, \ldots, U_{\mu-1}) \mapsto \{(T_0(v), \ldots, T_{\mu-1}(v)) \mid v \in \mathcal{B}_\nu\} = \{(U_0(v), \ldots, U_{\mu-1}(v)) \mid v \in \mathcal{B}_\nu\}$, where the equality is *of multisets*. HyperPlonk [Che+23, § 3.4] defines a protocol for $\mathsf{Multiset}(\mu)_{\iota,\nu}$ in two steps, first by reducing $\mathsf{Multiset}(1)_{\iota,\nu}$ to $\mathsf{Product}_{\iota,\nu}$, and then by reducing $\mathsf{Multiset}(\mu)_{\iota,\nu}$, for $k > 1$, to $\mathsf{Multiset}(1)_{\iota,\nu}$. Though our treatment is similar to HyperPlonk's, we reproduce the details for self-containedness.

PROTOCOL 4.16 (1-dimensional multiset check [Che+23, § 3.4])**.**
Parameters ι, ν and τ in $\mathbb{N}$, and ν-variate virtual polynomials $[T_0]$ and $[U_0]$ over $\mathcal{T}_\iota$, are fixed.

- $\mathcal{V}$ samples $r \leftarrow \mathcal{T}_\tau$, and sends r to $\mathcal{P}$.
- $\mathcal{P}$ and $\mathcal{V}$ run a product check on the virtual polynomials $[T'] \coloneqq r - [T_0]$ and $[U'] \coloneqq r - [U_0]$.

Theorem 4.17. *Protocol 4.16 securely decides the predicate* $\mathsf{Multiset}(1)_{\iota,\nu}$ *on* $[T_0]$ *and* $[U_0]$.

Proof. We follow [Che+23, Thm. 3.4], with appropriate adaptations. Assuming $\mathsf{Multiset}(1)_{\iota,\nu}(T_0, U_0) = 0$, we argue that $\mathsf{Product}_{\tau,\nu}(T', U') = 1$ holds with negligible probability over the verifier's random coins. Our hypothesis entails directly that the degree-2^ν, univariate polynomials $\widehat{T}(Y) \coloneqq \prod_{v \in \mathcal{B}_\nu}(Y - T_0(v))$ and $\widehat{U}(Y) \coloneqq \prod_{v \in \mathcal{B}_\nu}(Y - U_0(v))$, which we now view as elements of $\mathcal{T}_\tau[Y]$, are unequal. We see that the difference $\widehat{T}(Y) - \widehat{U}(Y)$ is not identically zero, and moreover of degree at most 2^ν; we write $R \subset \mathcal{T}_\tau$ for its roots. If $r \notin R$, then $\prod_{v \in \mathcal{B}_\nu}(r - T_0(v)) \neq \prod_{v \in \mathcal{B}_\nu}(r - U_0(v))$, so that $\mathsf{Product}_{\tau,\nu}(T', U') = 0$ necessarily holds. □

Remark 4.18. We compare our treatment of the product and multiset predicates to HyperPlonk's [Che+23, §§ 3.3–3.4]. HyperPlonk's product protocol [Che+23, § 3.3] purports to securely decide the predicate $(T, U) \mapsto \bigwedge_{v \in \mathcal{B}_\nu} U(v) \neq 0 \wedge \prod_{v \in \mathcal{B}_\nu} \frac{T(v)}{U(v)} = e$, where $e \in \mathcal{T}_\iota$ is a statement. In words, HyperPlonk's *stated* predicate requires that the denominator U be nowhere-vanishing on the cube, as well as that the product, over the cube, of the pointwise quotient between T and U equal e. In actuality, that protocol decides a significantly-more-complicated predicate, as we presently explain. The predicate actually decided by that protocol *allows* U to vanish on the cube, albeit with caveats. Indeed, it requires in this case merely that the numerator T *also* vanish wherever U does, and moreover stipulates that, *if* U vanishes anywhere on the cube, then T and U fulfill a weaker variant of the product relationship whereby, if $e \neq 0$, then T is *nonzero* wherever U is. In simple terms, by setting T and U both equal to 0 at $v^* \in \mathcal{B}_\nu$, say, the prover may cause the verifier to accept for *arbitrary* e (provided, again, that T is *nonzero* wherever U is, a circumstance which the prover can easily arrange). This breaks the security guarantees of [Che+23, § 3.3] as stated. We note that, in this situation, our relation $\prod_{v \in \mathcal{B}_\nu} T(v) = \prod_{v \in \mathcal{B}_\nu} U(v)$ *does* hold, while HyperPlonk's does not; the issue is an illegal "division by 0". In fact, our relation $\mathsf{Product}_{\iota,\nu}$ above is precisely the specialization of the "complicated" relation just described to the case $e \coloneqq 1$ (where significant simplifications emerge). We note that the $k = 1$ multiset check of Protocol 4.16 above—which is identical to [Che+23, § 3.4]—*is* nonetheless still secure, and with a simpler proof of security no less. Indeed, if HyperPlonk's product protocol *actually* decided the stated relation of [Che+23, § 3.3], then its multiset protocol would fail to be perfectly complete.

We now present the protocol for $2 \cdot \mu$-ary multiset check; our treatment of this protocol is identical to HyperPlonk's [Che+23, § 3.4].

PROTOCOL 4.19 (μ-dimensional multiset check [Che+23, § 3.4])**.**
Parameters ι, ν, and τ in $\mathbb{N}$, as well as ν-variate virtual polynomials $[T_0], \ldots, [T_{\mu-1}]$ and $[U_0], \ldots, [U_{\mu-1}]$ over $\mathcal{T}_\iota$, where $\mu > 1$, are fixed.

- $\mathcal{V}$ samples random scalars $r_1, \ldots, r_{\mu-1}$ from $\mathcal{T}_\tau$, and sends them to $\mathcal{P}$.
- $\mathcal{P}$ and $\mathcal{V}$ run a 1-dimensional multiset check on the virtual polynomials $[T'] \coloneqq [T_0] + r_1 \cdot [T_1] + \cdots + r_{\mu-1} \cdot [T_{\mu-1}]$ and $[U'] \coloneqq [U_0] + r_1 \cdot [U_1] + \cdots + r_{\mu-1} \cdot [U_{\mu-1}]$.

Theorem 4.20. *Protocol 4.19 securely decides the predicate* $\mathsf{Multiset}(\mu)_{\iota,\nu}$ *on* $([T_i])_{i=0}^{\mu-1}$ *and* $([U_i])_{i=0}^{\mu-1}$.

Proof. Assuming that $\mathsf{Multiset}(\mu)_{\iota,\nu}(T_0,\dots,T_{\mu-1},U_0,\dots,U_{\mu-1})=0$, we show that $\mathsf{Multiset}(1)_{\iota,\nu}(T',U')=1$ holds with negligible probability over $\mathcal{V}$'s random coins. We follow the proof strategy of [Che+23, Thm. 3.5]. We write $T \coloneqq \{(T_0(v),\dots,T_{\mu-1}(v)) \mid v\in\mathcal{B}_\nu\}$ and $U \coloneqq \{(U_0(v),\dots,U_{\mu-1}(v)) \mid v\in\mathcal{B}_\nu\}$ for the multisets at hand, as well as $\widehat{T} \coloneqq T \setminus U$ and $\widehat{U} \coloneqq U \setminus T$, where we understand all set-differences as *multiset* operations. Since T and U are equally sized as multisets, $\widehat{T}$ and $\widehat{U}$ necessarily also are; moreover, our hypothesis entails precisely that $\widehat{T}$ and $\widehat{U}$ are nonempty. We fix an element $t^*\in\widehat{T}$. We write $R \coloneqq \left\{(1,r_1,\dots,r_{\mu-1}) \,\middle|\, (r_1,\dots,r_{\mu-1})\in\mathcal{T}_\tau^{\mu-1}\right\}$; moreover, for each $r\in R$, we write $\varphi_r : \mathcal{T}_\tau^\mu \to \mathcal{T}_\tau$ for the map $\varphi_r : (a_0,\dots,a_{\mu-1}) \mapsto a_0 + r_1\cdot a_1 + \dots + r_{\mu-1}\cdot a_{\mu-1}$. Finally, for each $u\in\widehat{U}$, we set $R_u \coloneqq \left\{r\in R \,\middle|\, \varphi_r(t^*) \stackrel{?}{=} \varphi_r(u)\right\}$. If the verifier's challenge $r\notin\bigcup_{u\in\widehat{U}} R_u$, then $\mathsf{Multiset}(1)_{\iota,\nu}(T',U')=0$ certainly holds; indeed, in this case, the count of the element $\varphi_r(t^*)$ in the multiset $\{\varphi_r(t)\mid t\in T\}$ necessarily exceeds by at least 1 the count of this element in $\{\varphi_r(u)\mid u\in U\}$, so that $\{\varphi_r(t)\mid t\in T\}\neq\{\varphi_r(u)\mid u\in U\}$. On the other hand, each R_u is precisely the intersection in $\mathcal{T}_\tau^\mu$ between the affine hyperplane R and the normal hyperplane $\{r\in\mathcal{T}_\tau^\mu \mid r\cdot(t^*-u)=0\}$ (which is necessarily non-degenerate, by our choice of t^*). Each R_u is thus a *proper* affine subspace of R, and so covers a proportion consisting of at most $\frac{1}{|\mathcal{T}_\tau|}$ of R's points. The union $\bigcup_{u\in\widehat{U}} R_u$ thus covers at most $|\widehat{U}|\cdot\frac{1}{|\mathcal{T}_\tau|}\leq\frac{2^\nu}{|\mathcal{T}_\tau|}$ among R's points (where $|\widehat{U}|$ here is a *multiset* cardinality). This completes the proof. □

Permutation check. We finally describe a protocol for the predicate $\mathsf{Permutation}(\sigma)_{\iota,\nu} : (T_0,\dots,T_{\mu-1}) \mapsto \bigwedge_{(i,v)\in\{0,\dots,\mu-1\}\times\mathcal{B}_\nu} T_{i'}(v') = T_i(v)$ above; here, as before, we fix a bijection $\sigma : \{0,\dots,\mu-1\}\times\mathcal{B}_\nu \to \{0,\dots,\mu-1\}\times\mathcal{B}_\nu$. Though we follow HyperPlonk [Che+23, § 3.5], our protocol decides a more sophisticated variant of that work's predicate, which, in particular, allows *multiple* inputs, as well as permutations which act *across* these inputs.

Our protocol takes as common input a list $[T_0],\dots,[T_{\mu-1}]$ of virtual polynomials. It also—unlike the protocols already given above—makes use of the *indexer*; specifically, the protocol takes as common input further handles $[s_{\mathrm{id}}]$ and $[s_\sigma]$, which jointly capture the permutation $\sigma : \{0,\dots,\mu-1\}\times\mathcal{B}_\nu \to \{0,\dots,\mu-1\}\times\mathcal{B}_\nu$. We argue first that we may freely assume that $\mu=2^\alpha$ is a power of 2; indeed, we may always extend σ by the identity map, as well as pad the list $[T_0],\dots,[T_{\mu-1}]$ with further virtual polynomials (set to be identically zero, say). Clearly, the padded predicate holds if and only if the unpadded one does.

We fix an arbitrary injection $s : \{0,\dots,\mu-1\}\times\mathcal{B}_\nu \hookrightarrow \mathcal{T}_\tau$ (we assume without further comment that τ is sufficiently large). For each $i\in\{0,\dots,\mu-1\}$, we define mappings $\mathrm{id}_i : \mathcal{B}_\nu\to\mathcal{T}_\tau$ and $\sigma_i : \mathcal{B}_\nu\to\mathcal{T}_\tau$ by setting $\mathrm{id}_i : v\mapsto s(i,v)$ and $\sigma_i : v\mapsto s(\sigma(i,v))$. We finally write $s_{\mathrm{id}} \coloneqq \mathtt{merge}(\mathrm{id}_0,\dots,\mathrm{id}_{\mu-1})$ and $s_\sigma \coloneqq \mathtt{merge}(\sigma_0,\dots,\sigma_{\mu-1})$, following Example 4.10. We stipulate that the indexer output $[s_{\mathrm{id}}]$ and $[s_\sigma]$ *directly* as $\nu+\alpha$-variate handles (though this latter measure is not necessary, it improves efficiency).

PROTOCOL 4.21 (Permutation check [Che+23, § 3.4])**.**
Parameters ι, ν, and τ in $\mathbb{N}$, a bijection $\sigma : \{0,\dots,\mu-1\}\times\mathcal{B}_\nu \to \{0,\dots,\mu-1\}\times\mathcal{B}_\nu$, ν-variate polynomials $[T_0],\dots,[T_{\mu-1}]$, and finally the further handles $[s_{\mathrm{id}}]$ and $[s_\sigma]$ constructed above, are fixed.

- $\mathcal{P}$ and $\mathcal{V}$ construct the virtual polynomial $[T] \coloneqq \mathtt{merge}(T_0,\dots,T_{\mu-1})$.
- $\mathcal{P}$ and $\mathcal{V}$ run a 4-ary multicheck check on the $\nu+\alpha$-variate pairs $([s_{\mathrm{id}}],[T])$ and $([s_\sigma],[T])$.

Theorem 4.22. *Protocol 4.21 securely decides the predicate* $\mathsf{Permutation}(\sigma)_{\iota,\nu}$ *on* $[T_0],\dots,[T_{\mu-1}]$.

Proof. Assuming that $\mathsf{Multiset}(2)_{\iota,\nu}(s_{\mathrm{id}},T,s_\sigma,T)=1$, we show that $\mathsf{Permutation}(\sigma)_{\iota,\nu}(T_0,\dots,T_{\mu-1})=1$ holds with probability 1. We write $\widehat{T}_{\mathrm{id}} \coloneqq \{(s_{\mathrm{id}}(u),T(u)) \mid u\mathcal{B}_{\nu+\alpha}\}$ and $\widehat{T}_\sigma \coloneqq \{(s_\sigma(u),T(u)) \mid u\in\mathcal{B}_{\nu+\alpha}\}$ (both viewed as multisubsets of $\mathcal{T}_\tau^2$). We let $(i,v)\in\{0,\dots,\mu-1\}\times\mathcal{B}_\nu$ be arbitrary, and write $(i',v') \coloneqq \sigma(i,v)$. We note that the multisets $\widehat{T}_{\mathrm{id}}$ and $\widehat{T}_\sigma$ each admit precisely one element whose 0^{th} component equals $s(i',v')$; indeed, these elements are exactly $(s(i',v'),T_{i'}(v'))$ and $(s(i',v'),T_i(v))$, respectively, by construction of s_{id}, s_σ, and T. By the assumed equality of $\widehat{T}_{\mathrm{id}}$ and $\widehat{T}_\sigma$ of multisets, we conclude that $T_{i'}(v')=T_i(v)$. □

4.3 New Virtual Polynomials

We now introduce a handful of *new* virtual polynomial constructions. Each of these constructions—on input a handle, or even a further virtual polynomial—materializes a virtual polynomial, which relates to its input in a specified way.

Packed virtual polynomials. We fix integers ι, κ, τ, and ν in $\mathbb{N}$. We recall from Subsection 2.3 the multilinear $\mathcal{T}_\iota$-basis $(\beta_v)_{v\in\mathcal{B}_\kappa}$ of $\mathcal{T}_{\iota+\kappa}$. We finally fix a vector $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$.

We define the *packing operator* $\mathtt{pack}_\kappa : \mathcal{T}_\iota^{\mathcal{B}_\nu} \to \mathcal{T}_{\iota+\kappa}^{\mathcal{B}_{\nu-\kappa}}$ in the following way:

$$\mathtt{pack}_\kappa(f) \coloneqq \left(\sum_{v\in\mathcal{B}_\kappa} f(v \parallel u)\cdot\beta_v\right)_{u\in\mathcal{B}_{\nu-\kappa}}.$$

Intuitively, $\mathtt{pack}_\kappa$ iteratively processes "chunks" consisting of 2^κ lexicographically adjacent $\mathcal{T}_\iota$-elements; it assembles the constituents of each such chunk into a single $\mathcal{T}_{\iota+\kappa}$-element.

We now record a virtual polynomial materialization of $\mathtt{pack}_\kappa(f)$. For $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$ again as above, we write $\widetilde{f} \in \mathcal{T}_\iota[X_0,\ldots,X_{\nu-1}]^{\preceq 1}$ for the MLE of f; we moreover write $\widetilde{\mathtt{pack}_\kappa}(f) \in \mathcal{T}_{\iota+\kappa}[X_0,\ldots,X_{\nu-\kappa-1}]^{\preceq 1}$ for the MLE of $\mathtt{pack}_\kappa(f)$. We finally note the following explicit expression for $\widetilde{\mathtt{pack}_\kappa}(f)$:

$$\widetilde{\mathtt{pack}_\kappa}(f)(X_0,\ldots,X_{\nu-\kappa-1}) = \sum_{v\in\mathcal{B}_\kappa} \widetilde{f}(v_0,\ldots,v_{\kappa-1},X_0,\ldots,X_{\nu-\kappa-1})\cdot\beta_v,$$

where we destructure $(v_0,\ldots,v_{\kappa-1}) = v$ for each $v \in \mathcal{B}_\kappa$. Indeed, for each $(u_0,\ldots,u_{\nu-\kappa-1}) = u \in \mathcal{B}_{\nu-\kappa}$, $\widetilde{\mathtt{pack}_\kappa}(f)(u_0,\ldots,u_{\nu-\kappa-1}) = \mathtt{pack}_\kappa(f)(u)$ necessarily holds; moreover, the polynomial above is multilinear.

