Polylogarithmic Proofs for Multilinears over Binary Towers

Benjamin E. Diamond
Irreducible
bdiamond@irreducible.com

Jim Posen
Irreducible
jposen@irreducible.com

Abstract

The use of small fields has come to typify the design of modern, efficient SNARKs. In recent work, Diamond and Posen (EUROCRYPT '25) break a key *trace-length barrier*, by treating multilinear polynomials even over *tiny fields*—fields with fewer elements than the polynomial has coefficients. In this work, we make that advance applicable globally, by generically reducing the problem of tiny-field commitment to that of large-field commitment. We introduce a sumcheck-based technique—called "ring-switching"—which, on input a multilinear polynomial commitment scheme over a large extension field, yields a further scheme over that field's ground field. The resulting tiny-field scheme, like Diamond and Posen's, lacks "embedding overhead", in the sense that its commitment cost is identical to that of the large-field scheme on each input size (measured in *bits*). Its evaluation protocol's overhead is linear for the prover and logarithmic for the verifier, and is essentially optimal.

Instantiating our compiler on the *BaseFold* (CRYPTO '24) large-field multilinear polynomial commitment scheme—or more precisely, on a characteristic-2 adaptation of that scheme, which we develop at length—we obtain an extremely efficient scheme for multilinears over tiny binary fields. Our scheme outperforms Diamond–Posen and Hashcaster ('24) and represents a new state-of-the-art.

1 Introduction

The small-field revolution in SNARK design continues apace. The *ethSTARK* [Sta21] and *Plonky2* [Pol22] systems were the first to decouple their respective *arithmetization* and *cryptographic* fields. Those SNARKs use *small fields*—prime fields sized roughly like a 64-bit register—in their arithmetizations; each, during its security-critical portions, opportunistically passes to a cryptographically large field extension of its arithmetization field. Subsequent production-oriented SNARKs, like *Plonky3* and *RISC Zero*, have embraced similar designs, based on prime fields of just under 32 bits; `Stwo` has adopted a related architecture based on Haböck, Levit and Papini's *Circle STARK* [HLP24]. These techniques have delivered strong prover performance, which surpasses that available in elliptic curve-based SNARKs like *Sonic* [MBKM19], $\mathcal{P}\mathfrak{lon}\mathcal{K}$ [GWC19] and *Marlin* [Chi+20] (which all use the *KZG* [KZG10] univariate polynomial commitment scheme).

These SNARKs all use arithmetization fields which—though relatively "small"—are nonetheless at least as large as the statements they're capable of proving. This fact is not a coincidence. Indeed, all of them operate by, roughly, arranging their witness data into a *trace table* and Reed–Solomon-encoding that table's columns, before ultimately using a low-degree test based on *FRI* [BBHR18a]. Reed–Solomon codes exist only for alphabet–block-length pairs for which the alphabet is at least as large as the block length.

A recent work of Diamond and Posen [DP25] breaks this *trace-length barrier*, in that it treats even polynomials over *tiny fields*—fields smaller than the statement's trace length. Crucially, that work does so without "embedding overhead", a phenomenon we briefly recall. One might trivially commit to a tiny-field polynomial simply by tacitly embedding its coefficients into a sufficiently large field, and then blackbox-applying a standard scheme on the resulting object. That approach would face at least two deficiencies. On the efficiency front, it would induce a cost profile no better than that attached to an input polynomial *without* tiny coefficients. Rather, it would impose an artificial penalty proportional to the difference in size between its input polynomial's tiny coefficient field and the scheme's native field. On the security front, it would fail to guarantee the tininess of the prover's input, a security desideratum which, in practice, turns out to be essential. As Diamond and Posen [DP25] argue, many of today's production-oriented SNARKs suffer from some form of embedding overhead.

In this work, we introduce a *generic* reduction from the problem of tiny-field multilinear polynomial commitment to that of large-field multilinear commitment. Our techniques are rather different from those of Diamond–Posen [DP25]. Our reduction, applied to any large-field scheme, yields a corresponding tiny-field scheme, which moreover lacks embedding overhead. In fact, our reduction in principle applies even to brand-new, concurrently-developed large-field schemes like *Blaze* [Bre+25] and *WHIR* [ACFY24], and even to large-field schemes that haven't been created yet. Its overhead over the underlying large-field scheme is essentially optimal, and beats that associated with alternative constructions (we survey those in detail in Subsection 1.5 below). This work thus solves the problem of tiny-field multilinear polynomial commitment.

1.1 Some Historical Remarks

Diamond and Posen [DP25] break the trace-length barrier by further decoupling two fields which, in each of those small-field schemes cited above, coincide: the *arithmetization field* and the *alphabet field*. All of those schemes use just one prime field—again, sized just under 32 or 64 bits—both as the coefficient field of the polynomials committed and as the alphabet of the Reed–Solomon code used to encode them. The scheme [DP25] makes possible the simultaneous use of a *tiny* arithmetization field and a *small* alphabet field. Separately, that work reintroduces the use of *binary fields*, fields of characteristic 2; these fields have figured in various previous works, like *FRI* [BBHR18a] and *STARK* [BBHR18b]. Finally, that work treats exclusively *multilinear* polynomials; in this capacity, it extends an important line of work which includes *Libra* [Xie+19], *Virgo* [ZXZS20], *Spartan* [Set20], *Brakedown* [Gol+23], and *HyperPlonk* [CBBZ23].

To make their technique work, Diamond and Posen [DP25] tie together these various threads. They introduce a data-casting operation—which they call *packing*, and which is based on field extensions—which serves to recast a witness defined over $\mathbb{F}_2$, say, into a shorter witness over the larger field $\mathbb{F}_{2^{32}}$. They then apply a Brakedown-like multilinear commitment procedure to the resulting witness, whose coefficient field, crucially, is large enough to be used as a Reed–Solomon alphabet. Using various mathematical techniques, those authors manage to make that scheme work (using Brakedown in a naïve, "fire-and-forget" manner on the packed, $\mathbb{F}_{2^{32}}$-witness would lead to information loss). That work, therefore, treats three generally distinct fields at once: the *tiny* coefficient field, the *small* alphabet field, and the *huge* cryptographic field.

We mention a further observation essential to that work. In those small-field schemes above—which are themselves based on the *DEEP-ALI* [BGKS19] paradigm—the Reed–Solomon code plays two separate roles at once. On the one hand, it plays the role of an error-correcting linear block code, a mathematical object which amplifies errors and corruptions and makes them efficiently detectable. On the other hand, it serves the distinct end of polynomial extrapolation. It is essential to those DEEP-ALI-based schemes that the Reed–Solomon codewords that arise within them be, semantically, evaluations of polynomials. In particular, those *constraint* polynomials which, if the prover is honest, must vanish identically over its witnesses must likewise vanish identically over the Reed–Solomon encodings of those witnesses.

As Diamond and Posen [DP25] implicitly observe, unlike DEEP-ALI [BGKS19], Brakedown [Gol+23]—as does its predecessor works Ron-Zewi and Rothblum [RR24] and Bootle, Chiesa and Groth [BCG20]—decouples the coding-theoretic aspects of its code from the semantics of its code. That is, Brakedown's Ligero-inspired [AHIV23] polynomial commitment scheme uses its error-correcting code only for error-amplification; the semantics of that code are irrelevant to it. (That protocol could freely substitute its code with an otherwise-arbitrary code of identical alphabet, message length, block length, and distance, to no effect.) This decoupling makes Diamond and Posen's packing procedure coherent, since that procedure garbles the semantics of Reed–Solomon extrapolation.

On the other hand, most transparent, hash-based proofs which achieve polylogarithmic verifiers—like *Aurora* [Ben+19], or those based on DEEP-ALI [BGKS19]—use univariate *quotienting*. (We refer also to Haböck [Hab22] for a useful survey of these techniques.) As Diamond and Posen [DP25] note, quotienting seems incompatible with their packing technique.

Zeilberger, Chen and Fisch's *BaseFold PCS* [ZCF24, § 5] seems to be the first multilinear polynomial commitment scheme with a polylogarithmic verifier that doesn't use quotienting. That scheme, as written, works only for large fields of odd characteristic. That scheme is simple, elegant, and efficient, and represents a compelling candidate for adaptation to the binary case.

This work's small-field PCS works in a drop-in way with the PIOP of [DP25]; in conjunction with that work's PIOP, our PCS stands to yields an efficient SNARK for binary witnesses.

1.2 Our Contributions

We sketch our contributions here; in Subsections 1.3 and 1.4 below, we explain them in more detail.

A reduction from tiny-field commitment to huge-field commitment. We fix a field extension L / K. For technical reasons, we require that the extension degree of L over K be a power of 2, say 2^κ. We allow K to be arbitrarily small and of arbitrary characteristic. We describe a protocol which, given blackbox access to a secure polynomial commitment scheme for multilinears over L, yields a secure polynomial commitment scheme for multilinears over K. The resulting scheme lacks "embedding overhead", in a sense we presently explain. We recall first the *packing* procedure of [DP25, § 4]; that procedure recasts each ℓ-variate multilinear $t(X_0, \ldots, X_{\ell-1})$ over K into an $\ell-\kappa$-variate multilinear $t'(X_0, \ldots, X_{\ell-\kappa-1})$ over L. (Packing proceeds by reinterpreting each 2^κ-element chunk of $t(X_0, \ldots, X_{\ell-1})$'s Lagrange coefficient vector as a single L-element, using a fixed K-basis of L.) The multilinears $t(X_0, \ldots, X_{\ell-1})$ and $t'(X_0, \ldots, X_{\ell-\kappa-1})$ are of equal size, "in bits"; they contain "the same amount of information". Our K-scheme's commitment procedure, on the input $t(X_0, \ldots, X_{\ell-1})$, simply invokes the underlying L-scheme's commitment procedure on the packed polynomial $t'(X_0, \ldots, X_{\ell-\kappa-1})$. Our K-scheme's evaluation protocol invokes the underlying L-scheme's evaluation protocol once; it adds to the cost of that protocol essentially just that of an $\ell-\kappa$-variate sumcheck over L, as well as a small further premium for the verifier (dependent only on the security parameter). Our protocol, "ring-switching", is loosely inspired by Ron-Zewi and Rothblum [RR24, Fig. 2]'s "code-switching" technique. Under the hood, ring-switching intermediates between K-evaluation and L-evaluation using the *tensor algebra* $L \otimes_K L$, the tensor product of L with itself over its own subfield K.

BaseFold in characteristic 2. The *BaseFold* multilinear polynomial commitment scheme of Zeilberger, Chen and Fisch [ZCF24, § 5] identifies a new connection between Ben-Sasson, Bentov, Horesh and Riabzev's [BBHR18a] celebrated *FRI* IOP of proximity and multilinear evaluation. That work observes that the FRI prover's final constant message doesn't merely serve the verifier's proximity test, but moreover conveys important information about the message hidden beneath the prover's word. That is, in a certain restricted setting—that of *odd prime* characteristic, where FRI is carried out over a power-of-2-sized subgroup of multiplicative units, and the FRI folding arity is fixed at $\eta = 1$—the prover's message relates to its final FRI constant just as a multilinear's vector of coefficients relates to its evaluation. In other words, prime-field FRI with $\eta = 1$ implicitly contains multilinear evaluation "built in". Using this idea, as well as a further innovation which interleaves the FRI protocol with a sumcheck (using the same stream of verifier challenges for both), BaseFold [ZCF24, § 5] delivers a new and highly interesting PCS.

BaseFold PCS fails to work straightforwardly in characteristic 2. Indeed, while FRI [BBHR18a] certainly works in characteristic 2—and was originally presented that way—binary FRI is more complicated than prime-field FRI is. It replaces that variant's squaring maps $X \mapsto X^2$ with degree-2 *linear subspace polynomials* $X \mapsto q^{(i)}(X)$. For linear subspace maps $q^{(0)}, \ldots, q^{(\ell-1)}$ chosen arbitrarily—and, we emphasize, FRI does not suggest a choice—BaseFold's built-in multilinear evaluator fails to work, in general.

Below, we suggest FRI domain maps $q^{(0)}, \ldots, q^{(\ell-1)}$ which serve to recover BaseFold PCS in characteristic 2. Interestingly, our maps are related to the *novel polynomial basis* of Lin, Chung and Han [LCH14], and surface a before-unnoted connection between that work and binary FRI. This technique appears to independently interesting, and has already been used in various subsequent works [Fs24] [Bre+25].

A competitive PCS for multilinears over tiny binary fields. Putting the two parts above together, we obtain a tiny-field multilinear PCS with compelling performance characteristics. We present our combined scheme in Section 5 below, and benchmark our Rust implementation of it, which is production-grade. Our multithreaded implementation commits to and opening-proves a 28-variate multilinear over $\mathbb{F}_{2^{32}}$ in just 6 and 10 seconds, respectively (see Table 3). It commits and opens a 28-variate multilinear over $\mathbb{F}_2$ in a stunning 0.15 and 0.33 seconds. The resulting proofs are 0.359 MiB and 0.228 MiB, respectively.

Diamond and Posen [DP25]'s scheme is somewhat faster, especially during proving (as opposed to committing) and over larger fields. That scheme commits and proves a 28-variate multilinear over $\mathbb{F}_{2^{32}}$ in just 5 and 1 seconds, respectively. Over $\mathbb{F}_2$, it commits and proves a 28-variate multilinear in 0.15 and 0.17 seconds (these latter figures are similar to ours). On the other hand, it has much larger proofs: of 3.884 and 2.849 MiB, respectively, for 28-variate multilinears over $\mathbb{F}_{2^{32}}$ and $\mathbb{F}_2$ (see Table 1).

1.3 Ring-Switching

In this subsection, we gently introduce ring-switching, prioritizing technical simplicity and accuracy. We fix a field extension $L \,/\, K$ of power-of-2 degree 2^κ. Though ring-switching works for any such field extension, of any characteristic, we note the important special case $K = \mathbb{F}_2$ and $L = \mathbb{F}_{2^{128}}$. We write $\mathcal{B}_\kappa \coloneqq \{0,1\}^\kappa$ for the κ-dimensional unit cube.

The problem. We *assume* access to a large-field multilinear polynomial commitment scheme, for multilinears over L. How might we obtain a small-field commitment scheme for multilinears over K, assuming access to that large-field scheme?

We begin with a small-field multilinear, say $t(X_0,\ldots,X_{\ell-1}) \in K[X_0,\ldots,X_{\ell-1}]^{\preceq 1}$, that we'd like to commit to. Following Diamond and Posen [DP25], we fix a basis $(\beta_v)_{v\in\mathcal{B}_\kappa}$ of L over K, write $\ell' \coloneqq \ell - \kappa$, and define the *packed multilinear*:

$$t'(X_0,\ldots,X_{\ell'-1}) \coloneqq \sum_{v\in\mathcal{B}_\kappa} t(v_0,\ldots,v_{\kappa-1},X_0,\ldots,X_{\ell'-1}) \cdot \beta_v. \tag{1}$$

The multilinear $t'(X_0,\ldots,X_{\ell'-1})$'s coefficients relate to $t(X_0,\ldots,X_{\ell-1})$'s by a "packing" operation, as Diamond and Posen explain. That is, $t'(X_0,\ldots,X_{\ell'-1})$'s Lagrange coefficient vector arises from $t(X_0,\ldots,X_{\ell-1})$'s by a procedure which reinterprets each 2^κ-element chunk of that latter vector's elements as a single L-element.

How should we commit to $t(X_0,\ldots,X_{\ell-1})$? Our commitment procedure is simple: it just invokes the underlying large-field scheme's commitment procedure on $t'(X_0,\ldots,X_{\ell'-1})$. This procedure lacks embedding overhead, since $t(X_0,\ldots,X_{\ell-1})$ and $t'(X_0,\ldots,X_{\ell'-1})$ are of the same size (in bits).

To check an evaluation claim on $t(X_0,\ldots,X_{\ell-1})$, we must reduce it to one on $t'(X_0,\ldots,X_{\ell'-1})$. We fix a point $(r_0,\ldots,r_{\ell-1})$ and an evaluation claim $s \stackrel{?}{=} t(r_0,\ldots,r_{\ell-1})$. We emphasize that the evaluation point r is defined over L, and not over K (we refer to [DP25, § 3.2], as well as to Subsection 2.8 below, for security definitions).

A strawman approach. We begin with a tempting "strawman" approach, which exhibits the difficulty of the problem. This first approach, though correct, is insecure. On the other hand, it prefigures a few of our techniques, and serves as a jumping-off point.

This simple technique proceeds in the following way. First, the prover sends values $(\hat{s}_v)_{v\in\mathcal{B}_\kappa}$ which—it claims—respectively satisfy

$$\hat{s}_v \stackrel{?}{=} t(v_0,\ldots,v_{\kappa-1},r_\kappa,\ldots,r_{\ell-1}), \tag{2}$$

for each $v \in \mathcal{B}_\kappa$. The verifier begins by checking whether

$$s \stackrel{?}{=} \sum_{v\in\mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0,\ldots,v_{\kappa-1},r_0,\ldots,r_{\kappa-1}) \cdot \hat{s}_v \tag{3}$$

holds. This equality will certainly hold if the prover is honest, since

$$t(r_0,\ldots,r_{\ell-1}) = \sum_{v\in\mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0,\ldots,v_{\kappa-1},r_0,\ldots,r_{\kappa-1}) \cdot t(v_0,\ldots,v_{\kappa-1},r_\kappa,\ldots,r_{\ell-1}).$$

On the other hand, if $s \neq t(r_0,\ldots,r_{\ell-1})$, then the prover can cause (3) to pass only by sending claims $(\hat{s}_v)_{v\in\mathcal{B}_\kappa}$ for which at least one of the equalities (2) does *not* hold.

The linear combination, over the basis $(\beta_v)_{v\in\mathcal{B}_\kappa}$, of all 2^κ instances of (2) is:

$$\sum_{v\in\mathcal{B}_\kappa} \hat{s}_v \cdot \beta_v \stackrel{?}{=} \sum_{v\in\mathcal{B}_\kappa} t(v_0,\ldots,v_{\kappa-1},r_\kappa,\ldots,r_{\ell-1}) \cdot \beta_v. \tag{4}$$

On the other hand, by (1), the right-hand side of (4) is simply $t'(r_\kappa,\ldots,r_{\ell-1})$, which the verifier has direct access to. The verifier may thus use the underlying large-field scheme to compare $\sum_{v\in\mathcal{B}_\kappa} \beta_v \cdot \hat{s}_v$ to $t'(r_\kappa,\ldots,r_{\ell-1})$. We summarize this approach in Figure 1 below.

$\underline{\mathcal{P}(r,s;t)}$		$\underline{\mathcal{V}(r,s)}$
for each $v \in \mathcal{B}_\kappa$, set $\hat{s}_v := t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$.	$\xrightarrow{(\hat{s}_v)_{v \in \mathcal{B}_\kappa}}$	check $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.
		set $s' := \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \cdot \beta_v$.
		check $s' \stackrel{?}{=} t'(r_\kappa, \ldots, r_{\ell-1})$ using large-field scheme.

Figure 1: A simple—but insecure—straw-man variant of ring-switching.

While this protocol is complete, it's not secure. The problem is the linear combination (4) of (2). The set $(\beta_v)_{v \in \mathcal{B}_\kappa}$ is a basis of L over K. In particular, it's linearly independent over K: each distinct K-vector yields a unique combination of $(\beta_v)_{v \in \mathcal{B}_\kappa}$. But it's not linearly independent over L! It's easy to write down two L-combinations of $(\beta_v)_{v \in \mathcal{B}_\kappa}$, with unequal coefficient vectors, which nonetheless equal the same value.

The problem is that the individual equations (2) are defined over L, and not K. Thus, combining them is not secure. In particular, the prover can easily contrive to construct values $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$ which don't *individually* equal $(t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}))_{v \in \mathcal{B}_\kappa}$, but for which $\sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \cdot \beta_v = t'(r_\kappa, \ldots, r_{\ell-1})$ nonetheless holds.

Our solution. Our idea is—very roughly—to *further* decompose the claims (2), until they *are* defined over K. We may then apply the "tempting" linear combination (4) to the *resulting* decomposed claims, proceeding "slice-wise". We explain the details now.

For each $v \in \mathcal{B}_\kappa$, the verifier can freely basis-decompose the prover's quantity $\hat{s}_v$, writing

$$\hat{s}_v = \sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u \tag{5}$$

for appropriate K-elements $(\hat{s}_{u,v})_{u \in \mathcal{B}_\kappa}$. Moreover, for each $v \in \mathcal{B}_\kappa$,

$$t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}) = \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}). \tag{6}$$

Combining (5) and (6), we basis-decompose (2) as follows. For each $v \in \mathcal{B}_\kappa$, we have the claim:

$$\sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}). \tag{7}$$

We're not quite done. While, for each $w \in \mathcal{B}_\kappa$, $t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1})$ is of course a K-element, $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})$ is not. On the other hand, we can basis-decompose these latter quantities too. For each $w \in \mathcal{B}_{\ell'}$, we can freely write down K-elements $(A_{w,u})_{u \in \mathcal{B}_\kappa}$ for which

$$\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u. \tag{8}$$

Using these, we further re-express (7) in the following way. For each $v \in \mathcal{B}_\kappa$, we obtain the claim:

$$\begin{aligned}
\sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u &\stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) && \text{(this is (7).)} \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u \right) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) && \text{(by definition of } (A_{w,u})_{u \in \mathcal{B}_\kappa} \text{ (8).)} \\
&= \sum_{u \in \mathcal{B}_\kappa} \left(\sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \right) \cdot \beta_u && \text{(rearrange the sum.)}
\end{aligned}$$

We've finally reached something we can basis-decompose!

Indeed, everything in the left-hand and right-hand sides of the above identity is defined over K, *except* for the basis-elements $(\beta_u)_{u \in \mathcal{B}_\kappa}$. To check the claim above for each $v \in \mathcal{B}_\kappa$, it's thus *equivalent* for the verifier to check whether, for each $u \in \mathcal{B}_\kappa$ *and* each $v \in \mathcal{B}_\kappa$,

$$\hat{s}_{u,v} \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}). \tag{9}$$

Here is the key point: unlike (2), the claims (9) *are* purely defined over K. We can thus basis-combine the claims (9) using $(\beta_v)_{v \in \mathcal{B}_\kappa}$, as opposed to the claims (2). We do exactly this. For each $u \in \mathcal{B}_\kappa$, combining (9) over $v \in \mathcal{B}_\kappa$, we obtain the claim:

$$\begin{aligned}
\sum_{v \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_v &\stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \left(\sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \right) \cdot \beta_v && \text{(combine (9) over } (\beta_v)_{v \in \mathcal{B}_\kappa}.) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot \left(\sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \cdot \beta_v \right) && \text{(rearrange the sum.)} \\
&= \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t'(w_0, \ldots, w_{\ell'-1}). && \text{(by the definition (1).)}
\end{aligned}$$

Up to defining

$$\hat{s}_u \coloneqq \sum_{v \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_v, \tag{10}$$

we thus have the claims

$$\hat{s}_u \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t'(w_0, \ldots, w_{\ell'-1}) \tag{11}$$

for each $u \in \mathcal{B}_\kappa$. This combination *is* secure. Moreover, its right-hand side depends only on $t'(X_0, \ldots, X_{\ell'-1})$, as well as on $(A_{w,u})_{w \in \mathcal{B}_{\ell'}}$. We're getting close to something that we can run the sumcheck on.

The verifier must check (11) for *each* $u \in \mathcal{B}_\kappa$. In light of standard sumcheck batching techniques, though, this fact presents no obstacle. In practice, up to a soundness error of just $\frac{\kappa}{|L|}$, the verifier may simply sample further random scalars $(r''_0, \ldots, r''_{\kappa-1})$, and batch both sides of (11) by the L-vector $\left(\widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1})\right)_{u \in \mathcal{B}_\kappa}$, over varying $u \in \mathcal{B}_\kappa$. That is, the verifier may check whether

$$\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u, r'') \cdot \hat{s}_u \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u, r'') \cdot A_{w,u} \right) \cdot t'(w) \tag{12}$$

holds. We summarize our amended protocol in Figure 2 below.

$\mathcal{P}(r, s; t)$		$\mathcal{V}(r, s)$
for each $v \in \mathcal{B}_\kappa$, set $\hat{s}_v \coloneqq t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$.	$\xrightarrow{(\hat{s}_v)_{v \in \mathcal{B}_\kappa}}$	check $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.
		for each $v \in \mathcal{B}_\kappa$, decompose $\hat{s}_v = \sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u$.
		for each $u \in \mathcal{B}_\kappa$, combine $\hat{s}_u \coloneqq \sum_{v \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_v$.
	$\xleftarrow{(r''_0, \ldots, r''_{\kappa-1})}$	sample batching scalars $(r''_0, \ldots, r''_{\kappa-1}) \leftarrow L^\kappa$.
.... $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck (12).		
		at the end of the sumcheck, evaluate $\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u, r'') \cdot A_u(r')$ and $t'(r')$, where $r = (r'_0, \ldots, r'_{\ell'-1})$ is the sumcheck challenge.

Figure 2: A rough sketch of our full ring-switching protocol.

There is one, final issue that we've glossed over. In Figure 2 above, for each $u \in \mathcal{B}_\kappa$, we abbreviate $A_u(X_0, \ldots, X_{\ell'-1})$ for the multilinear extension of the function $A_u : w \mapsto A_{w,u}$ on $\mathcal{B}_{\ell'}$. At the end of the sumcheck, the verifier may learn $t'(r'_0, \ldots, r'_{\ell'-1})$ by invoking the underlying large-field scheme's evaluation protocol once. How might the verifier locally—and efficiently—obtain the evaluations $\left(A_u(r'_0, \ldots, r'_{\ell'-1})\right)_{u \in \mathcal{B}_\kappa}$? Here, $(r'_0, \ldots, r'_{\ell-1}) \in L^{\ell'}$ is the random point sampled by the verifier during the course of the sumcheck.

This issue brings us to the trickiest part of our entire theory. We claim that the multilinears $(A_u(X_0, \ldots, X_{\ell'-1}))_{u \in \mathcal{B}_\kappa}$ are *succinct*. In fact, the verifier may learn *all* of the evaluations $(A_u(r_0, \ldots, r_{\ell'-1}))_{u \in \mathcal{B}_\kappa}$, in one shot, by expending just $2 \cdot 2^\kappa \cdot \ell'$ L-multiplications. This fact is non-obvious; we sketch it now.

To do this, we recall how the multilinears $A_u(X_0, \ldots, X_{\ell'-1})$ are defined. Indeed, for each $u \in \mathcal{B}_\kappa$, $A_u(X_0, \ldots, X_{\ell'-1})$ is defined, in the Lagrange basis, as the u^{th} "coordinate slice" of the partially-specialized equality indicator $\widetilde{\mathsf{eq}}(X_0, \ldots, X_{\ell'-1}, r_\kappa, \ldots, r_{\ell-1})$. Moreover, for each $u \in \mathcal{B}_\kappa$,

$$A_u(r'_0, \ldots, r'_{\ell-1}) = \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell-1}) \tag{13}$$

obviously holds. Writing all 2^κ copies—i.e., over varying $u \in \mathcal{B}_\kappa$—of the sum (13) into the rows of a square, $2^\kappa \times 2^\kappa$ matrix, we obtain the sum expression depicted in Figure 3 below.

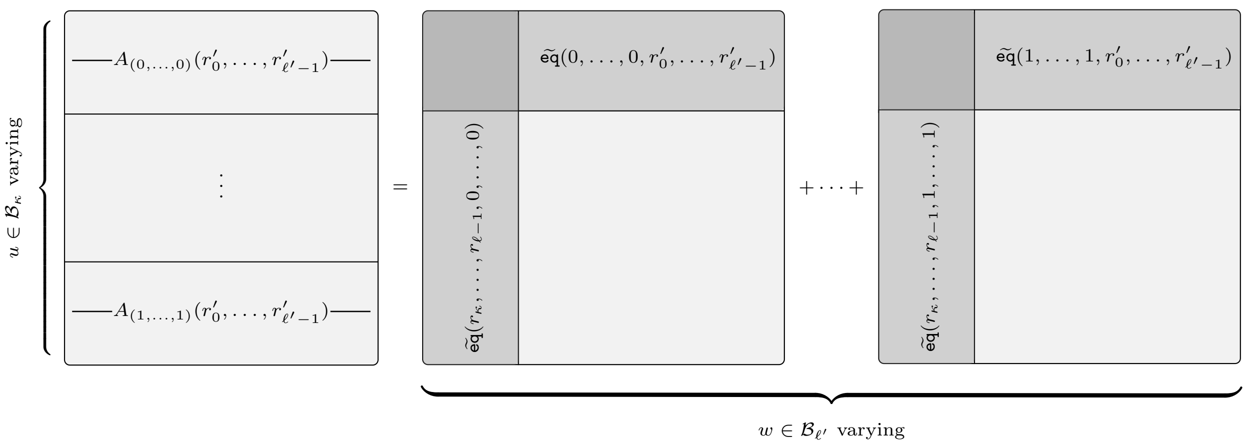

Figure 3: Vertically stacking (13), we express $\left(A_u(r'_0, \ldots, r'_{\ell'-1})\right)_{u \in \mathcal{B}_\kappa}$ as the rows of an unusual sum.