When $\widetilde{f}$ is given as a handle $[f]$, the expression $\widetilde{\mathtt{pack}_\kappa}(f)$ above defines a $\nu-\kappa$-variate virtual polynomial, in the sense of Definition 4.6. In fact, this virtual polynomial moreover admits an evaluation protocol, as we now argue. We fix a query $\mathsf{Query}(r',s')_{\nu-\kappa,\iota+\kappa}$. We note that the evaluation $\widetilde{\mathtt{pack}_\kappa}(f)(r') = \sum_{v\in\mathcal{B}_\kappa}\beta_v\cdot\widetilde{f}(v_0,\ldots,v_{\kappa-1},r'_0,\ldots,r'_{\nu-\kappa-1})$ is itself the sum, over the cube $\mathcal{B}_\kappa$, of the κ-variate polynomial:

$$\widetilde{\mathtt{pack}_\kappa}(f,r')(Y_0,\ldots,Y_{\kappa-1}) \coloneqq \widetilde{f}(Y_0,\ldots,Y_{\kappa-1},r'_0,\ldots,r'_{\nu-\kappa-1})\cdot\widetilde{\beta}(Y_0,\ldots,Y_{\kappa-1}),$$

where we write $\widetilde{\beta} \in \mathcal{T}_{\iota+\kappa}[X_0,\ldots,X_{\kappa-1}]^{\preceq 1}$ for the MLE of $(\beta_u)_{u\in\mathcal{B}_\kappa}$. It thus suffices for the verifier to decide $\mathsf{Sum}(s')_{\nu-\kappa,\tau}$ on $\widetilde{\mathtt{pack}_\kappa}(f,r')$. Using the sumcheck protocol, the verifier may in turn reduce this predicate to $\mathsf{Query}(r,s)_{\nu-\kappa,\tau}$, say, on $\widetilde{\mathtt{pack}_\kappa}(f,r')$. To decide this latter predicate, $\mathcal{V}$ may simply check $\widetilde{\beta}(r)\cdot\widetilde{f}(r \parallel r') \stackrel{?}{=} s$. We assume that κ is sufficiently small that $\mathcal{V}$ may evaluate $\widetilde{\beta}(r)$ itself; on the other hand, $\mathcal{V}$ may ascertain $\widetilde{f}(r \parallel r')$ by means of one query to $[f]$.

Shifted virtual polynomials. We again write $\mathcal{T}_\iota \subset \mathcal{T}_\tau$ for an arbitrary tower subfield, and fix an integer $\nu \in \mathbb{N}$. We recall the identification introduced in Section 2, which, for each $k \in \{0,\ldots,\nu\}$, maps $v \in \mathcal{B}_k$ to $\{v\} \coloneqq \sum_{i=0}^{k-1} 2^i\cdot v_i$.

For each *block size parameter* $b \in \{0,\ldots,\nu\}$ and each *shift offset* $o \in \mathcal{B}_b$, the shift operator, on input $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$, partitions f's index set $\mathcal{B}_\nu$ into b-dimensional subcubes, and then circularly rotates each resulting sub-array by o steps (where we, implicitly, flatten each sub-array lexicographically). We make this precise in the following way. For $b \in \{0,\ldots,\nu\}$ and $o \in \mathcal{B}_b$ and above, we define the *shift mapping* $s_{b,o} : \mathcal{B}_\nu \to \mathcal{B}_\nu$ by declaring, for each input $v = (v_0,\ldots,v_{\nu-1}) \in \mathcal{B}_\nu$, that $s_{b,o}(v) \coloneqq u$, where $u = (u_0,\ldots,u_{\nu-1})$ is such that the most-significant substrings $(u_b,\ldots,u_{\nu-1})$ and $(v_b\ldots,v_{\nu-1})$ agree, and $\{u\} + \{o\} \equiv \{v\} \pmod{2^b}$ moreover holds. We define the *shift operator* $\mathtt{shift}_{b,o} : \mathcal{T}_\iota^{\mathcal{B}_\nu} \to \mathcal{T}_\iota^{\mathcal{B}_\nu}$ by mapping each $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$ to the vector $\mathtt{shift}_{b,o}(f) \coloneqq (f(s_{b,o}(v)))_{v\in\mathcal{B}_\nu}$. We note that, provided that we write down the mappings f and $\mathtt{shift}_{b,o}(f)$ as flattened vectors—that is, using the lexicographic identification $v \mapsto \sum_{i=0}^{\nu-1} v_i\cdot 2^i$—we find that $\mathtt{shift}_{b,o}(f)$ has precisely the effect of circularly rotating each contiguous 2^b-sized block of f downward by $\{o\}$ steps. We sometimes abuse notation, below, by writing $\mathtt{shift}_{b,o}(f)$ and $\mathtt{shift}_{b,\{o\}}(f)$ interchangeably.

We initiate an *arithmetic* characterization of the shift operator, which expresses each shifted vector $\texttt{shift}_{b,o}(f)$ as a virtual polynomial on its input f. In fact, our construction moreover admits a linear-time—that is, a $\Theta(b)$-time—evaluation algorithm, as we explain below. Our approach is inspired by, and generalizes, the "adding 1 in binary" multilinear indicator of Setty, Thaler, and Wahby [STW23a, Sec. 5.1].

We first define the length-b, o-step *shift indicator* function $\texttt{s-ind}_{b,o} : \mathcal{B}_b \times \mathcal{B}_b \to \{0,1\}$, in the following way:

$$\texttt{s-ind}_{b,o}(x,y) \mapsto \begin{cases} 1 & \text{if } \{y\} \stackrel{?}{\equiv} \{x\} + \{o\} \pmod{2^b} \\ 0 & \text{otherwise.} \end{cases}$$

For b and $(o_0, \ldots, o_{b-1}) = o \in \mathcal{B}_b$ again fixed, we realize the shift indicator function $\texttt{s-ind}_{b,o}$ inductively, by means of a *sequence* of functions $\texttt{s-ind}'_{k,o}$ and $\texttt{s-ind}''_{k,o}$, each mapping $\mathcal{B}_k \times \mathcal{B}_k \to \{0,1\}$, for $k \in \{0, \ldots, b\}$. That is, for each $k \in \{0, \ldots, b\}$, on arguments x and y in $\mathcal{B}_k$, we define the function $\texttt{s-ind}'_{k,o}(x,y)$ so as to detect the condition $\{y\} \stackrel{?}{=} \{x\} + \{o\}$, and define $\texttt{s-ind}''_{k,o}(x,y)$ so as to detect the condition $\{y\} \stackrel{?}{=} \{x\} + \{o\} - 2^k$, where, in both expressions, we interpret $o = (o_0, \ldots, o_{k-1})$ as an element of $\mathcal{B}_k$ by truncating its bits. In words, $\texttt{s-ind}'_{k,o}$ detects the condition whereby the k-bit strings x and y differ exactly by the *binary addition* of o's least-significant k bits, and without overflow no less; $\texttt{s-ind}''_{k,o}$ detects the analogous condition, modulo an overflow into the k^{th}-indexed bit position. We finally note that $\texttt{s-ind}_{b,o} \coloneqq \texttt{s-ind}'_{b,o} \vee \texttt{s-ind}''_{b,o}$.

We now supply an inductive—and arithmetically friendly—description of the functions $\texttt{s-ind}'_{k,o}$ and $\texttt{s-ind}''_{k,o}$, for $k \in \{0, \ldots, b\}$. For typographical convenience, we give meaning to expressions of the form $\texttt{s-ind}'_{k-1,o}(x,y)$ and $\texttt{s-ind}''_{k-1,o}(x,y)$, for arguments x and y in $\mathcal{B}_k$—i.e., rather than in the domain of definition $\mathcal{B}_{k-1}$—by stipulating that the functions simply ignore their arguments' respective most-significant (that is, $k-1$-indexed) bits. Finally, below, we understand the expression $x_{k-1} \stackrel{?}{=} o_{k-1} + y_{k-1}$ and its variants *over the integers* (i.e., as integer expressions over the arguments x_{k-1}, y_{k-1}, and o_{k-1} in $\{0,1\} \subset \mathbb{Z}$).

case $k = 0$.
$\texttt{s-ind}'_{0,o} = 1$.
$\texttt{s-ind}''_{0,o} = 0$.

case $k > 0$.

$$\texttt{s-ind}'_{k,o}(x,y) = \begin{cases} \texttt{s-ind}'_{k-1,o}(x,y) & \text{if } x_{k-1} + o_{k-1} \stackrel{?}{=} y_{k-1} \\ \texttt{s-ind}''_{k-1,o}(x,y) & \text{if } x_{k-1} + o_{k-1} + 1 \stackrel{?}{=} y_{k-1} \\ 0 & \text{otherwise.} \end{cases}$$

$$\texttt{s-ind}''_{k,o}(x,y) = \begin{cases} \texttt{s-ind}'_{k-1,o}(x,y) & \text{if } x_{k-1} + o_{k-1} \stackrel{?}{=} y_{k-1} + 2 \\ \texttt{s-ind}''_{k-1,o}(x,y) & \text{if } x_{k-1} + o_{k-1} + 1 \stackrel{?}{=} y_{k-1} + 2 \\ 0 & \text{otherwise.} \end{cases}$$

The correctness of this inductive description may be explicitly checked. We note that certain among the case-expressions above can only hold in particular settings; for example, $x_{k-1} + o_{k-1} \stackrel{?}{=} y_{k-1} + 2$ holds if and only if x_{k-1} and o_{k-1} both equal 1 and y_{k-1} is 0. Finally, we note that $\texttt{s-ind}_{b,o}(x,y) = \texttt{s-ind}'_{b,o}(x,y) + \texttt{s-ind}''_{b,o}(x,y)$ holds for each $(x,y) \in \mathcal{B}_b \times \mathcal{B}_b$.

Exploiting the inductive description just given, we now *arithmetically* characterize the MLEs in $\mathcal{T}_\iota[X_0, \ldots, X_{k-1}, Y_0, \ldots, Y_{k-1}]^{\preceq 1}$ of the shift-indicator functions $\texttt{s-ind}'_{k,o}$ and $\texttt{s-ind}''_{k,o}$.

case $k = 0$.
$\widetilde{\texttt{s-ind}}'_{0,o} = 1$.
$\widetilde{\texttt{s-ind}}''_{0,o} = 0$.

case $k > 0$.

$$\widetilde{\texttt{s-ind}}'_{k,o}(X,Y) = \begin{cases} \widetilde{\texttt{eq}}(X_{k-1}, Y_{k-1}) \cdot \widetilde{\texttt{s-ind}}'_{k-1,o}(X,Y) + (1 - X_{k-1}) \cdot Y_{k-1} \cdot \widetilde{\texttt{s-ind}}''_{k-1,o}(X,Y) & o_{k-1} \stackrel{?}{=} 0. \\ (1 - X_{k-1}) \cdot Y_{k-1} \cdot \widetilde{\texttt{s-ind}}'_{k-1,o}(X,Y) & o_{k-1} \stackrel{?}{=} 1. \end{cases}$$

$$\widetilde{\texttt{s-ind}}''_{k,o}(X,Y) = \begin{cases} X_{k-1}\cdot(1-Y_{k-1})\cdot \widetilde{\texttt{s-ind}}''_{k-1,o}(X,Y) & o_{k-1} \stackrel{?}{=} 0. \\ X_{k-1}\cdot(1-Y_{k-1})\cdot \widetilde{\texttt{s-ind}}'_{k-1,o}(X,Y) + \widetilde{\texttt{eq}}(X_{k-1},Y_{k-1})\cdot \widetilde{\texttt{s-ind}}''_{k-1,o}(X,Y) & o_{k-1} \stackrel{?}{=} 1. \end{cases}$$

Above, we denote by $\texttt{eq} : \mathcal{B}_2 \to \{0,1\}$ the boolean equality function $(X_{k-1}, Y_{k-1}) \mapsto X_{k-1} \stackrel{?}{=} Y_{k-1}$, and by $\widetilde{\texttt{eq}}$ its $\mathcal{T}_\iota$-multilinear extension. We again stipulate that the functions $\widetilde{\texttt{s-ind}}'_{k-1,o}$ and $\widetilde{\texttt{s-ind}}''_{k-1,o}$, upon being fed k-variate arguments, simply ignore these arguments' respective last variables.

Finally, we define $\widetilde{\texttt{s-ind}}_{b,o} \coloneqq \widetilde{\texttt{s-ind}}'_{b,o} + \widetilde{\texttt{s-ind}}''_{b,o}$.

Theorem 4.23. *The polynomial $\widetilde{\texttt{s-ind}}_{b,o} \in \mathcal{T}_\iota[X_0,\ldots,X_{b-1},Y_0,\ldots,Y_{b-1}]$ just given is the MLE of $\texttt{s-ind}_{b,o}$.*

Proof. The function $\widetilde{\texttt{s-ind}}_{b,o}$'s pointwise agreement with $\texttt{s-ind}_{b,o}$ over $\mathcal{B}_b \times \mathcal{B}_b$ is self-evident. Its multilinearity holds by induction; indeed, for each $k \in \{1,\ldots,b\}$, we note that both $\widetilde{\texttt{s-ind}}'_{k,o}$ and $\widetilde{\texttt{s-ind}}''_{k,o}$ are sums of products between some *multilinear* function of X_{k-1} and Y_{k-1} and either $\widetilde{\texttt{s-ind}}'_{k-1,o}$ or $\widetilde{\texttt{s-ind}}''_{k-1,o}$, functions which themselves are—by induction—multilinear in the variables $(X_0,\ldots,X_{k-2},Y_0,\ldots,Y_{k-2})$. Each such product expression is necessarily multilinear in the variables $(X_0,\ldots,X_{k-1},Y_0,\ldots,Y_{k-1})$. □

We finally note that the polynomial $\widetilde{\texttt{s-ind}}_{b,o}$ admits a $\Theta(b)$-sized, layered arithmetic circuit; this circuit's description arises straightforwardly from the function's inductive characterization just given above.

We return to the shift operator $\texttt{shift}_{b,o} : \mathcal{T}_\iota^{\mathcal{B}_\nu} \to \mathcal{T}_\iota^{\mathcal{B}_\nu}$ already introduced. Leveraging the arithmetized shift-indicator functions just treated, we now present an arithmetical description of $\texttt{shift}_{b,o}$. Indeed, for each $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$ and each $v \in \mathcal{B}_\nu$, we have the equality:

$$\texttt{shift}_{b,o}(f)(v) = \sum_{u\in\mathcal{B}_b} f(u_0,\ldots,u_{b-1},v_b,\ldots,v_{\nu-1})\cdot \texttt{s-ind}_{b,o}(u_0,\ldots,u_{b-1},v_0,\ldots,v_{b-1}).$$

Finally, we write $\widetilde{\texttt{shift}}_{b,o}(f) \in \mathcal{T}_\iota[X_0,\ldots,X_{\nu-1}]^{\preceq 1}$ for the MLE of $\texttt{shift}_{b,o}(f)$. We note the explicit expression:

$$\widetilde{\texttt{shift}}_{b,o}(f)(X_0,\ldots,X_{\nu-1}) = \sum_{u\in\mathcal{B}_b} \widetilde{f}(u_0,\ldots,u_{b-1},X_b,\ldots,X_{\nu-1})\cdot \widetilde{\texttt{s-ind}}_{b,o}(u_0,\ldots,u_{b-1},X_0,\ldots,X_{b-1}).$$

Indeed, the polynomial above is clearly multilinear, and agrees pointwise with $\texttt{shift}_{b,o}(f)$ over $\mathcal{B}_\nu$.

When $[f]$ is a handle, the expression $\widetilde{\texttt{shift}}_{b,o}(f)$ defines a virtual polynomial, which we again claim is efficiently evaluable. We fix a query $\mathsf{Query}(r',s')_{\iota,\nu}$. We note that the evaluation $\widetilde{\texttt{shift}}_{b,o}(f)(r')$ is *itself* the sum, over the cube $\mathcal{B}_b$, of the b-variate polynomial:

$$\widetilde{\texttt{shift}}_{b,o}(f,r')(Y_0,\ldots,Y_{b-1}) \coloneqq \widetilde{f}(Y_0,\ldots,Y_{b-1},r'_b,\ldots,r'_{\nu-1})\cdot \widetilde{\texttt{s-ind}}_{b,o}\big(Y_0,\ldots,Y_{b-1},r'_0,\ldots,r'_{b-1}\big).$$

It thus suffices for $\mathcal{V}$ to decide $\mathsf{Sum}(s')_{\tau,b}$ on $\widetilde{\texttt{shift}}_{b,o}(f,r')$; using a sumcheck, $\mathcal{V}$ may in turn may reduce this predicate to $\mathsf{Query}(r,s)_{\iota,b}$ on $\widetilde{\texttt{shift}}_{b,o}(f,r')$, for values $r \in \mathcal{T}_\tau^b$ and $s \in \mathcal{T}_\tau$ derived during the sumcheck. As before, $\mathcal{V}$ may decide this latter predicate itself, by locally evaluating $\widetilde{\texttt{s-ind}}_{b,o}(r_0,\ldots,r_{b-1},r'_0,\ldots,r'_{b-1})$ and querying $f(r_0,\ldots,r_{b-1},r'_b,\ldots,r'_{\nu-1})$.