In the right-hand sum of Figure 3, for each $w \in \mathcal{B}_{\ell'}$, we've drawn a square in which $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})$ is written vertically and $\widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1})$ horizontally. By each such square, we mean the $2^\kappa \times 2^\kappa$ K-matrix whose cells are the "exterior product" of $\widetilde{\mathsf{eq}}(r_\kappa \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})$ and $\widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1})$. That is, we basis-decompose each of these L-elements, and take *all* cross-products between the resulting two K-vectors.

Why did we do such a thing? Because this is simply the definition of (13)! Indeed, if we "zoom into" a single row $u \in \mathcal{B}_\kappa$ of Figure 3, then we obtain (13) on the nose (we recall that $A_{w,u}$ is *defined* to be the u^{th} coordinate slice of $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})$).

The point of Figure 3 is that it lets us explain our succinct algorithm for $(A_u(r_0, \ldots, r_{\ell'-1}))_{u \in \mathcal{B}_\kappa}$. If we write $\star$ for the "exterior product" operation between K-vectors, then Figure 3 shows us that:

$$(A_u(r_0, \ldots, r_{\ell'-1}))_{u \in \mathcal{B}_\kappa} = \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \star \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1}). \tag{14}$$

The right-hand side of (14) *almost* looks like the equality indicator expression $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell'-1})$; on the other hand, it differs from that simple expression in its use of cross-products. As we will argue below, (14) *does* become expressible as an equality indicator evaluation, once we pass to a suitable ring—one whose multiplication operation appropriately captures the exterior product. As we argue below, this ring arises in the form of the *tensor product of algebras* of L by itself, over its subfield K—that is, the ring $A \coloneqq L \otimes_K L$. Below, we resurface rigorously this algebraic object, and show that it gives us exactly the structure we need.

Hashcaster. We undertake a detailed comparison of ring-switching to Hashcaster. Soukhanov's *Hashcaster* [Sou24] is a SNARK for binary (i.e., specifically $\mathbb{F}_2$-valued) witnesses. At the PIOP level, that work introduces a number of innovations, including an efficient "ternary" sumcheck for domains of power-of-3 size. For the purposes of this work, we survey just that work's ideas at the PCS level, which are also important. Indeed, that work yields an alternative reduction from the problem of evaluating the K-multilinear $t(X_0, \ldots, X_{\ell-1})$ to that of evaluating its packed L-multilinear $t'(X_0, \ldots, X_{\ell'-1})$. For self-containedness, we reproduce that work's technique here in some detail; we then compare it to ring-switching, the approach of this work.

We again fix a degree-2^κ field extension L / K, and write $(\beta_v)_{v \in \mathcal{B}_\kappa}$ for a basis of L over K. We write $\sigma \in \text{Gal}(L / K)$ for the *Frobenius automorphism* of L over K. As a notational device, for each $v \in \mathcal{B}_\kappa$, we write $\{v\} \coloneqq \sum_{i=0}^{\kappa-1} 2^i \cdot v_i$.

Hashcaster begins with the same observation as Subsection 1.3 does. That is, for the verifier to assess the evaluation claim $s \stackrel{?}{=} t(r_0, \ldots, r_{\ell-1})$, it suffices for the prover to send it quantities $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$ respectively *claimed to* equal $(t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}))_{v \in \mathcal{B}_\kappa}$ (recall (2)). Indeed, given these, the verifier can check

$$s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\texttt{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v, \tag{15}$$

as usual (just as in (3)).

At this point, Hashcaster diverges. Hashcaster's idea is to relate the claimed partial evaluations $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$ to the respective evaluations of the packed multilinear $t'(X_0, \ldots, X_{\ell'-1})$ at $\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right)_{v \in \mathcal{B}_\kappa}$, the *componentwise Galois orbit* of the suffix $(r_\kappa, \ldots, r_{\ell-1})$.

There are various ways to make this task precise. Hashcaster's idea hinges on the following matrix identity:

$$\begin{bmatrix} & & \\ & \sigma^{\{u\}}(\beta_v) & \\ & & \end{bmatrix} \cdot \begin{bmatrix} | \\ \hat{s}_v \\ | \end{bmatrix} \stackrel{?}{=} \begin{bmatrix} | \\ \sigma^{\{u\}}(t')(r_\kappa, \ldots, r_{\ell-1}) \\ | \end{bmatrix}. \tag{16}$$

On the left, we have the $2^\kappa \times 2^\kappa$ matrix whose $(\{u\}, \{v\})^{\text{th}}$ entry is $\sigma^{\{u\}}(\beta_v)$, the $\{u\}^{\text{th}}$ Galois image of the v^{th} basis vector. On the right-hand side, we have the vector containing the respective evaluations at $(r_\kappa, \ldots, r_{\ell-1})$ of $t'(X_0, \ldots, X_{\ell'-1})$'s various "Galois twists". Indeed, for each $u \in \mathcal{B}_\kappa$, we define $\sigma^{\{u\}}(t')(X_0, \ldots, X_{\ell'-1})$ by the Lagrange basis prescription $\sigma^{\{u\}} : w \mapsto \sigma^{\{u\}}(t'(w))$, for each $w \in \mathcal{B}_{\ell'}$. It is a nontrivial fact of field theory that the matrix $\left[\sigma^{\{u\}}(\beta_v)\right]$ is nonsingular; we refer to Lidl and Niederreiter [LN96, Lem. 3.51]. Assuming this fact, it's not too hard to show that (16) holds *if and only if* the prover is honest (i.e., if each of its claims $\hat{s}_v \stackrel{?}{=} t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$, for $v \in \mathcal{B}_\kappa$, is true).

Upon receiving the prover's vector of claims $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$, the verifier could thus compute the matrix transformation (16) on that vector. By moreover "peeling off" the twists $\sigma^{\{u\}}$, for each $u \in \mathcal{B}_\kappa$, the verifier could thereby obtain a further vector supposedly equal to $\left(t'\left(\sigma^{\{-u\}}(r_\kappa), \ldots, \sigma^{\{-u\}}(r_{\ell-1})\right)\right)_{u \in \mathcal{B}_\kappa}$, the vector of evaluations of $t'(X_0, \ldots, X_{\ell'-1})$ on the Galois orbit of $(r_\kappa, \ldots, r_{\ell-1})$.

In fact, we will do Hashcaster one better. We claim that, up to performing a transposition of the sort already discussed above in the context of ring-switching, the verifier may simplify its job; specifically, it might directly relate $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$ to $\left(t'\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right)\right)_{v \in \mathcal{B}_\kappa}$ (with no twists necessary). Indeed, given $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$, the verifier may freely, as before, decompose $\hat{s}_v = \sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u$ (just as in (5)), and write $\hat{s}_u \coloneqq \sum_{v \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_v$ (just as in (10)). In this setting, we obtain the further identity:

$$\underbrace{\begin{bmatrix} \text{———} \hat{s}_u \text{———} \end{bmatrix}}_{u \in \mathcal{B}_\kappa \text{ varying}} \cdot \begin{bmatrix} & & \\ & \sigma^{\{v\}}(\beta_u) & \\ & & \end{bmatrix} \stackrel{?}{=} \underbrace{\begin{bmatrix} \text{———} t'\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right) \text{———} \end{bmatrix}}_{v \in \mathcal{B}_\kappa \text{ varying}}. \tag{17}$$

We claim that (17) *also* holds if and only if the prover is honest (i.e., if $\hat{s}_v \stackrel{?}{=} t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$ for each $v \in \mathcal{B}_\kappa$). Importantly, the right-hand side of (17) directly yields the evaluations of $t'(X_0, \ldots, X_{\ell'-1})$ over $(r_\kappa, \ldots, r_{\ell-1})$'s Galois orbit, with no "twisting" or "peeling off" necessary.

To check the validity of the prover's claim, then, it's thus enough for the verifier to compute

$$\underbrace{\begin{bmatrix} \text{——}\overline{s}_v\text{——} \end{bmatrix}}_{v \in \mathcal{B}_\kappa \text{ varying}} \coloneqq \underbrace{\begin{bmatrix} \text{——}\hat{s}_u\text{——} \end{bmatrix}}_{u \in \mathcal{B}_\kappa \text{ varying}} \cdot \begin{bmatrix} \\ \sigma^{\{v\}}(\beta_u) \\ \\ \end{bmatrix}, \tag{18}$$

i.e. the left-hand side of (17), and then check whether, for each $v \in \mathcal{B}_\kappa$,

$$\overline{s}_v \stackrel{?}{=} t'\Big(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\Big) \tag{19}$$

holds.

We've made progress, since each right-hand side of (19) (for $v \in \mathcal{B}_\kappa$) is an evaluation of $t'(X_0, \ldots, X_{\ell'-1})$. On the other hand, we'd prefer to evaluate $t'(X_0, \ldots, X_{\ell'-1})$ at just one place, as opposed to over the entire orbit $\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right)_{v \in \mathcal{B}_\kappa}$. To this end, we again apply a sumcheck-based batching technique, due this time to Ron-Zewi and Rothblum [RR24, Fig. 3]. Indeed, we note that, for each $v \in \mathcal{B}_\kappa$, (19) is equivalent to the sum claim:

$$\overline{s}_v \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}\Big(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}\Big) \cdot t'(w). \tag{20}$$

After sampling batching scalars $(r''_0, \ldots, r''_{\kappa-1})$, the verifier may batch both sides of (20) by the vector $\left(\widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1})\right)_{v \in \mathcal{B}_\kappa}$, and in this way obtain the batched sum claim:

$$\sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v, r'') \cdot \overline{s}_v \stackrel{?}{=} \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v, r'') \cdot \widetilde{\mathsf{eq}}\Big(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}\Big) \right) \cdot t'(w). \tag{21}$$

This is finally something we can run the sumcheck on. Combining these observations, we obtain the protocol sketched in Figure 4.

$\underline{\mathcal{P}(r, s; t)}$		$\underline{\mathcal{V}(r, s)}$
for each $v \in \mathcal{B}_\kappa$, set $\hat{s}_v \coloneqq t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$.	$\xrightarrow{(\hat{s}_v)_{v \in \mathcal{B}_\kappa}}$	check $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.
		for each $v \in \mathcal{B}_\kappa$, decompose $\hat{s}_v = \sum_{u \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_u$.
		for each $u \in \mathcal{B}_\kappa$, combine $\hat{s}_u \coloneqq \sum_{v \in \mathcal{B}_\kappa} \hat{s}_{u,v} \cdot \beta_v$.
		set $[\text{—}\overline{s}_v\text{—}] \coloneqq [\text{—}\hat{s}_u\text{—}] \cdot \left[\sigma^{\{v\}}(\beta_u)\right]$ as in (18)
	$\xleftarrow{(r''_0, \ldots, r''_{\kappa-1})}$	sample batching scalars $(r''_0, \ldots, r''_{\kappa-1}) \leftarrow L^\kappa$.
	.... $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck (21).	
		at the end of the sumcheck, evaluate $\sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v, r'') \cdot \widetilde{\mathsf{eq}}\left(\sigma^{\{v\}}(r), r'\right)$ and $t'(r')$, where $r = (r'_0, \ldots, r'_{\ell'-1})$ is the sumcheck challenge, and we set $\sigma^{\{v\}}(r) \coloneqq \left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right)$.

Figure 4: A sketch of the Hashcaster [Sou24] protocol.

This is a perfectly correct and sound protocol, which moreover wards off the tensor-algebraic complexities of ring-switching (cf. Figure 3). But what about its efficiency? We claim that ring-switching is more efficient than Hashcaster for both the prover and the verifier. We note that the comparison we undertake below pits ring-switching against our *improved* variant of Hashcaster (see above for details). The "original" version of Hashcaster is still worse, albeit just by a bit. (We expand this analysis further in Subsection 3.2 below.)

We begin with our protocols' verifiers. The verifier complexities of Figures 2 and 4 turn out to be almost identical, *except* for the Hashcaster verifier's matrix transformation (18). Hashcaster's verifier must compute this matrix product; ours has no analogue of this task. This transformation entails a *quadratic* number of L-multiplications in the extension degree 2^κ. (There might be an "NTT analogue" for this matrix; we haven't investigated this thoroughly.) Thus Hashcaster's verifier's number of L-multiplications grows quadratically in the extension degree 2^κ; ours grows only linearly. We do not know how to modify Hashcaster so as to make its verifier complexity match ours (other than by replacing it entirely with ring-switching).

Our protocols' provers tell a similar story. To calculate $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$, both provers must begin by tensor-expanding $(r_\kappa, \ldots, r_{\ell-1})$, which takes $2^{\ell'}$ L-by-L multiplications, and then performing 2^ℓ L-by-K multiplications and just under 2^ℓ L-additions. To prepare the sumcheck (12), our prover must further basis-decompose this tensor, and row-combine the resulting $2^\kappa \times 2^{\ell'}$ K-matrix by the L-vector $(\widetilde{\texttt{eq}}(u, r''))_{u \in \mathcal{B}_\kappa}$. We thus further obtain again 2^ℓ L-by-K multiplications and just under 2^ℓ L-additions. To prepare the analogous sumcheck (21), Hashcaster's prover must do something more complex. It must first obtain the respective tensor-expansions of *each* of the elements of the Galois orbit $\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1})\right)_{v \in \mathcal{B}_\kappa}$. This task, at least implemented naively, entails $2^{\ell'}$ L-by-L multiplications and 2^ℓ Frobenius applications. Finally, it must multiply the row-combination vector $(\widetilde{\texttt{eq}}(u, r''))_{u \in \mathcal{B}_\kappa}$ by the resulting $2^\kappa \times 2^{\ell'}$ L-matrix, thereby spending 2^ℓ L-by-L multiplications. Hashcaster's prover, if implemented naively, is about 2^κ-fold more costly than ours.

On the other hand, Hashcaster develops various concrete optimizations, which serve to make its prover more efficient. These all have the same flavor, however; to explain it, we need to develop a bit more theory.

For each Galois extension $L \,/\, K$, the L-algebras

$$L \otimes_K L \cong \prod_{\rho \in \mathrm{Gal}(L/K)} L \tag{22}$$

are isomorphic. This fact is of some importance in arithmetic geometry (see e.g. Waterhouse [Wat79]). We record a proof here. The left-hand ring should be familiar to the reader as the tensor algebra. We understand that ring as an L-algebra by letting L act on the right-hand tensor factor; that is, we define $\alpha \cdot (a_0 \otimes a_1) \coloneqq (a_0 \otimes \alpha \cdot a_1)$ on simple tensors (we refer to Subsection 2.5 below for more details on the tensor algebra). We understand the right-hand ring as an L-algebra via the standard componentwise action. As for the isomorphism itself, we map simple tensors in the following way:

$$(a_0 \otimes a_1) \mapsto \left(\sigma^0(a_0) \cdot a_1, \ldots, \sigma^{2^\kappa - 1}(a_0) \cdot a_1\right). \tag{23}$$

To describe this isomorphism in coordinates, we must pick bases for both algebras in (22). In the left-hand ring, we pick the L-basis $(\beta_u \otimes 1)_{u \in \mathcal{B}_\kappa}$; as for the right, we use the standard basis.

It's not hard to show that the matrix of the map (23)—once expressed in coordinates with respect to these bases (and acting on row-vectors by right-multiplication)—is nothing other than the matrix

$$\begin{bmatrix} & & \\ & \sigma^{\{v\}}(\beta_u) & \\ & & \end{bmatrix}; \tag{24}$$

of (17); that is, the isomorphism (22) and the transformation (17) are *one and the same map*. Of course, since we already know that (24) is nonsingular (see again [LN96, Lem. 3.51]), (22) is indeed an isomorphism.

Hashcaster turns out to be a parallel instantiation of our theory which takes place on the right-hand ring of (22), as opposed to on the left. (Hashcaster's various "optimizations" serve to move certain among its steps to the left.) In this dictionary, (11) and (20) correspond, (12) and (21) correspond, and finally the final evaluations $(A_u(r_0, \ldots, r_{\ell'-1}))_{u \in \mathcal{B}_\kappa}$ and $\left(\widetilde{\texttt{eq}}\left(\sigma^{\{v\}}(r_\kappa), \ldots, \sigma^{\{v\}}(r_{\ell-1}), r'_0, \ldots, r'_{\ell'-1}\right)\right)_{v \in \mathcal{B}_\kappa}$ correspond. These "correspondences" are not merely suggestive, but are rather entirely rigorous; in each case, the relevant quantities differ *exactly* by the isomorphism (22). On the other hand, the check (15) (see also (3)) can only be performed in the left-hand side of (22). That isomorphism's left-hand side is thus the *most natural* side within which to remain throughout. By failing to stay there, Hashcaster imposes upon its prover and verifier the costs—which are ultimately artificial—of traversing the isomorphism (22). The hard part, of course, is to find out an analogue on the left-hand side of (22) of the verifier's final check, and to remain on the left for good. This is what we've done by identifying the tensor algebra, and by developing the theory of this work.

1.4 Binary BaseFold

In order to apply ring-switching, we need a large-field scheme to invoke it on. To this end, we adapt BaseFold PCS [ZCF24, § 5] to the characteristic 2 setting. In the process, we revisit binary-field FRI [BBHR18a], and surface a new connection between that protocol and the *additive NTT* of Lin, Chung and Han [LCH14]. We carry out this work rigorously in Section 4 below.

Some background. Each honest FRI prover begins with the evaluation of some polynomial $P(X) \coloneqq \sum_{j=0}^{2^\ell-1} a_j \cdot X^j$ over its initial domain $S^{(0)}$. Under certain mild conditions—specifically, if the folding factor η divides ℓ, and the recursion is carried out to its end—the prover's final oracle will be identically constant over its domain; in fact, the prover will rather send the verifier this latter constant in the clear. What will the value of this constant be, as a function of $P(X)$ and of the verifier's folding challenges?

In the setting of prime field multiplicative FRI, in which the folding arity η moreover equals 1, the folding maps $q^{(i)}$ all take the especially simple form $X \mapsto X^2$. BaseFold [ZCF24, § 5] makes the interesting observation whereby—again, in the prime field setting, and for $q^{(0)}, \ldots, q^{(\ell-1)}$ defined in just this way—the prover's final FRI response will be nothing other than $a_0 + a_1 \cdot r_0 + a_2 \cdot r_1 + \cdots + a_{2^\ell-1} \cdot r_0 \cdot \cdots \cdot r_{\ell-1}$, where $(r_0, \ldots, r_{\ell-1})$ are the verifier's FRI folding challenges. That is, it will be exactly the evaluation of the multilinear polynomial $a_0 + a_1 \cdot X_0 + a_2 \cdot X_1 + \cdots + a_{2^\ell-1} \cdot X_0 \cdot \cdots \cdot X_{\ell-1}$ at the point $(r_0, \ldots, r_{\ell-1})$.

What about in characteristic 2? In this setting, the simple folding maps $X \mapsto X^2$ no longer work, as [BBHR18a, § 2.1] already notes. (These maps are nowhere 2-to-1, and in fact are field isomorphisms.) Rather, we must set as our $q^{(i)}$ certain *linear subspace polynomials* of degree 2. FRI does not suggest precise choices for these polynomials, beyond merely demanding that they feature the right linear-algebraic syntax. That is, each $q^{(i)}$'s kernel must reside entirely inside the domain $S^{(i)}$ (see also Subsection 2.4). Given syntactically valid subspace polynomials $q^{(i)}$ chosen otherwise arbitrarily—and, we emphasize, FRI does not suggest a choice—the constant value of the prover's final oracle will relate in a complicated way to the coefficient vector $(a_0, \ldots, a_{2^\ell-1})$ and to the verifier's folding challenges r_i.

We recall Lin, Chung and Han [LCH14]'s *novel polynomial basis* $(X_j(X))_{j=0}^{2^\ell-1}$ and *additive NTT* (see Subsection 2.3). That algorithm computes, from the vector $(a_0, \ldots, a_{2^\ell-1})$ of coefficients of a polynomial $P(X) \coloneqq \sum_{j=0}^{2^\ell-1} a_j \cdot X_j(X)$ expressed with respect to the novel basis, $P(X)$'s vector of evaluations over $S \subset L$, itself an affine $\mathbb{F}_2$-linear subspace of L (and in quasilinear time in the size of S).

Our goal is to recover in the binary setting the "classical" FRI folding pattern identified above. Essentially, our insight is that, if we choose the FRI subspace maps $q^{(0)}, \ldots, q^{(\ell-1)}$ appropriately, then the prover's final FRI oracle becomes once again meaningfully related to $P(X)$'s initial coefficient vector $(a_0, \ldots, a_{2^\ell-1})$; that is, it becomes—as it was in the prime-field setting—$a_0 + a_1 \cdot r_0 + a_2 \cdot r_1 + \cdots + a_{2^\ell-1} \cdot r_0 \cdot \cdots \cdot r_{\ell-1}$.

The problem. BaseFold PCS doesn't "just work" in characteristic 2. To achieve our adaptation, we must develop a degree of machinery. Specifically, we must specialize binary FRI; that is, we must carefully choose that protocol's codeword domains $S^{(i)} \subset L$ (for $i \in \{0, \ldots, \ell\}$) and its two-to-one collapsing maps $q^{(i)} : S^{(i)} \to S^{(i+1)}$ (for $i \in \{0, \ldots, \ell-1\}$). Our choice serves to make FRI compatible with Lin, Chung and Han's [LCH14] *additive NTT*.

In this technical overview, we sketch *why* it doesn't work to just "do the simple thing" in characteristic 2. We also sketch our solution to this problem. For the sake of this illustration, we work in a toy-sized, 8-bit field: the *AES field*. That is, we work in the field

$$\mathbb{F}_2[X] \,/\, \left(x^8 + x^4 + x^3 + x + 1\right) \cong \mathbb{F}_{2^8}.$$

We set $\ell = 2$ and $\mathcal{R} = 1$. We fix the 2-variate input multilinear

$$t(X_0, X_1) \coloneqq \texttt{0xde} \cdot 1 + \texttt{0xad} \cdot X_0 + \texttt{0xbe} \cdot X_1 + \texttt{0xef} \cdot X_0 \cdot X_1.$$

This field's elements correspond in a one-to-one way with *bytes*. We finally fix the evaluation point

$$(r_0, r_1) \coloneqq (\texttt{0xab}, \texttt{0xcd}).$$

We note that

$$t(r_0, r_1) = \texttt{0x89}. \tag{25}$$

In order to set up FRI, we need a Reed–Solomon domain $S^{(0)} \subset L$ of dimension $\ell + \mathcal{R} = 3$, together with a system of collapsing maps.

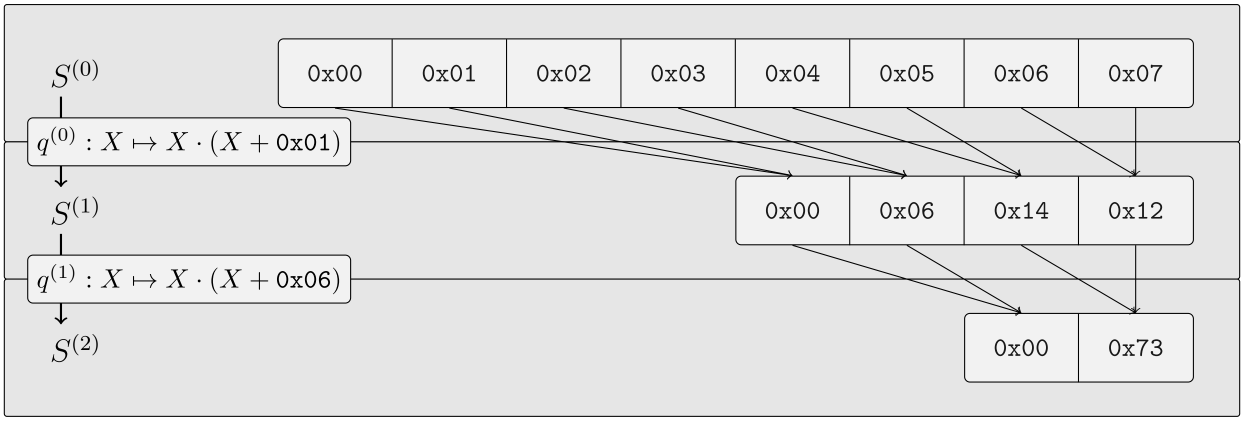

Figure 5: A possible system of domains and collapsing maps in the AES field.

In Figure 5, we sketch a plausible choice for these maps in the 8-bit AES field. We initialize $S^{(0)} := \langle \texttt{0x01}, \texttt{0x02}, \texttt{0x04} \rangle$. For our first collapsing map $q^{(0)} : S^{(0)} \to S^{(1)}$, we use $X \mapsto X^2 + X$; this map annihilates the one-dimensional subspace of $S^{(0)}$ generated by `0x01`. For our next map, we annihilate the image in $S^{(1)}$ (namely `0x06`) of the *next* basis vector of $S^{(0)}$ (namely `0x02`).

Figure 5's parameterization, while coherent—and perfectly suitable for FRI—fails to work for binary BaseFold. To show why, we give the thing a try. As prescribed by BaseFold PCS, we begin by "flattening" the input multilinear $t(X_0, X_1)$ into a *univariate* polynomial of degree less than 2^ℓ. In this way, we obtain

$$P(X) = \texttt{0xde} \cdot 1 + \texttt{0xad} \cdot X + \texttt{0xbe} \cdot X^2 + \texttt{0xef} \cdot X^3.$$

We next Reed–Solomon-encode—that is, we evaluate—$P(X)$ on the domain $S^{(0)}$; finally, we FRI-fold the resulting codeword using the challenges $r_0 = \texttt{0xab}$ and $r_1 = \texttt{0xcd}$. This process appears in Figure 6 below.

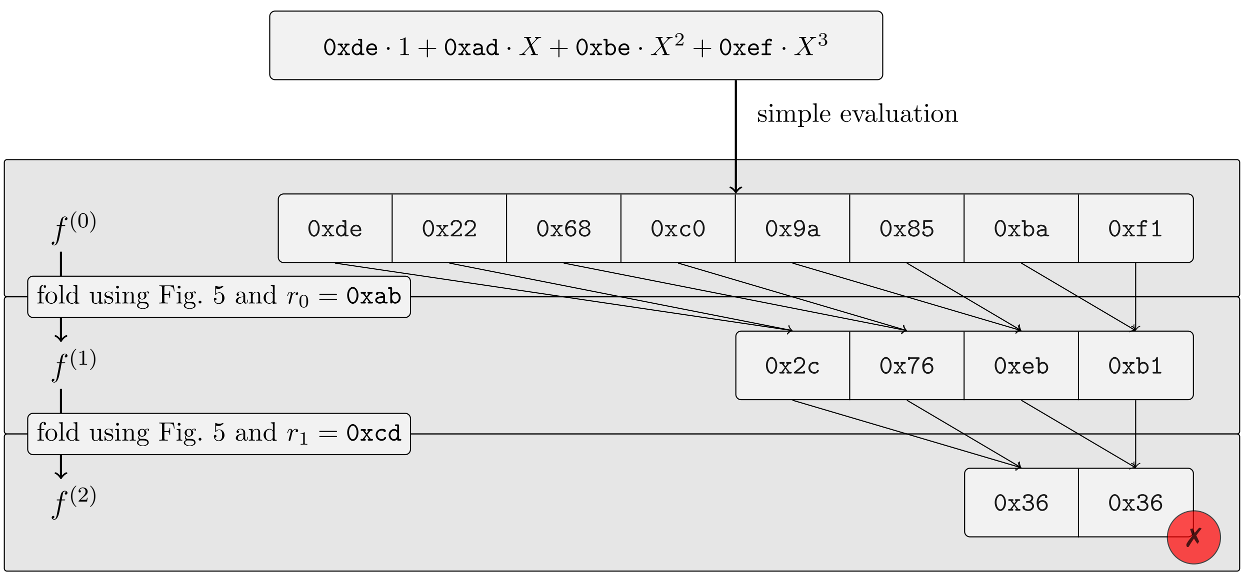

Figure 6: If we don't choose our domains carefully, then FRI-folding *fails* to capture multilinear evaluation.

The final FRI oracle in Figure 6 is constant, as is liable to hold *generically* in FRI. On the other hand, its value is wrong. We already agreed in (25) that $t(r_0, r_1) = \texttt{0x89}$; on the other hand, we obtained `0x36` above. BaseFold works only when these values are equal.

In Figure 7 below, we reveal *our* choice of domains $S^{(0)}$, $S^{(1)}$ and $S^{(2)}$ and folding maps $q^{(0)}$ and $q^{(1)}$.

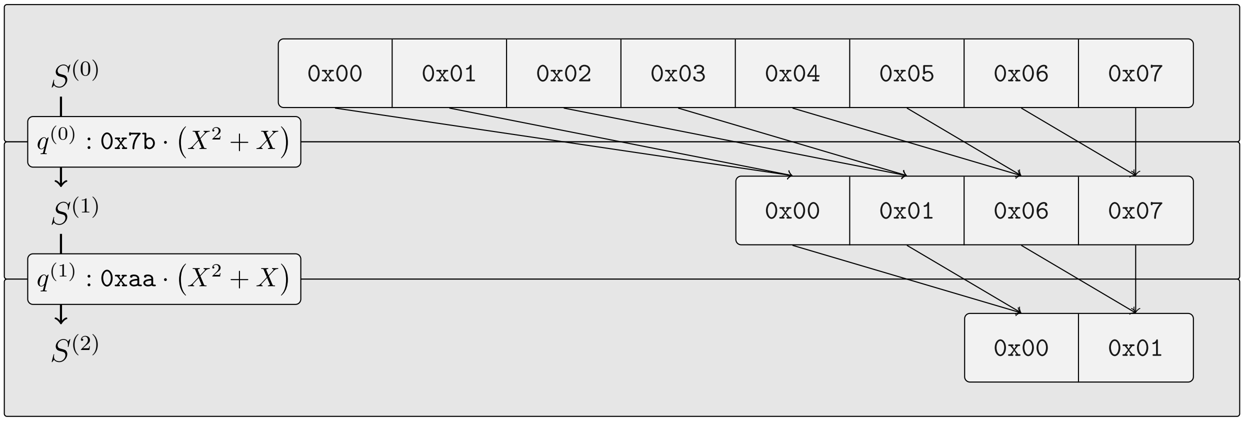

Figure 7: A further binary FRI configuration, this time BaseFold-compatible.

The choice procedure underlying Figure 7 is given rigorously in Subsection 4.1 below (see Definition 4.1). Actually, the method is not hard to describe. At each stage i, we *begin* with the simple map $q^{(i)} : X \mapsto X^2+X$, which annihilates the subspace of $S^{(i)}$ generated by 1. Then, however, we "twist" the map $q^{(i)}$, so as to make the first element *of its image* 1. This choice guarantees to boot that 1 will be in $S^{(i+1)} = q^{(i)}\left(S^{(i)}\right)$, so that the initial choice $q^{(i+1)} : X \mapsto X^2 + X$ makes sense (that map too, though, will need to be twisted).

This process can be seen in Figure 7 above. Under the map $X \mapsto X^2 + X$, `0x02` maps to `0x06`, whose inverse is the scaling factor `0x7b`. Similarly, under $X \mapsto X^2 + X$, `0x06` maps to `0x12`, whose inverse is `0xaa`.

The point of our theory is that, if we choose our collapsing maps in the right way—that is, as Figure 7 does—then we recover multilinear evaluation after all. Crucially, we must also replace our initial Reed–Solomon encoding with Lin–Chung–Han's variant. We depict this "happy path" in Figure 8.

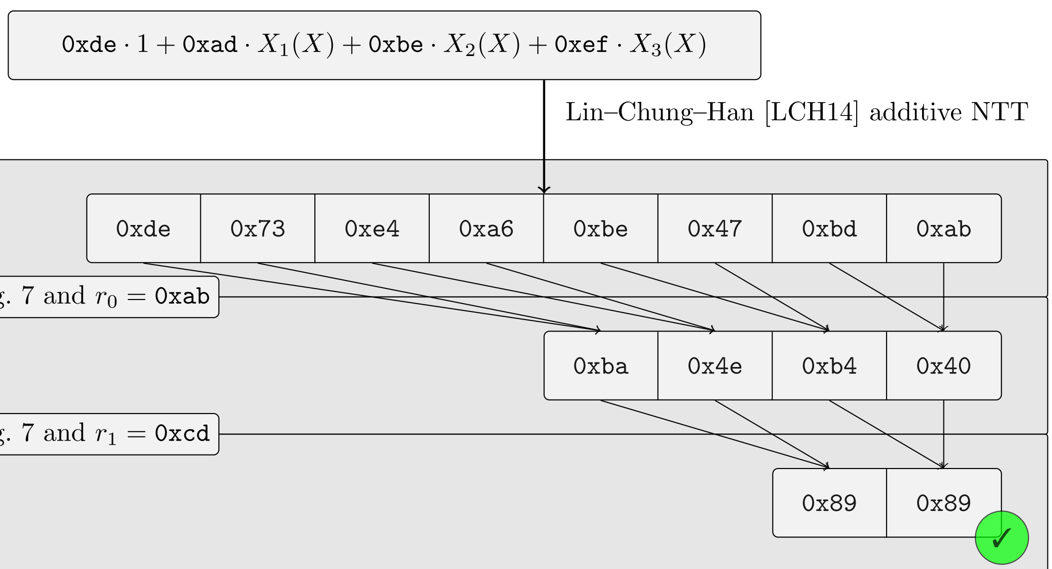

Figure 8: Upon parameterizing FRI carefully, we recover that protocol's built-in multilinear evaluator.

In Section 4 below, we prove that our collapsing maps work out this way, in general. To carry out that proof, we must study Lin–Chung–Han's additive NTT with some care (the key result is Theorem 4.12). In fact, we further enrich our binary BaseFold variant in various interesting ways. For example, using an "oracle-skipping" optimization—which itself exploits a recent tensor-style proximity gap, due to Diamond and Gruen [DG25]—we shrink that scheme's proofs by over half. We explain these ideas in full in Section 4.

1.5 Concurrent and Subsequent Works

In this subsection, we discuss an important subsequent work, Brehm et al.'s *Blaze* [Bre+25].

Blaze. Brehm et al.'s *Blaze* [Bre+25] is a polynomial commitment scheme for multilinears over large binary fields.

We fix an ℓ-variate multilinear $t(X_0, \ldots, X_{\ell-1})$ over a large binary field L. Using a technique based in code-switching [RR24], Blaze obtains a strictly linear-time commitment procedure, a linear-time prover, and a polylogarithmic verifier; we presently sketch its approach. Blaze begins how Brakedown does, except with a wide matrix—shaped something like $2^a \times 2^{\ell-a}$, where the matrix height 2^a is just polynomial in ℓ. That is, Blaze inscribes $t(X_0, \ldots, X_{\ell-1})$'s Lagrange coefficients, in row-major order, into that wide matrix. Its prover encodes that matrix row-wise under a *RAA (repeat, accumulate accumulate) code*—or under a "packed" variant of that code—and commits to the resulting matrix, which we presently call M.

To begin its evaluation procedure, the verifier samples a vector of 2^a random coefficients. Just as Brakedown [Gol+23] does, Blaze reduces the problem of evaluating $t(X_0, \ldots, X_{\ell-1})$ at some point $(r_0, \ldots, r_{\ell-1})$ to that of evaluating the message underneath $r^T \cdot M$—whatever it may be—at the suffix $(r_a, \ldots, r_{\ell-1})$.

As of this point, Blaze has shrunk its problem size by a polylogarithmic factor, and so can freely begin using "heavier"—i.e., quasilinear-time—techniques. The overhead to the verifier of this reduction is proportional to 2^a, which is just polylogarithmic in 2^ℓ. (This is code-switching in action.) Blaze, indeed, must now securely evaluate a multilinear whose coefficients are themselves encoded underneath the RAA code. To this end, it introduces a further protocol, which is based on BaseFold (and in fact on this work's binary variant). That is, it commits using binary BaseFold to the claimed RAA codeword $r^T \cdot M$, to the message supposedly underneath that codeword, and finally to all of the intermediate RAA encoding steps which intervene between those two quantities. It then uses sumcheck-based techniques, as well as the native evaluation procedure of BaseFold PCS, to check the validity of the RAA encoding and evaluate the committed message.

The Blaze PCS is functionally an alternative to the binary BaseFold PCS construction we present in Section 4. Operating over $\mathbb{F}_{2^{128}}$ throughout, Blaze [Bre+25, § 8] reports commitment and proving times which improve upon binary BaseFold's by roughly threefold at the $\ell = 28$ problem size, though its proofs are larger. The key point is that though Blaze's RAA code is very fast to encode, its relative distance is middling (e.g., just 0.19 at the rate $\rho = \frac{1}{4}$). Blaze reports a proof of 2.5 MiB in the $\ell = 28$ case, compared with 1.4 MiB for their binary BaseFold benchmark. In this work, we develop a binary BaseFold variant that further reduces proof sizes, using the nontrivial enhancement whereby we skip FRI round oracles (see Section 4). Incorporating our oracle-skipping technique, as well as various further concrete proof size optimizations (described in Subsection 5.2), we obtain a proof size of 0.633 MiB in the $\ell = 28$ case. It seems plausible that these enhancements could improve Blaze's proof sizes too.

Blaze's RAA code depends on a randomized, transparent setup, which involves an expensive verification procedure, itself necessary to bound that setup's probability of failure. That test must be independently rerun by each of the protocol's users; it requires more than a day of computation on a laptop [Bre+25, § 1.1]. The outcome of that test could, in theory, be attested to by a further SNARK, or else checked more quickly with the aid of a GPU-accelerated implementation. Blaze's RAA setup procedure must be carried out independently for each instance size and each code rate.

Blaze's failure analysis assumes that its sampler's coins are uniform. In order to make that analysis applicable in practice, Blaze's sampler must use public, nothing-up-my-sleeve randomness. Blaze's security analysis becomes voided if some protocol administrator is able to influence its setup's choice of random seed.

While Blaze supports only cryptographically large fields, its authors note the compatibility of Blaze PCS with this work's ring-switching compiler. This compatibility calls to mind the small-field scheme which would arise upon the instantiation of our ring-switching reduction (see Section 3) on the Blaze large-field PCS; the resulting small-field scheme would yield an interesting alternative to our combined scheme (see Section 5).

Acknowledgements. We would like to acknowledge our colleagues at Irreducible for their insights and contributions to the *Binius* implementation of these techniques. We would like to gratefully thank Benedikt Bünz, Giacomo Fenzi, Angus Gruen, Ulrich Haböck, Joseph Johnston, Raju Krishnamoorthy, Eugene Rabinovich, Justin Thaler and Benjamin Wilson, whose collective comments and suggestions contributed significantly to this work. We thank Ron Rothblum for patiently explaining code-switching to us.

2 Background and Notation

We write $\mathbb{N}$ for the nonnegative integers. All fields in this work are finite. We fix a binary field L. For each $\ell \in \mathbb{N}$, we write $\mathcal{B}_\ell$ for the ℓ-dimensional *boolean hypercube* $\{0,1\}^\ell \subset L^\ell$. We occasionally identify $\mathcal{B}_\ell$ with the integer range $\{0, \ldots 2^\ell - 1\}$ by mapping $v \mapsto \{v\} \coloneqq \sum_{i=0}^{\ell-1} 2^i \cdot v_i$. The *rings* we treat are nonzero and commutative with unit. For our purposes, an *algebra* A over a field L is a commutative ring A together with an embedding of rings $L \hookrightarrow A$. For L a field and $R \subset L^\vartheta$ a subset, we write $\mu(R) \coloneqq \frac{|R|}{|L|^\vartheta}$.

2.1 Multilinear Polynomials

We review various normal forms for multilinear polynomials, following [DP25, § 2.1]. An ℓ-variate polynomial in $L[X_0, \ldots, X_{\ell-1}]$ is *multilinear* if each of its indeterminates appears with individual degree at most 1; we write $L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$ for the set of multilinear polynomials over L in ℓ indeterminates. Clearly, the set of monomials $(1, X_0, X_1, X_0 \cdot X_1, \ldots, X_0 \cdot \cdots \cdot X_{\ell-1})$ yields a L-basis for $L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$; we call this basis the *multilinear monomial basis* in ℓ variables.

We introduce the $2 \cdot \ell$-variate polynomial

$$\widetilde{\texttt{eq}}(X_0, \ldots, X_{\ell-1}, Y_0, \ldots, Y_{\ell-1}) \coloneqq \prod_{i=0}^{\ell-1} (1 - X_i) \cdot (1 - Y_i) + X_i \cdot Y_i.$$

It is essentially the content of Thaler [Tha22, Fact. 3.5]) that the set $(\widetilde{\texttt{eq}}(X_0, \ldots, X_{\ell-1}, w_0, \ldots, w_{\ell-1}))_{w \in \mathcal{B}_\ell}$ yields a further L-basis of the space $L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.

For each fixed $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, the vector $(\widetilde{\texttt{eq}}(r_0, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell-1}))_{w \in \mathcal{B}_\ell}$ takes the form

$$\left(\prod_{i=0}^{\ell-1} r_i \cdot w_i + (1 - r_i) \cdot (1 - w_i) \right)_{w \in \mathcal{B}_\ell} = ((1 - r_0) \cdot \cdots \cdot (1 - r_{\ell-1}), \ldots, r_0 \cdot \cdots \cdot r_{\ell-1}).$$

We call this vector the *tensor product expansion* of $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and denote it by $\bigotimes_{i=0}^{\ell-1}(1 - r_i, r_i)$. We note that it can be computed in 2^ℓ L-additions and 2^ℓ L-multiplications (see e.g. [Tha22, Lem. 3.8]).

As a notational device, we introduce the further $2 \cdot \ell$-variate polynomial:

$$\widetilde{\texttt{mon}}(X_0, \ldots, X_{\ell-1}, Y_0, \ldots, Y_{\ell-1}) \coloneqq \prod_{i=0}^{\ell-1} 1 + (X_i - 1) \cdot Y_i;$$

we note that $(\widetilde{\texttt{mon}}(X_0, \ldots, X_{\ell-1}, w_0, \ldots, w_{\ell-1}))_{w \in \mathcal{B}_\ell}$ is the multilinear monomial basis in ℓ indeterminates.

2.2 Error-Correcting Codes

We recall details on codes, referring throughout to Guruswami [Gur06]. 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 fix a field L. A linear $[n, k, d]$*-code* over L is a k-dimensional linear subspace $C \subset L^n$ for which $d(v_0, v_1) \geq d$ holds for each unequal pair of elements v_0 and v_1 of C. The *unique decoding radius* of the $[n, k, d]$-code $C \subset L^n$ is $\left\lfloor \frac{d-1}{2} \right\rfloor$; indeed, we note that, for each word $u \in L^n$, at most one codeword $v \in C$ satisfies $d(u, v) < \frac{d}{2}$ (this fact is a direct consequence of the triangle inequality). For $u \in L^n$ arbitrary, we write $d(u, C) \coloneqq \min_{v \in C} d(u, v)$ for the *distance* between u and the code C.

For each linear code $C \subset L^n$ and each integer $m \geq 1$, we define C's *m-fold interleaved code* as the subset $C^m \subset (L^n)^m \cong (L^m)^n$. We understand this latter set as a length-n block code over the alphabet L^m. In particular, its elements are essentially matrices in $L^{m \times n}$ each of whose rows is a C-element. We write matrices $(u_i)_{i=0}^{m-1} \in L^{m \times n}$ row-wise. By definition of C^m, two matrices in $L^{m \times n}$ *differ* at a column if they differ at any of that column's components. That a matrix $(u_i)_{i=0}^{m-1} \in L^{m \times n}$ is within distance e to the code C^m—in which event we write $d^m\left((u_i)_{i=0}^{m-1}, C^m\right) \leq e$—thus entails precisely that there exists a subset $D \coloneqq \Delta^m\left((u_i)_{i=0}^{m-1}, C^m\right)$, 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 recall Reed–Solomon codes (see [Gur06, Def. 2.3]). For notational convenience, we consider only Reed–Solomon codes whose message and block lengths are powers of two. We fix nonnegative *message length* and *rate* parameters ℓ and $\mathcal{R}$, as well as a subset $S \subset L$ of size $2^{\ell+\mathcal{R}}$. We write $C \subset L^{2^{\ell+\mathcal{R}}}$ for the *Reed–Solomon* code $\mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$, itself defined to be the set $\left\{ (P(x))_{x \in S} \;\middle|\; P(X) \in L[X]^{\prec 2^\ell} \right\}$. That is, $\mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$ is the set of those $2^{\ell+\mathcal{R}}$-tuples which arise as the evaluations of some polynomial of degree less than 2^ℓ over S. The distance of $\mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$ is $d = 2^{\ell+\mathcal{R}} - 2^\ell + 1$. We write $\mathsf{Enc} : L[X]^{\prec 2^\ell} \to L^S$ for the *encoding function* which maps $P(X)$ to its tuple of evaluations over S.

We recall the *Berlekamp–Welch* algorithm for Reed–Solomon decoding within the unique decoding radius (see [Gur06, Rem. 4]).

Algorithm 1 (Berlekamp–Welch [Gur06, Rem. 4].)

procedure DecodeReedSolomon$\left((f(x))_{x \in S}\right)$
  allocate $A(X)$ and $B(X)$ of degrees $\left\lfloor \frac{d-1}{2} \right\rfloor$ and $2^{\ell+\mathcal{R}} - \left\lfloor \frac{d-1}{2} \right\rfloor - 1$; write $Q(X,Y) \coloneqq A(X) \cdot Y + B(X)$.
  interpret the equalities $Q(x, f(x)) = 0$, for $x \in S$, as a system of $2^{\ell+\mathcal{R}}$ equations in $2^{\ell+\mathcal{R}}+1$ unknowns.
  by finding a nonzero solution of this linear system, obtain values for the polynomials $A(X)$ and $B(X)$.
  if $A(X) \nmid B(X)$ **then return** $\perp$.
  write $P(X) \coloneqq -B(X)/A(X)$.
  return $P(X)$.

We note that the unknown polynomial $Q(X,Y)$ above indeed has $\left\lfloor \frac{d-1}{2} \right\rfloor + 1 + 2^{\ell+\mathcal{R}} - \left\lfloor \frac{d-1}{2} \right\rfloor = 2^{\ell+\mathcal{R}} + 1$ coefficients, as required.

Upon being given an input word $f : S \to L$ for which $d(f, C) < \frac{d}{2}$, Algorithm 1 necessarily returns the unique polynomial $P(X)$ of degree less than 2^ℓ for which $d(f, \mathsf{Enc}(P(X))) < \frac{d}{2}$ holds. Indeed, this is simply the correctness of Berlekamp–Welch algorithm on input assumed to reside within the unique decoding radius; we refer to [Gur06, Rem. 4] for a thorough treatment. (We note that the $(1, 2^\ell - 1)$-weighted degree of $Q(X,Y)$ is at most $D = 2^{\ell+\mathcal{R}} - \left\lfloor \frac{d-1}{2} \right\rfloor - 1$, while $t = 2^{\ell+\mathcal{R}} - \left\lfloor \frac{d-1}{2} \right\rfloor$; the hypothesis of [Gur06, Lem. 4.3] is therefore fulfilled. We conclude that $Q(X, P(X))$, of degree at most D with at least t zeros, is in fact identically zero, so that $Y - P(X) \mid Q(X,Y)$.)

We discuss this algorithm further in Remark 4.26 below.

2.3 The Novel Polynomial Basis

We recall in detail the *novel polynomial basis* of Lin, Chung and Han [LCH14, § II. C.]. We fix again a binary field L, of degree r, say, over $\mathbb{F}_2$. For our purposes, a *subspace polynomial* over L is a polynomial $W(X) \in L[X]$ which splits completely over L, and whose roots, each of multiplicity 1, form an $\mathbb{F}_2$-linear subspace of L. For a detailed treatment of subspace polynomials, we refer to Lidl and Niederreiter [LN96, Ch. 3. § 4.]. For each subspace polynomial $W(X) \in L[X]$, the evaluation map $W : L \to L$ is $\mathbb{F}_2$-linear.

For each fixed $\ell \in \{0, \ldots, r-1\}$, the set $L[X]^{\prec 2^\ell}$ of polynomials of degree less than 2^ℓ is a 2^ℓ-dimensional vector space over L. Of course, the set $(1, X, X^2, \ldots, X^{2^\ell - 1})$ yields a natural L-basis of $L[X]^{\prec 2^\ell}$. Lin, Chung and Han define a further L-basis of $L[X]^{\prec 2^\ell}$—called the *novel polynomial basis*—in the following way. We fix once and for all an $\mathbb{F}_2$-basis $(\beta_0, \ldots, \beta_{r-1})$ of L (which we view as an r-dimensional vector space over its subfield $\mathbb{F}_2$). For each $i \in \{0, \ldots, \ell\}$, we write $U_i \coloneqq \langle \beta_0, \ldots, \beta_{i-1} \rangle$ for the $\mathbb{F}_2$-linear span of the prefix $(\beta_0, \ldots, \beta_{i-1})$, and define the *subspace vanishing polynomial* $W_i(X) \coloneqq \prod_{u \in U_i} (X - u)$, as well as its *normalized* variant $\widehat{W}_i(X) \coloneqq \frac{W_i(X)}{W_i(\beta_i)}$ (we note that $\beta_i \notin U_i$, so that $W_i(\beta_i) \neq 0$). In words, for each $i \in \{0, \ldots, \ell\}$, $W_i(X)$ vanishes precisely on $U_i \subset L$; $\widehat{W}_i(X)$ moreover satisfies $\widehat{W}_i(X)(\beta_i) = 1$. Finally, for each $j \in \{0, \ldots, 2^\ell - 1\}$, we write $(j_0, \ldots, j_{\ell-1})$ for the bits of j—so that $j = \sum_{k=0}^{\ell-1} 2^k \cdot j_k$ holds—and set $X_j(X) \coloneqq \prod_{i=0}^{\ell-1} \widehat{W}_i(X)^{j_i}$. We note that, for each $j \in \{0, \ldots, 2^\ell - 1\}$, $X_j(X)$ is of degree j. We conclude that the change-of-basis matrix from $(1, X, \ldots, X^{2^\ell - 1})$ to $(X_0(X), X_1(X), \ldots, X_{2^\ell - 1}(X))$ is triangular (with an everywhere-nonzero diagonal), so that this latter list indeed yields a L-basis of $L[X]^{\prec 2^\ell}$.

We now fix moreover a rate parameter $\mathcal{R} \in \{1, \ldots, r - \ell\}$ and a union $S \subset L$ of $2^{\mathcal{R}}$ distinct cosets of $U_\ell = \langle \beta_0, \ldots, \beta_{\ell-1} \rangle$. For example, we may take as $S \subset L$ any affine translate of the $\ell + \mathcal{R}$-dimensional subspace $\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle$. For each $S \subset L$ of this form, Lin, Chung and Han [LCH14, § III. B.]'s $O(\ell \cdot 2^{\ell+\mathcal{R}})$-time algorithm serves to compute, on input the polynomial $P(X) \coloneqq \sum_{j=0}^{2^\ell-1} a_j \cdot X_j(X)$ (expressed in coordinates with respect to the novel polynomial basis), its encoding $(P(x))_{x \in S}$. In fact, that algorithm takes exactly $\ell \cdot 2^{\ell+\mathcal{R}}$ L-additions and $\ell \cdot 2^{\ell+\mathcal{R}-1}$ L-multiplications [LCH14, § III. D.].

In Remark 4.14 below, we suggest a new interpretation of Lin, Chung and Han's algorithm [LCH14, § III.] based on the techniques of this paper. For now, for self-containedness, we record here the key algorithm in full, in our notation. We note that Algorithm 2's equivalence with [LCH14, § III.] is not obvious; we explain the correctness of our description in Remark 4.14 below. In what follows, we fix as above the degree and rate parameters ℓ and $\mathcal{R}$. We finally fix a polynomial $P(X) = \sum_{j=0}^{2^\ell-1} a_j \cdot X_j(X)$; we write $b : \mathcal{B}_{\ell+\mathcal{R}} \to L$ for $(a_j)_{j=0}^{2^\ell-1}$'s $2^{\mathcal{R}}$-fold tiling; in other words, for each $v \in \mathcal{B}_{\ell+\mathcal{R}}$, we set $b(v_0, \ldots, v_{\ell+\mathcal{R}-1}) \coloneqq a_{\{(v_0, \ldots, v_{\ell-1})\}}$.

Algorithm 2 (Lin–Chung–Han [LCH14, § III.].)

```
procedure AdditiveNTT((b(v))_{v ∈ B_{ℓ+R}})
    for i ∈ {ℓ − 1, ..., 0} (i.e., in downward order) do
        for (u, v) ∈ B_{ℓ+R−i−1} × B_i do
            define the twiddle factor t := Σ_{k=0}^{ℓ+R−i−2} u_k · Ŵ_i(β_{i+1+k}).
            overwrite first b(u ‖ 0 ‖ v) += t · b(u ‖ 1 ‖ v) and then b(u ‖ 1 ‖ v) += b(u ‖ 0 ‖ v).
    return (b(v))_{v ∈ B_{ℓ+R}}.
```

We note that the twiddle factor t above depends only on u, and not on v, and can be reused accordingly. Finally, in the final return statement above, we implicitly identify $\mathcal{B}_{\ell+\mathcal{R}} \cong S$ using the standard basis $\beta_0, \ldots, \beta_{\ell+\mathcal{R}-1}$ of the latter space (see also Subsection 4.1 below).

2.4 FRI

We recall Ben-Sasson, Bentov, Horesh and Riabzev's [BBHR18a] *Fast Reed–Solomon Interactive Oracle Proof of Proximity* (FRI). For L a binary field, and size and rate parameters ℓ and $\mathcal{R}$ fixed, FRI yields an IOP of proximity for the Reed–Solomon code $\mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$; here, we require that $S \subset L$ be an affine, $\mathbb{F}_2$-linear subspace (of dimension $\ell + \mathcal{R}$, of course). That is, FRI yields an IOP for the claim whereby some oracle $[f]$—i.e., representing a function $f : S \to L$—is close to a codeword $(P(x))_{x \in S}$ (here, $P(X) \in L[X]^{\prec 2^\ell}$ represents a polynomial of degree less than 2^ℓ). FRI's verifier complexity is polylogarithmic in 2^ℓ. We abbreviate $\rho \coloneqq 2^{-\mathcal{R}}$, so that $\mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$ is of rate ρ.

Internally, FRI makes use of a folding constant η—which we fix to be 1—as well as a fixed, global sequence of subspaces and maps of the form:

$$S = S^{(0)} \xrightarrow{q^{(0)}} S^{(1)} \xrightarrow{q^{(1)}} S^{(2)} \xrightarrow{q^{(2)}} \cdots \xrightarrow{q^{(\ell-1)}} S^{(\ell)}. \tag{26}$$

Here, for each $i \in \{0, \ldots, \ell - 1\}$, $q^{(i)}$ is a subspace polynomial of degree $2^\eta = 2$, whose kernel, which is 1-dimensional, is moreover contained in $S^{(i)}$. By linear-algebraic considerations, we conclude that $S^{(i+1)}$'s $\mathbb{F}_2$-dimension is 1 less than $S^{(i)}$'s is; inductively, we conclude that each $S^{(i)}$ is of dimension $\ell + \mathcal{R} - i$. In particular, $S^{(\ell)}$ is of dimension $\mathcal{R}$.

2.5 Tensor Products of Fields

We record algebraic preliminaries, referring throughout to Lang [Lan02, Ch. XVI]. We fix a field extension L / K. We define the *tensor product* $A \coloneqq L \otimes_K L$ of L with itself over K as in [Lan02, Ch. XVI § 6]. Here, we view L as a K-algebra; the resulting object $A \coloneqq L \otimes_K L$ is likewise a K-algebra. We would like to sincerely thank Benjamin Wilson for first suggesting to us this tensor-theoretic perspective on the tower algebra of [DP25, § 3.4].

We recall from [Lan02, Ch. XVI, § 1] the natural K-bilinear mapping $\varphi : L \times L \to L \otimes_K L$ which sends $\varphi : (\alpha_0, \alpha_1) \mapsto \alpha_0 \otimes \alpha_1$. We write φ_0 and φ_1 for φ's restrictions to the subsets $L \times \{1\}$ and $\{1\} \times L$ of $L \times L$, and moreover identify these latter subsets with L. That is, we write $\varphi_0 : \alpha \mapsto \alpha \otimes 1$ and $\varphi_1 : \alpha \mapsto 1 \otimes \alpha$, both understood as maps $L \to A$. We claim that these maps are injective (i.e., that they're not identically zero). We follow Lang [Lan02, Ch. XVI, § 2, Prop. 2.3]. The mapping $f : L \times L \to L$ sending $f : (\alpha_0, \alpha_1) \mapsto \alpha_0 \cdot \alpha_1$ is K-bilinear; by the universal property of the tensor product, f induces a K-linear map $h : L \otimes_K L \to L$, for which, for each $\alpha \in L$, $h(\alpha \otimes 1) = f(\alpha, 1) = \alpha \cdot 1 = \alpha$ holds; we see that $\alpha \otimes 1 = 0$ if and only if $\alpha = 0$.