We will occasionally find reason to insist on the nonexistence or the existence of an overflow. In these cases, respectively, we may simply replace the shift-indicator function $\texttt{s-ind}_{b,o}$ with its simpler analogues $\texttt{s-ind}'_{b,o}$ and $\texttt{s-ind}''_{b,o}$, in the expression for $\texttt{shift}_{b,o}$ above. We write $\texttt{shift}'_{b,o}$ and $\texttt{shift}''_{b,o}$ for the resulting *overflow-free* and *overflow-mandated* shift operators.

Example 4.24. We set $\iota \coloneqq 0$ and $b \coloneqq 5$, and fix $\nu \geq 5$ and $o \coloneqq (o_0,\ldots,o_4) \in \mathcal{B}_5$ arbitrarily. For each vector $f \in \mathcal{T}_0^{\mathcal{B}_\nu}$—which we view as a length-2^ν column of bits, by means of the lexicographic flattening $v \mapsto \sum_{i=0}^{\nu-1} 2^i \cdot v_i$—the operator $\texttt{shift}_{b,o}(f)$ breaks f into 32-elements chunks, and then *circularly rotates* each chunk downwards by $\{o\}$ steps (or equivalently, upward by $32-\{o\}$ steps). On the other hand, the overflow-free shift operator $\texttt{shift}'_{b,o}(f)$ rotates each chunk downwards by $\{o\}$ steps, without rotation; that is, it 0-fills the first $\{o\}$ components of each chunk. Finally, the operator $\texttt{shift}''_{b,o}(f)$ acts by upwardly shifting f by $32-\{o\}$ steps, 0-filling the *bottom* $32-\{o\}$ elements of each chunk.

The saturation operator. We record a final, and very simple, virtual polynomial construction, which we use in our multiplication gadget below (see Subsection 5.3). For $\nu \geq 0$ fixed, and given *block size* and *offset* parameters $b \in \{0, \ldots, \nu\}$ and $o \in \mathcal{B}_b$ as above, the saturation operator, on input $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$, partitions f's index set $\mathcal{B}_\nu$ into b-dimensional subcubes, and "saturates" each resulting block with a single value (i.e., that which f takes at the block's o^{th} position). More precisely, we define the *saturation mapping* $r_{b,o} : \mathcal{B}_\nu \to \mathcal{B}_\nu$ by setting $r_{b,o}(v_0, \ldots, v_{\nu-1}) \coloneqq (o_0, \ldots, o_{b-1}, v_b, \ldots, v_{\nu-1})$, for each $v = (v_0, \ldots, v_{\nu-1}) \in \mathcal{B}_\nu$; finally, we define the *saturation operator* $\mathtt{sat}_{b,o} : \mathcal{T}_\iota^{\mathcal{B}_\nu} \to \mathcal{T}_\iota^{\mathcal{B}_\nu}$ by setting $\mathtt{sat}_{b,o}(f) \coloneqq (f(r_{b,o}(v)))_{v \in \mathcal{B}_\nu}$.

We record a virtual polynomial realization of the saturation operator. Indeed, for $b \in \{0, \ldots, \nu\}$ and $o \in \mathcal{B}_b$ as above, and for each $f \in \mathcal{T}_\iota^{\mathcal{B}_\nu}$ and $v \in \mathcal{B}_\nu$, we have that:

$$\mathtt{sat}_{b,o}(f)(v) = f(o_0, \ldots, o_{b-1}, v_b, \ldots, v_{\nu-1});$$

writing $\widetilde{\mathtt{sat}}_{b,o}(f) \in \mathcal{T}_\iota[X_0, \ldots, X_{\nu-1}]^{\preceq 1}$ for the MLE of $\mathtt{sat}_{b,o}(f)$, we conclude that:

$$\widetilde{\mathtt{sat}}_{b,o}(f)(X_0, \ldots, X_{\nu-1}) = \widetilde{f}(o_0, \ldots, o_{b-1}, X_b, \ldots, X_{\nu-1}).$$

The polynomial above is clearly multilinear, and agrees pointwise with $\mathtt{sat}_{b,o}(f)$ over $\mathcal{B}_\nu$.

When f is a virtual polynomial, $\widetilde{\mathtt{sat}}_{b,o}(f)$ clearly also is, and can be evaluated as efficiently as f can.

4.4 Binary-Field Lasso

In this subsection, we discuss the work *Lasso* of Setty, Thaler and Wahby [STW23b] and adapt that work to our binary tower setting.

We develop a distilled, conceptually minimal approach to Lasso, by teasing apart its various components. Indeed, we contend that Lasso ultimately amounts to a combination of the following components:

- **A virtual polynomial abstraction, which materializes large tables.** Indeed, Lasso's "SOS tables" [STW23b, § 3.2] can be viewed as *composition virtual polynomials* in the sense of Example 4.9 above, which, operating over subtables, virtually materialize large tables.
- **A small-table lookup procedure.** Lasso's core contribution is, arguably, its single-table lookup procedure [STW23a, Claim. 3], a virtual polynomial protocol which—using *offline memory-checking* in the sense of Blum, Evans, Gemmell, Kannan, and Naor [Blu+91]—proves that the values taken over the cube by one virtual polynomial represent a subset of the values taken over the cube by another virtual polynomial. More precisely, Lasso's lookup procedure reduces precisely this predicate to a multiset predicate on certain further virtual polynomials.
- **A multiset-check.** Finally, Lasso employs a virtual protocol which securely decides the multiset predicate [STW23a, Claim. 4], in the sense already developed in Subsections 4.1 and 4.2 above.

Leveraging our abstractions, already developed above, for virtual polynomials and multisets, we thus record a minimal rendition of Lasso, which, in particular, isolates its memory-checking component. Importantly, we excise the table-virtualization process from the jurisdiction of the lookup itself, and subsume it into the constraint-satisfaction apparatus already furnished by the higher-level SNARK (see Section 5 below). This separation of concerns yields a conceptually simpler framework.

Separately, we adapt Lasso to the setting of binary fields. On the one hand, the remark [STW23a, Rem. 2] addresses the small-characteristic setting; it suggests that the memory-checker, as opposed to *additively incrementing* the prover's vector of read-counts, instead multiply them element-wise by a multiplicative generator. We note, however, that this approach is insecure; indeed, by exploiting the degenerate element 0—whose multiplicative orbit is, of course, of size 1—the prover may attack multiplicative Lasso as written. We remedy this issue below, at the cost of requiring that the prover commit to an additional polynomial; indeed, our approach forces the prover to submit an everywhere-nonzero vector of read-counts.

We now record our protocol for the lookup predicate $\mathsf{Lookup}_{\iota,\nu} : (T, U) \mapsto \bigwedge_{v \in \mathcal{B}_\nu} \exists v' \in \mathcal{B}_\nu : U(v) = T(v')$.

PROTOCOL 4.25 (Lasso-based lookup [STW23a])**.**
Parameters ι and ν in $\mathbb{N}$, and ν-variate virtual polynomials $[T]$ and $[U]$ over $\mathcal{T}_\iota$, are fixed.

- $\mathcal{P}$ and $\mathcal{V}$ set $\zeta \geq 0$ minimally so that $|T_\zeta| - 1 > 2^\nu$ holds (equivalently, they set $\zeta := \lceil \log(\nu + 1) \rceil$); $\mathcal{P}$ and $\mathcal{V}$ moreover fix a generator $\alpha \in \mathcal{T}_\zeta^*$ of $\mathcal{T}_\zeta$'s multiplicative group of units $\mathcal{T}_\zeta^*$.
- $\mathcal{P}$ defines arrays R and F in $\mathcal{T}_\zeta^{\mathcal{B}_\nu}$ as follows. $\mathcal{P}$ initializes $F := (1)_{v \in \mathcal{B}_\nu}$, and then executes:

1: **for** each $v \in \mathcal{B}_\nu$, in *any* order, **do**
2: pick an arbitrary $v' \in \mathcal{B}_\nu$ for which $U(v) = T(v')$ holds.
3: assign $R[v] := F[v']$.
4: overwrite $F[v'] \mathrel{\times}= \alpha$.

$\mathcal{P}$ sets $R' := \left(\frac{1}{R(v)}\right)_{v \in \mathcal{B}_\nu}$ to be the pointwise reciprocal of the vector R. $\mathcal{P}$ submits the multilinear extensions $\left(\texttt{submit}, \zeta, \nu, \widetilde{R}\right)$, $\left(\texttt{submit}, \zeta, \nu, \widetilde{R'}\right)$, and $\left(\texttt{submit}, \zeta, \nu, \widetilde{F}\right)$ to the oracle.

- $\mathcal{P}$ and $\mathcal{V}$ run a zerocheck on the ν-variate virtual polynomial $R \cdot R' - 1$ over $\mathcal{T}_\zeta$.
- $\mathcal{P}$ and $\mathcal{V}$ define further ν-variate virtual polynomials as follows. They set $O : (X_0, \ldots, X_{\nu-1}) \mapsto 1$ to be the identically-1 polynomial, and set $W := \alpha \cdot R$. $\mathcal{P}$ and $\mathcal{V}$ finally run a 4-ary multiset check on the $\nu + 1$-variate pairs $(\texttt{merge}(T, U), \texttt{merge}(O, W))$ and $(\texttt{merge}(T, U), \texttt{merge}(F, R))$.

Above, our F corresponds to Lasso's array of "final counts", R correspond to its array of inline read counts, and W corresponds to its array of inline write counts. We refer to [STW23a, Claim 3]. Protocol 4.25's completeness amounts to a more-or-less straightforward, albeit slightly subtle exercise, and essentially follows from [STW23a, Claim 2]. We suggest the following inductive proof. Indeed, assuming that R is *initialized* to the all-zero array $(0)_{v \in \mathcal{B}_\nu}$, we argue that the the relevant equality

$$\{(T(v), 1) \mid v \in \mathcal{B}_\nu\} \cup \{(U(v), \alpha \cdot R(v)) \mid v \in \mathcal{B}_\nu\} = \{(T(v), F(v)) \mid v \in \mathcal{B}_\nu\} \cup \{(U(v), R(v)) \mid v \in \mathcal{B}_\nu\} \quad (1)$$

of multisets is an *algorithmic invariant* of the main loop above (that is, it holds as of the beginning of each iteration). The base case is clear (both multisets at hand equal $\{(T(v), 1) \mid v \in \mathcal{B}_\nu\} \cup \{(U(v), 0) \mid v \in \mathcal{B}_\nu\}$). We fix an iteration index $v \in \mathcal{B}_\nu$ of the above loop. The assignment 3 entails removing $(U(v), 0)$ from *both* multisets, as well as adding $(U(v), \alpha \cdot F[v'])$ to the left multiset and $(U(v), F[v'])$ to the right. On the other hand, the update 4 entails removing $(T(v'), F[v'])$ from the right multiset and adding $(T(v'), \alpha \cdot F[v'])$ to the right multiset. Since $T(v') = U(v)$, these changes balance; assuming that the equality held at the loop's beginning, we conclude that it likewise holds as of the loop's end. This completes the proof of completeness.

Theorem 4.26. *Protocol 4.25 securely decides the lookup predicate on T and U.*

Proof. We adapt Setty, Thaler, and Wahby [STW23a, Claim 3] to the multiplicative setting. Assuming that $\mathsf{Zero}_{\zeta,\nu}(R \cdot R' - 1) = 1$ and $\mathsf{Multiset}(4)_{\tau,\nu+1}(\texttt{merge}(T, U), \texttt{merge}(O, W), \texttt{merge}(T, U), \texttt{merge}(F, R)) = 1$ both hold, we show that $\mathsf{Lookup}_{\iota,\nu}(T, U) = 1$ holds with probability 1. Our assumption $\mathsf{Zero}_{\zeta,\nu}(R \cdot R' - 1) = 1$ immediately implies that, for each $v \in \mathcal{B}_\nu$, $R(v) \cdot R'(v) = 1$, so that $R(v) \neq 0$. Our second assumption is precisely the equality (1) of multisets. From it, we conclude *a fortiori* that

$$\{(U(v), R(v)) \mid v \in \mathcal{B}_\nu\} \subset \{(T(v), 1) \mid v \in \mathcal{B}_\nu\} \cup \{(U(v), \alpha \cdot R(v)) \mid v \in \mathcal{B}_\nu\} \quad (2)$$

as multisets. We suppose, for contradiction, that $v_0 \in \mathcal{B}_\nu$, say, were such that, for each $v' \in \mathcal{B}_\nu$, $U(v_0) \neq T(v')$ held. Since certainly $(U(v_0), R(v_0)) \notin \{(T(v), 1) \mid v \in \mathcal{B}_\nu\}$, by hypothesis on v_0, we conclude from (2) that $(U(v_0), R(v_0)) \in \{(U(v), \alpha \cdot R(v)) \mid v \in \mathcal{B}_\nu\}$, so that $v_1 \in \mathcal{B}_\nu$, say, is such that $(U(v_0), R(v_0)) = (U(v_1), \alpha \cdot R(v_1))$. Since $U(v_1) = U(v_2)$, applying (2) again to the pair $(U(v_1), R(v_1))$, we find as before an element $v_2 \in \mathcal{B}_\nu$, say, for which $(U(v_1), R(v_1)) = (U(v_2), \alpha \cdot R(v_2))$. Proceeding in this way, we obtain a sequence of elements $v_i \in \mathcal{B}_\nu$, for $i \in \{0, \ldots, 2^\nu\}$, for which, for each $i \in \{0, \ldots, 2^\nu - 1\}$, we have $R(v_{i+1}) \cdot \alpha = R(v_i)$.

Since $|\mathcal{B}_\nu| = 2^\nu$, by the pidgeonhole principle, we must have a collision $v_i = v_j$, for unequal indices $i < j$, say, in $\{0, \ldots, 2^\nu\}$. We conclude that $R(v_i) = \alpha^{j-i} \cdot R(v_j) = \alpha^{j-i} \cdot R(v_i)$; using our guarantee whereby $R(v_i) \neq 0$, we finally conclude that $\alpha^{j-i} = 1$. Since $j - i \in \{1, \ldots, 2^\nu\}$, and α's multiplicative order is exactly $|\mathcal{T}_\zeta| - 1 > 2^\nu$, we obtain a contradiction. We conclude that $\mathsf{Lookup}_{\iota,\nu}(T, U) = 1$, as desired. □

5 A SNARK over Binary Tower Fields

We now present a practical SNARK, suitable for general statements over binary tower fields. Its arithmetization scheme—that is, the method by which it algebraically captures general computations—refines the *PLONKish* scheme of Grigg, Bowe, Hopwood and Lai [Gri+22], and in particular its adaptation, due to Chen, Bünz, Boneh and Zhang's *HyperPlonk* [Che+23, Def. 4.1], to the multivariate setting. The PLONKish arithmetization arranges its witness data into a computational trace, called a *trace matrix*, of field elements. The scheme moreover makes use of a plurality of *gate constraints*; these are multivariate polynomials, to be evaluated over certain subsets of the trace matrix.

Our treatment differs from HyperPlonk's primarily in that we do not confine ourselves to a single finite field. Rather, we partition our trace matrix's column set into regions, each in turn corresponding to *different* subfields in the tower. For example, our trace matrix might feature certain columns defined over $\mathbb{F}_2$, others over $\mathbb{F}_{2^8}$, and still others over $\mathbb{F}_{2^{32}}$, say. Our gate constraints, moreover, may express polynomial relations defined over *particular* subfields of the tower. In fact, even a single gate constraint may freely act on columns which themselves belong to *unequally* sized subfields; indeed, each among the tower's subfields embeds unambiguously into each of its larger fields.

We pause to emphasize the utility of the virtual polynomial abstraction constructions in our SNARK. Indeed, the "virtual polynomial-centric" approach we pursue serves to vastly reduce the number of trace columns which the prover must explicitly commit to. That is, instead of requiring that the prover commit to certain auxiliary columns, and only then ensuring that they relate as prescribed to the trace columns, the verifier may instead directly materialize the needed auxiliary columns virtually. The verifier may then, finally, check that the relevant polynomial relations—each defined over a collection consisting of both explicit *and* virtual columns—hold.