We assume once and for all that $\deg(L \,/\, K)$ is a power of 2, say 2^κ. We fix a K-basis $(\beta_v)_{v \in \mathcal{B}_\kappa}$ of L. We moreover impose the simplifying assumption whereby $\beta_{(0,\ldots,0)} = 1$. By [Lan02, Ch. XVI, § 2, Cor. 2.4], the set $(\beta_u \otimes \beta_v)_{(u,v) \in \mathcal{B}_\kappa \times \mathcal{B}_\kappa}$ yields a K-basis of A. We thus see that each A-element is, concretely, a $2^\kappa \times 2^\kappa$ array of K-elements. For each $a \in A$ given, there is a unique 2^κ-tuple of L-elements $(a_v)_{v \in \mathcal{B}_\kappa}$ for which $a = \sum_{v \in \mathcal{B}_\kappa} a_v \otimes \beta_v$ holds. (Indeed, this is just [Lan02, Ch. XVI, § 2, Prop. 2.3].) Similarly, there is a unique 2^κ-tuple of L-elements $(a_u)_{u \in \mathcal{B}_\kappa}$ for which $a = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes a_u$ holds. We call the tuples $(a_v)_{v \in \mathcal{B}_\kappa}$ and $(a_u)_{u \in \mathcal{B}_\kappa}$ a's *column* and *row* representations, respectively. We depict the tensor algebra in Figure 9 below.

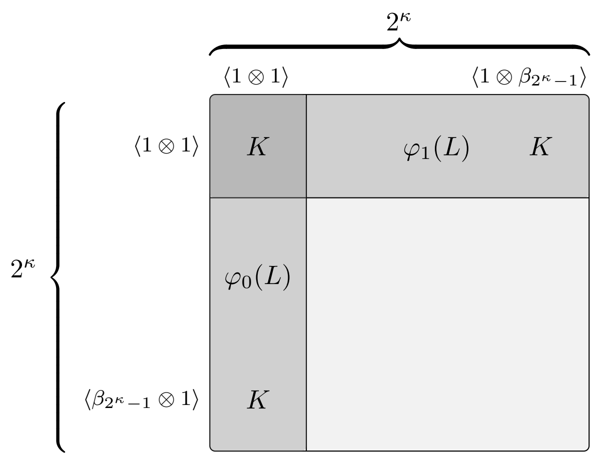

Figure 9: A depiction of our "tensor algebra" data structure.

The maps φ_0 and φ_1 respectively embed L into A's left-hand column and top row. That is, the image of $\varphi_0 : L \hookrightarrow A$ is the set of K-arrays which are 0 except in their respective left-most columns; the image of $\varphi_1 : L \hookrightarrow A$ is the set of K-arrays which are 0 outside of their top rows. We finally characterize concretely the products $\varphi_0(\alpha) \cdot a$ and $\varphi_1(\alpha) \cdot a$, for elements $\alpha \in L$ and $a \in A$ arbitrary. It is a straightforward to show that $\varphi_0(\alpha) \cdot a = \sum_{v \in \mathcal{B}_\kappa} (\alpha \cdot a_v) \otimes \beta_v$ and $\varphi_1(\alpha) \cdot a = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes (\alpha \cdot a_u)$ both hold; here, we again write $(a_v)_{v \in \mathcal{B}_\kappa}$ and $(a_u)_{u \in \mathcal{B}_\kappa}$ for a's column and row representations. That is, $\varphi_0(\alpha) \cdot a$ differs from a by column-wise multiplication by α; $\varphi_1(\alpha) \cdot a$ differs from a by row-wise multiplication by α. In short, φ_0 operates on columns; φ_1 operates on rows.

Below, the tensor algebra $A \coloneqq L \otimes_K L$ plays a critical role in our "ring-switching" technique (see Section 3). For now, we record a simple polynomial-packing operation, which is implicit in [DP25, § 3.4]. We obtain a natural K-isomorphism $K^{2^\kappa} \to L$ via the basis-combination procedure $(\alpha_v)_{v \in \mathcal{B}_\kappa} \mapsto \sum_{v \in \mathcal{B}_\kappa} \alpha_v \cdot \beta_v$. By applying this map in chunks, we may associate to each ℓ-variate K-multilinear an $\ell - \kappa$-variate L-multilinear.

Definition 2.1. For each extension L/K, with K-basis $(\beta_v)_{v \in \mathcal{B}_\kappa}$ say, and each multilinear $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, we write $\ell' \coloneqq \ell - \kappa$, and define the *packed polynomial* $t'(X_0, \ldots, X_{\ell'-1}) \in L[X_0, \ldots, X_{\ell'-1}]^{\preceq 1}$ by declaring, for each $w \in \mathcal{B}_{\ell'}$, that $t' : w \mapsto \sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \cdot \beta_v$.

Definition 2.1 replaces each little-endian chunk—consisting of 2^κ adjacent K-elements—of $t(X_0, \ldots, X_{\ell-1})$'s Lagrange coefficient vector with a single L-element, by basis-combining that chunk.

We emphasize that Definition 2.1's packing procedure is reversible (see also [DP25, Thm. 3.9]); that is, $t'(X_0, \ldots, X_{\ell'-1})$ can be "unpacked". We note that Definition 2.1 is essentially the same as [DP25, § 4.3].

We finally write $\varphi_1(t')(X_0, \ldots, X_{\ell'-1}) \in A[X_0, \ldots, X_{\ell'-1}]$ for the result of embedding $t'(X_0, \ldots, X_{\ell'-1})$ componentwise along the inclusion $\varphi_1 : L \hookrightarrow A$.

2.6 Binary Towers

We recall towers of binary fields, referring throughout to [DP25, § 2.3]. For simplicity, we present only Wiedemann's tower [Wie88]; on the other hand, our results go through without change on other binary towers (cf. e.g. the *Cantor tower* given in Li et al. [Li+18, § 2.1]). That is, we set $\mathcal{T}_0 := \mathbb{F}_2$ and $\mathcal{T}_1 := \mathbb{F}_2[X_0]/(X_0^2 + X_0 + 1)$, as well as, for each $\iota > 1$, $\mathcal{T}_\iota := \mathcal{T}_{\iota-1}/(X_{\iota-1}^2 + X_{\iota-2} \cdot X_{\iota-1} + 1)$. Fan and Paar [FP97] observe that the multiplication and inversion operations in Wiedemann's tower admit $O\left(2^{\log(3)\cdot\iota}\right)$-time algorithms.

The *monomial* $\mathbb{F}_2$-*basis* of the binary tower $\mathcal{T}_\tau$ is $(\beta_v)_{v\in\mathcal{B}_\tau} := (\widetilde{\texttt{mon}}(X_0, \ldots, X_{\tau-1}, v_0, \ldots, v_{\tau-1}))_{v\in\mathcal{B}_\tau}$. More generally, for each pair of integers $\iota \geq 0$ and $\tau \geq \iota$, the set $(\widetilde{\texttt{mon}}(X_\iota, \ldots, X_{\tau-1}, v_0, \ldots, v_{\tau-\iota-1})_{v\in\mathcal{B}_{\tau-\iota}}$ likewise yields a $\mathcal{T}_\iota$-basis of $\mathcal{T}_\tau$; we again write $(\beta_v)_{v\in\mathcal{B}_{\tau-\iota}}$ for this basis.

The *tower algebra* data structure of Diamond and Posen [DP25, § 3.4] is essentially nothing other than $\mathcal{T}_\tau \otimes_{\mathcal{T}_\iota} \mathcal{T}_{\iota+\kappa}$. We use tensor-notation in this work; we thus avoid referring to that algebra directly. In this work, we moreover only consider "square" tensors (i.e., of the same field with itself). That work's "constant" and "synthetic" embeddings correspond to our embeddings φ_0 and φ_1, respectively.

2.7 Proximity Gaps

We turn to proximity gaps, following Ben-Sasson, et al., [Ben+23], Diamond and Posen [DP24], and Diamond and Gruen [DG25]. Throughout this subsection, we again fix a Reed–Solomon code $C := \mathsf{RS}_{L,S}[2^{\ell+\mathcal{R}}, 2^\ell]$; we moreover write $d := 2^{\ell+\mathcal{R}} - 2^\ell + 1$ for C's distance. In the following results, for notational convenience, we abbreviate $n := 2^{\ell+\mathcal{R}}$ for the Reed–Solomon code C's block length.

We recall the notion of *proximity gaps*, both over affine lines [DG25, Def. 1] and over tensor combinations [DG25, Def. 2]. The following key result entails that Reed–Solomon codes exhibit proximity gaps for affine lines, for each proximity parameter $e \in \left\{0, \ldots, \left\lfloor \frac{d-1}{2} \right\rfloor\right\}$ within the unique decoding radius.

Theorem 2.2 (Ben-Sasson, et al. [Ben+23, Thm. 4.1]). *For each proximity parameter $e \in \left\{0, \ldots, \left\lfloor \frac{d-1}{2} \right\rfloor\right\}$ and each pair of words u_0 and u_1 in $L^{2^{\ell+\mathcal{R}}}$, if*

$$\Pr_{r\in L}[d((1-r)\cdot u_0 + r \cdot u_1, C) \leq e] > \frac{n}{|L|},$$

then $d^2\left((u_i)_{i=0}^1, C^2\right) \leq e$.

Diamond and Gruen [DG25, Thm. 2], making use of a result of Angeris, Evans and Roh [AER24] (see also [DG25, Thm. 3]), show that each code C for which the conclusion of Theorem 2.2 holds also exhibits *tensor-style proximity gaps* in the sense of Diamond and Posen [DP24, Thm. 2] (although they sharpen by a factor of two that result's false witness probability). Applying their result to Theorem 2.2, those authors obtain:

Theorem 2.3 (Diamond–Gruen [DG25, Cor. 1]). *For each proximity parameter $e \in \left\{0, \ldots, \left\lfloor \frac{d-1}{2} \right\rfloor\right\}$, each tensor arity $\vartheta \geq 1$, and each list of words $u_0, \ldots, u_{2^\vartheta-1}$ in $L^{2^{\ell+\mathcal{R}}}$, if*

$$\Pr_{(r_0,\ldots,r_{\vartheta-1})\in L^\vartheta}\left[d\left(\begin{bmatrix} & \bigotimes_{i=0}^{\vartheta-1}(1-r_i, r_i) & \end{bmatrix} \cdot \begin{bmatrix} - & u_0 & - \\ & \vdots & \\ - & u_{2^\vartheta-1} & - \end{bmatrix}, C\right) \leq e\right] > \vartheta \cdot \frac{n}{|L|},$$

then $d^{2^\vartheta}\left((u_i)_{i=0}^{2^\vartheta-1}, C^{2^\vartheta}\right) \leq e$.

2.8 Security Definitions

We record security definitions. We begin by defining various abstract oracles, following [DP25, § 4.1].

FUNCTIONALITY 2.4 ($\mathcal{F}_{\mathsf{Vec}}^{L}$—vector oracle)**.**
An arbitrary alphabet L is given.

- Upon receiving $(\texttt{submit}, m, f)$ from $\mathcal{P}$, where $m \in \mathbb{N}$ and $f \in L^{\mathcal{B}_m}$, output $(\texttt{receipt}, L, [f])$ to all parties, where $[f]$ is some unique handle onto the vector f.
- Upon receiving $(\texttt{query}, [f], v)$ from $\mathcal{V}$, where $v \in \mathcal{B}_m$, send $\mathcal{V}$ $(\texttt{result}, f(v))$.

FUNCTIONALITY 2.5 ($\mathcal{F}_{\mathsf{Poly}}^{\lambda,\ell}$—polynomial oracle)**.**
A security parameter $\lambda \in \mathbb{N}$ and a number-of-variables parameter $\ell \in \mathbb{N}$ are given. The functionality constructs and fixes a field L (allowed to depend on λ and ℓ).

- Upon receiving $(\texttt{submit}, t)$ from $\mathcal{P}$, where $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, output $(\texttt{receipt}, [t])$ to all parties, where $[t]$ is some unique handle onto the polynomial t.
- On input $(\texttt{query}, [t], r)$ from $\mathcal{V}$, where $r \in L^{\ell}$, send $\mathcal{V}$ $(\texttt{result}, t(r_0, \ldots, r_{\ell-1}))$.

FUNCTIONALITY 2.6 ($\mathcal{F}_{\mathsf{SFPoly}}^{\lambda,K,\ell}$—small-field polynomial oracle)**.**
A security parameter $\lambda \in \mathbb{N}$, a number-of-variables parameter $\ell \in \mathbb{N}$, and a ground field K are given. The functionality constructs and fixes a field extension L / K (allowed to depend on λ, ℓ and K).

- Upon receiving $(\texttt{submit}, t)$ from $\mathcal{P}$, where $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, output $(\texttt{receipt}, [t])$ to all parties, where $[t]$ is some unique handle onto the polynomial t.
- On input $(\texttt{query}, [t], r)$ from $\mathcal{V}$, where $r \in L^{\ell}$, send $\mathcal{V}$ $(\texttt{result}, t(r_0, \ldots, r_{\ell-1}))$.

An *IOP*, by definition, is a protocol in which $\mathcal{P}$ and $\mathcal{V}$ may make free use of the abstract Functionality 2.4; in a *PIOP*, the parties may instead use Functionality 2.5. *Interactive oracle polynomial commitment schemes* (IOPCSs) serve to bridge these two models. They're IOPs; that is, they operate within the abstract computational model in which Functionality 2.4 is assumed to exist. On the other hand, they "emulate" the more-powerful Functionality 2.5, in the sense that each given PIOP—by inlining in place of each of its calls to Functionality 2.5 an execution of the IOPCS—stands to yield an equivalently secure IOP.

Departing slightly from previous works, we treat polynomial commitment in the IOP model; that is, for our purposes, a "polynomial commitment scheme" is an IOP (i.e., a protocol in which a string oracle is available to both parties) which captures the commitment, and subsequently the evaluation, of a polynomial. Our approach contrasts with that taken by various previous works (we note e.g. Diamond and Posen [DP25] and Setty [Set20]). Those works opt to define polynomial commitment schemes in the *plain* (random oracle) model, noting that a plain PCS, upon being inlined into a secure PIOP, yields a sound argument. This latter approach absorbs the Merklization process both into the PCS and into the composition theorem. Our approach bypasses this technicality, and separates the relevant concerns. Indeed, given a PIOP, we may first inline our IOPCS into it; on the resulting IOP, we may finally invoke generically the compiler of Ben-Sasson, Chiesa and Spooner [BCS16]. This "two-step" compilation process serves to transform any secure PIOP into a secure argument in the random oracle model.

We also define the security of IOPCSs differently than do [Set20, Def. 2.11] and [DP25, § 3.5]. Our definition below requires that $\mathcal{E}$ extract $t(X_0, \ldots, X_{\ell-1})$ immediately after seeing $\mathcal{A}$'s commitment (that is, before seeing r, or observing any evaluation proofs on the part of $\mathcal{A}$). This work's IOPCS constructions indeed meet this stricter requirement, owing essentially to their use of Reed–Solomon codes, which are efficiently decodable. (In the setting of general—that is, not-necessarily-decodable—codes, extraction becomes much more complicated, and requires rewinding.) On the other hand, our strict rendition of the *IOPCS* notion makes its key composability property—that is, the fact whereby a secure IOPCS, upon being inlined into a secure PIOP, yields a secure IOP—easier to prove. (We believe that this composability property should, on the other hand, nonetheless hold even under various weakenings of Definition 2.8.)

Definition 2.7. An *interactive oracle polynomial commitment scheme* (IOPCS) is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ with the following syntax:

- $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell)$. On input the security parameter $\lambda \in \mathbb{N}$ and a number-of-variables parameter $\ell \in \mathbb{N}$, outputs params, which includes, among other things, a field L.
- $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input params and a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, outputs a handle $[f]$ to a vector.
- $b \leftarrow \langle \mathcal{P}([f], s, r; t), \mathcal{V}([f], s, r) \rangle$ is an IOP, in which the parties may jointly leverage the machine $\mathcal{F}^L_{\mathsf{Vec}}$. The parties have as common input a vector handle $[f]$, an evaluation point $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and a claimed evaluation $s \in L$. $\mathcal{P}$ has as further input a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$. $\mathcal{V}$ outputs a success bit $b \in \{0, 1\}$.

The IOPCS Π is *complete* if the obvious correctness property holds. That is, for each multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$ and each honestly generated commitment $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$, it should hold that, for each $r \in L^\ell$, setting $s := t(r_0, \ldots, r_{\ell-1})$, the honest prover algorithm induces the verifier to accept with probability 1, so that $\langle \mathcal{P}([f], s, r; t), \mathcal{V}([f], s, r) \rangle = 1$.

We now define the security of IOPCSs.

Definition 2.8. For each interactive oracle polynomial commitment scheme Π, security parameter $\lambda \in \mathbb{N}$, number-of-variables parameter $\ell \in \mathbb{N}$, PPT adversary $\mathcal{A}$, and PPT emulator $\mathcal{E}$, we define the following experiment:

- The experimenter samples $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell)$, and gives params, including L, to $\mathcal{A}$ and $\mathcal{E}$.
- The adversary, after interacting arbitrarily with the vector oracle, outputs a handle $[f] \leftarrow \mathcal{A}(\mathsf{params})$.
- On input $\mathcal{A}$'s record of interactions with the oracle, $\mathcal{E}$ outputs $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.
- The verifier outputs $(r_0, \ldots, r_{\ell-1}) \leftarrow \mathcal{V}(\mathsf{params}, [f])$; $\mathcal{A}$ responds with an evaluation claim $s \leftarrow \mathcal{A}(r)$.
- By running the evaluation IOP with $\mathcal{A}$ as $\mathcal{V}$, the experimenter obtains the bit $b \leftarrow \langle \mathcal{A}(s, r), \mathcal{V}([f], s, r) \rangle$.
- The experimenter defines two quantities:
  - $\mathsf{Real}^{\Pi,\ell}_{\mathcal{A}}(\lambda)$: is defined to be s if $b = 1$, and $\perp$ otherwise.
  - $\mathsf{Ideal}^{\Pi,\ell}_{\mathcal{E},\mathcal{A}}(\lambda)$: is defined to be $t(r_0, \ldots, r_{\ell-1})$ if $t(X_0, \ldots, X_{\ell-1}) \neq \perp$ and $b = 1$, and $\perp$ otherwise.

The IOPCS Π is said to be *secure* if, for each PPT adversary $\mathcal{A}$, there exists a PPT emulator $\mathcal{E}$ and a negligible function negl such that, for each $\lambda \in \mathbb{N}$ and each $\ell \in \mathbb{N}$, $\Pr\left[\mathsf{Real}^{\Pi,\ell}_{\mathcal{A}}(\lambda) \neq \mathsf{Ideal}^{\Pi,\ell}_{\mathcal{E},\mathcal{A}}(\lambda)\right] \leq \mathsf{negl}(\lambda)$.

We finally record a variant of Definition 2.7 in which the parties may fix a small coefficient field K.

Definition 2.9. A *small-field interactive oracle polynomial commitment scheme* (small-field IOPCS) is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ with the following syntax:

- $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, K)$. On input the security parameter $\lambda \in \mathbb{N}$, a number-of-variables parameter $\ell \in \mathbb{N}$ and a field K, outputs params, which includes, among other things, a field extension L / K.
- $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input params and a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, outputs a handle $[f]$ to a vector.
- $b \leftarrow \langle \mathcal{P}([f], s, r; t), \mathcal{V}([f], s, r) \rangle$ is an IOP, in which the parties may jointly leverage the machine $\mathcal{F}^L_{\mathsf{Vec}}$. The parties have as common input a vector handle $[f]$, an evaluation point $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and a claimed evaluation $s \in L$. $\mathcal{P}$ has as further input a multilinear polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$. $\mathcal{V}$ outputs a success bit $b \in \{0, 1\}$.

We define the *security* of small-field IOPCSs Π exactly as in Definition 2.8, except that we require that $\mathcal{E}$ output a polynomial $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.

3 Ring-Switching

In this section, we formally present ring-switching. First, we review in a bit more detail the material of Subsection 1.3. Our main goal is to reframe that subsection's content tensor-theoretically.

As usual, we fix a field extension L / K and a basis $(\beta_v)_{v \in \mathcal{B}_\kappa}$ of L over K. We moreover fix a K-multilinear $t(X_0, \ldots, X_{\ell-1})$ and an evaluation point $(r_0, \ldots, r_{\ell-1})$ over L. Finally, we again write $t'(X_0, \ldots, X_{\ell'-1})$ for $t(X_0, \ldots, X_{\ell-1})$'s packed multilinear (see (1)). Above, we argued that, in order for the verifier to assess the claim $t(r_0, \ldots, r_{\ell-1}) \stackrel{?}{=} s$, it's enough for the prover to send values $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$ respectively *claimed* to equal $(t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}))_{v \in \mathcal{B}_\kappa}$. Provided it can verify these latter claims, the verifier must simply check (3).

At the core of our theory rests the following characterization of the partial evaluations $(t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}))_{v \in \mathcal{B}_\kappa}$.

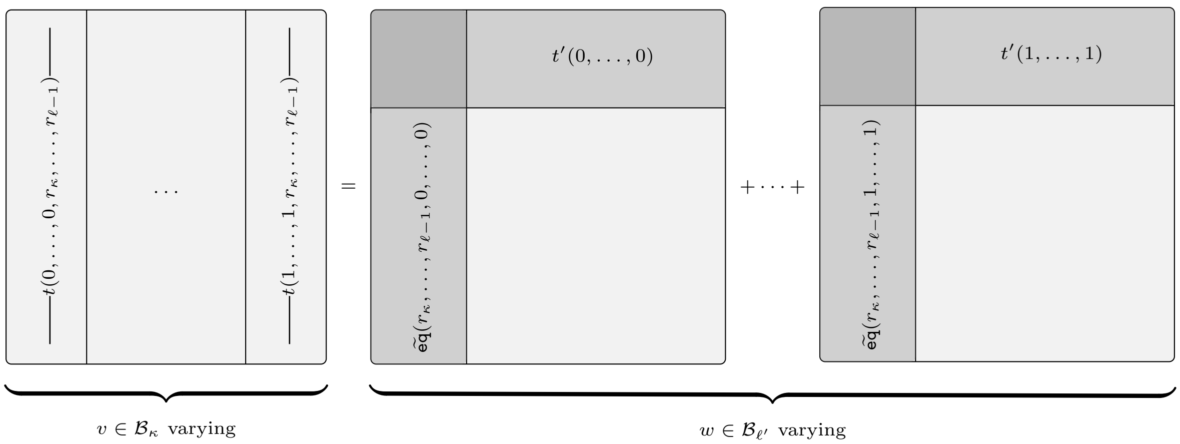

Figure 10: A graphical depiction of $t(X_0, \ldots, X_{\ell-1})$'s partial evaluations.

The partial evaluations $(t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}))_{v \in \mathcal{B}_\kappa}$ appear as the *columns* of the left-hand matrix of Figure 10. In Figure 10's right-hand side, we have the sum, over varying $w \in \mathcal{B}_{\ell'}$, of the matrices $\widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r_\kappa, \ldots, r_{\ell-1}) \star t'(w)$. Here, we again use the $\star$ symbol to denote the "exterior product" between two L-elements (recall Subsection 1.3). That is, we basis-decompose both operands into K-vectors, and then take the $2^\kappa \times 2^\kappa$ K-matrix of cross-products between these vectors.

Figure 10 is true essentially by definition of $t'(X_0, \ldots, X_{\ell'-1})$. Indeed, we fix a column index $v \in \mathcal{B}_\kappa$, and "zoom into" the v^{th} column of Figure 10. We first note that

$$t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}) = \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell'-1}, w_0, \ldots, w_{\ell-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}). \quad (27)$$

Moreover, for each $w \in \mathcal{B}_{\ell'}$, $t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1})$ is exactly the v^{th} horizontal slice of of $t'(w)$ (this is (1) again). Figure 10 thus amounts to the horizontal "stacking" of all 2^κ instances of the relationship (27), as the column index $v \in \mathcal{B}_\kappa$ varies.

On the other hand, by taking the *exact same picture* and viewing it row-wise, we obtain relationships that we sumcheck. As in Subsection 1.3, for each $w \in \mathcal{B}_{\ell'}$, we write $(A_{w,u})_{u \in \mathcal{B}_\kappa}$ for the basis-decomposition of $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})$; that is,

$$\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u \quad (28)$$

holds for each $w \in \mathcal{B}_{\ell'}$ (recall also (8)). Interpreting Figure 10 row-wise, we obtain Figure 11 below, which, again, is true essentially by definition. Indeed, "zooming into" some row $u \in \mathcal{B}_\kappa$ of Figure 11, we note that that Figure 11's right-hand side sums the slice-product $A_{w,u} \cdot t'(w)$ over $w \in \mathcal{B}_{\ell'}$.

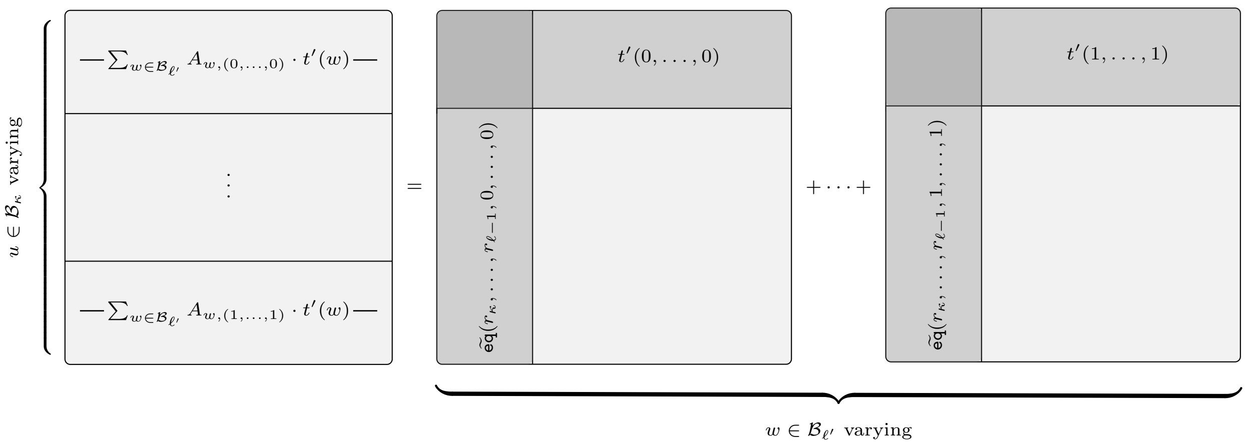

Figure 11: A row-wise viewpoint into the same exact expression.

The point is that the rows of Figure 11 are amenable to the sumcheck protocol—provided, that is, that the verifier can efficiently evaluate $\left(A_u(r'_0,\ldots,r'_{\ell'-1})\right)_{u\in\mathcal{B}_\kappa}$, for some random point $(r'_0,\ldots,r'_{\ell'-1})$ that arises during the sumcheck. Here, for each $u \in \mathcal{B}_\kappa$, we define $A_u(X_0,\ldots,X_{\ell'-1})$ to be the multilinear extension of the map $A_u : w \mapsto A_{w,u}$, for $w \in \mathcal{B}_{\ell'}$ varying. This gets us to where we left off in Subsection 1.3 (recall again Figure 3).

In fact, Figures 3, 10 and 11 all arise naturally as tensor-algebraic expressions. We recall the *tensor algebra* $A := L \otimes_K L$ from Subsection 2.5. Each element of the tensor algebra looks, concretely, like a $2^\kappa \times 2^\kappa$ array of K-elements. We may freely interpret each such element, say $\hat{s} \in A$, both column-wise and row-wise. That is, we can interpret each of $\hat{s}$'s *columns* as an L-element, and so obtain a list $(\hat{s}_v)_{v\in\mathcal{B}_\kappa}$ of L-elements; alternatively, we can interpret each of $\hat{s}$'s *rows* as an L-element, thereby obtaining a further list $(\hat{s}_u)_{u\in\mathcal{B}_\kappa}$. As should be clear by now, our theory hinges on a viewpoint that maintains both of these perspectives at once. The tensor algebra furnishes the arena within which we might best do this.

The important thing about the tensor algebra is its multiplication operation, and in particular how that operation handles columns and rows. Given some L-element α, we can obtain an A-element by inscribing α's K-decomposition into the left-hand *column* of a $2^\kappa \times 2^\kappa$ array. The resulting A-element is $\varphi_0(\alpha)$. Similarly, we may equally inscribe α's K-decomposition into the top *row* of a $2^\kappa \times 2^\kappa$ K-array; this operation yields $\varphi_1(\alpha)$. (We again recall Subsection 2.5.) The important thing about the tensor algebra's multiplication structure is that it captures the $\star$ operation. In fact, for each pair of L-elements α_0 and α_1,

$$\alpha_0 \star \alpha_1 = \varphi_0(\alpha_0) \cdot \varphi_1(\alpha_1). \tag{29}$$

In (29), the left-hand product is of course the exterior product, whereas the right-hand product is the ambient multiplication operation in the tensor algebra.

Figures 10 and 11 represent the tensor-algebric identity

$$\varphi_1(t')(\varphi_0(r_\kappa),\ldots,\varphi_0(r_{\ell-1})) = \sum_{w\in\mathcal{B}_{\ell'}} \varphi_0(\widetilde{\mathsf{eq}}(r_\kappa,\ldots,r_{\ell-1},w_0,\ldots,w_{\ell'-1})) \cdot \varphi_1(t'(w_0,\ldots,w_{\ell'-1})). \tag{30}$$

The identity (30) captures, in just one expression, "both views" (i.e., the views respectively expressed by Figures 10 and 11). Here, we write $\varphi_1(t')(X_0,\ldots,X_{\ell'-1})$ for the A-valued multilinear defined by the Lagrange prescription $w \mapsto \varphi_1(t'(w))$ (i.e., for $w \in \mathcal{B}_{\ell'}$ varying).

Our key claim is that the right-hand side of Figure 3 is simply

$$\widetilde{\mathsf{eq}}\left(\varphi_0(r_\kappa),\ldots,\varphi_0(r_{\ell-1}),\varphi_1(r'_0),\ldots,\varphi_1(r'_{\ell'-1})\right) \tag{31}$$

(recall also (14)). This nontrivial fact is key to our theory; it implies that the verifier may locally compute the right-hand side of Figure 3 succinctly. In fact, the verifier may compute (31) concretely in just $2 \cdot \ell' \cdot 2^\kappa$ L-multiplications, a fact we explain rigorously in Remark 3.4 below. In Subsection 3.2 below, we argue that ring-switching is concretely and asymptotically efficient for both the prover and the verifier, and is essentially optimal.

3.1 Ring-Switching Protocol

We now record our ring-switching reduction.

CONSTRUCTION 3.1 (Ring-Switching Compiler)**.**
A large-field scheme $\Pi' = \big(\mathsf{Setup}', \mathsf{Commit}', \mathcal{P}', \mathcal{V}'\big)$ is given as input. We define the small-field scheme $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ in the following way.

1. $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, K)$. On input 1^λ, ℓ, and K, run and output $\Pi'.\mathsf{Setup}'(1^\lambda, \ell')$, where ℓ' is such that the field $L \,/\, K$ returned by that routine, of degree 2^κ over K say, satisfies $\ell' = \ell - \kappa$.
2. $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, fix the packed polynomial $t'(X_0, \ldots, X_{\ell'-1}) \in L[X_0, \ldots, X_{\ell'-1}]^{\preceq 1}$ as in Definition 2.1; output $\Pi'.\mathsf{Commit}'(\mathsf{params}, t')$.

We define $(\mathcal{P}, \mathcal{V})$ as the following IOP, in which both parties have the common input $[f]$, $s \in L$, and $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and $\mathcal{P}$ has the further input $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.

1. $\mathcal{P}$ computes $\hat{s} \coloneqq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ and sends $\mathcal{V}$ the A-element $\hat{s}$.
2. $\mathcal{V}$ decomposes $\hat{s} =: \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \otimes \beta_v$. $\mathcal{V}$ requires $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.
3. $\mathcal{V}$ samples batching scalars $(r''_0, \ldots, r''_{\kappa-1}) \leftarrow L^\kappa$ and sends them to $\mathcal{P}$.
4. For each $w \in \mathcal{B}_{\ell'}$, $\mathcal{P}$ decomposes $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) =: \sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u$. $\mathcal{P}$ defines the function $A : w \mapsto \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot A_{w,u}$ on $\mathcal{B}_{\ell'}$ and writes $A(X_0, \ldots, X_{\ell'-1})$ for its multilinear extension. $\mathcal{P}$ defines $h(X_0, \ldots, X_{\ell'-1}) \coloneqq A(X_0, \ldots, X_{\ell'-1}) \cdot t'(X_0, \ldots, X_{\ell'-1})$.
5. $\mathcal{V}$ decomposes $\hat{s} =: \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \hat{s}_u$, and sets $s_0 \coloneqq \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \hat{s}_u$.
6. $\mathcal{P}$ and $\mathcal{V}$ execute the following loop:
   for $i \in \{0, \ldots, \ell' - 1\}$ **do**
   $\quad$ $\mathcal{P}$ sends $\mathcal{V}$ the polynomial $h_i(X) \coloneqq \sum_{w \in \mathcal{B}_{\ell'-i-1}} h\big(r'_0, \ldots, r'_{i-1}, X, w_0, \ldots, w_{\ell'-i-2}\big)$.
   $\quad$ $\mathcal{V}$ requires $s_i \stackrel{?}{=} h_i(0) + h_i(1)$. $\mathcal{V}$ samples $r'_i \leftarrow L$, sets $s_{i+1} \coloneqq h_i(r'_i)$, and sends $\mathcal{P}$ r'_i.
7. $\mathcal{P}$ computes $s' \coloneqq t'(r'_0, \ldots, r'_{\ell'-1})$ and sends $\mathcal{V}$ s'.
8. $\mathcal{V}$ sets $e \coloneqq \widetilde{\mathsf{eq}}\big(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), \varphi_1(r'_0), \ldots, \varphi_1(r'_{\ell'-1})\big)$ and decomposes $e =: \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes e_u$.
9. $\mathcal{V}$ requires $s_{\ell'} \stackrel{?}{=} \big(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u\big) \cdot s'$.
10. $\mathcal{P}$ and $\mathcal{V}$ engage in the evaluation protocol $b' \leftarrow \langle \mathcal{P}'([f], s', r'; t'), \mathcal{V}'([f], s', r') \rangle$; $\mathcal{V}$ outputs $b \coloneqq b'$.

Theorem 3.2. *If* $\Pi' = \big(\mathsf{Setup}', \mathsf{Commit}', \mathcal{P}', \mathcal{V}'\big)$ *is complete, then* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *also is.*

Proof. We must prove three main things. First, we must show that, if $\mathcal{P}$ constructs $\hat{s} \in A$ honestly, then $\mathcal{V}$'s check $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\kappa-1}, v_0, \ldots, v_{\kappa-1}) \cdot \hat{s}_v$ will pass. Further, we must show that $\mathcal{V}$'s quantity $s_0 \coloneqq \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \hat{s}_u$ will satisfy $s_0 = \sum_{w \in \mathcal{B}_{\ell'}} h(w)$, so that $\mathcal{V}$ will accept throughout its sumcheck. Finally, we must show that $\mathcal{V}$'s final check will pass; this task amounts to showing that e's row-representation $e = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes e_u$ will satisfy $A(r'_0, \ldots, r'_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} e_u \cdot \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1})$.

We begin with the first fact above. If $\mathcal{P}$ operates as prescribed, then its initial message $\hat{s} \in A$ will satisfy:

$$\hat{s} \coloneqq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1})) = \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) \cdot \varphi_1(t')(w). \tag{32}$$

By the definition of $\varphi_1(t')(X_0, \ldots, X_{\ell'-1})$, for each $w \in \mathcal{B}_{\ell'}$, we have the column decomposition $\varphi_1(t')(w) = \sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \otimes \beta_v$. On the other hand, $\widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) = \varphi_0(\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}))$. Using the column-multiplication rule, we obtain, for each summand

$w \in \mathcal{B}_{\ell'}$ of the sum (32) above, the column decomposition $\varphi_0(\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})) \cdot \varphi_1(t')(w) = \sum_{v \in \mathcal{B}_\kappa} (\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1})) \otimes \beta_v$. Inlining this expression into the sum (32) above, we obtain:

$$
\begin{aligned}
\hat{s} &= \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) \cdot \varphi_1(t')(w) && \text{(by (32).)} \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{v \in \mathcal{B}_\kappa} (\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1})) \otimes \beta_v \right) && \text{(column values.)} \\
&= \sum_{v \in \mathcal{B}_\kappa} \left(\sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \right) \otimes \beta_v && \text{(rearranging sums.)} \\
&= \sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}) \otimes \beta_v. && \text{(fundamental property of multilinears.)}
\end{aligned}
$$

That is, $\mathcal{V}$'s column-decomposition $\hat{s} = \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \otimes \beta_v$ will satisfy $\hat{s}_v = t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$ for each $v \in \mathcal{B}_\kappa$. Assuming now that $\mathcal{P}$'s initial claim $s \stackrel{?}{=} t(r_0, \ldots, r_{\ell-1})$ is true, we obtain:

$$
\begin{aligned}
s &= t(r_0, \ldots, r_{\ell-1}) && \text{(by the truth of } \mathcal{P}\text{'s claim.)} \\
&= \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}) && \text{(partial multilinear expansion.)} \\
&= \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v. && \text{(by the calculation just carried out.)}
\end{aligned}
$$

In particular, $\mathcal{V}$ will accept its first check $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.

We turn to the sumcheck. As a notational device, we define the A-valued polynomial:

$$\hat{h}(X_0, \ldots, X_{\ell'-1}) \coloneqq \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), X_0, \ldots, X_{\ell'-1})) \cdot \varphi_1(t')(X_0, \ldots, X_{\ell'-1}).$$

Informally, we must show that $\mathcal{P}$'s polynomial $h(X_0, \ldots, X_{\ell'-1})$ above is a "row-combination" of $\hat{h}(X_0, \ldots, X_{\ell'-1})$ by the vector $\left(\widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1})\right)_{u \in \mathcal{B}_\kappa}$.

On the one hand, we note immediately that

$$\sum_{w \in \mathcal{B}_{\ell'}} \hat{h}(w) = \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) \cdot \varphi_1(t')(w) = \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1})) = \hat{s};$$

the last equality holds precisely when $\mathcal{P}$ constructs $\hat{s}$ honestly.

On the other hand, for each $w \in \mathcal{B}_{\ell'}$:

$$
\begin{aligned}
\hat{h}(w) &= \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) \cdot \varphi_1(t')(w) && \text{(by definition of } \hat{h}.) \\
&= \left(\sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes A_{w,u} \right) \cdot (1 \otimes t'(w)) && \text{(by the definitions of } \varphi_1(t') \text{ and of } A_{w,u}.) \\
&= \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes (A_{w,u} \cdot t'(w)). && \text{(distributing and using the multiplicative structure of } A.)
\end{aligned}
$$

We explain in slightly further detail the second equality above. Indeed, we use first the fact—already noted above—whereby $\widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) = \varphi_0(\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}))$. On the other hand, since the basis decomposition $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u$ holds by definition of the elements $A_{w,u}$, the row representation of this quantity's image under φ_0 can be none other than $\sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes A_{w,u}$, which is what appears above.

Combining the above two calculations, we conclude that, if $\mathcal{P}$ is honest, then

$$\hat{s} = \sum_{w \in \mathcal{B}_{\ell'}} \hat{h}(w) = \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes (t'(w) \cdot A_{w,u}) \right) = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \left(\sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t'(w) \right) \tag{33}$$

will hold, so that $\mathcal{V}$'s row decomposition $\hat{s} = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \hat{s}_u$ will satisfy $\hat{s}_u = \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t'(w)$ for each $u \in \mathcal{B}_\kappa$. We conclude that, if $\mathcal{P}$ constructs $\hat{s}$ correctly, then

$$\begin{aligned}
\sum_{w \in \mathcal{B}_{\ell'}} h(w) &= \sum_{w \in \mathcal{B}_{\ell'}} A(w) \cdot t'(w) && \text{(by definition of } h(X_0, \ldots, X_{\ell'-1}).) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot A_{w,u} \right) \cdot t'(w) && \text{(by definition of } A(X_0, \ldots, X_{\ell'-1}).) \\
&= \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot t'(w) && \text{(interchanging the above sums.)} \\
&= \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \hat{s}_u && \text{(by (33) and the remarks below it.)} \\
&= s_0 && \text{(by definition of the verifier.)}
\end{aligned}$$

will hold, so that $\mathcal{P}$'s sumcheck claim $s_0 = \sum_{w \in \mathcal{B}_{\ell'}} h(w)$ will be valid, and $\mathcal{V}$ will accept throughout the course of its sumcheck, by the completeness of that latter protocol.

We turn to $\mathcal{V}$'s final check. If $\mathcal{P}$ is honest, then $s' = t'(r'_0, \ldots, r'_{\ell'-1})$ will hold; moreover, by definition of the sumcheck, we will have $s_{\ell'} = h(r'_0, \ldots, r'_{\ell'-1})$. To treat $\mathcal{V}$'s final check, it thus suffices to argue that $h(r'_0, \ldots, r'_{\ell'-1}) = t'(r'_0, \ldots, r'_{\ell'-1}) \cdot \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u$ will hold; to show this, it in turn suffices, by definition of $h(X_0, \ldots, X_{\ell'-1})$, to prove that

$$A(r'_0, \ldots, r'_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u.$$

We proceed as follows. We note first that:

$$\begin{aligned}
e &= \widetilde{\mathsf{eq}}\big(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), \varphi_1(r'_0), \ldots, \varphi_1(r'_{\ell'-1})\big) && \text{(by definition.)} \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), w_0, \ldots, w_{\ell'-1}) \cdot \widetilde{\mathsf{eq}}\big(\varphi_1(r'_0), \ldots, \varphi_1(r'_{\ell'-1}), w_0, \ldots, w_{\ell'-1}\big) && \text{(see below.)} \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \varphi_0(\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1})) \cdot \varphi_1\big(\widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1})\big) && \text{(pulling out } \varphi_0 \text{ and } \varphi_1.) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes A_{w,u} \right) \cdot \big(1 \otimes \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1})\big) && \text{(again by definition of the } A_{w,u}.) \\
&= \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \left(\sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1}) \right). && \text{(multiplying in } A \text{ and rearranging.)}
\end{aligned}$$

To achieve the second equality above, we note that the multilinears $\widetilde{\mathsf{eq}}(X_0, \ldots, X_{\ell'-1}, Y_0, \ldots, Y_{\ell'-1})$ and $\sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(X_0, \ldots, X_{\ell'-1}, w_0, \ldots, w_{\ell'-1}) \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, Y_0, \ldots, Y_{\ell'-1})$ are necessarily identical, since they agree identically on the cube $\mathcal{B}_{2 \cdot \ell'}$.

We see that the verifier's row-decomposition $e = \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes e_u$ will satisfy $e_u = \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1})$ for each $u \in \mathcal{B}_\kappa$. We conclude finally $\mathcal{V}$ will have

$$\begin{aligned}
\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(r'', u) \cdot e_u &= \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \sum_{w \in \mathcal{B}_{\ell'}} A_{w,u} \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1}) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \left(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot A_{w,u} \right) \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1}) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} A(w_0, \ldots, w_{\ell'-1}) \cdot \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell'-1}, r'_0, \ldots, r'_{\ell'-1}) \\
&= A(r'_0, \ldots, r'_{\ell'-1}),
\end{aligned}$$

which is exactly what we needed to show. This completes the proof of completeness. □

Remark 3.3. We explain in slightly more rigorous terms the "information loss" which would result if the parties *merely* evaluated $t'(r_\kappa, \ldots, r_{\ell-1})$, as opposed to using the tensor algebra. During Theorem 3.2's proof, we show that $\hat{s}_v = t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1})$ holds for each $v \in \mathcal{B}_\kappa$. On the other hand,

$$\begin{aligned}
t'(r_\kappa, \ldots, r_{\ell-1}) &= \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t'(w) \\
&= \sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot \sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \cdot \beta_v \\
&= \sum_{v \in \mathcal{B}_\kappa} \left(\sum_{w \in \mathcal{B}_{\ell'}} \widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \cdot t(v_0, \ldots, v_{\kappa-1}, w_0, \ldots, w_{\ell'-1}) \right) \cdot \beta_v \\
&= \sum_{v \in \mathcal{B}_\kappa} t(v_0, \ldots, v_{\kappa-1}, r_\kappa, \ldots, r_{\ell-1}) \cdot \beta_v \\
&= \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \cdot \beta_v.
\end{aligned}$$

We see that, while the information contained in $\hat{s} = \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \otimes \beta_v$ suffices to recover $t(r_0, \ldots, r_{\ell-1})$ (as the proof of Theorem 3.2 above shows), the datum $t(r_\kappa, \ldots, r_{\ell-1})$ would yield, rather, the basis-combination $\sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \cdot \beta_v$ of $\hat{s}$'s columns. Since the K-basis $(\beta_v)_{v \in \mathcal{B}_\kappa}$ is certainly not linearly independent over L, this latter combination reflects $\hat{s}$ only "lossfully". We note that, interestingly, $\sum_{v \in \mathcal{B}_\kappa} \hat{s} \cdot \beta_v = h(\hat{s})$ holds; here, $h : L \otimes_K L \to L$ is the canonical K-linear map defined on simple tensors by multiplication (we recall Subsection 2.5 above). That is, $t(r_\kappa, \ldots, r_{\ell-1})$ relates to $\varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ exactly by the map h, which is of course not injective. We would like to thank Raju Krishnamoorthy for explaining this fact to us.

Remark 3.4. We discuss the verifier's computation of $e \coloneqq \widetilde{\mathsf{eq}}\big(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), \varphi_1(r'_0), \ldots, \varphi_1(r'_{\ell'-1})\big)$. Clearly, this computation amounts to $O(\ell')$ arithmetic operations in the algebra A, and so can be carried out in polylogarithmic time for the verifier in the worst case (we defer our full efficiency analysis to Subsection 5.2 below). Here, we discuss a concretely efficient procedure by whose aid the verifier may compute e, at least in the characteristic 2 case. Indeed, we note first the following identity, valid only in characteristic 2:

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

This identity suggests the correctness of the following algorithm:

1: initialize the A-element $e \coloneqq 1$.
2: **for** $i \in \{0, \ldots, \ell' - 1\}$ **do** update $e \mathrel{-}= e \cdot \varphi_0(r_{\kappa+i}) + e \cdot \varphi_1(r'_i)$.
3: **return** e.

This algorithm computes e using just $2 \cdot \ell'$ "scaling operations", ℓ' vertical and ℓ' horizontal. Here, we mean by the term "scaling operation" the multiplication of an A-element by an L-element, itself embedded into A either by φ_0 or φ_1 (as the case may be). As is made clear in Subsection 2.5, multiplications of these latter sorts are easier to carry out than general A-by-A multiplications are, and in fact amount to 2^κ L-multiplications a piece (i.e., either of each column of A or of each row of A by the fixed L-multiplicand).

We now prove the security of ring-switching.

Theorem 3.5. *If* $\Pi' = \big(\mathsf{Setup}', \mathsf{Commit}', \mathcal{P}', \mathcal{V}'\big)$ *is secure, then* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *also is.*

Proof. We write $\mathcal{E}'$ for the emulator for Π'. We define an emulator $\mathcal{E}$ for Π as follows.

1. On input $\mathcal{A}$'s record of interactions with the vector oracle, $\mathcal{E}$ internally runs $t'(X_0, \ldots, X_{\ell'-1}) \leftarrow \mathcal{E}'$.
2. If $t'(X_0, \ldots, X_{\ell'-1}) = \bot$, then $\mathcal{E}$ outputs $\bot$ and aborts.
3. By reversing Definition 2.1, $\mathcal{E}$ obtains $t(X_0, \ldots, X_{\ell-1}) \in K[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, which it outputs.

We argue that the emulator $\mathcal{E}$ defined in this way is secure. If $\mathcal{V}'$ rejects, then $\mathcal{V}$ also does. The probability with which $\mathcal{E}'$ outputs $\bot$ and $\mathcal{V}'$ accepts is negligible, by the security of Π'. So too, therefore, is the probability with which $\mathcal{E}$ outputs $\bot$ and $\mathcal{V}$ accepts. We thus fix our attention on those executions of the experiment for which $t'(X_0, \ldots, X_{\ell'-1}) \neq \bot$; in particular, we assume that $t(X_0, \ldots, X_{\ell-1}) \neq \bot$. Similarly, the probability with which $t'(X_0, \ldots, X_{\ell'-1}) \neq \bot$, $t'(r') \neq s'$, and $b' = 1$ all hold is negligible, by the security of Π'. We thus focus our attention on those executions for which $t'(r') = s'$. We must show that the probability with which $t(r) \neq s$ and $\mathcal{V}$ accepts is negligible. We assume now that $t(r) \neq s$.

We may further restrict our considerations to the set of executions within which $\mathcal{P}$ computes its first message $\hat{s} \neq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ *incorrectly.* Indeed, it is shown directly in the course of our proof of Theorem 3.2 above that, if $\hat{s} = \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ holds, then $\sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v = t(r_0, \ldots, r_{\ell-1})$ also will. In this latter setting,

$$\begin{aligned} s &\neq t(r_0, \ldots, r_{\ell-1}) && \text{(by our initial assumption above whereby } \mathcal{P}\text{'s claim is false.)} \\ &= \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v && \text{(a consequence of } \hat{s} = \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1})).) \end{aligned}$$

will hold, so that $\mathcal{V}$ will reject and we're done. We thus assume that $\hat{s} \neq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$. For the sake of notation, we abbreviate $\overline{s} \coloneqq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ for this latter quantity, and write $\overline{s} =: \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \overline{s}_u$ for its row-decomposition.

Our hypothesis whereby $\hat{s} \neq \overline{s}$ entails that the κ-variate polynomial over L

$$S(X_0, \ldots, X_{\kappa-1}) \coloneqq \sum_{u \in \mathcal{B}_\kappa} (\hat{s}_u - \overline{s}_u) \cdot \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, X_0, \ldots, X_{\kappa-1})$$

is not identically zero. Applying Schwartz–Zippel to $S(X_0, \ldots, X_{\kappa-1})$, we conclude that the probability, over $\mathcal{V}$'s choice of $(r''_0, \ldots, r''_{\kappa-1}) \leftarrow L^\kappa$, that $S(r''_0, \ldots, r''_{\kappa-1}) = 0$ will hold is at most $\frac{\kappa}{|L|}$, which is negligible. We thus assume that $S(r''_0, \ldots, r''_{\kappa-1}) \neq 0$, which itself immediately entails that:

$$s_0 \coloneqq \sum_{u \in \mathcal{B}_\kappa} \hat{s}_u \cdot \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \neq \sum_{u \in \mathcal{B}_\kappa} \overline{s}_u \cdot \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1})$$

will hold. On the other hand, by an argument identical to one already given during the proof of Theorem 3.2 above, we have that:

$$\sum_{w \in \mathcal{B}_{\ell'}} h(w) = \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \overline{s}_u;$$

we again write $h(X_0, \ldots, X_{\ell'-1}) \coloneqq A(X_0, \ldots, X_{\ell'-1}) \cdot t'(X_0, \ldots, X_{\ell'-1})$ (as usual, $t(X_0, \ldots, X_{\ell-1})$ here refers to what $\mathcal{E}$ extracted). Combining the above two equations, we conclude—again under our hypothesis whereby $S(r''_0, \ldots, r''_{\kappa-1}) \neq 0$—that $s_0 \neq \sum_{w \in \mathcal{B}_{\ell'}} h(w)$. By the soundness of the sumcheck, we conclude that the probability with which $\mathcal{V}$ accepts throughout that protocol and $s_{\ell'} = h\big(r'_0, \ldots, r'_{\ell'-1}\big)$ holds is at most $\frac{2 \cdot \ell'}{|L|}$, which is negligible. We thus assume that $s_{\ell'} \neq h\big(r'_0, \ldots, r'_{\ell'-1}\big)$, or in other words that:

$$s_{\ell'} \neq A(r'_0, \ldots, r'_{\ell'-1}) \cdot t(r'_0, \ldots, r'_{\ell'-1}).$$

The proof of Theorem 3.2 already shows that $A(r'_0, \ldots, r'_{\ell'-1}) = \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u$. On the other hand, we've already justified our consideration just of those executions within which $s' = t'(r'_0, \ldots, r'_{\ell'-1})$ holds. Under exactly this latter condition, therefore, the verifier will obtain:

$$s_{\ell'} \neq A(r'_0, \ldots, r'_{\ell'-1}) \cdot t(r'_0, \ldots, r'_{\ell'-1}) = \left(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u \right) \cdot s',$$

and so will once again reject. This completes the proof. □

3.2 Efficiency

We examine the efficiency of Protocol 3.1. Throughout, we count K-operations. We view the ground field K as constant in both λ and ℓ. We must first clarify how L relates to K. The soundness error of Protocol 3.1 is $\frac{2\cdot\ell'+\kappa}{|L|}$. In order for this quantity to be negligible, it's enough that $|L| \geq 2^{\omega(\log\lambda)}$ hold (we note that $\ell = O(\log\lambda)$ must hold, lest the prover's input length 2^ℓ fail to be polynomial in the security parameter). For simplicity, we assume that $\deg(L \,/\, K) = \lambda$. In this case, $|L| = |K|^\lambda \geq 2^\lambda$, so that Protocol 3.1 in fact becomes exponentially secure. We moreover obtain $2^\kappa = \lambda$ and $2^{\ell'} = \frac{2^\ell}{\lambda}$.

We refer to von zur Gathen and Gerhard for [GG13] for complexity-theoretic background. For the purposes of this subsection, we understand L as a *univariate* extension of K (see [GG13, § 25.4]). Following [GG13, Def. 8.26], we write $\mathsf{M}(\lambda)$ for the complexity—measured in K-operations—of multiplying two polynomials in $K[X]$ of degree at most λ. It is known [GG13, Thm. 8.23] that we may take $\mathsf{M}(\lambda) = \widetilde{O}(\lambda)$. It can be shown that multiplication in L can be carried out in at most $6\cdot\mathsf{M}(\lambda)+O(\lambda) = \widetilde{O}(\lambda)$ K-operations [GG13, Cor. 11.11]. We write $\mathsf{Q}(\lambda) = O(\mathsf{M}(\lambda)) = \widetilde{O}(\lambda)$ for the cost, in K-operations, of L-multiplication. We note that K-by-L multiplication takes exactly λ K-multiplications; each L-addition costs exactly λ K-additions.

Prover cost. The prover's main cost is that of computing the tensor-expansion

$$\underbrace{\left[\text{———} \bigotimes_{i=\kappa}^{\ell-1}(1-r_i, r_i) \text{———} \right]}_{2^{\ell'} \text{ elements}}. \tag{34}$$

In view of the standard algorithm for it (see Subsection 2.1), this task takes $2^{\ell'}$ L-multiplications and $2^{\ell'}$ L-additions. We see that the total number of K-operations associated with this task is

$$O\left(2^{\ell'}\right)\cdot\mathsf{Q}(\lambda) = O\left(\frac{2^\ell}{\lambda}\right)\cdot\mathsf{Q}(\lambda) = O(2^\ell)\cdot\frac{\mathsf{Q}(\lambda)}{\lambda} = O(2^\ell)\cdot\widetilde{O}(1),$$

which is linear in the input length and polylogarithmic in λ.

Having computed (34), the prover must use it in two places. First, to compute the partial evaluations $(t(v_0,\ldots,v_{\kappa-1},r_\kappa,\ldots,r_{\ell-1}))_{v\in\mathcal{B}_\kappa}$ in step 1 above, the prover must perform *all* 2^κ dot-products of the form

$$\hat{s}_v \coloneqq \left\langle \left[\text{———} \bigotimes_{i=\kappa}^{\ell-1}(1-r_i, r_i) \text{———} \right], \underbrace{\left[\text{———} t(v_0,\ldots,v_{\kappa-1},w_0,\ldots,w_{\ell'-1}) \text{———} \right]}_{w\in\mathcal{B}_{\ell'} \text{ varying}} \right\rangle, \tag{35}$$

for $v \in \mathcal{B}_\kappa$ varying. Each dot-product (35) takes $2^{\ell'}$ L-by-K multiplications and $2^{\ell'}$ L-additions. The total cost associated with the dot-products (35) is thus

$$2^\kappa\cdot O(2^{\ell'})\cdot\lambda = O(2^\ell)\cdot\lambda$$

K-operations.

Finally, to prepare its sumcheck in step 4, the prover must first tensor-expand $\bigotimes_{i=0}^{\kappa-1}(1-r''_i, r''_i)$—a task whose cost is polynomial in λ alone, and which we ignore—and then compute the L-by-K matrix product:

$$\left[\text{——} \bigotimes_{i=0}^{\kappa-1}(1-r''_i, r''_i) \text{——} \right] \cdot \left.\begin{bmatrix} | & & | \\ A_{(0,\ldots,0),u} & \cdots & A_{(1,\ldots,1),u} \\ | & & | \end{bmatrix}\right\} {\scriptstyle u\in\mathcal{B}_\kappa \text{ varying}}. \tag{36}$$

The product (36) yields exactly the table of values of $A(X_0,\ldots,X_{\ell'-1})$ on the cube $\mathcal{B}_{\ell'}$.