5.1 The PLONK relation

We define the indexed relation R_{PLONK} on tuples of the form $(\mathtt{i}, \mathtt{x}; \mathtt{w})$, where the *index* $\mathtt{i}$ captures the public parameters of the constraint system, the *statement* $\mathtt{x}$ represents the circuit's public inputs, and the *witness* $\mathtt{w}$ includes further inputs to the circuit.

The index is defined to be a tuple of the form $\mathtt{i} = (\tau, \nu, \xi, n_\varphi, n_\psi, \iota, a, g, \sigma)$, where:

- $\tau \in \mathbb{N}$ is the height of the maximally-indexed tower step $\mathcal{T}_\tau$ in use,
- $\nu \in \mathbb{N}$ is the base-2 logarithm of the number of trace rows (we require $\nu \leq 2^\tau$),
- $\xi \in \{0, \ldots, \nu\}$ is the base-2 logarithm of the statement length,
- $n_\varphi \in \mathbb{N}$ is the number of *fixed columns*,
- $n_\psi \in \mathbb{N}$ is the number of *witness columns*,
- $\iota : \{0, \ldots, n_\psi - 1\} \to \{0, \ldots, \tau\}$ is a mapping, which assigns to each witness column a tower field index,
- $a \in \left(\mathcal{T}_\tau^{\mathcal{B}_\nu}\right)^{n_\varphi}$ is the array of *fixed columns*,
- $(g_0, \ldots, g_{\mu-1})$ is a list of ν-variate virtual polynomials, each of which operates over $n_\varphi + n_\psi$ handles,
- $\sigma : \{0, \ldots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu \to \{0, \ldots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu$ defines a plurality of global copy constraints.

The statement is $\mathtt{x} \coloneqq x \in \mathcal{T}_\tau^{\mathcal{B}_\xi}$, a vector of input values. The witness is $\mathtt{w} \coloneqq w \in \left(\mathcal{T}_\tau^{\mathcal{B}_\nu}\right)^{n_\psi}$, an array of witness columns.

We record several remarks. We assume throughout that τ is sufficiently large that an injection $s : \{0, \dots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu \hookrightarrow \mathcal{T}_\tau$ exists. Above, we slightly abuse notation by calling the objects $(g_0, \dots, g_{\mu-1})$ "virtual polynomials"; more properly, these are circuits which operates over "placeholder" handles (i.e., handles which don't exist yet). Once the relevant handles become available—i.e., after the indexer and prover commit to the fixed and witness columns, respectively—the verifier may, by "plugging in" the appropriate handles, create from each of these circuits a genuine virtual polynomial. On the other hand, upon being fed real polynomials (as opposed to handles), these virtual polynomials of course become standard polynomials.

For convenience, we write $\mathtt{pad}_\nu(x) \in \mathcal{T}_\tau^{\mathcal{B}_\nu}$ for the zero-extension of the vector $x \in \mathcal{T}_\tau^{\mathcal{B}_\iota}$ to the domain $\mathcal{B}_\nu$; we moreover write $c \in \left(\mathcal{T}_\tau^{\mathcal{B}_\nu}\right)^{n_\varphi + n_\psi}$ for the concatenation of columns $c \coloneqq a \parallel w \parallel \mathtt{pad}_\nu(x)$. The indexed relation $\mathcal{R}_{\mathsf{PLONK}}$ holds, by definition, if and only if:

1. For each $i \in \{0, \dots, \mu - 1\}$, the polynomial $g_i\big(c_0, \dots, c_{n_\varphi + n_\psi - 1}\big)$ is identically zero over $\mathcal{B}_\nu$.
2. For each $(i, v) \in \{0, \dots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu$, it holds that $c_i(v) = c_{i'}(v')$, where we write $(i', v') \coloneqq \sigma(i, v)$.
3. For each $(i, v) \in \{0, \dots, n_\psi - 1\} \times \mathcal{B}_\nu$, it holds that $w_i(v) \in \mathcal{T}_{\iota(i)} \subset \mathcal{T}_\tau$.

These three conditions capture, respectively, the witness's satisfaction of all gate constraints, its satisfaction of all global copy constraints, and finally its satisfaction of all subfield constraints. The first two conditions are standard across PLONKish variants (see e.g. [Che+23, Def. 4.1]); the third condition is new, and pertains specifically to our tower setting. We note that we do *not* isolate so-called *selector columns*, as prior formalizations do (see e.g. [Che+23, Sec. 4.1] and [STW23a, Sec. 2.2]); instead, we subsume these into our fixed columns a.

5.2 Our Protocol

We now present a tower field multilinear polynomial IOP for the indexed relation R_{PLONK}. On the input $\mathtt{i}$, the indexer $\mathcal{I}$, for each $i \in \{0, \dots, n_\varphi - 1\}$, submits $(\mathtt{submit}, \tau, \nu, \widetilde{a_i})$ to the oracle, where $\widetilde{a_i}$ is the MLE of the fixed column $a_i \in \mathcal{T}_\tau^{\mathcal{B}_\nu}$, and receives $(\mathtt{receipt}, \tau, \nu, [a_i])$. Moreover, $\mathcal{I}$ performs the *permutation check*'s setup procedure—already described in detail in advance of Protocol 4.21 above—with respect to the permutation $\sigma : \{0, \dots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu \to \{0, \dots, n_\varphi + n_\psi\} \times \mathcal{B}_\nu$; in this way, $\mathcal{I}$ obtains further handles $[s_{\mathrm{id}}]$ and $[s_\sigma]$. Finally, $\mathcal{I}$ outputs the list of handles $\mathtt{vp} \coloneqq \big([a_0], \dots, [a_{n_\varphi - 1}], [s_{\mathrm{id}}], [s_\sigma]\big)$.

PROTOCOL 5.1 (main polynomial IOP for R_{PLONK}).
On the security parameter λ, and common input $\mathtt{i}$ and $\mathtt{x}$, $\mathcal{P}$ and $\mathcal{V}$ proceed as follows.

- Both $\mathcal{P}$ and $\mathcal{V}$ compute the zero-extension $\mathtt{pad}_\nu(c_\iota)$, as well as its MLE $\widetilde{\mathtt{pad}}_\nu(c_\iota)$.
- For each $i \in \{0, \dots, n_\psi - 1\}$, $\mathcal{P}$ sends $(\mathtt{submit}, \iota(i), \nu, w_i)$ to the polynomial oracle.
- For each $i \in \{0, \dots, n_\psi - 1\}$, upon receiving $(\mathtt{receipt}, \iota_i, \nu, [w_i])$ from the oracle, $\mathcal{V}$ checks $\iota_i \stackrel{?}{=} \iota(i)$.

We abbreviate $\big([c_0], \dots, [c_{n_\varphi + n_\psi - 1}]\big) \coloneqq \big([a_0], \dots, [a_{n_\varphi - 1}], [w_0], \dots, [w_{n_\psi - 1}]\big)$.

- For each $i \in \{0, \dots, \mu - 1\}$, $\mathcal{P}$ and $\mathcal{V}$ zerocheck the virtual polynomial $g_i\big([c_0], \dots, [c_{n_\varphi + n_\psi - 1}]\big)$.
- $\mathcal{P}$ and $\mathcal{V}$ run a permutation check, with statement σ, on the input $\big([c_0], \dots, [c_{n_\varphi + n_\psi}]\big)$.

Theorem 5.2. *Protocol 5.1 securely computes the relation R_{PLONK}.*

Proof. We construct an emulator $\mathcal{E}$. Our emulator $\mathcal{E}$ operates as follows, given access to $\mathcal{A}$, and to $\mathtt{i}$ and $\mathtt{x}$:

1. $\mathcal{E}$ independently runs $\mathtt{vp} \coloneqq \mathcal{I}(\mathtt{i})$, internally simulating the existence of the polynomial oracle.
2. Using $\mathtt{vp}$, and playing the role of $\mathcal{V}$, $\mathcal{E}$ runs $\mathcal{A}$ internally, and in particular intercepts its submissions $(\mathtt{submit}, \iota_i, \nu, w_i)$ intended for the polynomial oracle. As $\mathcal{V}$ would, $\mathcal{E}$ aborts if, for any index $i \in \{0, \dots, n_\psi - 1\}$, either $\mathcal{A}$ fails to submit the expected witness w_i or the tower height $\iota_i \neq \iota(i)$ is wrong.

3. $\mathcal{E}$ continues to simulate the role of $\mathcal{V}$ internally to $\mathcal{A}$, during the course of the virtual protocols prescribed by Protocol 5.1. If, at any point, the verifier $\mathcal{V}$ in $\mathcal{E}$'s head aborts, then $\mathcal{E}$ does too (i.e., it outputs $\perp$).

4. $\mathcal{E}$ outputs $\mathbb{w} \coloneqq \left(w_0, \ldots, w_{n_\psi - 1}\right)$ and terminates.

We argue that $\mathcal{E}$ fulfills the requirements of Definition 4.3 with respect to the relation R_{PLONK}. We note first that $\mathcal{E}$ outputs $\mathbb{w}$ in step 4 above with *the same* probability with which $\mathcal{V}$ accepts. It follows that the relevant discrepancy $|\Pr[\langle \mathcal{A}(\mathbb{i}, \mathbb{x}), \mathcal{V}(\mathsf{vp}, \mathbb{x})\rangle = 1] - \Pr[R(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 1]|$ is equal to the probability with which $\mathcal{A}$ submits a well-formed witness $\mathbb{w}$ for which $R(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 0$ holds *and* $\mathcal{V}$ nonetheless accepts. This latter probability is negligible precisely in virtue of the security—in the sense of Definition 4.7—of the *zerocheck* and *permutation check* virtual protocols which $\mathcal{V}$ runs with $\mathcal{A}$ in Protocol 5.1. □

5.3 Gadgets

In this subsection, we record composable gadgets for various key operations, including (arithmetic) addition and multiplication. In our setting, a "gadget" is a special sort of virtual polynomial protocol in which the predicate at hand may apply not just to input columns—that is, to polynomials known to both parties before the protocol begins—but moreover to further virtual polynomials which arise during the protocol. Informally, a gadget is a virtual protocol in which, if the verifier doesn't abort, the parties output a *further* virtual polynomial, which necessarily relates to the protocol's inputs in a prescribed way. This slight relaxation of Definition 4.7 doesn't change the spirit of that definition.

Addition. We record a simple gadget for the *arithmetic* addition of unsigned integers. Our construction, informally, captures the raw relationship at the level of *bits*, using a few simple $\mathbb{F}_2$-constraints, as well as a shift (see Subsection 4.3); it then uses the packing operator to materialize the relevant bit-columns into virtual columns of blocks.

We fix a column size $\nu \geq 0$ and a bit-width $b \in \{0, \ldots, \nu\}$. On inputs X, Y, and Z in $\mathcal{T}_0^{\mathcal{B}_\nu}$, the *addition predicate* sends $\mathsf{Add}_{\nu,b} : (X, Y, Z) \mapsto \bigwedge_{v \in \mathcal{B}_{\nu-b}} \{\mathtt{pack}_b(X)(v)\} + \{\mathtt{pack}_b(Y)(v)\} \equiv \{\mathtt{pack}_b(Z)(v)\} \pmod{2^{2^b}}$. That is, the addition predicate requires that, for each $v \in \mathcal{B}_{\nu-b}$, the elements $\mathtt{pack}_b(X)(v)$, $\mathtt{pack}_b(Y)(v)$, and $\mathtt{pack}_b(Z)(v)$ of $\mathcal{T}_b$ respectively have monomial basis representations $x = (x_0, \ldots, x_{2^b-1})$, $y = (z_0, \ldots, z_{2^b-1})$, and $z = (z_0, \ldots, z_{2^b-1})$ for which $\{x\} + \{y\} \equiv \{z\} \pmod{2^{2^b}}$ holds (as usual, we write $\{v\} \coloneqq \sum_{i=0}^{2^b-1} 2^i \cdot v_i$).

PROTOCOL 5.3 (Addition gadget)**.**
Parameters $\nu \in \mathbb{N}$ and $b \in \{0, \ldots, \nu\}$ and ν-variate virtual polynomials $[X]$ and $[Y]$ over $\mathcal{T}_0$ are fixed.

- By performing $2^{\nu-b}$ independent 2^b-bit ripple-carry additions, $\mathcal{P}$ obtains the vector of carry-outs $c_{\text{in}} \in \mathcal{T}_0^{\mathcal{B}_\nu}$. $\mathcal{P}$ submits $(\mathtt{submit}, 0, \nu, \widetilde{c_{\text{in}}})$ to the oracle.
- $\mathcal{P}$ and $\mathcal{V}$ define ν-variate virtual polynomials $c_{\text{in}} \coloneqq \mathtt{shift}'_{b,1}(c_{\text{out}})$ and $Z \coloneqq X + Y + c_{\text{in}}$ over $\mathcal{T}_0$.
- $\mathcal{P}$ and $\mathcal{V}$ zerocheck the ν-variate virtual polynomial $X \cdot Y + X \cdot c_{\text{in}} + Y \cdot c_{\text{in}} - c_{\text{out}}$ over $\mathcal{T}_0$.
- $\mathcal{P}$ and $\mathcal{V}$ output Z.

Theorem 5.4. *Protocol 5.3 securely decides the predicate* $\mathsf{Add}_{\nu,b}$ *on* $[X]$, $[Y]$ *and* $[Z]$.

Proof. Indeed, assuming that $\mathsf{Zero}_{0,\nu}(X \cdot Y + X \cdot c_{\text{in}} + Y \cdot c_{\text{in}} - c_{\text{out}}) = 1$, we show that $\mathsf{Add}_{\nu,b}(X, Y, Z)$ holds with probability 1. Protocol 5.3 captures the action of a ripple-carry adder on each 2^b-bit chunk of the inputs X and Y. Indeed, our hypothesis entails exactly that the $\mathbb{F}_2$-identity $X \cdot Y + X \cdot c_{\text{in}} + Y \cdot c_{\text{in}} = c_{\text{out}}$ holds identically over $\mathcal{B}_\nu$. This shows that, logically, $c_{\text{out}} = X \wedge Y \vee X \wedge c_{\text{in}} \vee Y \wedge c_{\text{in}}$ holds identically over $\mathcal{B}_\nu$, so that c_{out} relates as required to X, Y and c_{in}. On the other hand, the relationship between c_{in} and c_{out} is correct, by definition of $\mathtt{shift}'_{b,1}$. We conclude that $Z \coloneqq X + Y + c_{\text{in}}$ has the required property. □

Multiplication. We now describe a gadget which captures unsigned integer multiplication. For each $\nu \geq 0$ and $b \in \{0, \ldots, \nu\}$, we define the *multiplication predicate* $\mathsf{Mult}_{\nu,b}(X, Y, Z) \mapsto \bigwedge_{v \in \mathcal{B}_{\nu-b}} \{\mathtt{pack}_b(X)(v)\} \cdot \{\mathtt{pack}_b(Y)(v)\} \equiv \{\mathtt{pack}_b(Z)(v)\} \pmod{2^{2^b}}$.

Informally, our multiplication gadget executes the schoolbook algorithm on a-bit words, where a, a tunable parameter, controls the size of a certain lookup table. We check the relevant word-by-word multiplications using lookups; the remaining work amounts to appropriately combining the results of the various word-wise multiplications.

We fix a *lookup table size parameter* $a \in \{0, \dots, b-1\}$. We recall the multilinear $\mathcal{T}_a$-basis $1, X_a, X_{a+1}, X_a \cdot X_{a+1}$ of T_{a+2}. We define the *multiplication lookup table* $T \in \mathcal{T}_{a+2}^{\mathcal{B}_{2^a} \times \mathcal{B}_{2^a}}$ as follows:

$$T : (x, y) \mapsto x \cdot 1 + y \cdot X_a + z_0(x, y) \cdot X_{a+1} + z_1(x, y) \cdot X_a \cdot X_{a+1}.$$

Above, we first identify the $\mathcal{B}_{2^a}$-elements x and y with $\mathcal{T}_a$-elements, by means of the multilinear $\mathbb{F}_2$-basis of $\mathcal{T}_a$. Moreover, we write $z_0(x, y)$ and $z_1(x, y)$ for the unique $\mathcal{T}_a$-elements for which $\{z_0(x, y)\} + 2^{2^a} \cdot \{z_1(x, y)\} = \{x\} \cdot \{y\}$ holds; here, the right-hand quantity is a simple product of integers. In words, $z_0(x, y)$ and $z_1(x, y)$, on the level of bits, respectively give the lower and upper halves of the $2 \cdot 2^a$-bit integer product $\{x\} \cdot \{y\}$. Informally, the lookup table T takes, as its values over the $2^a + 2^a$-dimensional hypercube, precisely the concatenations $x \parallel y \parallel x \cdot y$ of the "legal" multiplication triples (this concatenation takes place in T_{a+2}).

We how have the following virtual polynomial protocol:

PROTOCOL 5.5 (Multiplication gadget)**.**
Parameters $\nu \in \mathbb{N}$, $b \in \{0, \dots, \nu\}$, and $a \in \{0, \dots, b-1\}$, as above, the lookup table $T \in \mathcal{T}_{a+2}^{\mathcal{B}_{2^{a+1}}}$, and finally ν-variate virtual polynomials $[X]$ and $[Y]$ over $\mathcal{T}_0$ are fixed.