To obtain the $2^\kappa \times 2^{\ell'}$ matrix in the right-hand side of (36), the prover must use (34) again; specifically, it must simply basis-decompose each component of that tensor. This task is simply a rewiring operation, and is free (recall also (28)). To compute the matrix product (36), the prover must again perform $2^\kappa \cdot 2^{\ell'} = 2^\ell$ L-by-K multiplications and 2^ℓ L-additions; the total cost of (36) is thus again $O(2^\ell) \cdot \lambda$ K-operations.

The sumcheck 6 is ℓ'-dimensional over L. In view of standard algorithms for this task (we recall the treatment of Thaler [Tha22, § 4.1]), the prover can carry it out in $O(2^{\ell'})$ total L-operations. Just as in (34) above, the prover's total cost during the sumcheck is therefore again $O(2^\ell) \cdot \widetilde{O}(1)$ total K-operations.

In sum, we see that our prover's cost is $O(2^\ell) \cdot \lambda$ K-operations. Our prover is thus linear (in both the statement size and the security parameter). Interestingly, our prover is dominated by the respective costs of K-by-L multiplication and L-addition. These tasks are, in practice, cheaper than L-multiplication is. Upon ignoring these former costs—and considering *only* the cost of L-multiplication—we would in fact obtain a prover costing just $O(2^\ell) \cdot \widetilde{O}(1)$ K-multiplications (that is, with just a *polylogarithmic*, and not a linear, multiplicative dependence on the security parameter λ).

Verifier cost. To receive $\hat{s}$, the verifier must receive 2^κ L-elements, whose total size is that of $2^{2\cdot\kappa} = \lambda^2$ K-elements. In its step 2 above, the verifier must compute the tensor-expansion $\bigotimes_{i=0}^{\kappa-1}(1 - r_i, r_i)$, as well as dot the resulting tensor with $(\hat{s}_v)_{v \in \mathcal{B}_\kappa}$. These tasks, together, amount to $O(2^\kappa)$ L-operations, and so cost in total

$$O(2^\kappa) \cdot \mathsf{Q}(\lambda) = O(\lambda) \cdot \mathsf{Q}(\lambda) = \widetilde{O}(\lambda^2)$$

K-operations. The verifier's respective combination tasks in 5 and 9 are structurally identical to that of 2, and have identical costs; we skip our analyses of them.

During the sumcheck 6, the verifier must expend $O(\ell')$ L-operations; these cost in total $O(\ell') \cdot \mathsf{Q}(\lambda) \leq O(\ell) \cdot \mathsf{Q}(\lambda)$ K-operations. To compute e in step 8, the verifier must perform $2 \cdot 2^\kappa \cdot \ell'$ L-operations (recall Remark 3.4). These in total cost

$$O(\ell') \cdot 2^\kappa \cdot \mathsf{Q}(\lambda) \leq O(\ell) \cdot \widetilde{O}(\lambda^2)$$

K-operations. Our verifier is thus logarithmic in the statement size and quadratic in λ (up to a polylogarithmic multiplicative factor).

Other costs. Protocol 3.1's prover must invoke Π''s commitment procedure exactly once. During Protocol 3.1's evaluation phase, the prover and the verifier must jointly invoke the underlying large-field commitment scheme Π''s evaluation procedure exactly once. We ignore all of these costs in this subsection, and only measure the efficiency of Protocol 3.1 over and above them.

Comparison to Hashcaster. To prepare the sumcheck (21) naively, Hashcaster [Sou24]'s prover would need to compute the matrix product

$$\begin{bmatrix} \text{———} \bigotimes_{i=0}^{\kappa-1}(1 - r''_i, r''_i) \text{———} \end{bmatrix} \cdot \left.\begin{bmatrix} \text{————} \bigotimes_{i=\kappa}^{\ell-1}\left(1 - \sigma^0(r_i), \sigma^0(r_i)\right) \text{————} \\ \vdots \\ \text{————} \bigotimes_{i=\kappa}^{\ell-1}\left(1 - \sigma^{2^\kappa-1}(r_i), \sigma^{2^\kappa-1}(r_i)\right) \text{————} \end{bmatrix}\right\} v \in \mathcal{B}_\kappa \text{ varying}. \tag{37}$$

To compute that matrix product, Hashcaster's prover would need to expend $\Theta(2^\ell)$ L-multiplications, thereby incurring $\Theta(2^\ell) \cdot \mathsf{Q}(\lambda)$ K-operations. That prover's cost would thus be worse than ours by a polylogarithmic factor in λ, at least. As it turns out, by nontrivially using the Galois transformation (22), Hashcaster is able to bring its prover cost closer to ours, albeit with various *additive* polylogarithmic penalties in λ. Its prover, on the other hand, is much more complicated, and runs the sumcheck using a non-blackbox algorithm.

To compute the Galois transformation (18), Hashcaster's verifier must expend $2^{2\cdot\kappa}$ L-multiplications, for a total cost of

$$2^{2\cdot\kappa} \cdot \mathsf{Q}(\lambda) = \lambda^2 \cdot \mathsf{Q}(\lambda) = \widetilde{O}(\lambda^3)$$

K-operations. We thus see that Hashcaster's verifier's dependence on λ is cubic, as opposed to quadratic.

4 Binary BaseFold

In this section, we present our large-field PCS. We have already sketched the main problem in Subsection 1.4 above; here, we record a few further details.

The additive NTT. The *number-theoretic transform* entails evaluating a polynomial $P(X) = \sum_{j=0}^{2^\ell - 1} a_j \cdot X^j$ of degree less than 2^ℓ on some 2^ℓ-sized subset of its coefficient field. In fields containing a 2^ℓ-sized *multiplicative coset*, the Cooley–Tukey algorithm computes the number-theoretic transform in $O(\ell \cdot 2^\ell)$ time. Binary fields, on the other hand, have no 2-adicity at all—their multiplicative groups of units are odd.

In a classic and farsighted work, Cantor [Can89] developed an "additive" variant of the FFT: an algorithm that evaluates $P(X)$ not on a multiplicative coset of its coefficient field, but on an additive subgroup of it. Indeed, each binary field $\mathbb{F}_{2^r}$ can be viewed as a vector space over its own subfield $\mathbb{F}_2$. Here, by an *additive subgroup* of $\mathbb{F}_{2^r}$, we mean an $\mathbb{F}_2$-vector subspace $S \subset \mathbb{F}_{2^r}$. Cantor's algorithm evaluates $P(X)$ on any such 2^ℓ-sized domain $S \subset \mathbb{F}_{2^r}$ in $O(\ell^{\log_2(3)} \cdot 2^\ell)$ time.

For some time, it was not known whether the Cooley–Tukey algorithm's characteristic $O(\ell \cdot 2^\ell)$ time complexity could be recovered in the additive, binary case. In a key and important work, Lin, Chung and Han [LCH14] attain exactly this feat, with a caveat: they interpret their input vector $(a_0, \ldots, a_{2^\ell - 1})$ as $P(X)$'s coefficients not in the standard monomial basis, but in a *novel polynomial basis* that those authors introduce. That is, the polynomial which their algorithm evaluates over S is not $\sum_{j=0}^{2^\ell - 1} a_j \cdot X^j$, but $\sum_{j=0}^{2^\ell - 1} a_j \cdot X_j(X)$; here, for each $j \in \{0, \ldots, 2^\ell - 1\}$, $X_j(X)$ is an alternate basis polynomial of degree j that those authors describe. Lin, Chung and Han [LCH14] build their basis polynomials $(X_j(X))_{j=0}^{2^\ell - 1}$ out of *subspace vanishing polynomials*. These are polynomials $\widehat{W}_i(X)$, for $i \in \{0, \ldots, \ell\}$, which respectively vanish identically on an ascending chain of $\mathbb{F}_2$-subspaces $U_0 \subset \cdots \subset U_\ell$ of $\mathbb{F}_{2^r}$.

Our binary adaptation of BaseFold PCS ties together two disparate threads: Lin, Chung and Han [LCH14]'s *additive NTT* and FRI [BBHR18a]. We recall that binary-field FRI works with the aid of a sequence of $\mathbb{F}_2$-subspaces $S^{(0)}, \ldots, S^{(\ell)}$ of $\mathbb{F}_{2^r}$, themselves connected by linear subspace polynomials:

$$S^{(0)} \xrightarrow{q^{(0)}} S^{(1)} \xrightarrow{q^{(1)}} S^{(2)} \xrightarrow{q^{(2)}} \cdots \xrightarrow{q^{(\ell-1)}} S^{(\ell)}.$$

Here, the maps $q^{(0)}, \ldots, q^{(\ell-1)}$ are linear subspace polynomials of degree 2.

Choosing the folding maps. To use BaseFold PCS in characteristic 2, we must use the additive NTT instead of the standard one, and use binary-field FRI instead of prime-field FRI. Simple enough, but which domains $S^{(0)}, \ldots, S^{(\ell)}$ and which maps $q^{(0)}, \ldots, q^{(\ell-1)}$ should we use in the latter protocol? FRI [BBHR18a] does *not* suggest a canonical choice; each choice there works as well as any other. But BaseFold's FRI subprotocol is not just a proximity test; it's also a built-in multilinear evaluator (see Subsection 1.2). BaseFold PCS relies on the fact whereby a FRI execution which begins on the Reed–Solomon encoding of $(a_0, \ldots, a_{2^\ell - 1})$ will end on the constant polynomial whose value on $S^{(\ell)}$ is identically

$$a_0 + a_1 \cdot r'_0 + a_2 \cdot r'_1 + a_3 \cdot r'_0 \cdot r'_1 + \cdots + a_{2^\ell - 1} \cdot r'_0 \cdot \cdots \cdot r'_{\ell-1}, \tag{38}$$

where $(r'_0, \ldots, r'_{\ell-1})$ are the verifier's FRI challenges. For maps $q^{(0)}, \ldots, q^{(\ell-1)}$ generically chosen, this fact will simply cease to hold. We discuss this problem further in the introduction of Section 4 below.

We recover BaseFold PCS in characteristic 2 by introducing a specialization of binary FRI that works compatibly with [LCH14]. That is, we introduce a particular choice of the maps $q^{(0)}, \ldots, q^{(\ell-1)}$ which causes the equality (38) to re-emerge. Interestingly, the right choice of $q^{(0)}, \ldots, q^{(\ell-1)}$ turns out to be that for which, for each $i \in \{0, \ldots, \ell\}$, the composition identity

$$\widehat{W}_i = q^{(i-1)} \circ \cdots \circ q^{(0)}$$

holds. That is, we choose our FRI folding maps $q^{(0)}, \ldots, q^{(\ell-1)}$ so that they "factor" Lin, Chung and Han [LCH14]'s subspace polynomials. Our binary BaseFold variant is independently important, and has already figured in key subsequent works, including Brehm et al.'s *Blaze* [Bre+25] and Frigo and shelat [Fs24].

More remarks on FRI. We opt moreover to modify FRI itself, so as to induce a Lagrange-style, as opposed to a monomial-style, 2-to-1 folding pattern in the coefficient domain. In our FRI variant, the value of the prover's final oracle becomes rather

$$a_0 \cdot (1 - r_0) \cdot \cdots \cdot (1 - r_{\ell-1}) + \cdots + a_{2^\ell - 1} \cdot r_0 \cdot \cdots \cdot r_{\ell-1},$$

the evaluation at $(r_0, \ldots, r_{\ell-1})$ of the polynomial whose coefficients in the multilinear Lagrange basis—as opposed to in the multilinear monomial basis—are $(a_0, \ldots, a_{2^\ell-1})$. (We explain this further in Remark 4.7.)

Separately, standard FRI [BBHR18a, § 3.2] supports *arbitrary-arity folding*, controlled by a folding arity parameter $\eta \geq 1$. The parameter η mediates a tradeoff between the number of oracles committed (which grows like $\frac{\ell}{\eta}$) and the size of each Merkle leaf (which grows like 2^η). The "sweet spot" tends to be around $\eta = 4$, in practical deployments. The effect on proof size at stake—i.e., which one stands to induce, upon changing η from 1 to something better—is significant (amounting to a halving at least, if not better). In rough terms, FRI stipulates that, to fold a given oracle using the parameter η, the prover interpolate a *univariate* polynomial of degree less than 2^η on each coset of the relevant oracle, and finally evaluate the resulting univariate polynomials collectively at the verifier's challenge point.

BaseFold, as written, does not support the use of higher-arity folding, for straightforward reasons. Upon using a parameter $\eta > 1$, one would cause (38) to fail in essentially two ways. For one, the number of challenges r'_i available would become too few (something like $\frac{\ell}{\eta}$, as opposed to ℓ). Moreover, the relationship between the list $(a_0, \ldots, a_{2^\ell-1})$ of initial coefficients and the value of the final constant FRI oracle would become that of a multivariate evaluation—of individual degree at most $2^\eta - 1$—over the challenges r'_i, as opposed to of a multilinear one. For this reason, BaseFold as written remains unable to draw on the proof-size gains available at the hands of higher-arity folding (which are significant).

We introduce a new, *multilinear* style of many-to-one FRI folding, which contrasts with FRI's univariate approach [BBHR18a, § 3.2]. We describe our FRI folding variant in Subsection 4.2 below (see in particular Definitions 4.6 and 4.8). We parameterize our method by a constant ϑ, which plays a role analogous to η's. Informally, we stipulate that the verifier send ϑ folding challenges, and that the prover fold its oracle, again coset-wise, using a length-2^ϑ *tensor combination* of the verifier's challenges over each coset. Our folding technique is equivalent to that in which the prover repeatedly performs standard, 2-to-1 folding ϑ times in succession—consuming ϑ challenges in the process, as opposed to 1—and commits only to the final result. (For this reason, we informally call it "oracle-skipping".) Interestingly, our FRI-folding variant makes necessary a sort of proximity gap different from that invoked by the original FRI protocol. Indeed, while the soundness proof [Ben+23, § 8.2] of FRI uses the proximity gap result [Ben+23, Thm. 1.5] for *low-degree parameterized curves*, our security treatment below instead uses a *tensor-folding* proximity gap of the sort recently established by Diamond and Posen [DP24, Thm. 2] (in fact, we use a sharpening of that result due to Diamond and Gruen [DG25, Cor. 1]).

Practical matters. We examine in various further aspects of binary-field FRI. For example, even in the abstract IOP model, we must fix $\mathbb{F}_2$-bases of the respective Reed–Solomon domains $S^{(i)}$, in order to interpret our committed functions $f^{(i)} : S^{(i)} \to L$ as L-valued strings. That is, we must implicitly lexicographically flatten each domain $S^{(i)}$, using some ordered $\mathbb{F}_2$-basis of it, known to both the prover and the verifier. The choice of these bases matters. Indeed, for $\mathbb{F}_2$-bases of $S^{(i)}$ and $S^{(i+1)}$ chosen arbitrarily, the fundamental operation which associates to each $y \in S^{(i+1)}$ its fiber ${q^{(i)}}^{-1}(\{y\}) \subset S^{(i)}$—which both the prover and the verifier must perform repeatedly—could come to assume complexity on the order of $\dim\left(S^{(i)}\right)^2$ bit-operations, even after a linear-algebraic preprocessing phase.

We suggest a family of bases for the respective domains $S^{(i)}$ with respect to which the maps $q^{(i)}$ come to act simply by projecting away their first coordinate. In particular, the application of each map $q^{(i)}$—in coordinates—becomes free; the preimage operation ${q^{(i)}}^{-1}(\{y\})$ comes to amount simply to that of prepending an arbitrary boolean coordinate to y's coordinate representation. While bases with these properties can of course be constructed in FRI even for maps $q^{(i)}$ chosen arbitrarily, our procedure moreover yields a basis of the initial domain $S^{(0)}$ which coincides with that expected by the additive NTT [LCH14]. In particular, our prover may use *as is* the output of the additive NTT as its 0^{th} FRI oracle, without first subjecting that output to the permutation induced by an appropriate change-of-basis transformation on $S^{(0)}$. We believe that these observations stand to aid all implementers of binary-field FRI.

4.1 Using FRI in Novel Polynomial Basis

We begin by proposing a specific construction of those subspace polynomials $q^{(0)}, \ldots, q^{(\ell-1)}$ invoked internally by FRI. Throughout this section, we fix a binary field L, with $\mathbb{F}_2$-basis $(\beta_0, \ldots, \beta_{r-1})$. Throughout the remainder of this subsection—and in fact, the entire paper—we impose the simplifying assumption whereby $\beta_0 = 1$. We fix moreover a size parameter $\ell \in \{0, \ldots, r-1\}$ and a rate parameter $\mathcal{R} \in \{1, \ldots, r-\ell\}$. We finally recall the subspace vanishing polynomials $\widehat{W}_i(X) \in L[X]$, for $i \in \{0, \ldots, \ell\}$, which we now view as $\mathbb{F}_2$-linear maps $\widehat{W}_i : L \to L$, as well as their non-normalized counterparts $W_i : L \to L$ (see Subsection 2.3).

We begin by defining our FRI domains and folding maps.

Definition 4.1. For each $i \in \{0, \ldots, \ell\}$, we define the domain

$$S^{(i)} := \widehat{W}_i(\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle).$$

Moreover, for each $i \in \{0, \ldots, \ell-1\}$, we define

$$q^{(i)}(X) := \frac{W_i(\beta_i)^2}{W_{i+1}(\beta_{i+1})} \cdot X \cdot (X+1).$$

For each $i \in \{0, \ldots, \ell-1\}$, the map $q^{(i)}(X)$ is a linear subspace polynomial of degree 2. *A priori*, this map's kernel could relate arbitrarily to the domain $S^{(i)} \subset L$; moreover, the image of its restriction to $S^{(i)}$ could relate arbitrarily to $S^{(i+1)}$. In the following sequence of results, we prove that in fact $q^{(i)}\big(S^{(i)}\big) = S^{(i+1)}$ holds for each $i \in \{0, \ldots, \ell-1\}$. In particular, the chain of maps $q^{(0)}, \ldots, q^{(\ell-1)}$ and the spaces $S^{(0)}, \ldots, S^{(\ell)}$ yield a valid global parameterization of the FRI protocol (in the sense of Subsection 2.4).

Lemma 4.2. *For each $i \in \{0, \ldots, \ell-1\}$, we have the equality $q^{(i)} \circ \widehat{W}_i = \widehat{W}_{i+1}$ of polynomials.*

Proof. We invoke the following direct calculation:

$$\begin{aligned}
\left(q^{(i)} \circ \widehat{W}_i\right)(X) &= \frac{W_i(\beta_i)^2}{W_{i+1}(\beta_{i+1})} \cdot \widehat{W}_i(X) \cdot \left(\widehat{W}_i(X) + 1\right) && \text{(by definition of } q^{(i)}.\text{)} \\
&= \frac{W_i(\beta_i)^2}{W_{i+1}(\beta_{i+1})} \cdot \frac{W_i(X)}{W_i(\beta_i)} \cdot \frac{W_i(X) + W_i(\beta_i)}{W_i(\beta_i)} && \text{(by definition of } \widehat{W}_i.\text{)} \\
&= \frac{W_i(X) \cdot (W_i(X) + W_i(\beta_i))}{W_{i+1}(\beta_{i+1})} && \text{(cancellation of } W_i(\beta_i)^2.\text{)} \\
&= \frac{W_{i+1}(X)}{W_{i+1}(\beta_{i+1})} && \text{(recursive characterization of } W_{i+1}(X).\text{)} \\
&= \widehat{W}_{i+1}(X). && \text{(by definition of } \widehat{W}_{i+1}(X).\text{)}
\end{aligned}$$

In the second-to-last step, we exploit the recursive identity $W_{i+1}(X) = W_i(X) \cdot (W_i(X) + W_i(\beta_i))$, itself a basic consequence of the definitions of W_{i+1} and W_i and of the linearity of W_i. □

Theorem 4.3. *For each $i \in \{0, \ldots, \ell-1\}$, $q^{(i)}\big(S^{(i)}\big) = S^{(i+1)}$.*

Proof. Using Lemma 4.2, we obtain:

$$\begin{aligned}
q^{(i)}\left(S^{(i)}\right) &= q^{(i)}\left(\widehat{W}_i(\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle)\right) && \text{(by definition of } S^{(i)}.\text{)} \\
&= \widehat{W}_{i+1}(\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle) && \text{(by Lemma 4.2.)} \\
&= S^{(i+1)}. && \text{(again by definition of } S^{(i+1)}.\text{)}
\end{aligned}$$

This completes the proof of the theorem. □

In the following further corollary of Lemma 4.2, we argue that the polynomials $q^{(0)}, \ldots, q^{(\ell-1)}$ collectively "factor" the normalized subspace polynomials $\widehat{W}_0, \ldots, \widehat{W}_\ell$, at least provided we assume $\beta_0 = 1$.

Corollary 4.4. *For each $i \in \{0, \ldots, \ell\}$, $\widehat{W}_i = q^{(i-1)} \circ \cdots \circ q^{(0)}$ holds.*

Proof. In the base case $i = 0$, we must show that $\widehat{W}_0$ equals the empty composition (namely X itself). To show this, we recall first that $W_0(X) = X$. Moreover:

$$\widehat{W}_0(X) = \frac{X}{W_0(\beta_0)} = \frac{X}{\beta_0} = X;$$

in the last step, we use our global simplifying assumption $\beta_0 = 1$.

For $i \in \{0, \ldots, \ell - 1\}$ arbitrary, Lemma 4.2 shows that $\widehat{W}_{i+1} = q^{(i)} \circ \widehat{W}_i$. Applying induction to $\widehat{W}_i$, we conclude that this latter map in turn equals $q^{(i)} \circ \cdots \circ q^{(0)}$. □

We note finally the following result.

Corollary 4.5. *For each $i \in \{0, \ldots, \ell\}$, the set $\left(\widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{\ell+\mathcal{R}-1})\right)$ is an $\mathbb{F}_2$-basis of the space $S^{(i)}$.*

Proof. Indeed, the subspace $V_i \coloneqq \langle \beta_i, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle$ is clearly a subspace of $\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle$, so that in turn $\widehat{W}_i(V_i) \subset \widehat{W}_i(\langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1} \rangle)$, which itself equals $S^{(i)}$ (by Definition 4.1). On the other hand, the restriction of $\widehat{W}_i$ to V_i is necessarily injective, since $\widehat{W}_i$'s kernel $\langle \beta_0, \ldots, \beta_{i-1} \rangle$ intersects V_i trivially. Since $S^{(i)}$ is $\ell + \mathcal{R} - i$-dimensional, we conclude by a dimension count that $\left(\widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{\ell+\mathcal{R}-1})\right)$ spans $S^{(i)}$. □

The bases $\left\langle \widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{\ell+\mathcal{R}-1}) \right\rangle = S^{(i)}$, for $i \in \{0, \ldots, \ell\}$, serve to simplify various aspects of our protocol's implementation. For example, expressed in coordinates with respect to these bases, each map $q^{(i)} : S^{(i)} \to S^{(i+1)}$ acts simply by projecting away its 0^{th}-indexed component (indeed, for each $i \in \{0, \ldots, \ell - 1\}$, $q^{(i)}$ maps $(\widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{\ell+\mathcal{R}-1}))$ to $(0, \widehat{W}_{i+1}(\beta_{i+1}), \ldots, \widehat{W}_{i+1}(\beta_{\ell+\mathcal{R}-1}))$). Similarly, for each $i \in \{0, \ldots, \ell - 1\}$ and each $y \in S^{(i+1)}$, the two L-elements $x \in S^{(i)}$ for which $q^{(i)}(x) = y$ differ precisely at their 0^{th} components, and elsewhere agree with y's coordinate representation. Below, we often identify $S^{(i)} \cong \mathcal{B}_{\ell+\mathcal{R}-i}$ as sets, using these bases; moreover, where possible, we eliminate altogether the maps $q^{(0)}, \ldots, q^{(\ell-1)}$ from our descriptions. These measures make our protocol's description and implementation more transparent.

4.2 FRI Folding, Revisited

We now introduce a new FRI-like folding mechanism. Below, we again write L for a binary field.

Definition 4.6. We fix an index $i \in \{0, \ldots, \ell - 1\}$ and a map $f^{(i)} : S^{(i)} \to L$. For each $r \in L$, we define the map $\mathsf{fold}\left(f^{(i)}, r\right) : S^{(i+1)} \to L$ by setting, for each $y \in S^{(i+1)}$:

$$\mathsf{fold}\left(f^{(i)}, r\right) : y \mapsto \begin{bmatrix} 1 - r & r \end{bmatrix} \cdot \begin{bmatrix} x_1 & -x_0 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) \\ f^{(i)}(x_1) \end{bmatrix},$$

where we write $(x_0, x_1) \coloneqq {q^{(i)}}^{-1}(\{y\})$ for the fiber of $q^{(i)}$ over $y \in S^{(i+1)}$.

Remark 4.7. Definition 4.6's quantity $\mathsf{fold}\left(f^{(i)}, r\right)(y)$ is closely related—and yet not equivalent—to FRI's expression $\mathsf{interpolant}\left(f^{(i)}\big|_{{q^{(i)}}^{-1}(\{y\})}\right)(r)$. (FRI's variant, however, admits a similar matrix expression.) The essential point is that FRI's variant induces a monomial fold, as opposed to a Lagrange fold; that is, if we were to use FRI's variant instead of our own, then our Lemma 4.13 below would remain true, albeit with the alternate conclusion $P^{(i+1)}(X) = \sum_{j=0}^{2^{\ell-i-1}-1} (a_{2j} + r'_i \cdot a_{2j+1}) \cdot X_j^{(i+1)}(X)$. Our entire theory admits a parallel variant in this latter setting, though that variant introduces further complications.

We finally record the following iterated extension of Definition 4.6.

Definition 4.8. We fix a positive folding factor ϑ, an index $i \in \{0, \ldots, \ell - \vartheta\}$, and a map $f^{(i)} : S^{(i)} \to L$. For each tuple $(r_0, \ldots, r_{\vartheta-1}) \in L^{\vartheta}$, we abbreviate $\mathsf{fold}\left(f^{(i)}, r_0, \ldots, r_{\vartheta-1}\right) \coloneqq \mathsf{fold}\left(\cdots \mathsf{fold}\left(f^{(i)}, r_0\right), \cdots, r_{\vartheta-1}\right)$.

We have the following mathematical characterization of this iterated folding operation:

Lemma 4.9. *For each positive folding factor ϑ, each index $i \in \{0, \ldots, \ell - \vartheta\}$, and each $y \in S^{(i+\vartheta)}$, there is a $2^\vartheta \times 2^\vartheta$ invertible matrix M_y,which depends only on $y \in S^{(i+\vartheta)}$, such that, for each function $f^{(i)} : S^{(i)} \to L$ and each tuple $(r_0 \ldots, r_{\vartheta-1}) \in L^\vartheta$ of folding challenges, we have the matrix identity:*

$$\mathsf{fold}\Big(f^{(i)}, r_0, \ldots, r_{\vartheta-1}\Big)(y) = \begin{bmatrix} \bigotimes_{j=0}^{\vartheta-1}(1-r_j, r_j) \end{bmatrix} \cdot \begin{bmatrix} & & \\ & M_y & \\ & & \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) \\ \vdots \\ f^{(i)}(x_{2^\vartheta-1}) \end{bmatrix},$$

where the right-hand vector's values $(x_0, \ldots, x_{2^\vartheta-1})$ represent the fiber $\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\}) \subset S^{(i)}$.

Proof. We prove the result by induction on ϑ. In the base case $\vartheta = 1$, the claim is a tautology, in view of Definition 4.6. We note that that definition's matrix $\begin{bmatrix} x_1 & -x_0 \\ -1 & 1 \end{bmatrix}$ is invertible, since its determinant $x_1 - x_0$ is nonzero (and in fact equals 1, a fact we shall use below).