- $\mathcal{P}$ and $\mathcal{V}$ initialize the identically-zero ν-variate virtual column $[Z]$ over $\mathcal{T}_0$.
- For each $u \in \mathcal{B}_{b-a}$, $\mathcal{P}$ and $\mathcal{V}$ proceed as follows:
  - $\mathcal{P}$ and $\mathcal{V}$ define $X_u \coloneqq \mathtt{shift}'_{b-a,u}(\mathtt{pack}_a(X))$ and $Y_u \coloneqq \mathtt{sat}_{b-a,u}(\mathtt{pack}_a(Y))$.
  - $\mathcal{P}$ constructs the array $\mathbf{cross}_u \coloneqq (z_0(X_u(v), Y_u(v)) + z_1(X_u(v), Y_u(v)) \cdot X_a)_{v \in \mathcal{B}_{\nu-a}}$ in $\mathcal{T}_{a+1}^{\mathcal{B}_{\nu-a}}$, where z_0 and z_1 are as above, and submits $(\mathtt{submit}, a+1, \nu-a, \widetilde{\mathbf{cross}}_u)$ to the oracle.
  - $\mathcal{P}$ and $\mathcal{V}$ perform a lookup on $U_u \coloneqq X_u \cdot 1 + Y_u \cdot X_a + \mathbf{cross}_u \cdot X_{a+1}$ against T.
  - For each parity bit $j \in \{0, 1\}$, $\mathcal{P}$ defines $\mathbf{aux}_{u,j} \in \mathcal{T}_0^{\mathcal{B}_\nu}$ by concatenating the bits of the $2^{\nu-a-1}$ $\mathcal{T}_{a+1}$-elements of the substring $(\mathbf{cross}_u(j, v_1, \dots, v_{\nu-a-1}))_{v \in \mathcal{B}_{\nu-a-1}}$. $\mathcal{P}$ submits $(\mathtt{submit}, 0, \nu, \widetilde{\mathbf{aux}}_0)$ and $(\mathtt{submit}, 0, \nu, \widetilde{\mathbf{aux}}_1)$ to the oracle. $\mathcal{P}$ and $\mathcal{V}$ run a zerocheck on the $\nu - a$-variate polynomial $\mathtt{merge}\big(\mathtt{pack}_{a+1}(\mathbf{aux}_{u,0}), \mathtt{pack}_{a+1}(\mathbf{aux}_{u,1})\big) - \mathbf{cross}_u$ over $\mathcal{T}_{a+1}$.
  - By running the addition gadget twice, each time with block parameter b, $\mathcal{P}$ and $\mathcal{V}$ update $Z \mathrel{+}= \mathbf{aux}_{u,0} + \mathtt{shift}'_{b,2^a}(\mathbf{aux}_{u,1})$.
- $\mathcal{P}$ and $\mathcal{V}$ output the virtual polynomial $[Z]$.

In the last line of the main loop, we slightly abuse notation by writing $\mathtt{shift}'_{b,2^a}$ to signify $\mathtt{shift}'_{b,o}$, where $o \in \mathcal{B}_b$ is chosen so that $\{o\} = 2^a$ holds; that is, we set $o \coloneqq (0, \dots, 0, 1, \underbrace{0 \quad \dots \quad 0}_{a})$.

The completeness of Protocol 5.5 is a straightforward, though delicate, exercise. We note that, for each $u \in \mathcal{B}_{b-a}$, the elements $\mathbf{aux}_0$ and $\mathbf{aux}_1$ of $\mathcal{T}_0^{\mathcal{B}_\nu}$ are defined precisely so that $\mathtt{pack}_{a+1}(\mathbf{aux}_{u,0})$ and $\mathtt{pack}_{a+1}(\mathbf{aux}_{u,1})$ respectively yield the *even* and *odd* substrings of $\mathbf{cross}_u$. In other words, the equality $\mathtt{merge}\big(\mathtt{pack}_{a+1}(\mathbf{aux}_{u,0}), \mathtt{pack}_{a+1}(\mathbf{aux}_{u,1})\big) = \mathbf{cross}_u$ ensured during the zerocheck holds essentially by fiat. The completeness of the lookups follows directly from the construction of $\mathbf{cross}_u$.

Theorem 5.6. *Protocol 5.5 securely decides the predicate* $\mathsf{Mult}_{\nu,b}$ *on* $[X]$, $[Y]$ *and* $[Z]$.

Proof. If $\mathcal{V}$ accepts, then $\mathsf{Lookup}(T, U_u) = 1$ holds for each $u \in \mathcal{B}_{b-a}$; in particular, for each $u \in \mathcal{B}_{b-a}$ and each $v \in \mathcal{B}_{\nu-a}$, we have the equality $\{\mathbf{cross}_u(v)\} = \Big\{\mathtt{shift}'_{b-a,u}(\mathtt{pack}_a(X))(v)\Big\} \cdot \{\mathtt{sat}_{b-a,u}(\mathtt{pack}_a(Y))(v)\}$ of unsigned, 2^{a+1}-bit integers. Unwinding the definitions of $\mathtt{shift}'_{b-a,u}$ and $\mathtt{sat}_{b-a,u}$, we conclude further that, for each iteration $u \in \mathcal{B}_{b-a}$ as above and each chunk index $(v_{b-a}, \dots, v_{\nu-a-1})$, the vector

$\mathbf{cross}_u$, restricted to the chunk indexed by $(v_{b-a}, \ldots, v_{\nu-a-1})$, contains exactly the 2^{b-a} double-width cross terms $\{\mathtt{pack}_a(X)(s_0, \ldots, s_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1})\} \cdot \{\mathtt{pack}_a(Y)(u_0, \ldots, u_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1})\}$, where the strings $s \coloneqq (s_0, \ldots, s_{b-a-1})$ are such that $\{s\}$ ranges through the list $\big(0, \ldots, 0, 1, 2, \ldots, 2^{b-a} - 1 - \{u\}\big)$. In words, $(\mathbf{cross}_u(w_0, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a}))_{w \in \mathcal{B}_{b-a}}$ yields precisely the $\{u\}^{\text{th}}$ row of the triangular array associated with the schoolbook multiplication of $(\mathtt{pack}_a(X)(w_0, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1}))_{w \in \mathcal{B}_{b-a}}$ and $(\mathtt{pack}_a(Y)(w_0, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1}))_{w \in \mathcal{B}_{b-a}}$, which we understand 2^{b-a}-limb integers. Each cell of this array, moreover, is double-width—that is, 2^{a+1} bits—and needs to be reduced.

To "fold" the elements of this row, with carries, we use a trick. Working modulo 2^{2^b}, we must add the 2^{b-a} 2^{a+1}-bit elements of each chunk of $\mathbf{cross}_u$, after shifting each successive element moreover by 2^a *further* positions (and truncating the most-significant 2^a bits of the last-indexed element in each chunk). Upon writing each integer in its proper place, we find that the even-indexed components of the chunk—corresponding to those indices $(0, w_1, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1})$, for $(w_1, \ldots, w_{b-a-1}) \in \mathcal{B}_{b-a-1}$—don't overlap; the odd-indexed components $(1, w_1, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1})$ similarly lack overlaps. We thus "lift" both of these respective substrings to bit-vectors, so that we can add them. The bit-vectors $\mathbf{aux}_{u,0}$ and $\mathbf{aux}_{u,1}$, we see, are defined to be the respective lifts to $\mathcal{T}_0^{\mathcal{B}_\nu}$ of the even and odd substrings of $\mathbf{cross}_u$; the verifier's zerocheck ensures that they take exactly this form. It thus remains only to show that, for each block index $(v_{b-a}, \ldots, v_{\nu-a-1})$, we have that $\sum_{w \in \mathcal{B}_{b-a}} 2^{\{w\} \cdot a} \cdot \{\mathbf{cross}_u(w_0, \ldots, w_{b-a-1}, v_{b-a}, \ldots, v_{\nu-a-1})\} \equiv \{\mathtt{pack}_b(\mathbf{aux}_{u,0})(v_{b-a}, \ldots, v_{\nu-a-1})\} + \Big\{\mathtt{pack}_b\Big(\mathtt{shift}'_{b,2^a}(\mathbf{aux}_{u,1})\Big)(v_{b-a}, \ldots, v_{\nu-a-1})\Big\} \pmod{2^{2^b}}$. We observe finally that the two terms on this expression's right-hand correspond precisely to its left-hand sum's even-indexed and odd-indexed subset sums. This completes the proof. □

We finally remark upon the efficiency of Protocol 5.5. Protocol 5.5 entails $O(2^{b-a})$ executions of the addition gadget, as well as 2^{b-a} lookups, each into tables T and U_u sized $2^{2^{a+1}}$ and $2^{\nu-a}$, respectively. The parameter $a \in \{0, \ldots, b-1\}$, we see, mediates a tradeoff between *more lookups* and *more expensive lookups*; those choices of a for which these costs become similar appear to be the best.

Example 5.7. In the case $\nu \coloneqq 20$ and $b \coloneqq 5$, Protocol 5.5 yields a multiplication gadget for 32-bit integers. Setting $a \coloneqq 3$, we obtain a *limb size* of $2^3 = 8$ bits, as well as a lookup table T—of size $2^{2^{3+1}} = 2^{16}$—which contains all possible byte-by-byte products. Finally, each looked-up column U_u is of size $2^{20-3} = 2^{17}$. These sizes thus all-but balance, and seem to be optimal. In this setting, we see that Protocol 5.5 proceeds by performing the schoolbook method, chunk-wise, on pairs of 4-limb integers, using lookups to handle each individual product of bytes.

6 Performance Evaluation

We implemented certain key subroutines of our system in Rust. We focus our performance evaluation here on the polynomial commitment scheme and sumcheck protocol, which together dominate the prover's overall computation. Our software of course implements the required tower field arithmetic primitives. We use the Intel *Galois Field New Instructions* (GFNI) instruction set extension to accelerate the fundamental multiplication and inversion operations. The GFNI extension includes the SIMD instruction `GF2P8MULB`, which multiplies elements of the field $\mathbb{F}_{2^8} \cong \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$. Though this instruction assumes a *monomial*—as opposed to a tower—$\mathbb{F}_2$-basis of $\mathbb{F}_{2^8}$, we convert between these representations using the further instruction `GF2P8AFFINEQB`; that is, by using both instructions together, we obtain a short sequence of CPU instructions which multiplies $\mathbb{F}_{2^8}$-elements expressed in coordinates with respect to the tower basis. For those towers $\mathcal{T}_\iota$ for which $\iota > 3$, we use the binary, recursive Karatsuba approach discussed above (see Subsection 2.3), though we now terminate its recursion at the base case $\mathbb{F}_{2^8}$.

We benchmark our polynomial commitment scheme's performance on polynomials over 1-bit, 8-bit, 32-bit, and 64-bit binary fields. The schemes of Section 3 allow the use of any collision-resistant hash function; we instantiate them with the Grøstl hash function, which, as discussed in the introduction, is both performant and recursion-friendly. We use the Reed–Solomon code with rate $\frac{1}{2}$. We present benchmarks for both single-threaded execution, in Table 2, and multi-threaded execution, in Table 3, though we note that our code does not yet implement multi-threading in a thoroughly optimized manner.

We compare the performance results of our polynomial commitment scheme to two high-performance software implementations of alternative schemes. We benchmark against the *Lasso* open-source project's[1] implementation of Wahby, et al.'s *Hyrax* [Wah+18], using the BN254 elliptic curve. This library was chosen as a point of comparison because it is built atop the well-known *arkworks* project [con22] and includes further optimizations when committing small-valued field elements. The Lasso project makes efficient use of multithreading in the commitment phase. We also benchmark against *plonky2*[2], which implements FRI-PCS, parameterized with a 100-bit *conjectured* security target. Even though plonky2's FRI-PCS is univariate and the other schemes benchmarked here are multivariate, we find it as useful point of comparison because of its highly-regarded performance characteristics. Like our construction, the FRI-PCS allows a choice of hash function, and so we show performance results for both the Poseidon and Keccak-256 hash functions, noting that Poseidon is slower but often used in practice for its recursion-friendliness. The plonky2 system also implements multithreading, but optimizes for batched polynomial commitments, as opposed to single polynomial commitments. Seeking a more fair comparison, we also show how FRI-PCS performs with batched commitments and opening proofs on a batch of 256 committed polynomials, each with 256-fold fewer coefficents. For the FRI-PCS benchmarks, we observed that, as expected, the time required to commit elements smaller than 64 bits is nearly identical to the time to commit 64-bit elements, and so only show the unified results.

Table 4 shows our performance results for the sumcheck prover. The sumcheck protocol's performance profile depends on the form of the virtual polynomial that the protocol is applied to. We adopt the standard course whereby we benchmark only multivariate polynomials defined as the product of several multilinears. Specifically, we include in the table results corresponding to the products of 2, 3, and 4 multilinears. One sumcheck prover benchmarked below uses the tower field $\mathcal{T}_7 \cong \mathbb{F}_{2^{128}}$. We moreover benchmark the sumcheck prover over the *monomial-basis* binary field $\mathbb{F}_{2^{128}} \cong \mathbb{F}_2[X]/(X^{128} + X^{127} + X^{126} + X^{121} + 1)$. This field—and in particular, its irreducible polynomial—appears in Gueron, Langley and Lindell's *RFC 8452* [GLL19, Sec. 3], and figures in that document's *POLYVAL* polynomial authenticator. This particular field makes available a certain highly optimized multiplication algorithm, which appears, for both the *x86_64* and the *ARM64* instruction sets, in Thomas Pornin's *BearSSL*[3] (these implementations use the `PCLMULQDQ` and `PMULL` instructions, respectively). We note that a prover may explictly convert between the tower field $\mathcal{T}_7$ and the isomorphic monomial-basis field $\mathbb{F}_{2^{128}}$ for the sake computational efficiency *without* necessitating any change to the protocol verification logic. Finally, we compare performance against the implementation of sumcheck over the BN254 scalar field, as implemented in *Lasso*.

We ran all benchmarks on an Amazon Web Services `c7i.16xlarge` compute-optimized cloud instance, with a 4th-generation Intel *Xeon Scalable* ("Sapphire Rapids 8488C") processor, 64 virtual cores, and 128 GiB of RAM.

[1] `https://github.com/a16z/Lasso.git`
[2] `https://github.com/0xPolygonZero/plonky2`
[3] `https://bearssl.org/`

Commitment Scheme	Num. Coefficients	Size (bits)	Commit (s)	Prove (s)	Verify (s)
Hyrax, BN254 G_1	2^{20}	1	0.3328	0.2817	0.02251
		8	0.5550	0.2792	0.02243
		32	1.606	0.2790	0.02261
		64	3.258	0.2795	0.02270
	2^{24}	1	5.240	2.816	0.07090
		8	7.641	2.922	71.232
		32	22.38	2.917	0.07100
		64	40.959	2.894	0.07082
	2^{28}	1	84.65	54.72	0.2378
		8	118.5	55.72	0.2389
		32	287.1	55.73	0.2394
		64	597.7	55.08	0.2392
FRI-PCS, Goldilocks-64, Poseidon	2^{20}	64	3.068	1.217	0.01016
	2^{24}	64	50.11	19.93	0.01358
	2^{28}	64	824.5	326.6	0.001728
FRI-PCS, Goldilocks-64, Keccak-256	2^{20}	64	1.168	0.5371	0.003998
	2^{24}	64	19.89	8.884	0.005193
	2^{28}	64	342.4	149.4	0.006509
Batched FRI-PCS, Goldilocks-64, Poseidon	2^{20}	64	0.3552	0.1438	0.00783
	2^{24}	64	5.752	0.3860	0.01041
	2^{28}	64	97.654	8.228	0.01327
Batched FRI-PCS, Goldilocks-64, Keccak-256	2^{20}	64	0.07485	0.06851	0.002408
	2^{24}	64	1.326	0.1815	0.003438
	2^{28}	64	28.617	8.168	0.004466
Our construction	2^{20}	1	0.006090	0.007672	0.01159
		8	0.04163	0.005218	0.02261
		32	0.1564	0.01547	0.03611
		64	0.2230	0.05640	0.03685
	2^{24}	1	0.04337	0.1180	0.03353
		8	0.3629	0.07936	0.06968
		32	1.470	0.2412	0.1286
		64	2.575	0.8953	0.1370
	2^{28}	1	0.5703	1.856	0.1265
		8	4.934	1.252	0.2722
		32	18.67	3.381	0.1827
		64	37.63	14.28	0.3044

Table 2: Single-threaded comparison of commitment schemes for polynomials with coefficients of varying bit-lengths.