We thus fix a folding factor $\vartheta > 1$, and suppose that the claim holds for $\vartheta - 1$. We write $(z_0, z_1) \coloneqq {q^{(i+\vartheta-1)}}^{-1}(\{y\})$, as well as $(x_0, \ldots, x_{2^\vartheta-1}) \coloneqq \left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})$. Unwinding Definition 4.8, we recursively express the relevant quantity $\mathsf{fold}\big(f^{(i)}, r_0, \ldots, r_{\vartheta-1}\big)(y)$—which, for typographical reasons, we call $\mathfrak{f}$—in the following way:

$$\mathfrak{f} = \begin{bmatrix} 1 - r_{\vartheta-1} & r_{\vartheta-1} \end{bmatrix} \cdot \begin{bmatrix} z_1 & -z_0 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathsf{fold}\big(f^{(i)}, r_0, \ldots, r_{\vartheta-2}\big)(z_0) \\ \mathsf{fold}\big(f^{(i)}, r_0, \ldots, r_{\vartheta-2}\big)(z_1) \end{bmatrix}$$

$$= \begin{bmatrix} 1 - r_{\vartheta-1} & r_{\vartheta-1} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} z_1 & -z_0 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \bigotimes_{j=0}^{\vartheta-2}(1-r_j, r_j) & \\ & \bigotimes_{j=0}^{\vartheta-2}(1-r_j, r_j) \end{bmatrix}}_{\text{these matrices may be interchanged.}} \cdot \begin{bmatrix} M_{z_0} & \\ & M_{z_1} \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) \\ \vdots \\ f^{(i)}(x_{2^\vartheta-1}) \end{bmatrix}.$$

In the second step above, we apply the inductive hypothesis on both z_0 and z_1. That hypothesis furnishes the nonsingular, $2^{\vartheta-1} \times 2^{\vartheta-1}$ matrices M_{z_0} and M_{z_1}; we note moreover that the union of the fibers $\left(q^{(i+\vartheta-2)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{z_0\})$ and $\left(q^{(i+\vartheta-2)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{z_1\})$ is precisely $\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})$. Interchanging the two matrices bracketed above, we further reexpress this quantity as:

$$= \begin{bmatrix} 1 - r_{\vartheta-1} & r_{\vartheta-1} \end{bmatrix} \cdot \begin{bmatrix} \bigotimes_{j=0}^{\vartheta-2}(1-r_j, r_j) & \\ & \bigotimes_{j=0}^{\vartheta-2}(1-r_j, r_j) \end{bmatrix} \cdot \begin{bmatrix} \mathsf{diag}(z_1) & \mathsf{diag}(-z_0) \\ \mathsf{diag}(-1) & \mathsf{diag}(1) \end{bmatrix} \cdot \begin{bmatrix} M_{z_0} & \\ & M_{z_1} \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) \\ \vdots \\ f^{(i)}(x_{2^\vartheta-1}) \end{bmatrix}.$$

By the standard recursive substructure of the tensor product, the product of the left-hand two matrices equals exactly $\bigotimes_{j=0}^{\vartheta-1}(1-r_j, r_j)$. On the other hand, the product of the two $2^\vartheta \times 2^\vartheta$ nonsingular matrices above is itself nonsingular, and supplies the required $2^\vartheta \times 2^\vartheta$ matrix M_y. □

We emphasize that, in Lemma 4.9, the matrix M_y depends only on $y \in S^{(i+\vartheta)}$—and of course on ϑ and $i \in \{0, \ldots, \ell - \vartheta\}$—and not on the map $f^{(i)}$ or the folding challenges $(r_0, \ldots, r_{\vartheta-1}) \in L^\vartheta$.

Remark 4.10. Interestingly, the matrix M_y of Lemma 4.9 is nothing other than that of the *inverse additive NTT* [LCH14, § III. C.] on the coset $(x_0, \ldots, x_{2^\vartheta-1})$; i.e., it's the matrix which, on input the evaluations of some polynomial of degree less than 2^ϑ on the set of elements $(x_0, \ldots, x_{2^\vartheta-1})$, returns the coefficients—with respect to the i^{th}-order novel basis (see Remark 4.15 below)—of that polynomial.

4.3 Our Large-Field IOPCS

We now present our binary adaptation of BaseFold's IOPCS [ZCF24, § 5], itself based on the material of our Subsections 4.1 and 4.2 above. In order to present a notationally simpler variant of our protocol, we assume below that $\vartheta \mid \ell$; this requirement is not necessary.

CONSTRUCTION 4.11 (Binary BaseFold IOPCS)**.**
We define $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ as follows.

1. $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell)$. On input 1^λ and ℓ, choose a constant, positive rate parameter $\mathcal{R} \in \mathbb{N}$ and a binary field $L/\mathbb{F}_2$ whose degree r (say) over $\mathbb{F}_2$ satisfies $r = \omega(\log \lambda)$ and $r \geq \ell + \mathcal{R}$. Initialize the vector oracle $\mathcal{F}_{\mathsf{Vec}}^L$. Fix a folding factor $\vartheta \mid \ell$ and a repetition parameter $\gamma = \omega(\log \lambda)$. Fix an arbitrary $\mathbb{F}_2$-basis $(\beta_0, \ldots, \beta_{r-1})$ of L. Write $(X_0(X), \ldots, X_{2^\ell - 1}(X))$ for the resulting novel L-basis of $L[X]^{\prec 2^\ell}$, and fix the domains $S^{(0)}, \ldots, S^{(\ell)}$ and the polynomials $q^{(0)}, \ldots, q^{(\ell-1)}$ as in Subsection 4.1. Write $C^{(0)} \subset L^{2^{\ell+\mathcal{R}}}$ for the Reed–Solomon code $\mathsf{RS}_{L,S^{(0)}}[2^{\ell+\mathcal{R}}, 2^\ell]$.

2. $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, use t's Lagrange coefficients $(t(w))_{w \in \mathcal{B}_\ell}$ as the coefficients, in the novel polynomial basis, of a univariate polynomial $P(X) := \sum_{w \in \mathcal{B}_\ell} t(w) \cdot X_{\{w\}}(X)$, say. Using Algorithm 2, compute the Reed–Solomon codeword $f : S^{(0)} \to L$ defined by $f : x \mapsto P(x)$. Submit $(\texttt{submit}, \ell + \mathcal{R}, f)$ to the vector oracle $\mathcal{F}_{\mathsf{Vec}}^L$. Upon receiving $(\texttt{receipt}, \ell + \mathcal{R}, [f])$ from $\mathcal{F}_{\mathsf{Vec}}^L$, output the handle $[f]$.

We define $(\mathcal{P}, \mathcal{V})$ as the following IOP, in which both parties have the common input $[f]$, $s \in L$, and $(r_0, \ldots, r_{\ell-1}) \in L^\ell$, and $\mathcal{P}$ has the further input $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.

1. $\mathcal{P}$ writes $h(X_0, \ldots, X_{\ell-1}) := \widetilde{\texttt{eq}}(r_0, \ldots, r_{\ell-1}, X_0, \ldots, X_{\ell-1}) \cdot t(X_0, \ldots, X_{\ell-1})$.

2. $\mathcal{P}$ and $\mathcal{V}$ both abbreviate $f^{(0)} := f$ and $s_0 := s$, and execute the following loop:

   1: **for** $i \in \{0, \ldots, \ell - 1\}$ **do**
   2: $\quad$ $\mathcal{P}$ sends $\mathcal{V}$ the polynomial $h_i(X) := \sum_{w \in \mathcal{B}_{\ell-i-1}} h(r'_0, \ldots, r'_{i-1}, X, w_0, \ldots, w_{\ell-i-2})$.
   3: $\quad$ $\mathcal{V}$ requires $s_i \stackrel{?}{=} h_i(0) + h_i(1)$. $\mathcal{V}$ samples $r'_i \leftarrow L$, sets $s_{i+1} := h_i(r'_i)$, and sends $\mathcal{P}$ r'_i.
   4: $\quad$ $\mathcal{P}$ defines $f^{(i+1)} : S^{(i+1)} \to L$ as the function $\mathsf{fold}\big(f^{(i)}, r'_i\big)$ of Definition 4.6.
   5: $\quad$ **if** $i + 1 = \ell$ **then** $\mathcal{P}$ sends $c := f^{(\ell)}(0, \ldots, 0)$ to $\mathcal{V}$.
   6: $\quad$ **else if** $\vartheta \mid i + 1$ **then** $\mathcal{P}$ submits $(\texttt{submit}, \ell + \mathcal{R} - i - 1, f^{(i+1)})$ to the oracle $\mathcal{F}_{\mathsf{Vec}}^L$.

3. $\mathcal{V}$ requires $s_\ell \stackrel{?}{=} \widetilde{\texttt{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}) \cdot c$.

4. $\mathcal{V}$ executes the following querying procedure:

   1: **for** γ repetitions **do**
   2: $\quad$ $\mathcal{V}$ samples $v \leftarrow \mathcal{B}_{\ell+\mathcal{R}}$ randomly.
   3: $\quad$ **for** $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$ (i.e., taking ϑ-sized steps) **do**
   4: $\quad\quad$ for each $u \in \mathcal{B}_\vartheta$, $\mathcal{V}$ sends $\big(\texttt{query}, [f^{(i)}], (u_0, \ldots, u_{\vartheta-1}, v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})\big)$ to the oracle.
   5: $\quad\quad$ **if** $i > 0$ **then** $\mathcal{V}$ requires $c_i \stackrel{?}{=} f^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1})$.
   6: $\quad\quad$ $\mathcal{V}$ defines $c_{i+\vartheta} := \mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$.
   7: $\quad$ $\mathcal{V}$ requires $c_\ell \stackrel{?}{=} c$.

In our commitment procedure above, we give meaning to the commitment of f by implicitly identifying $S^{(0)} \cong \mathcal{B}_{\ell+\mathcal{R}}$ as sets (as discussed above); similarly, in the prover's line 6 above, we identify $\mathcal{B}_{\ell+\mathcal{R}-i-1} \cong S^{(i+1)}$. Conversely, in its lines 4 and 6 above, the verifier must implicitly identify the $\mathcal{B}_{\ell+\mathcal{R}-i}$-elements $(u_0, \ldots, u_{\vartheta-1}, v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})_{u \in \mathcal{B}_\vartheta}$ with $S^{(i)}$-elements—and the $\mathcal{B}_{\ell+\mathcal{R}-i-\vartheta}$-element $(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$ with an $S^{(i+\vartheta)}$-element—in order to appropriately apply Definition 4.8. We note that, in line 6, $\mathcal{V}$ has precisely the information it needs to compute $\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$ (namely, the values of $f^{(i)}$ on the fiber $(u_0, \ldots, u_{\vartheta-1}, v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})_{u \in \mathcal{B}_\vartheta} \cong \big(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\big)^{-1}(\{(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})\})$).

The completness of Construction 4.11's evaluation IOP is not straightforward. For instance, it is simply not obvious what the folding operation of line 4 does to the coefficients of the low-degree polynomial $P^{(i)}(X)$ underlying $f^{(i)}$. (Though our folding operation departs slightly from FRI's—we refer to Remark 4.7 for a discussion of this fact—the conceptual obstacle is essentially the same.) Indeed, the completeness proof of generic FRI [BBHR18a, § 4.1.1] tells us that the folded function $f^{(i+1)}$ represents the evaluations of *some* polynomial $P^{(i+1)}(X)$ of appropriate degree on the domain $S^{(i+1)}$. But which one? The proof of [BBHR18a, § 4.1.1] fails to constructively answer this question, in that it invokes the generic characteristics of the multivariate reduction—called $Q^{(i)}(X,Y)$—of $P^{(i)}(X)$ by $Y - q^{(i)}(X)$. (We refer to e.g. von zur Gathen and Gerhard [GG13, Alg. 21.11] for a thorough treatment of multivariate division.) It seems simply infeasible to analyze by hand the execution of the multivariate division algorithm with sufficient fidelity as to determine with any precision the result $P^{(i+1)}(Y) = Q^{(i)}(r'_i, Y)$ (though we don't rule out the prospect whereby a proof could in principle be achieved in this way).

Instead, we introduce certain, carefully-selected L-bases of the spaces $L[X]^{\prec 2^{\ell-i}}$, for $i \in \{0,\ldots,\ell\}$ (so-called "higher-order" novel polynomial bases). As it turns out, the respective coefficients of $P^{(i)}(X)$ and $P^{(i+1)}(X)$ *with respect to these bases* are tractably related; their relationship amounts to an even–odd tensor-fold by the FRI challenge r'_i. Proceeding by induction, we obtain the desired characterization of c.

Theorem 4.12. *The IOPCS* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *of Construction 4.11 is complete.*

Proof. Provided that $\mathcal{P}$ is honest, $s = t(r_0,\ldots,r_{\ell-1})$ will hold. Since $t(r_0\ldots,r_{\ell-1}) = \sum_{w\in\mathcal{B}_\ell} h(w)$, this guarantee implies that $s = s_0 = \sum_{w\in\mathcal{B}_\ell} h(w)$ will hold, so that, by the completeness of the sumcheck, $\mathcal{V}$'s checks $s_i \stackrel{?}{=} h_i(0)+h_i(1)$ will pass. Finally, $s_\ell = h(r'_0,\ldots,r'_{\ell-1}) = \widetilde{\mathsf{eq}}(r_0,\ldots,r_{\ell-1},r'_0,\ldots,r'_{l-1})\cdot t(r'_0,\ldots,r'_{\ell-1})$ too will hold. To argue the completeness of $\mathcal{V}$'s check $s_\ell \stackrel{?}{=} \widetilde{\mathsf{eq}}(r_0,\ldots,r_{\ell-1},r'_0,\ldots,r'_{\ell-1})\cdot c$ above, it thus suffices to argue that, for $\mathcal{P}$ honest, $c = t(r'_0,\ldots,r'_{\ell-1})$ will hold.

We introduce a family of further polynomial bases. For each $i \in \{0,\ldots,\ell-1\}$, we define the i^{th}*-order subspace vanishing polynomials* $\widehat{W}^{(i)}_0,\ldots,\widehat{W}^{(i)}_{\ell-i-1}$ as the polynomials $X, q^{(i)}, q^{(i+1)}\circ q^{(i)},\ldots,q^{(\ell-2)}\circ\cdots\circ q^{(i)}$, respectively (that is, $\widehat{W}^{(i)}_k \coloneqq q^{(i+k-1)}\circ\cdots\circ q^{(i)}$, for each $k\in\{0,\ldots,\ell-i-1\}$). Finally, we define the i^{th}*-order novel polynomial basis* by setting $X^{(i)}_j \coloneqq \prod_{k=0}^{\ell-i-1} \widehat{W}^{(i)\,j_k}_k$, for each $j\in\{0,\ldots,2^{\ell-i}-1\}$ (here, again, we write $(j_0,\ldots,j_{\ell-i-1})$ for the bits of j). We adopt the notational convention whereby the ℓ^{th}-order basis consists simply of the constant polynomial $X^{(\ell)}_0(X) = 1$. Below, we use a certain inductive relationship between the bases $\left(X^{(i)}_j(X)\right)_{j=0}^{2^{\ell-i}-1}$ and $\left(X^{(i+1)}_j(X)\right)_{j=0}^{2^{\ell-i-1}-1}$; that is, for each $j\in\{0,\ldots,2^{\ell-i-1}-1\}$, the polynomials $X^{(i)}_{2j}(X)$ and $X^{(i)}_{2j+1}(X)$ respectively equal $X^{(i+1)}_j\big(q^{(i)}(X)\big)$ and $X\cdot X^{(i+1)}_j\big(q^{(i)}(X)\big)$.

Lemma 4.13. *Fix an index* $i\in\{0,\ldots,\ell-1\}$*. If* $f^{(i)}: S^{(i)}\to L$ *is exactly the evaluation over* $S^{(i)}$ *of the polynomial* $P^{(i)}(X) = \sum_{j=0}^{2^{\ell-i}-1} a_j\cdot X^{(i)}_j(X)$*, then, under honest prover behavior,* $f^{(i+1)}: S^{(i+1)}\to L$ *is exactly the evaluation over* $S^{(i+1)}$ *of the polynomial* $P^{(i+1)}(X) = \sum_{j=0}^{2^{\ell-i-1}-1}((1-r'_i)\cdot a_{2j} + r'_i\cdot a_{2j+1})\cdot X^{(i+1)}_j(X)$*.*

Proof. Given $P^{(i)}(X)$ as in the hypothesis of the lemma, we introduce the *even and odd refinements* $P^{(i+1)}_0(X) \coloneqq \sum_{j=0}^{2^{\ell-i-1}-1} a_{2j}\cdot X^{(i+1)}_j(X)$ and $P^{(i+1)}_1(X) \coloneqq \sum_{j=0}^{2^{\ell-i-1}-1} a_{2j+1}\cdot X^{(i+1)}_j(X)$ of $P^{(i)}(X)$. We note the following key polynomial identity:

$$P^{(i)}(X) = P^{(i+1)}_0(q^{(i)}(X)) + X\cdot P^{(i+1)}_1(q^{(i)}(X)); \tag{39}$$

This identity is a direct consequence of the definitions of the higher-order novel polynomial bases.

We turn to the proof of the lemma. We claim that $f^{(i+1)}(y) = P^{(i+1)}(y)$ holds for each $y\in S^{(i+1)}$, where $P^{(i+1)}(X)$ is as in the lemma's hypothesis. To this end, we let $y\in S^{(i+1)}$ be arbitrary; we moreover write $(x_0,x_1) \coloneqq {q^{(i)}}^{-1}(\{y\})$ for the fiber of $q^{(i)}$ over y. We begin by examining the values $P^{(i)}(x_0)$ and $P^{(i)}(x_1)$. For each $b\in\{0,1\}$ we have:

$$\begin{aligned} P^{(i)}(x_b) &= P^{(i+1)}_0\Big(q^{(i)}(x_b)\Big) + x_b\cdot P^{(i+1)}_1\Big(q^{(i)}(x_b)\Big) && \text{(by the identity (39).)} \\ &= P^{(i+1)}_0(y) + x_b\cdot P^{(i+1)}_1(y). && \text{(using } q^{(i)}(x_b) = y.\text{)} \end{aligned}$$

Using now our assumption whereby $f^{(i)}(x_b) = P^{(i)}(x_b)$ for each $b \in \{0, 1\}$, and unwinding the prescription of Definition 4.6, we obtain:

$$\begin{aligned}
f^{(i+1)}(y) &= \begin{bmatrix} 1 - r'_i & r'_i \end{bmatrix} \cdot \begin{bmatrix} x_1 & -x_0 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} P^{(i)}(x_0) \\ P^{(i)}(x_1) \end{bmatrix} && \text{(by our hypothesis on } f^{(i)}\text{, and by Definition 4.6.)} \\
&= \begin{bmatrix} 1 - r'_i & r'_i \end{bmatrix} \cdot \begin{bmatrix} x_1 & -x_0 \\ -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix} \cdot \begin{bmatrix} P_0^{(i+1)}(y) \\ P_1^{(i+1)}(y) \end{bmatrix} && \text{(by the calculation just performed above.)} \\
&= \begin{bmatrix} 1 - r'_i & r'_i \end{bmatrix} \cdot \begin{bmatrix} P_0^{(i+1)}(y) \\ P_1^{(i+1)}(y) \end{bmatrix} && \text{(cancellation of inverse matrices.)} \\
&= P^{(i+1)}(y). && \text{(by the definitions of } P_0^{(i+1)}(X)\text{, } P_1^{(i+1)}(X)\text{, and } P^{(i+1)}(X)\text{.)}
\end{aligned}$$

To achieve the third equality above, we note that the matrices $\begin{bmatrix} x_1 & -x_0 \\ -1 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix}$ are inverses; there, we use the guarantee $x_1 - x_0 = 1$, a basic consequence of Definition 4.1 (or rather of $\ker\left(q^{(i)}\right) = \{0, 1\}$). ☐

Applying Corollary 4.4, we note finally that $\left(\widehat{W}_k^{(0)}\right)_{k=0}^{\ell-1}$ and $\left(X_j^{(0)}\right)_{j=0}^{2^\ell-1}$ themselves equal precisely the standard subspace vanishing and novel basis polynomials, respectively. It follows that in the base case $i = 0$ of Lemma 4.13—and again assuming honest behavior by the prover—we have that $f^{(0)}$ will equal the evaluation over $S^{(0)}$ of $P^{(0)}(X) \coloneqq P(X) = \sum_{w \in \mathcal{B}_\ell} t(w) \cdot X_{\{w\}}^{(0)}(X)$. Applying Lemma 4.13 repeatedly, we conclude by induction that $f^{(\ell)}$ will equal the evaluation over $S^{(\ell)}$ of the constant polynomial $\sum_{w \in \mathcal{B}_\ell} \widetilde{\mathsf{eq}}(w_0, \ldots, w_{\ell-1}, r'_0, \ldots, r'_{\ell-1}) \cdot t(w) = t(r'_0, \ldots, r'_{\ell-1})$, so that $c = t(r'_0, \ldots, r'_{\ell-1})$ will hold, as desired.

The completeness of the verifier's query phase is self-evident (and is just as in [BBHR18a, § 4.1.1]); we note that $\mathcal{V}$ applies to each oracle $f^{(i)}$ the same folding procedure that $\mathcal{P}$ does. This completes the proof of completeness. ☐

Remark 4.14. Using the techniques of Subsection 4.1 and of Theorem 4.12 above, we are able to suggest a new explanation of the additive NTT algorithm of Lin, Chung and Han [LCH14, § III.], and of its correctness; we note also our Algorithm 2 above. (Li et al. [Li+18, Alg. 2] present yet a further—and very interesting—perspective, which differs both from ours and from Lin–Chung–Han's.) We fix an index $i \in \{0, \ldots, \ell - 1\}$ and a polynomial $P^{(i)}(X) \coloneqq \sum_{j=0}^{2^{\ell-i}-1} a_j \cdot X_j^{(i)}(X)$ expressed with respect to the i^{th}-order novel basis. The key idea is that the values of $P^{(i)}(X)$ on the domain $S^{(i)}$ can be derived—using only $O\left(2^{\ell+\mathcal{R}-i}\right)$ K-operations—given the values of $P^{(i)}(X)$'s even and odd refinements $P_0^{(i+1)}(X)$ and $P_1^{(i+1)}(X)$ (as in the proof of Lemma 4.13) over the domain $S^{(i+1)}$. This is a direct consequence of the identity (39) above. Indeed, applying that identity, we see that, for $y \in S^{(i+1)}$ arbitrary, with fiber $(x_0, x_1) \coloneqq {q^{(i)}}^{-1}(\{y\})$, say, we have the equalities $P^{(i)}(x_0) \coloneqq P_0^{(i+1)}(y) + x_0 \cdot P_1^{(i+1)}(y)$ and $P^{(i)}(x_1) \coloneqq P_0^{(i+1)}(y) + x_1 \cdot P_1^{(i+1)}(y)$. Since x_0 and x_1 in fact differ by exactly 1, we see that $P^{(i)}(x_1)$ can be computed from $P^{(i)}(x_0)$ using a single further K-addition. We recover the key butterfly diagram of [LCH14, Fig. 1. (a)] (see also Algorithm 2 above) upon carrying out this procedure recursively, with the convention whereby we flatten (using the space's canonical basis) and interleave the two copies of $S^{(i+1)}$ at each instance. The base case of the recursion consists of the 2^ℓ-fold interleaving of the domain $S^{(\ell)}$, into which $P^{(0)}$'s coefficients are tiled $2^{\mathcal{R}}$ times. The final stage of the butterfly diagram yields the desired evaluation of $P^{(0)}(X)$ on $S^{(0)}$. Algorithm 2's twiddle factors in its i^{th} stage, then, are nothing other than the respective first lifts x_0 of y, as the image $y = q^{(i)}(x_0)$ varies throughout $S^{(i+1)}$. These latter elements x_0, in turn, take precisely the form $\sum_{k=0}^{\ell+\mathcal{R}-i-2} u_k \cdot \widehat{W}_i(\beta_{i+1+k})$, for $u \in \mathcal{B}_{\ell+\mathcal{R}-i-1} \cong S^{(i+1)}$ arbitrary; indeed, we suppress throughout the 0^{th} canonical basis element $\widehat{W}_i(\beta_i) = 1$ of $S^{(i)}$, since that element is subsumed into each butterfly. We find it interesting that the same polynomial identity underlies both the correctness of [LCH14, § III.] and our above analysis of FRI's folding.

Remark 4.15. Though it seems inessential to the proof of Theorem 4.12, it is interesting to note that, for each $i \in \{0, \ldots, \ell-1\}$, the i^{th}-order basis $\left(X_j^{(i)}\right)_{j=0}^{2^{\ell-i}-1}$ is itself a novel polynomial basis in its own right, namely that attached to the set of vectors $\left(\widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{\ell-1})\right)$. Equivalently, the i^{th}-order subspace vanishing polynomials $\left(\widehat{W}_k^{(i)}\right)_{k=0}^{\ell-i-1}$ are simply the subspace vanishing polynomials attached to this latter set of vectors. Indeed, for each $k \in \{0, \ldots, \ell-i-1\}$, $\left\langle \widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{i+k-1}) \right\rangle \subset \ker\left(\widehat{W}_k^{(i)}\right)$ certainly holds, since $\widehat{W}_k^{(i)} \circ \widehat{W}_i = q^{(i+k-1)} \circ \cdots \circ q^{(i)} \circ \widehat{W}_i = \widehat{W}_{i+k}$, which annihilates $\langle \beta_0, \ldots, \beta_{i+k-1} \rangle$ (here, we use the definition of $\widehat{W}_k^{(i)}$ and Lemma 4.2). On the other hand, $\widehat{W}_k^{(i)} = q^{(i+k-1)} \circ \cdots \circ q^{(i)}$'s kernel can be of dimension at most k (say by degree considerations), while the vectors $\widehat{W}_i(\beta_i), \ldots, \widehat{W}_i(\beta_{i+k-1})$ are linearly independent (a consequence of Corollary 4.5). We conclude that the above containment is an equality. Finally, the subspace polynomials $\left(\widehat{W}_k^{(i)}\right)_{k=0}^{\ell-i-1}$ are normalized. Indeed, using Lemma 4.2 again, we see that, for each $k \in \{0, \ldots, \ell-i-1\}$, $\widehat{W}_k^{(i)}\left(\widehat{W}_i(\beta_{i+k})\right) = \left(q^{(i+k-1)} \circ \cdots \circ q^{(i)} \circ \widehat{W}_i\right)(\beta_{i+k}) = \widehat{W}_{i+k}(\beta_{i+k}) = 1$ holds.

We now prove the security of Construction 4.11. Our key technical results below (see Propositions 4.20 and 4.23), essentially, jointly constitute a variant of FRI's soundness statement [BBHR18a, § 4.2.2]. Our proofs of these results incorporate—albeit in an attenuated way—various ideas present in [BBHR18a, § 4.2.2] and [Ben+23, § 8.2]. We also introduce a number of new ideas, which, by and large, pertain to our new folding technique (see Subsection 4.2).

We note that our protocol seems not to admit a security proof which invokes that of FRI in a strictly blackbox manner. Rather, our security argument—and, it would seem, any conceivable analysis of Construction 4.11—must inevitably concern itself not merely with the distance from the code of $\mathcal{A}$'s initial committed word, but moreover with the consistency of its oracles, and in particular with whether its final oracle value c relates as it should to its initial oracle.

Theorem 4.16. *The IOPCS* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *of Construction 4.11 is secure.*

Proof. We define a straight-line emulator $\mathcal{E}$ as follows.

1. By inspecting $\mathcal{A}$'s messages to the vector oracle, $\mathcal{E}$ immediately recovers the function $f : S^{(0)} \to L$ underlying the handle $[f]$ output by $\mathcal{A}$.

2. $\mathcal{E}$ runs the Berlekamp–Welch decoder (i.e., Algorithm 1) on the word $f : S^{(0)} \to L$. If that algorithm outputs $P(X) = \bot$ or if $\deg(P(X)) \geq 2^\ell$, then $\mathcal{E}$ outputs $\bot$ and aborts.

3. Otherwise, $\mathcal{E}$ re-expresses the Berlekamp–Welch output polymomial $P(X) = \sum_{w \in \mathcal{B}_\ell} t_w \cdot X_{\{w\}}(X)$ in coordinates with respect to the novel polynomial basis. $\mathcal{E}$ writes $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$ for the multilinear whose Lagrange coordinates are $(t_w)_{w \in \mathcal{B}_\ell}$. $\mathcal{E}$ outputs $t(X_0, \ldots, X_{\ell-1})$ and halts.