Commitment Scheme	Num. Coefficients	Size (bits)	Commit (s)	Prove (s)	Verify (s)
Hyrax, BN254 G_1	2^{20}	1	0.02535	0.2357	0.02483
		8	0.02835	0.2431	0.02565
		32	0.06567	0.23749	0.002331
		64	0.1271	0.2410	0.02497
	2^{24}	1	0.1904	0.9233	0.07479
		8	0.2405	0.9294	0.07398
		32	0.7117	0.9152	0.07267
		64	1.266	0.9254	0.07140
	2^{28}	1	2.592	4.421	0.2374
		8	3.518	4.323	0.2367
		32	8.619	4.347	0.2367
		64	18.55	4.377	0.2352
FRI-PCS, Goldilocks-64, Poseidon	2^{20}	64	0.1690	0.3110	0.008183
	2^{24}	64	3.495	6.033	0.01128
	2^{28}	64	64.07	107.4	0.01501
FRI-PCS, Goldilocks-64, Keccak-256	2^{20}	64	0.1009	0.2867	0.003521
	2^{24}	64	2.386	5.676	0.004669
	2^{28}	64	44.87	99.40	0.005895
Batched FRI-PCS, Goldilocks-64, Poseidon	2^{20}	64	0.02384	0.01369	0.007848
	2^{24}	64	0.2407	0.1761	0.01039
	2^{28}	64	5.186	7.396	13.438
Batched FRI-PCS, Goldilocks-64, Keccak-256	2^{20}	64	0.004309	0.02017	0.002416
	2^{24}	64	0.04893	0.1724	0.003427
	2^{28}	64	2.545	7.430	0.004574
Our construction	2^{20}	1	0.03715	0.006876	0.03295
		8	0.2172	0.004718	0.1050
		32	0.8271	0.01548	0.1849
		64	1.052	0.05643	0.2231
	2^{24}	1	0.1640	0.1057	0.1181
		8	1.321	0.07364	0.3588
		32	5.283	0.2416	0.6586
		64	6.521	0.8947	0.6761
	2^{28}	1	1.025	1.672	0.4251
		8	8.306	1.149	1.301
		32	13.687	3.827	0.7312
		64	17.56	14.28	0.8522

Table 3: Multi-threaded performance of commitment schemes for polynomials with coefficients of varying bit-lengths.

Field	Num. Variables ℓ	Composition Degree	Time (s)
BN254 $\mathbb{F}_r$	20	2	0.119
		3	0.1401
		4	0.1756
	24	2	1.626
		3	1.908
		4	2.482
	28	2	24.83
		3	46.66
		4	72.01
$\mathcal{T}_7$, tower basis	20	2	0.08228
		3	0.081812
		4	0.1510
	24	2	0.7168
		3	1.3281
		4	1.928
	28	2	11.94
		3	19.99
		4	29.99
$\mathbb{F}_{2^{128}}$, monomial basis	20	2	0.06221
		3	0.08993
		4	0.09911
	24	2	0.6048
		3	0.6979
		4	1.158
	28	2	7.707
		3	12.00
		4	15.98

Table 4: Multi-threaded performance of the sumcheck prover on products of multilinear polynomials.

References

[Ame+23] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: lightweight sublinear arguments without a trusted setup". In: *Designs, Codes and Cryptography* (2023). DOI: 10.1007/s10623-023-01222-8.

[BC14] Daniel J. Bernstein and Tung Chou. "Faster Binary-Field Multiplication and Faster Binary-Field MACs". In: ed. by Antoine Joux and Amr Youssef. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-13051-4. DOI: 10.1007/978-3-319-13051-4_6.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. "Linear-Time Arguments with Sublinear Verification from Tensor Codes". In: *Theory of Cryptography*. Ed. by Rafael Pass and Krzysztof Pietrzak. Cham: Springer International Publishing, 2020, pp. 19–46. ISBN: 978-3-030-64378-2. DOI: 10.1007/978-3-030-64378-2_2.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *International Conference on Theory of Cryptography*. Vol. 9986. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 31–60. ISBN: 978-3-662-53644-5. DOI: 10.1007/978-3-662-53644-5_2.

[BCS97] Wieb Bosma, John Cannon, and Allan Steel. "Lattices of Compatibly Embedded Finite Fields". In: *Journal of Symbolic Computation* 24.3 (1997), pp. 351–369. DOI: https://doi.org/10.1006/jsco.1997.0138. URL: https://www.sciencedirect.com/science/article/pii/S0747717197901383.

[Ben+18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed–Solomon Interactive Oracle Proofs of Proximity". In: *International Colloquium on Automata, Languages, and Programming*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. Leibniz International Proceedings in Informatics. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 14:1–14:17.

[Ben+18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Paper 2018/046. 2018. URL: https://eprint.iacr.org/2018/046.

[Ben+19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. *DEEP-FRI: Sampling Outside the Box Improves Soundness*. Cryptology ePrint Archive, Paper 2019/336. 2019. URL: https://eprint.iacr.org/2019/336.

[Ben+23] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. "Proximity Gaps for Reed–Solomon Codes". In: *Journal of the ACM* 70.5 (Oct. 2023).

[Ber+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *The Keccak reference*. 2011. URL: https://keccak.team/files/Keccak-reference-3.0.pdf.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Cham: Springer International Publishing, 2020, pp. 677–706.

[BGT23] Jeremy Bruestle, Paul Gafni, and RISC Zero Team. *RISC Zero zkVM: Scalable, Transparent Arguments of RISC-V Integrity*. 2023. URL: https://dev.risczero.com/proof-system-in-detail.pdf (visited on 11/12/2023).

[Bla+93] Ian F. Blake, XuHong Gao, Ronald C. Mullin, Scott A. Vanstone, and Tomik Yaghoobian. *Applications of Finite Fields*. Ed. by Alfred J. Menezes. The Springer International Series in Engineering and Computer Science. Springer Science+Business Media, 1993.

[Blu+91] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. "Checking the Correctness of Memories". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1991, pp. 90–99. ISBN: 0818624450. DOI: 10.1109/SFCS.1991.185352.

[Che+18] Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang. *Faster Multiplication for Long Binary Polynomials*. 2018. URL: https://arxiv.org/abs/1708.09746.

[Che+23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *Advances in Cryptology – EUROCRYPT 2023*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2023.

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Lecture Notes in Computer Science. Full version. Cham: Springer International Publishing, 2020, pp. 738–768. ISBN: 978-3-030-45721-1. DOI: 10.1007/978-3-030-45721-1_26.

[Coh92] Stephen D. Cohen. "The Explicit Construction of Irreducible Polynomials Over Finite Fields". In: *Designs, Codes and Cryptography* 2.2 (1992), pp. 169–174.

[con22] arkworks contributors. *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[Cor+22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Fourth. The MIT Press, 2022.

[DP23] Benjamin E. Diamond and Jim Posen. *Proximity Testing with Logarithmic Randomness*. Cryptology ePrint Archive, Paper 2023/630. 2023. URL: https://eprint.iacr.org/2023/630.

[FP97] John L. Fan and Christof Paar. "On efficient inversion in tower fields of characteristic two". In: *Proceedings of IEEE International Symposium on Information Theory*. 1997.

[Gau+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. *Grøstl – a SHA-3 candidate*. 2011. URL: https://www.groestl.info/Groestl.pdf (visited on 11/12/2023).

[GLL19] Shay Gueron, Adam Langley, and Yehuda Lindell. *AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption*. RFC 8452. Apr. 2019.

[Gol+23] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. "Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS". In: *Advances in Cryptology – CRYPTO 2023*. Ed. by Helena Handschuh and Anna Lysyanskaya. Cham: Springer Nature Switzerland, 2023, pp. 193–226. ISBN: 978-3-031-38545-2. DOI: 10.1007/978-3-031-38545-2_7.

[Gra] Markus Grassl. *Bounds on the minimum distance of linear codes and quantum codes*. http://www.codetables.de.

[Gra+19] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. Cryptology ePrint Archive, Paper 2019/458. 2019. URL: https://eprint.iacr.org/2019/458.

[Gri+22] Jack Grigg, Sean Bowe, Daira Hopwood, and Ying Tong Lai. *The halo2 Book*. 2022. URL: https://zcash.github.io/halo2.

[Gur06] Venkatesan Guruswami. *Algorithmic Results in List Decoding*. Vol. 2. Foundations and Trends in Theoretical Computer Science 2. now publishers, 2006.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[Hab22] Ulrich Haböck. *A summary on the FRI low degree test*. Cryptology ePrint Archive, Paper 2022/1216. 2022. URL: https://eprint.iacr.org/2022/1216.

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1992, pp. 723–732.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Advances in Cryptology - ASIACRYPT 2010*. Ed. by Masayuki Abe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 177–194. ISBN: 978-3-642-17373-8.

[LCH14] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. "Novel Polynomial Basis and Its Application to Reed-Solomon Erasure Codes". In: *IEEE 55th Annual Symposium on Foundations of Computer Science*. 2014, pp. 316–325. DOI: 10.1109/FOCS.2014.41.

[Lun+92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39.4 (Oct. 1992), pp. 859–868. DOI: 10.1145/146585.146605. URL: https://doi.org/10.1145/146585.146605.

[Mal+19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. *Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings*. Cryptology ePrint Archive, Paper 2019/099. 2019. URL: https://eprint.iacr.org/2019/099.

[Pol22] Polygon Zero Team. *Plonky2: Fast Recursive Arguments with PLONK and FRI*. GitHub. 2022. URL: https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *Advances in Cryptology – CRYPTO 2020*. Ed. by Daniele Micciancio and Thomas Ristenpart. Cham: Springer International Publishing, 2020, pp. 704–737. ISBN: 978-3-030-56877-1. DOI: 10.1007/978-3-030-56877-1_25.

[SL20] Srinath Setty and Jonathan Lee. *Quarks: Quadruple-efficient transparent zkSNARKs*. Cryptology ePrint Archive, Paper 2020/1275. 2020. URL: https://eprint.iacr.org/2020/1275.

[Sta21] StarkWare. *ethSTARK Documentation*. Cryptology ePrint Archive, Paper 2021/582. 2021. URL: https://eprint.iacr.org/2021/582.

[STW23a] Srinath Setty, Justin Thaler, and Riad Wahby. *Customizable constraint systems for succinct arguments*. Cryptology ePrint Archive, Paper 2023/552. 2023. URL: https://eprint.iacr.org/2023/552.

[STW23b] Srinath Setty, Justin Thaler, and Riad Wahby. *Unlocking the lookup singularity with Lasso*. Cryptology ePrint Archive, Paper 2023/1216. 2023. URL: https://eprint.iacr.org/2023/1216.

[Tha22] Justin Thaler. *Proofs, Arguments and Zero-Knowledge*. Vol. 4. Foundations and Trends in Privacy and Security 2–4. now publishers, 2022.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Conference on Theory of Cryptography*. New York, USA: Springer-Verlag, 2008, pp. 1–18. ISBN: 354078523X.

[Wah+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat abhi, Justin Thaler, and Michael Walfish. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *IEEE Symposium on Security and Privacy*. 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[Wie88] Doug Wiedemann. "An Iterated Quadratic Extension of $GF(2)$". In: *The Fibonacci Quarterly* 26.4 (1988), pp. 290–295.

A Case Study: Keccak-256 Arithmetization

In this appendix, we supply a *PLONKish* arithmetization (in the sense of Section 5.1 above) for the KECCAK-$f[1600]$ permutation [Ber+11, § 1.2], which resides at the heart of the KECCAK family of sponge functions. The *Keccak-256* hash function in particular represents a core bottleneck facing modern efforts to scale Ethereum using SNARKs. Our arithmetization captures the correct computation of KECCAK-$f[1600]$, and exploits the unique advantages of our tower-field setting.

We recall the full KECCAK-$f[b]$ permutation. We follow the treatment of that algorithm given in [Ber+11, § 1.2], as well as a pseudo-code description given online[4]. The permutation state consists of a b-bit-array $A \in \mathbb{F}_2^{5\times 5\times w}$, where $w \coloneqq 2^\ell$ for some power $\ell \in \{0, \ldots, 6\}$ (below, we in fact fix $\ell \coloneqq 6$, so that $b = 1600$). We understand A throughout as a 5×5 array of *lanes*, or w-bit words (see [Ber+11, Fig. 1.1]), and index into it accordingly. We define addition and multiplication on lanes componentwise; to avoid confusion, we denote using the symbol $*$ the componentwise multiplication of lanes (i.e., the bitwise AND operation). The intermediate value B below takes the same shape as A does; the objects C and D below are one-dimensional, length-5 arrays of lanes. We understand all indices into these arrays modulo 5. We moreover make use of various constants. That is, we have a number n_r of *rounds* (per KECCAK's specification, we set $n_\mathrm{r} \coloneqq 12 + 2 \cdot \ell = 24$); as well as a 5×5 array $r \in \{0, \ldots, w-1\}^{5\times 5}$ of *rotation offsets*, whose derivation is explained in [Ber+11, § 1.2]. We finally have an array RC of *round constants*, whose construction is again detailed in [Ber+11, § 1.2]; for each $i_\mathrm{r} \in \{0, \ldots, n_\mathrm{r} - 1\}$, $\mathrm{RC}[i_\mathrm{r}] \in \mathbb{F}_2^w$ is a single binary word. We reproduce the KECCAK-$f[b]$ permutation in full in Algorithm 1 below.

[4] https://keccak.team/keccak_specs_summary.html

Algorithm 1 (KECCAK-$f[b]$ permutation [Ber+11].)

procedure KECCAK-$f[b](A)$
  for $i_\mathrm{r} \in \{0, \ldots, n_\mathrm{r} - 1\}$ **do**
    for $x \in \{0, \ldots, 4\}$ **do** $C[x] \coloneqq A[x,0] + A[x,1] + A[x,2] + A[x,3] + A[x,4]$. ▷ begin θ step
    for $x \in \{0, \ldots, 4\}$ **do** $D[x] \coloneqq C[x-1] + \mathrm{rot}(C[x+1], 1)$.
    for $(x,y) \in \{0, \ldots, 4\} \times \{0, \ldots, 4\}$ **do** $A[x,y] \mathrel{+}= D[x]$.
    for $(x,y) \in \{0, \ldots, 4\} \times \{0, \ldots, 4\}$ **do** $B[y, 2 \cdot x + 3 \cdot y] \coloneqq \mathrm{rot}(A[x,y], r[x,y])$. ▷ ρ and π steps
    for $(x,y) \in \{0, \ldots, 4\} \times \{0, \ldots, 4\}$ **do** $A[x,y] = (B[x+1,y] + 1) * B[x+2,y]$. ▷ χ step
    $A[0,0] \mathrel{+}= \mathrm{RC}[i_\mathrm{r}]$. ▷ ι step

We build our PLONK constraint system for KECCAK-$f[1600]$ over two tower fields, $\mathcal{T}_0$ and $\mathcal{T}_6$. We take the liberty of defining the constraint system over committed columns of different lengths, linking them by means of a packing argument (see Subsection 4.3). The constraint system has 2^5 rows, so that it can to accommodate $n_\mathrm{r} = 24$ rounds; it uses one row per round. It uses two fixed columns:

- $q_\mathrm{round} \in \mathcal{T}_0[X_0, ..., X_4]^{\preceq 1}$ is the selector for the round computation. It takes the value 1 on those cube points lexicographically indexed $\{0, \ldots, 23\}$ and 0 on the points indexed $\{24, \ldots, 31\}$. This polynomial has a simple description as a multilinear extension over the dimension-5 cube, and so can be efficiently evaluated locally by the verifier (i.e., without the aid of a commitment or an opening proof).
- $\mathbf{RC} \in \mathcal{T}_6[X_0, ..., X_4]^{\preceq 1}$ is the column of 64-bit round constants.

The permutation state is committed in the following group of columns:

- $\mathbf{A} \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5 \times 5}$ captures the 25 lanes of state, as of the beginning of each successive round.
- $\mathbf{C} \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5}$ represents C in the θ step above.
- $\mathbf{D} \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5}$ represents D in the θ step above.
- $\mathbf{A}^\chi \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5 \times 5}$ captures the state of A as of the conclusion of the χ step.

We further define a series of virtual polynomials:

- $\mathbf{A}^\theta \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5 \times 5}$ represents the state of A as of the conclusion of the θ step. For each $x \in \{0, \ldots, 4\}$ and $y \in \{0, \ldots, 4\}$, we define $\mathbf{A}^\theta_{x,y} \coloneqq \mathbf{A}_{x,y} + \mathbf{D}_x$.
- $\mathbf{B} \in \left(\mathcal{T}_0[X_0, ..., X_{10}]^{\preceq 1}\right)^{5 \times 5}$ represents the state of B as of the conclusion of the π and ρ steps. For each $x \in \{0, \ldots, 4\}$ and $y \in \{0, \ldots, 4\}$, we define $\mathbf{B}_{y, 2x+3y} \coloneqq \mathtt{shift}_{6, r(x,y)}(\mathbf{A}^\theta_{x,y})$.

We impose the following gate constraints:

- For each $x \in \{0, ..., 4\}$, $\mathbf{C}_x - \sum_{y=0}^{4} \mathbf{A}_{x,y} = 0$.
- For each $x \in \{0, ..., 4\}$, $\mathbf{C}_{x-1} + \mathtt{shift}_{6,1}(\mathbf{C}_{x+1}) - \mathbf{D}_x = 0$.
- For each $(x,y) \in \{0, ..., 4\} \times \{0, ..., 4\}$, $\mathbf{A}^\chi_{x,y} - ((1 - \mathbf{B}_{x+1,y}) \cdot \mathbf{B}_{x+2,y}) = 0$.
- $q_\mathrm{round} \cdot \left(\mathtt{pack}_6\left(\mathbf{A}^\chi_{0,0}\right) + \mathbf{RC} - \mathtt{shift}''_{5,-1}(\mathtt{pack}_6(\mathbf{A}_{0,0}))\right) = 0$.
- For each $(x,y) \in \{0, ..., 4\} \times \{0, ..., 4\} \setminus \{(0,0)\}$, $q_\mathrm{round} \cdot \left(\mathtt{shift}''_{5,-1}(\mathtt{pack}_6(\mathbf{A}_{x,y})) - \mathtt{pack}_6(\mathbf{A}^\chi_{x,y})\right) = 0$.

Provided that these constraints are fulfilled, we see that the arrays $(\mathtt{pack}_6(A_{x,y})(v))_{(x,y) \in \{0,...,4\} \times \{0,...,4\}}$, for the row-indices $\{v\} = 0$ and $\{v\} = 24$ respectively, are related exactly by the KECCAK-$f[1600]$ permutation.

B Deferred Proofs

Proof of Theorem 3.12. We fix a tuple (c, t_0, t_1, u_0, u_1) for which $\Pi.\mathsf{Open}(\mathsf{params}, c; t_0, s_0)$ and $\Pi.\mathsf{Open}(\mathsf{params}, c; t_1, s_1)$ both hold. Barring a collision in the random oracle, we have that $u_0 = u_1$; we write $(u_i)_{i=0}^{m_0-1}$ for the common hint. By hypothesis, the respective encodings $(\mathsf{Enc}(t_{0,i}))_{i=0}^{m_0-1}$ and $(\mathsf{Enc}(t_{1,i}))_{i=0}^{m_0-1}$ of t_0 and t_1 satisfy $d^{m_0}\Big((\mathsf{Enc}(t_{0,i}))_{i=0}^{m_0-1}, (u_i)_{i=0}^{m_0-1}\Big) < \frac{d}{3}$ and $d^{m_0}\Big((\mathsf{Enc}(t_{1,i}))_{i=0}^{m_0-1}, (u_i)_{i=0}^{m_0-1}\Big) < \frac{d}{3}$. By unique decoding, we conclude that $(\mathsf{Enc}(t_{0,i}))_{i=0}^{m_0-1} = (\mathsf{Enc}(t_{1,i}))_{i=0}^{m_0-1}$. Because Enc is injective, we finally conclude that $t_0 = t_1$ as *packed* matrices; because the natural embedding is injective (see also Theorem 3.9 above), we finally deduce the equality of t_0 and t_1 unpacked matrices, and hence as polynomials in $\mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]$. □

Proof of Theorem 3.13. We define an emulator $\mathcal{E}$ in the following way. Given access to $\mathcal{A}$, and on inputs params, c and $(r_0, \ldots, r_{\ell-1})$, $\mathcal{E}$ operates as follows:

1. $\mathcal{E}$ internally runs $\mathcal{A}$ on the further input $(r_0, \ldots, r_{\ell-1})$ in a straight-line manner, until $\mathcal{A}$ outputs s and π. If $\Pi.\mathsf{Verify}(\mathsf{params}, c, s, (r_0, \ldots, r_{\ell-1}), \pi) = 0$, then $\mathcal{E}$ outputs $(s, \pi; \perp, \perp)$ and terminates.

2. Having already obtained $\mathcal{A}$'s commitment c and observed $\mathcal{A}$'s random oracle queries, $\mathcal{E}$ runs the straight-line extractor of [BCS16, § A.1], and so obtains $u \coloneqq (u_i)_{i=0}^{m_0-1}$, or else outputs $(s, \pi; \perp, \perp)$ if it fails.

3. $\mathcal{E}$ writes $(r_{0,0}, \ldots, r_{0,\ell-1})$ for the randomness it used above and t'_0 for the message sent by $\mathcal{A}$ during the course of its initial proof, and moreover proceeds as follows:

Algorithm 2

```
1: assign i ≔ 1.
2: while i < m_0 do
3:     rewind A to its initial point (i.e., immediately after outputting c).
4:     freshly sample (r_{i,0}, ..., r_{i,ℓ−1}) ← Q(params).
5:     run A again on (r_{i,0}, ..., r_{i,ℓ−1}), with fresh verifier randomness, until it outputs (s, π).
6:     if Π.Verify(params, c, s, (r_{i,0}, ..., r_{i,ℓ−1}), π) then
7:         write t'_i for the message sent by A during its proof.
8:         increment i += 1.
```

   $\mathcal{E}$ checks if the $m_0 \times m_0$ matrix $\Big(\bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{i,j}, r_{i,j})\Big)_{i=0}^{m_0-1}$ is singular. If it is, $\mathcal{E}$ outputs $(s, \pi; \perp, u)$.

4. Otherwise, using the *constant* $\mathcal{T}_\tau$-vector space structure on $A_{\iota,\kappa,\tau}$, $\mathcal{E}$ performs the matrix operation:

$$\begin{bmatrix} — & t_0 & — \\ & \cdots & \\ — & t_{m_0-1} & — \end{bmatrix} \coloneqq \begin{bmatrix} — & \bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{0,j}, r_{0,j}) & — \\ & \cdots & \\ — & \bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{m_0-1,j}, r_{m_0-1,j}) & — \end{bmatrix}^{-1} \cdot \begin{bmatrix} — & t'_0 & — \\ & \cdots & \\ — & t'_{m_0-1} & — \end{bmatrix}.$$

   If the entries of $(t_i)_{i=0}^{m_0-1}$ do *not* reside entirely in the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$, then $\mathcal{E}$ outputs $(s, \pi; \perp, u)$. Otherwise, $\mathcal{E}$ recovers the *unpacked* matrix $(t_i)_{i=0}^{m_0-1}$ by reversing the $\mathcal{T}_\iota$-isomorphism of Theorem 3.9, sets as $t(X_0, \cdots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \cdots, X_{\ell-1}]$ the polynomial whose coefficients (in the multilinear Lagrange basis) are given by the concatenation of $(t_i)_{i=0}^{m_0-1}$'s rows, and outputs $(s, \pi; t, u)$.

We now argue that $\mathcal{E}$ meets the requirements of Definition 3.5. We first argue that $\mathcal{E}$ runs in expected polynomial time in λ. We write ε for the probability that $\mathcal{A}$ passes, *conditioned* on its state as of the point at which it outputs c (this latter probability is taken over the coins of both $\mathcal{Q}$ and $\mathcal{V}$, and over the further coins of $\mathcal{A}$). We note that, for each c, $\mathcal{E}$ enters Algorithm 2 above with probability exactly ε, and, moreover, satisfies the condition of line 6 with probability exactly ε per iteration of that algorithm (since $\mathcal{E}$ simulates the same view to $\mathcal{A}$ during each successive iteration of Algorithm 3 as it does during its initial execution of $\mathcal{A}$). We conclude that $\mathcal{E}$'s total expected runtime is at most $1 + \varepsilon \cdot \frac{m_0-1}{\varepsilon} = m_0$ times the time it takes to run Construction 3.7 once; this total time is thus polynomial in λ (and independent of c and ε).

We now analyze the distribution returned by $\mathcal{E}$. We note that the outputs (c, s, π) upon which $\mathcal{D}$ runs are identically distributed in the real and emulated distributions. It thus suffices to show that it holds in at most negligibly many executions of $\mathcal{A}$, $\mathcal{Q}$ and $\mathcal{E}$ that, simultaneously, $\Pi.\mathsf{Verify}(\mathsf{params}, c, s, (r_0, \ldots, r_{\ell-1}), \pi) = 1$ and *either* $\Pi.\mathsf{Open}(\mathsf{params}, c; t, u) = 0$ or $t(r_0, \ldots, r_{\ell-1}) \neq s$.

We write $Q(\lambda)$ for the number of queries $\mathcal{A}$ makes to the random oracle during one execution. By [BCS16, Lem. 3], it holds with probability at most $\frac{Q(\lambda)^2+1}{2^\lambda}$, which is negligible, that *either* $\mathcal{E}$ fails to extract c in step 2 *or* that $\mathcal{A}$ outputs inconsistent Merkle leaves during its initial proof π. We thus freely assume that $\mathcal{E}$ successfully extracts the opening hint $u \coloneqq (u_i)_{i=0}^{m_0-1}$ in step 2 and that, throughout its initial proof π, $\mathcal{A}$ outputs Merkle leaves $(u_{i,j})_{i=0}^{m_0-1}$ consistent with $(u_i)_{i=0}^{m_0-1}$.

We recall the extension code $\widehat{C} \subset A_{\iota,\kappa,\tau}$ of the code $C \subset \mathcal{T}_{\iota+\kappa}^n$ output by $\Pi.\mathsf{Setup}$ (i.e., see Subsection 3.1). The following lemma shows that we may, moreover, safely restrict our attention to the setting in which the extracted matrix $(u_i)_{i=0}^{m_0-1}$ features correlated agreement with $\widehat{C}$. Though the matrix $(u_i)_{i=0}^{m_0-1}$ has, by definition, entries in the synthetic subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$, for the purposes of the below lemma, we temporarily view it as a matrix with entries in $A_{\iota,\kappa,\tau}$.

Lemma B.1. *If its matrix satisfies* $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) \geq \frac{d}{3}$*, then* $\mathcal{A}$ *passes with negligible probability.*

Proof. We abbreviate $u' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (u_i)_{i=0}^{m_0-1}$. Under the hypothesis of the lemma, Theorem 3.10, applied with the proximity parameter $e \coloneqq \left\lfloor \frac{d-1}{3} \right\rfloor$, implies that, if the second part $(r_{\ell_1}, \ldots, r_{\ell-1}) \in \mathcal{T}_\tau^{\ell_0}$ of the verifier's random point resides *outside* a set of mass at most $2 \cdot \ell_0 \cdot \frac{d}{3 \cdot |\mathcal{T}_\tau|}$ in $\mathcal{T}_\tau^{\ell_0}$, then we have $d\Big(u', \widehat{C}\Big) \geq \frac{d}{3}$. For such $(r_{\ell_1}, \ldots, r_{\ell-1})$, we thus have in particular that $d(u', \mathsf{Enc}(t')) \geq \frac{d}{3}$, since $\mathsf{Enc}(t')$ is a codeword. It follows that $\Pr_{j \leftarrow \{0,\ldots,n-1\}}\left[u'_j = \mathsf{Enc}(t')_j\right] \leq 1 - \frac{d}{3 \cdot n}$. Finally, by our assumption about $\mathcal{A}$'s Merkle paths, we have that $\bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (u_{i,j})_{i=0}^{m_0-1} = u'_j$ for each column $(u_{i,j})_{i=0}^{m_0-1}$ output by $\mathcal{A}$. We see that $\mathcal{A}$'s chance of passing (over the coins of $\mathcal{Q}$ and $\mathcal{V}$) is at most $\ell_0 \cdot \frac{2 \cdot d}{3 \cdot |\mathcal{T}_\tau|} + \left(1 - \frac{d}{3 \cdot n}\right)^\rho$. As $|\mathcal{T}_\tau| \geq 2^{\omega(\log \lambda)}$ holds by construction, and d and ℓ_0 are polynomial in λ, $\ell_0 \cdot \frac{2 \cdot d}{3 \cdot |\mathcal{T}_\tau|}$ is negligible. On the other hand, because $d \in \Omega(n)$ and $\rho \in \Theta(\lambda)$, we likewise have that $\left(1 - \frac{d}{3 \cdot n}\right)^\rho \leq (1 - \Omega(1))^{\Theta(\lambda)}$ is negligible. This completes the proof of the lemma. □

Indeed, the lemma shows that it holds only for a negligible proportion of executions that $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) \geq \frac{d}{3}$ *and* $\mathcal{E}$ proceeds into step 3. We may thus safely ignore these executions. We accordingly assume throughout what follows that $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) < \frac{d}{3}$. In particular, it holds that the respective messages underlying $\mathcal{A}$'s initial matrix $(u_i)_{i=0}^{m_0-1}$ are well-defined; we write $(t_i)_{i=0}^{m_0-1}$ for these messages. The following lemma shows that we may *further* restrict our attention to the case in which $\mathcal{A}$ correctly outputs $t' = \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1}$ during its initial proof.

Lemma B.2. *If its message* $t' \neq \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1}$*, then* $\mathcal{A}$ *passes with negligible probability.*

Proof. We again fix $e \coloneqq \left\lfloor \frac{d-1}{3} \right\rfloor$ and abbreviate $u' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (u_i)_{i=0}^{m_0-1}$; we moreover write $v' \coloneqq \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (\mathsf{Enc}(t_i))_{i=0}^{m_0-1}$. By our assumption above, $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) \leq e$ holds; in particular, $d(u', v') \leq e$. On the other hand, our hypothesis implies that $\mathsf{Enc}(t') \neq v'$. Finally, since $\widehat{C}$ is $\mathcal{T}_\tau$-linear (as discussed above), $v' \in \widehat{C}$ is a codeword. By the reverse triangle inequality, we thus have:

$$d(u', \mathsf{Enc}(t')) \geq |d(\mathsf{Enc}(t'), v') - d(u', v')| \geq d - e.$$

As above, we conclude that $\Pr_{j \leftarrow \{0,\ldots,n-1\}}\left[u'_j = \mathsf{Enc}(t')_j\right] \leq 1 - \frac{d-e}{n}$; moreover, we again have that $\bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (u_{i,j})_{i=0}^{\ell-1} = u'_j$ for each $j \in J$ queried by the verifier. We thus upper-bound $\mathcal{A}$'s probability of passing by $\left(1 - \frac{2 \cdot d}{3 \cdot n}\right)^\rho$. This again completes the proof, in light of the guarantees $d \in \Omega(n)$ and $\rho \in \Theta(\lambda)$. □

The following lemma is specific to our setting, and doesn't appear in [DP23]. In the following lemma, we continue to assume that $(u_i)_{i=0}^{m_0-1}$ has entries in the synthetic subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$, as well as that $d^{m_0}\Big((u_i)_{i=0}^{m_0-1}, \widehat{C}^{m_0}\Big) < \frac{d}{3}$ holds.

Lemma B.3. *The matrix of messages $(t_i)_{i=0}^{m_0-1}$ also has entries in the subring $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$.*

Proof. We suppose for contradiction that some row t_{i^*}, say, where $i^* \in \{0,\ldots,m_0-1\}$, satisfies $t_{i^*,j^*} \notin A_{\iota,\kappa,\iota}$, for some component $j^* \in \{0,\ldots,\frac{m_1}{2^\kappa}-1\}$. We recall the $\mathcal{T}_{\iota+\kappa}$-basis $(\beta_v)_{v\in\mathcal{B}_{\tau-\iota}}$ of $A_{\iota,\kappa,\tau}$ introduced above. Expressing each component of t_{i^*} in coordinates with respect to this basis, we express t_{i^*} as a collection of $2^{\tau-\iota}$ vectors $t_{i^*,v} \in \mathcal{T}_{\iota+\kappa}^{m_1/2^\kappa}$, for $v \in \mathcal{B}_{\tau-\iota}$. Our hypothesis on t_{i^*} entails precisely that at least one *nonzero-indexed* slice—indexed $v^* \in \mathcal{B}_{\tau-\iota} \setminus \{(0,\ldots,0)\}$, say—is *not* identically zero (i.e., as $\mathcal{T}_{\iota+\kappa}^{m_1/2^\kappa}$-element).

Again exploiting the fact that $\widehat{C}$'s generator matrix has entries in $\mathcal{T}_{\iota+\kappa}$, we see that the encoding $\mathsf{Enc}(t_{i^*}) \in A_{\iota,\kappa,\tau}^n$ is precisely given, slice-wise, by the respective slice-encodings $\mathsf{Enc}(t_{i^*,v})$, for $v \in \mathcal{B}_{\tau-\iota}$. Since t_{i^*,v^*} is not identically zero, we conclude that $\mathsf{Enc}(t_{i^*,v^*}) \in \mathcal{T}_{\iota+\kappa}^n$ is necessarily nonzero at *at least* d positions.

Since u_{i^*} is defined over $A_{\iota,\kappa,\iota}$, its $v^{*\text{th}}$ slice u_{i^*,v^*} is identically zero. We conclude that $d(u_{i^*},\mathsf{Enc}(t_{i^*})) \geq d$; this contradicts the inequality $d(u_{i^*},\mathsf{Enc}(t_{i^*})) < \frac{d}{3}$, itself a direct consequence of t_{i^*}'s construction. □

Lemma B.3 shows that, under the hypothesis $d^{m_0}\left((u_i)_{i=0}^{m_0-1},\widehat{C}^{m_0}\right) < \frac{d}{3}$—and assuming, as usual, that $(u_i)_{i=0}^{m_0-1}$ is defined over $A_{\iota,\kappa,\iota} \subset A_{\iota,\kappa,\tau}$—we actually obtain the stronger conclusion $d^{m_0}\left((u_i)_{i=0}^{m_0-1},C^{m_0}\right) < \frac{d}{3}$. Indeed, for each $i \in \{0,\ldots,m_0-1\}$, since $t_i \in A_{\iota,\kappa,\iota}^{m_1/2^\kappa}$ (by Lemma B.3), we conclude that $\mathsf{Enc}(t_i) \in A_{\iota,\kappa,\iota}^n$.

We thus restrict our attention to the case in which $\mathcal{A}$'s initial proof π verifies, $\mathcal{E}$ successfully extracts $(u_i)_{i=0}^{m_0-1}$, and *both* $d^{m_0}\left((u_i)_{i=0}^{m_0-1},C^{m_0}\right) < \frac{d}{3}$ and $t' = \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i,r_i)\cdot(t_i)_{i=0}^{m_0-1}$ hold. We denote:

$$\delta \coloneqq \frac{\ell_0}{|\mathcal{T}_\tau|} + \left(1-\frac{2\cdot d}{3\cdot n}\right)^\rho + \frac{Q(\lambda)^2+1}{2^\lambda}.$$

Since δ is negligible in λ, $\sqrt{\delta}$ also is. In this light, we may simply ignore each execution for which $\mathcal{A}$'s probability of success $\varepsilon \leq \sqrt{\delta}$, since in that case $\mathcal{E}$ proceeds into step 3 in the first place with negligible probability. We thus assume that $\varepsilon > \sqrt{\delta}$ in what follows.

We first show that, with overwhelming probability, as of the conclusion of *each* iteration $i \in \{1,\ldots,m_0-1\}$ of Algorithm 2, we have $t'_i = \bigotimes_{j=\ell_1}^{\ell-1}(1-r_{i,j},r_{i,j})\cdot(t_i)_{i=0}^{m_0-1}$.

Lemma B.4. *The probability that $t'_i \neq \bigotimes_{j=\ell_1}^{\ell-1}(1-r_{i,j},r_{i,j})\cdot(t_i)_{i=0}^{m_0-1}$ for* any *$i \in \{1,\ldots,m_0-1\}$ is negligible.*

Proof. We write V for the event in which $\mathcal{A}$ submits an accepting proof, and E for the event in which $\mathcal{A}$ outputs the correct message $t' = \bigotimes_{i=\ell_1}^{\ell-1}(1-r_i,r_i)\cdot(t_i)_{i=0}^{m_0-1}$. By the argument of Lemma B.2, *if* $\mathcal{A}$ submits Merkle leaves consistent with $(u_i)_{i=0}^{m_0-1}$, then $\mathcal{V}$ accepts with probability at most $\left(1-\frac{2\cdot d}{3\cdot n}\right)^\rho$. By a union bound, we conclude that $\Pr[V \mid \neg E] \leq \frac{Q(\lambda)^2+1}{2^\lambda} + \left(1-\frac{2\cdot d}{3\cdot n}\right)^\rho \leq \delta$. Our assumption $\varepsilon > \sqrt{\delta}$ thus implies:

$$\sqrt{\delta} < \varepsilon = \Pr[V] = \Pr[V\wedge E] + \Pr[V \mid \neg E]\cdot\Pr[\neg E] \leq \Pr[V\wedge E] + \Pr[\neg E]\cdot\delta,$$

so that $\Pr[V\wedge E] > \sqrt{\delta} - \Pr[\neg E]\cdot\delta \geq \sqrt{\delta}-\delta$.

By Bayes' theorem—and using the facts noted above—we conclude that:

$$\Pr[E \mid V] = \frac{\Pr[V\wedge E]}{\Pr[V\wedge E] + \Pr[V \mid \neg E]\cdot\Pr[\neg E]} > \frac{\sqrt{\delta}-\delta}{\sqrt{\delta}-\delta+\delta} = 1-\sqrt{\delta}.$$

We thus see that, under our assumption, the probability that *all* of $\mathcal{E}$'s iterations $i \in \{1,\ldots,m_0-1\}$ satisfy $t'_i = \bigotimes_{j=\ell_1}^{\ell-1}(1-r_{i,j},r_{i,j})\cdot(t_i)_{i=0}^{m_0-1}$ is greater than $\left(1-\sqrt{\delta}\right)^{m_0-1}$, which is overwhelming. Indeed, by a standard binomial approximation, we have that $1-\left(1-\sqrt{\delta}\right)^{m_0-1} \leq (m_0-1)\cdot\sqrt{\delta}$, which is negligible. □

The following lemma is the last remaining step.

Lemma B.5. *The probability that the rows $\left(\bigotimes_{j=\ell_1}^{\ell-1}(1-r_{i,j},r_{i,j})\right)_{i=0}^{m_0-1}$ are linearly dependent is negligible.*

Proof. We fix an arbitrary *proper* $\mathcal{T}_\tau$-linear subspace $A \subset \mathcal{T}_\tau^{m_0}$, and moreover define its preimage $S \coloneqq \left\{(r_{\ell_1}, \ldots, r_{\ell-1}) \in \mathcal{T}_\tau^{\ell_0} \;\middle|\; \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \in A\right\}$ under the tensor map. We first argue that $\mu(S) \leq \frac{\ell_0}{|\mathcal{T}_\tau|}$. It suffices to prove the result only in case A is a hyperplane. We write $a = (a_0, \ldots, a_{m_0-1}) \in \mathcal{T}_\tau^{m_0}$ for a vector of coefficients, *not* all zero, for which $A = \{u \in \mathcal{T}_\tau^{m_0} \mid u \cdot a = 0\}$ holds. By construction, $(r_{\ell_1}, \ldots, r_{\ell-1}) \in S$ if and only if $\bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot a = 0$.

We note that $S \subset \mathcal{T}_\tau^{\ell_0}$ is nothing other than the vanishing locus of that combination of the ℓ_0-variate multilinear Lagrange polynomials given by the coefficient vector a. Because a is not identically zero and these polynomials are linearly independent, we conclude that the combination is itself nonzero. Applying Schwartz–Zippel, we conclude that its vanishing locus $S \subset \mathcal{T}_\tau^{\ell_0}$ is of mass at most $\mu(S) \leq \frac{\ell_0}{|\mathcal{T}_\tau|}$, as desired.

As before, we write V for the event in which $\mathcal{A}$ submits an accepting proof; slightly abusing notation, we denote also by S the event in which $\mathcal{Q}$'s query satisfies $(r_{\ell_1}, \ldots, r_{\ell-1}) \in S$. By the above argument, we have that $\mu(S) = \frac{\ell_0}{|\mathcal{T}_\tau|} \leq \delta$. On the other hand, by our assumption $\varepsilon > \sqrt{\delta}$, we have:

$$\sqrt{\delta} < \varepsilon = \Pr[V] \leq \mu(S) + \Pr[V \mid \neg S] \cdot (1 - \mu(S)),$$

so that $\Pr[V \mid \neg S] \cdot (1 - \mu(S)) > \sqrt{\delta} - \mu(S) \geq \sqrt{\delta} - \delta$.

Again using Bayes' theorem, and using the above facts, we have that:

$$\Pr[\neg S \mid V] = \frac{\Pr[V \mid \neg S] \cdot (1 - \mu(S))}{\Pr[V \mid \neg S] \cdot (1 - \mu(S)) + \Pr[V \mid S] \cdot \mu(S)} > \frac{\sqrt{\delta} - \delta}{\sqrt{\delta} - \delta + \delta} = 1 - \sqrt{\delta}.$$

We now consider the successive vectors $(r_{i,\ell_1}, \ldots, r_{i,\ell-1}) \in \mathcal{T}_\tau^{\ell_0}$ collected by $\mathcal{E}$ over the course of its *successful* executions of the main loop of Algorithm 3. For each $i^* \in \{1, \ldots, m_0 - 1\}$, writing $A \subset \mathcal{T}_\tau^{m_0}$ for the span of the vectors $\left(\bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{i,j}, r_{i,j})\right)_{i=0}^{i^*-1}$, we conclude that, with probability greater than $1 - \sqrt{\delta}$, the vector $(r_{i^*,\ell_1}, \ldots, r_{i^*,\ell-1}) \in \mathcal{T}_\tau^{\ell_0}$ collected during the $i^{*\text{th}}$ iteration satisfies $\bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{i^*,j}, r_{i^*,j}) \notin A$. We see that the rows $\left(\bigotimes_{j=\ell_1}^{\ell-1}(1 - r_{i,j}, r_{i,j})\right)_{i=0}^{m_0-1}$ are independent with probability greater than $\left(1 - \sqrt{\delta}\right)^{m_0-1}$, which is overwhelming, as $1 - \left(1 - \sqrt{\delta}\right)^{m_0-1} \leq (m_0 - 1) \cdot \sqrt{\delta}$ is negligible. This completes the proof of the lemma. □

We finally argue that the values t and $u = (u_i)_{i=0}^{m_0-1}$ extracted by $\mathcal{E}$ satisfy $\Pi.\mathsf{Open}(\mathsf{params}, c; t, u)$ and $t(r_0, \ldots, r_{\ell-1}) = s$. Indeed, under the condition guaranteed by the successful execution of step 2 above, $\mathcal{A}$'s initial matrix $(u_i)_{i=0}^{m_0-1}$ is well-defined, and hashes to c, and moreover is successfully extracted by $\mathcal{E}$. Under the condition guaranteed by Lemma B.1, a matrix $(t_i)_{i=0}^{m_0-1}$ with entries in $A_{\iota,\kappa,\tau}$ for which $d^{m_0}\left((\mathsf{Enc}(t_i))_{i=0}^{m_0-1}, (u_i)_{i=0}^{m_0-1}\right) < \frac{d}{3}$ exists and is unique. Under the conditions guaranteed by Lemmas B.4 and B.5, $\mathcal{E}$ extracts precisely this matrix $(t_i)_{i=0}^{m_0-1}$ in steps 3 and 4. By Lemma B.3, this matrix is *a posteriori* defined over $A_{\iota,\kappa,\iota}$, and—up to reversing Theorem 3.9's $\mathcal{T}_\iota$-isomorphism—defines a polynomial $t(X_0, \ldots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, as required. Finally, Lemma B.2 guarantees that $\mathcal{A}$'s first message satisfies $t' = \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1}$; on the other hand, $\Pi.\mathsf{Verify}(\mathsf{params}, c, s, (r_0, \ldots, r_{\ell-1}), \pi)$ implies that $s = t' \cdot \bigotimes_{i=0}^{\ell_0-1}(1 - r_i, r_i)$. We conclude that $s = \bigotimes_{i=\ell_1}^{\ell-1}(1 - r_i, r_i) \cdot (t_i)_{i=0}^{m_0-1} \cdot \bigotimes_{i=0}^{\ell_0-1}(1 - r_i, r_i) = t(r_0, \ldots, r_{\ell-1})$, as required. This completes the proof of the theorem. □

Proof of Theorem 3.14. We recall first of all the guarantee $\ell = O(\log \lambda)$, which holds by assumption throughout Construction 3.11; we moreover assume that, for $\iota \geq 0$ arbitrary, each $\mathcal{T}_\iota$-multiplication imposes a cost polynomial in the bit-length 2^ι.

We set $\rho \coloneqq \lambda$ and $\tau \coloneqq \left\lceil \log(\log^2(\lambda)) \right\rceil$, as well as as well as $\ell_1 \coloneqq \left\lceil \frac{1}{2} \cdot (\ell + \log \rho) \right\rceil$ and $\ell_0 \coloneqq \ell - \ell_1$, and finally write $m_0 \coloneqq 2^{\ell_0}$ and $m_1 \coloneqq 2^{\ell_1}$. Since the statement is asymptotic and ι is constant, we assume freely that $\tau \geq \iota$. We note that m_0 and m_1 differ by at most a factor of 2 from, respectively, $\frac{1}{\sqrt{\rho}} \cdot \sqrt{2^\ell}$ and $\sqrt{\rho \cdot 2^\ell}$. We fix the *rate* $\gamma \coloneqq \frac{1}{2}$, and moreover set $\kappa \geq 0$ *minimally* so that $|\mathcal{T}_{\iota+\kappa}| = 2^{2^{\iota+\kappa}} \geq \frac{1}{\gamma} \cdot \frac{m_1}{2^\kappa}$ holds (equivalently, so that $2^{\iota+\kappa} + \kappa \geq -\log \gamma + \ell_1$ holds). We note that this minimal κ necessarily satisfies $\kappa \leq \ell_1$. Indeed, the choice $\kappa = \ell_1$ itself certainly satisfies $2^{\iota+\ell_1} + \ell_1 \geq 1 + \ell_1 = -\log \gamma + \ell_1$, as required. We conclude that $2^\kappa \leq m_1$; we finally write $k \coloneqq \frac{m_1}{2^\kappa}$ and $n \coloneqq \frac{1}{\gamma} \cdot k$, and set as $C \subset \mathcal{T}_{\iota+\kappa}^n$ the Reed–Solomon code $\mathsf{RS}_{\mathcal{T}_{\iota+\kappa}}[n, k]$.

We note that, by our choice of κ, this code exists. On the other hand, by the *minimality* of κ, we moreover have the upper-bound $2^{\iota+\kappa} \leq 2 \cdot (-\log(\gamma) + \ell_1 - \kappa) = O(\ell)$. We note that the values ρ, τ, κ, and $C \subset \mathcal{T}_{\iota+\kappa}^n$ chosen in this way fulfill the requirements of $\Pi.\mathsf{Setup}$.

We assume that each bit received imposes constant cost; we see immediately that the protocol's total communication-attendant cost is $2^\tau \cdot m_1 + 2^{\iota+\kappa} \cdot m_0 \cdot \rho = O\Big(\log^2(\lambda) \cdot \sqrt{\rho \cdot 2^\ell} + \ell \cdot \sqrt{\frac{2^\ell}{\rho}} \cdot \rho\Big) = \widetilde{O}(\sqrt{2^\ell})$.

Again by our choice of τ, we see that each $\mathcal{T}_\tau$-operation costs polynomially in $2^\tau = O(\log^2 \lambda)$, and so costs $\widetilde{O}(1)$. We see that the total cost of all $\mathcal{T}_\tau$-operations is $\widetilde{O}(m_0 + m_1) = \widetilde{O}\Big(\sqrt{\lambda \cdot 2^\ell}\Big)$. We analyze the total cost of all $\mathcal{T}_\iota$ operations in the following way. Since $2^\kappa \leq 2^{\iota+\kappa} = O(\ell)$, and since the cost of $2^{\tau-\iota}$ $\mathcal{T}_\iota$-operations is certainly at most that of one $\mathcal{T}_\tau$-operation, we see that the total cost imposed by $2^\kappa \cdot 2^{\tau-\iota}$ $\mathcal{T}_\iota$-operations is at most $O(\ell) \cdot \widetilde{O}(1) = \widetilde{O}(1)$. The total cost of all $\mathcal{T}_\iota$-operations is thus $\widetilde{O}(\rho \cdot m_0) = \widetilde{O}(\sqrt{\rho \cdot 2^\ell}) = \widetilde{O}(\sqrt{\lambda \cdot 2^\ell})$.

Using the upper-bound $2^{\iota+\kappa} = O(\ell) = O(\log \lambda)$, we see that each $\mathcal{T}_{\iota+\kappa}$-operation costs $\widetilde{O}(1)$. On the other hand, the result [LCH14] ensures that the cost of $\mathfrak{Enc}$ is equal to that of $O(n \cdot \log k) = O\Big(\frac{1}{\gamma} \cdot \frac{m_1}{2^\kappa} \cdot \log(\frac{m_1}{2^\kappa})\Big) = \widetilde{O}\Big(\sqrt{2^\ell}\Big)$ $\mathcal{T}_{\iota+\kappa}$-operations, and so is itself $\widetilde{O}\Big(\sqrt{2^\ell}\Big)$. The total cost of all $2^{\tau-\iota}$ encoding operations is thus $O(\log^2 \lambda) \cdot \widetilde{O}\Big(\sqrt{2^\ell}\Big) = \widetilde{O}\Big(\sqrt{2^\ell}\Big)$, as required.

We assume, as Brakedown's analysis does (see [Gol+23, § 1]), that the cost $\mathfrak{Hash}$ of hashing a vector of m_0 $\mathcal{T}_{\iota+\kappa}$-elements is comparable to that of performing m_0 $\mathcal{T}_{\iota+\kappa}$-operations. The total cost of each $\mathfrak{Hash}$ operation is thus $\widetilde{O}\Big(\frac{1}{\sqrt{\rho}} \cdot \sqrt{2^\ell}\Big)$, so that the total cost of all ρ $\mathfrak{Hash}$-operations is $\widetilde{O}(\sqrt{\rho \cdot 2^\ell}) = \widetilde{O}(\sqrt{\lambda \cdot 2^\ell})$. □