We begin by defining various notions, adapting [BBHR18a, § 4.2.1]. For each $i \in \{0, \vartheta, \ldots, \ell\}$ (i.e., ascending in ϑ-sized steps), we write $C^{(i)} \subset L^{2^{\ell+\mathcal{R}-i}}$ for the Reed–Solomon code $\mathsf{RS}_{L,S^{(i)}}[2^{\ell+\mathcal{R}-i}, 2^{\ell-i}]$. We recall that $C^{(i)}$ is of distance $d_i \coloneqq 2^{\ell+\mathcal{R}-i} - 2^{\ell-i} + 1$. We write $f^{(0)}, f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ for the oracles committed by $\mathcal{A}$; we moreover write $f^{(\ell)} : S^{(\ell)} \to L$ for the identically-c function (here, $c \in L$ is $\mathcal{A}$'s final FRI message). For each $i \in \{0, \vartheta, \ldots, \ell-\vartheta\}$, we write $\Delta\left(f^{(i+\vartheta)}, g^{(i+\vartheta)}\right) \subset S^{(i+\vartheta)}$ for the *disagreement* set between the elements $f^{(i+\vartheta)}$ and $g^{(i+\vartheta)}$ of $L^{2^{\ell+\mathcal{R}-i-\vartheta}}$; that is, $\Delta\left(f^{(i+\vartheta)}, g^{(i+\vartheta)}\right)$ is the set of elements $y \in S^{(i+\vartheta)}$ for which $f^{(i+\vartheta)}(y) \neq g^{(i+\vartheta)}(y)$. We moreover write $\Delta^{(i)}\left(f^{(i)}, g^{(i)}\right) \subset S^{(i+\vartheta)}$ for the *fiber-wise disagreement set* of the elements $f^{(i)}$ and $g^{(i)}$ of $L^{2^{\ell+\mathcal{R}-i}}$. That is, $\Delta^{(i)}\left(f^{(i)}, g^{(i)}\right) \subset S^{(i+\vartheta)}$ denotes the set of elements $y \in S^{(i+\vartheta)}$ for which the respective restrictions of $f^{(i)}$ and $g^{(i)}$ to the fiber $\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\}) \subset S^{(i)}$ are not identically equal. We define $d^{(i)}\left(f^{(i)}, C^{(i)}\right) \coloneqq \min_{g^{(i)} \in C^{(i)}} \left|\Delta^{(i)}\left(f^{(i)}, g^{(i)}\right)\right|$. We note that, if $d^{(i)}\left(f^{(i)}, C^{(i)}\right) < \frac{d_{i+\vartheta}}{2}$, then $d\left(f^{(i)}, C^{(i)}\right) < \frac{d_i}{2}$ *a fortiori* holds. (Each offending fiber contributes at most 2^ϑ errors; on the other hand, $2^\vartheta \cdot \left\lfloor \frac{d_{i+\vartheta}-1}{2} \right\rfloor \leq \left\lfloor \frac{d_i-1}{2} \right\rfloor$.) In any case, in case the oracle $f^{(i)} : S^{(i)} \to L$ is such that $d\left(f^{(i)}, L^{(i)}\right) < \frac{d_i}{2}$ happens to hold, we write $\overline{f}^{(i)} \in L^{(i)}$ for the unique codeword for which $d\left(f^{(i)}, \overline{f}^{(i)}\right) < \frac{d_i}{2}$.

We record the following key compliance condition:

Definition 4.17. For each index $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$, we say that $\mathcal{A}$'s i^{th} oracle $f^{(i)}$ is *compliant* if the conditions $d^{(i)}\big(f^{(i)}, C^{(i)}\big) < \frac{d_i}{2}$, $d\big(f^{(i+\vartheta)}, C^{(i+\vartheta)}\big) < \frac{d_{i+\vartheta}}{2}$, and $\overline{f}^{(i+\vartheta)} = \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)$ all hold.

We first argue that if any among $\mathcal{A}$'s oracles $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$ is not compliant, then $\mathcal{V}$ will accept with negligible probability at most. This is exactly Proposition 4.23 below. In order to prepare for that proposition, we record a sequence of lemmas. We begin with the following elementary fact.

Lemma 4.18. *For each $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$, if $d\big(f^{(i)}, C^{(i)}\big) < \frac{d_i}{2}$, then, for each tuple of folding challenges $(r'_i, \ldots, r'_{i+\vartheta-1}) \in L^\vartheta$, we have that $\Delta\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)\big) \subset \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$.*

Proof. We proceed by contraposition; we fix an element $y \notin \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$. By definition of that latter set, we conclude immediately that the restrictions $f^{(i)}|_{\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})} = \overline{f}^{(i)}|_{\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})}$ are identically equal. Applying Definition 4.8, we see under this guarantee that, regardless of the challenges $(r'_i, \ldots, r'_{i+\vartheta-1})$, $\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(y) = \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(y)$ necessarily also holds. □

We now define a sequence of bad folding events. Our definition of E_i is case-based, and depends on the status of $f^{(i)}$. If $f^{(i)}$ is within the (fiber-wise) unique decoding radius, then E_i captures the event whereby the generic inclusion of Lemma 4.18 becomes strict. Otherwise, E_i captures the "bad batching" event whereby $\mathsf{fold}(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1})$ becomes close to $C^{(i+\vartheta)}$.

Definition 4.19. For each $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$, we define the *bad subset* $E_i \subset L^\vartheta$ as the set of tuples $(r'_i, \ldots, r'_{i+\vartheta-1}) \in L^\vartheta$ for which, as the case may be:

in case $\boldsymbol{d^{(i)}\big(f^{(i)}, C^{(i)}\big) < \frac{d_{i+\vartheta}}{2}}$: $\Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big) \not\subset \Delta\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)\big)$.

in case $\boldsymbol{d^{(i)}\big(f^{(i)}, C^{(i)}\big) \geq \frac{d_{i+\vartheta}}{2}}$: $d\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), C^{(i+\vartheta)}\big) < \frac{d_{i+\vartheta}}{2}$.

We now bound the bad subsets E_i of Definition 4.19. We recall that $\mu(E_i) \coloneqq \frac{|E_i|}{|L|^\vartheta}$ denotes the probability mass of the set $E_i \subset L^\vartheta$.

Proposition 4.20. *For each $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$, $\mu(E_i) \leq \vartheta \cdot \frac{\left|S^{(i+\vartheta)}\right|}{|L|}$ holds.*

Proof. We treat separately the two cases of Definition 4.19.

We begin with the first case. We fix an element $y \in \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$, we moreover write $E_i^y \subset L^\vartheta$ for the set of tuples $(r'_i, \ldots, r'_{i+\vartheta-1}) \in L^\vartheta$ for which $y \notin \Delta\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)\big)$. We argue that $\mu(E_i^y) \leq \frac{\vartheta}{|L|}$. This latter claim suffices to complete the proof of the first case; indeed, since $E_i = \bigcup_{y \in \Delta^{(i)}\left(f^{(i)}, \overline{f}^{(i)}\right)} E_i^y$, assuming the claim, we conclude that $\mu(E_i) \leq \left|\Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)\right| \cdot \frac{\vartheta}{|L|} \leq \left|S^{(i+\vartheta)}\right| \cdot \frac{\vartheta}{|L|}$.

For $y \in \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$ chosen as above, we apply Lemma 4.9 to the words $f^{(i)}$ and $\overline{f}^{(i)}$. Applying that lemma, we see that $(r'_i, \ldots, r'_{i+\vartheta-1}) \in E_i^y$ holds if and only if we have the following matrix identity:

$$0 = \begin{bmatrix} & \bigotimes_{j=0}^{\vartheta-1}(1 - r'_{i+j}, r'_{i+j}) & \end{bmatrix} \cdot \begin{bmatrix} & & \\ & M_y & \\ & & \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) - \overline{f}^{(i)}(x_0) \\ \vdots \\ f^{(i)}(x_{2^\vartheta-1}) - \overline{f}^{(i)}(x_{2^\vartheta-1}) \end{bmatrix}, \tag{40}$$

where we again write $(x_0, \ldots, x_{2^\vartheta-1}) \coloneqq \left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})$. Our hypothesis $y \in \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$ entails precisely that the right-hand vector of (40) is not identically zero. By Lemma 4.9, M_y is nonsingular; we conclude that the image of the right-hand vector of (40) under M_y is likewise not identically zero. Writing $(a_0, \ldots, a_{2^\vartheta-1})$ for this latter vector—which, we repeat, is not zero—we conclude that $E_i^y \subset L^\vartheta$ is precisely the vanishing locus in L^ϑ of the ϑ-variate polynomial $s(X_0, \ldots, X_{\vartheta-1}) \coloneqq \sum_{v \in \mathcal{B}_\vartheta} a_{\{v\}} \cdot \widetilde{\mathsf{eq}}(X_0, \ldots, X_{\vartheta-1}, v_0, \ldots, v_{\vartheta-1})$ over L. Since $s(X_0, \ldots, X_{\vartheta-1})$'s values on the cube $\{0,1\}^\vartheta \subset L^\vartheta$ are exactly $(a_0, \ldots, a_{2^\vartheta-1})$, $s(X_0, \ldots, X_{\vartheta-1})$ is certainly not zero. Applying the Schwartz–Zippel lemma to $s(X_0, \ldots, X_{\vartheta-1})$, we conclude that the relevant locus $E_i^y \subset L^\vartheta$ is of mass at most $\mu(E_i^y) \leq \frac{\vartheta}{|L|}$, as required.

We turn to the second case of Definition 4.19; in particular, we assume that $d^{(i)}\left(f^{(i)}, C^{(i)}\right) \geq \frac{d_{i+\vartheta}}{2}$. We define an interleaved word $\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$—i.e., a $2^\vartheta \times 2^{\ell+\mathcal{R}-i-\vartheta}$ matrix, with entries in L—in the following way. For each $y \in S^{(i+\vartheta)}$, writing M_y for the matrix guaranteed to exist by Lemma 4.9, we define the column:

$$\begin{bmatrix} f_0^{(i+\vartheta)}(y) \\ \vdots \\ f_{2^\vartheta-1}^{(i+\vartheta)}(y) \end{bmatrix} \coloneqq \begin{bmatrix} & & \\ & M_y & \\ & & \end{bmatrix} \cdot \begin{bmatrix} f^{(i)}(x_0) \\ \vdots \\ f^{(i)}(x_{2^\vartheta-1}) \end{bmatrix}. \tag{41}$$

We note that the resulting $2^\vartheta \times 2^{\ell+\mathcal{R}-i-\vartheta}$ matrix $\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$—i.e., that whose columns are given by the respective left-hand sides of (41), for $y \in S^{(i+\vartheta)}$ varying—satisfies, for each $(r'_i, \ldots, r'_{i+\vartheta-1}) \in L^\vartheta$,

$$\mathsf{fold}\left(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\right) = \begin{bmatrix} \bigotimes_{j=i}^{i+\vartheta-1}(1-r'_j, r'_j) \end{bmatrix} \cdot \begin{bmatrix} - & f_0^{(i+\vartheta)} & - \\ & \vdots & \\ - & f_{2^\vartheta-1}^{(i+\vartheta)} & - \end{bmatrix}. \tag{42}$$

Indeed, this is essentially the content of Lemma 4.9, which we apply here jointly to all elements $y \in S^{(i+\vartheta)}$.

We claim that the interleaved word $\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$ constructed in this way is far from the interleaved code ${C^{(i+\vartheta)}}^{2^\vartheta}$.

Lemma 4.21. *Under our hypothesis* $d^{(i)}\left(f^{(i)}, C^{(i)}\right) \geq \frac{d_{i+\vartheta}}{2}$*, we have* $d^{2^\vartheta}\left(\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}, {C^{(i+\vartheta)}}^{2^\vartheta}\right) \geq \frac{d_{i+\vartheta}}{2}$*.*

Proof. We fix an arbitrary interleaved codeword $\left(g_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1} \in {C^{(i+\vartheta)}}^{2^\vartheta}$. We define a "lift" $g^{(i)} \in C^{(i)}$ of $\left(g_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$ in the following way. Writing, for each $j \in \{0, \ldots, 2^\vartheta - 1\}$, $P_j^{(i+\vartheta)}(X) \coloneqq \sum_{k=0}^{2^{\ell-i-\vartheta}-1} a_{j,k} \cdot X_k^{(i+\vartheta)}(X)$ for the polynomial—expressed in coordinates with respect to the $i+\vartheta^{\text{th}}$-order novel polynomial basis—for which $g_j^{(i+\vartheta)} = \mathsf{Enc}(P_j^{(i+\vartheta)})$ holds, we define

$$P^{(i)}(X) \coloneqq \sum_{j=0}^{2^\vartheta-1} \sum_{k=0}^{2^{\ell-i-\vartheta}-1} a_{j,k} \cdot X_{k\cdot 2^\vartheta+j}^{(i)};$$

that is, $P^{(i)}$'s list of i^{th}-order coefficients is precisely the 2^ϑ-fold interleaving of the polynomials $P_0^{(i+\vartheta)}(X), \ldots, P_{2^\vartheta-1}^{(i+\vartheta)}(X)$'s respective lists of $i+\vartheta^{\text{th}}$-order coefficients. Finally, we define $g^{(i)} \coloneqq \mathsf{Enc}(P^{(i)})$.

We argue that the codeword $g^{(i)} \in C^{(i)}$ constructed in this way stands in relation to $\left(g_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$ just as $f^{(i)}$ does to $\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$ (i.e., it also satisfies a matrix identity analogous to (41) for each $y \in S^{(i+\vartheta)}$). To prove this, we fix an arbitrary element $y \in S^{(i+\vartheta)}$; we moreover fix a row-index $j \in \{0, \ldots, 2^\vartheta - 1\}$. We write $(j_0, \ldots, j_{\vartheta-1})$ for the bits of j (i.e., so that $j = \sum_{k=0}^{\vartheta-1} 2^k \cdot j_k$ holds). We first note that the functions $g_j^{(i+\vartheta)}$ and $\mathsf{fold}\left(g^{(i)}, j_0, \ldots, j_{\vartheta-1}\right)$ agree identically over the domain $S^{(i+\vartheta)}$. Indeed, this is a direct consequence of Lemma 4.13 and of the construction of $g^{(i)}$ ($g_j^{(i+\vartheta)}(y)$'s underlying polynomial's coefficients are the j^{th} refinement of $g^{(i)}$'s underlying polynomial's). On the other hand, applying Lemma 4.9 to $y \in S^{(i+\vartheta)}$ and $g^{(i)}$, with the folding tuple $(j_0, \ldots, j_{\vartheta-1})$, we see that the dot product between M_y's j^{th} row and $\left(g^{(i)}(x_0), \ldots, g^{(i)}(x_{2^\vartheta-1})\right)$ is exactly $\mathsf{fold}\left(g^{(i)}, j_0, \ldots, j_{\vartheta-1}\right)(y) = g_j^{(i+\vartheta)}(y)$, where the latter equality was just argued.

Since $g^{(i)} \in C^{(i)}$ is a codeword, our hypothesis $d^{(i)}\big(f^{(i)}, C^{(i)}\big) \geq \frac{d_{i+\vartheta}}{2}$ applies to it. That hypothesis entails precisely that, for at least $\frac{d_{i+\vartheta}}{2}$ elements $y \in S^{(i+\vartheta)}$, the restrictions $f^{(i)}|_{\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})}$ and $g^{(i)}|_{\left(q^{(i+\vartheta-1)} \circ \cdots \circ q^{(i)}\right)^{-1}(\{y\})}$ are *not* identically equal. For each such $y \in S^{(i+\vartheta)}$, since M_y is nonsingular (and since both $f^{(i)}$ and $g^{(i)}$ satisfy (41)), we conclude that the columns $\left(f_j^{(i+\vartheta)}(y)\right)_{j=0}^{2^\vartheta-1}$ and $\left(g_j^{(i+\vartheta)}(y)\right)_{j=0}^{2^\vartheta-1}$ are in turn unequal. Since $\left(g_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$ was arbitrary, we conclude that $d^{2^\vartheta}\left(\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}, C^{(i+\vartheta)^{2^\vartheta}}\right) \geq \frac{d_{i+\vartheta}}{2}$. □

Applying Lemma 4.21, we conclude directly that the contraposition of Theorem 2.3 is fulfilled with respect to the code $C^{(i+\vartheta)} \subset L^{2^{\ell+\mathcal{R}-i-\vartheta}}$, the proximity parameter $e := \left\lfloor \frac{d_{i+\vartheta}-1}{2} \right\rfloor$, and the interleaved word $\left(f_j^{(i+\vartheta)}\right)_{j=0}^{2^\vartheta-1}$. That theorem's contraposition immediately implies that the set $E_i \subset L^\vartheta$ consisting of those tuples $(r'_i, \ldots, r'_{i+\vartheta-1}) \in L^\vartheta$ for which $d\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), C^{(i+\vartheta)}\big) < \frac{d_{i+\vartheta}}{2}$ holds—and here, we use (42)—is of mass at most $\mu(E_i) \leq \vartheta \cdot \frac{2^{\ell+\mathcal{R}-i-\vartheta}}{|L|} = \vartheta \cdot \frac{\left|S^{(i+\vartheta)}\right|}{|L|}$, as required. This completes the proof of the proposition. □

Proposition 4.22. *The probability that any among the bad events $E_0, E_\vartheta, \ldots, E_{\ell-\vartheta}$ occurs is at most $\frac{2^{\ell+\mathcal{R}}}{|L|}$.*

Proof. Applying Proposition 4.20, we upper-bound the quantity of interest as:

$$\frac{\vartheta}{|L|} \cdot (|S_\vartheta| + \cdots + |S_\ell|) = \frac{\vartheta}{|L|} \cdot \left(2^{\ell+\mathcal{R}-\vartheta} + \cdots + 2^{\mathcal{R}}\right) \leq \frac{\vartheta}{|L|} \cdot \frac{2^\vartheta}{2^\vartheta - 1} \cdot 2^{\ell+\mathcal{R}-\vartheta} \leq \frac{2^{\ell+\mathcal{R}}}{|L|},$$

which completes the proof. In the last two steps, we use the geometric series formula and the inequality $\frac{\vartheta}{2^\vartheta-1} \leq 1$ (which holds for each $\vartheta \geq 1$), respectively. □

In light of Proposition 4.22, we freely assume that none of the events $E_0, E_\vartheta, \ldots, E_{\ell-\vartheta}$ occurs. Under this assumption, we finally turn to the following key proposition.

Proposition 4.23. *If any of $\mathcal{A}$'s oracles is not compliant, then $\mathcal{V}$ accepts with at most negligible probability.*

Proof. We suppose that at least one of $\mathcal{A}$'s oracles is not compliant; we write $i^* \in \{0, \vartheta, \ldots, \ell - \vartheta\}$ for the index of $\mathcal{A}$'s highest-indexed noncompliant oracle.

Lemma 4.24. *For $i^* \in \{0, \vartheta, \ldots, \ell - \vartheta\}$ as above, we have $d\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), \overline{f}^{(i^*+\vartheta)}\big) \geq \frac{d_{i^*+\vartheta}}{2}$.*

Proof. Assuming first that $d^{(i^*)}\big(f^{(i^*)}, C^{(i^*)}\big) < \frac{d_{i^*+\vartheta}}{2}$, we write $\overline{f}^{(i^*)} \in C^{(i^*)}$ for the codeword for which $\left|\Delta^{(i^*)}\big(f^{(i^*)}, \overline{f}^{(i^*)}\big)\right| < \frac{d_{i^*+\vartheta}}{2}$. We note that $d\big(f^{(i^*)}, \overline{f}^{(i^*)}\big) < \frac{d_{i^*}}{2}$ *a fortiori* holds; by Definition 4.17 and our choice of i^*, we thus must have in turn $\overline{f}^{(i^*+\vartheta)} \neq \mathsf{fold}\big(\overline{f}^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big)$. On the other hand, by Lemma 4.18, $\left|\Delta^{(i^*)}\big(f^{(i^*)}, \overline{f}^{(i^*)}\big)\right| < \frac{d_{i^*+\vartheta}}{2}$ implies that $d\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), \mathsf{fold}\big(\overline{f}^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big)\big) < \frac{d_{i^*+\vartheta}}{2}$. Finally, by the reverse triangle inequality, $d\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), \overline{f}^{(i^*+\vartheta)}\big)$ is at least:

$$d\Big(\overline{f}^{(i^*+\vartheta)}, \mathsf{fold}\Big(\overline{f}^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\Big)\Big) - d\Big(\mathsf{fold}\Big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\Big), \mathsf{fold}\Big(\overline{f}^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\Big)\Big).$$

Since $\overline{f}^{(i^*+\vartheta)}$ and $\mathsf{fold}\big(\overline{f}^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big)$ are unequal codewords in $C^{(i^*+\vartheta)}$, this quantity in turn is greater than or equal to $d_{i^*+\vartheta} - \frac{d_{i^*+\vartheta}}{2} \geq \frac{d_{i^*+\vartheta}}{2}$, and the proof of the first case is complete.

In the case $d^{(i^*)}\big(f^{(i^*)}, C^{(i^*)}\big) \geq \frac{d_{i^*+\vartheta}}{2}$, our assumption whereby E_{i^*} didn't occur implies, by definition, that $d\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), C^{(i^*+\vartheta)}\big) \geq \frac{d_{i^*+\vartheta}}{2}$. Since $\overline{f}^{(i^*+\vartheta)} \in C^{(i^*+\vartheta)}$ is a codeword, $d\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), \overline{f}^{(i^*+\vartheta)}\big) \geq \frac{d_{i^*+\vartheta}}{2}$ in particular holds, and the proof is again complete. □

Lemma 4.25. *Whenever its suffix $(v_{i^*+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1}) \in \Delta\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i+\vartheta-1}\big), \overline{f}^{(i^*+\vartheta)}\big)$, $\mathcal{V}$ rejects.*

Proof. We fix an iteration of the query phase's outer loop for which the lemma's hypothesis holds. We fix an arbitrary index $i \in \{i^*, i^* + \vartheta, \ldots, \ell - \vartheta\}$. If $\mathcal{V}$ rejects before finishing the inner loop 3's i^{th} iteration, then there's nothing to prove. We argue that, conditioned on $\mathcal{V}$ reaching the end of its i^{th} iteration, we have the inductive conclusion $c_{i+\vartheta} \neq \overline{f}^{(i+\vartheta)}(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$ as of the end of that iteration.

In the base case $i = i^*$, $\mathcal{V}$ assigns $c_{i^*+\vartheta} \coloneqq \mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big)(v_{i^*+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$ inline on line 6. On the other hand, the hypothesis of the lemma is precisely $\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i+\vartheta-1}\big)(v_{i^*+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1}) \neq \overline{f}^{(i^*+\vartheta)}(v_{i^*+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$; we conclude immediately that $c_{i^*+\vartheta} \neq \overline{f}^{(i^*+\vartheta)}(v_{i^*+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$ will hold as of the end of the $i^{*\text{th}}$ iteration, as desired.

We fix an index $i \in \{i^*+\vartheta, \ldots, \ell-\vartheta\}$. As of the beginning of the i^{th} iteration, by induction, we have the hypothesis $c_i \neq \overline{f}^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1})$. If $\overline{f}^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1}) = f^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1})$ moreover holds, then we see immediately that $\mathcal{V}$ will reject on line 5; indeed, in this case $c_i \neq \overline{f}^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1}) = f^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1})$ will hold. We conclude that, conditioned on $\mathcal{V}$ reaching the end of its i^{th} iteration, we necessarily have $\overline{f}^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1}) \neq f^{(i)}(v_i, \ldots, v_{\ell+\mathcal{R}-1})$, or in other words $(v_i, \ldots, v_{\ell+\mathcal{R}-1}) \in \Delta\big(f^{(i)}, \overline{f}^{(i)}\big)$. This guarantee implies *a fortiori* that $(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1}) \in \Delta^{(i)}\big(f^{(i)}, \overline{f}^{(i)}\big)$, by definition of this latter set. Using our assumption whereby the event E_i *didn't* occur, we conclude in turn that $(v_{i+\vartheta}, \ldots, v_{\ell-1}) \in \Delta\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)\big)$. Since $\overline{f}^{(i+\vartheta)} = \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)$ (a consequence of the maximality of i^*), this latter set itself equals $\Delta\big(\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big), \overline{f}^{(i+\vartheta)}\big)$. We conclude that $\mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1}) \neq \overline{f}^{(i+\vartheta)}(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$, so that, after its assignment on line 6, $\mathcal{V}$ will obtain $c_{i+\vartheta} \neq \overline{f}^{(i+\vartheta)}(v_{i+\vartheta}, \ldots, v_{\ell+\mathcal{R}-1})$, thereby preserving the inductive hypothesis.

Carrying through the induction, we see finally that either $\mathcal{V}$ will abort before finishing its inner loop 3, or else it will have $c_\ell \neq \overline{f}^{(\ell)}(v_\ell, \ldots, v_{\ell+\mathcal{R}-1})$ as of its final check 7. Since $c = \overline{f}^{(\ell)}(v_\ell, \ldots, v_{\ell+\mathcal{R}-1})$ holds identically for each $v \in \mathcal{B}_\mathcal{R}$ (by definition of this latter oracle), we see that $\mathcal{V}$ will reject its check $c_\ell \stackrel{?}{=} c$. □

We return to the proposition. Lemma 4.24 guarantees (i.e., assuming E_{i^*} doesn't occur) that $c_{i^*+\vartheta} \in \Delta\big(\mathsf{fold}\big(f^{(i^*)}, r'_{i^*}, \ldots, r'_{i^*+\vartheta-1}\big), \overline{f}^{(i^*+\vartheta)}\big)$ will hold with probability at least $\frac{1}{|S^{(i^*+\vartheta)}|} \cdot \frac{d_{i^*+\vartheta}}{2} \geq \frac{1}{2} - \frac{1}{2 \cdot 2^\mathcal{R}}$ in each of the verifier's query iterations. By Lemma 4.25, the verifier will reject in each such iteration (i.e., assuming none of the events $E_{i^*+\vartheta}, \ldots, E_{\ell-\vartheta}$ occurs). We see that $\mathcal{V}$ will accept with probability at most $\left(\frac{1}{2} + \frac{1}{2 \cdot 2^\mathcal{R}}\right)^\gamma$, which is negligible (we recall that $\mathcal{R}$ is a positive constant). This completes the proof of the proposition. □

In light of Proposition 4.23, we assume that all of $\mathcal{A}$'s oracles are compliant. Under this assumption, we note first that $d\big(f^{(0)}, C^{(0)}\big) < \frac{d_0}{2}$ will hold. We see that Algorithm 1 will terminate successfully in step 2 above. We write $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$ for the polynomial output by $\mathcal{E}$ in that step.

We now argue that $c = t(r'_0, \ldots, r'_{\ell-1})$ will hold. To this end, we apply Definition 4.17 repeatedly. In the base case $i = 0$, we note that $\overline{f}^{(0)}$ will be the encoding of $P^{(0)}(X) = \sum_{w \in \mathcal{B}_\ell} t(w) \cdot X^{(0)}_{\{w\}}(X)$, precisely by $\mathcal{E}$'s construction of $(t(w))_{w \in \mathcal{B}_\ell}$. On the other hand, for each $i \in \{0, \vartheta, \ldots, \ell - \vartheta\}$, writing $P^{(i)}(X) \in L[X]^{\prec 2^{\ell-i}}$ for the polynomial for which $\mathsf{Enc}(P^{(i)}) = \overline{f}^{(i)}$ holds, and using our assumption $\overline{f}^{(i+\vartheta)} = \mathsf{fold}\big(\overline{f}^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)$, we conclude that $\overline{f}^{(i+\vartheta)}$ will be exactly the encoding of that polynomial $P^{(i+\vartheta)}(X) \in L[X]^{\prec 2^{\ell-i-\vartheta}}$ which results from repeatedly applying to $P^{(i)}(X)$ the conclusion of Lemma 4.13 (with the folding challenges $r'_i, \ldots, r'_{i+\vartheta-1}$). Carrying out the induction, we see that $\overline{f}^{(\ell)}$ will itself be identically equal to $\sum_{w \in \mathcal{B}_\ell} \widetilde{\mathsf{eq}}\big(w_0, \ldots, w_{\ell-1}, r'_0, \ldots, r'_{\ell-1}\big) \cdot t(w) = t(r'_0, \ldots, r'_{\ell-1})$, so that $c = t(r'_0, \ldots, r'_{\ell-1})$ indeed will hold.

We write $(r_0, \ldots, r_{\ell-1}) \in L^\ell$ for the evaluation point output by $\mathcal{V}$ and $s \in L$ for $\mathcal{A}$'s response. To finish the proof, we argue that the probability with which $s \neq t(r_0, \ldots, r_{\ell-1})$ and $\mathcal{V}$ accepts is negligible. We assume that $s \neq t(r_0, \ldots, r_{\ell-1})$.

As in Construction 4.11, we write $h(X_0, \ldots, X_{\ell-1}) \coloneqq \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, X_0, \ldots, X_{\ell-1}) \cdot t(X_0, \ldots, X_{\ell-1})$ (here, $t(X_0, \ldots, X_{\ell-1})$ refers to what $\mathcal{E}$ extracted). Since $t(r_0, \ldots, r_{\ell-1}) = \sum_{w \in \mathcal{B}_\ell} h(w)$, our assumption $s \neq t(r_0, \ldots, r_{\ell-1})$ amounts to the condition $s \neq \sum_{w \in \mathcal{B}_\ell} h(w)$. The soundness analysis of the sumcheck (we refer to Thaler [Tha22, § 4.1]) states that, under this very assumption, the probability that the verifier accepts its checks $s_i \stackrel{?}{=} h_i(0) + h_i(1)$ *and* $s_\ell = h(r'_0, \ldots, r'_{\ell-1})$ holds is at most $\frac{2 \cdot \ell}{|L|}$ over $\mathcal{V}$'s choice of its folding challenges $(r'_0, \ldots, r'_{\ell-1})$. We thus assume that $s_\ell \neq h(r'_0, \ldots, r'_{\ell-\kappa-1}) = \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}) \cdot t(r'_0, \ldots, r'_{\ell-1})$.

Our conclusion whereby $c = t(r'_0, \ldots, r'_{\ell-1})$, established above, thus implies that $\mathcal{V}$ will reject its check $s_\ell \stackrel{?}{=} \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}) \cdot c$. This completes the proof of the theorem. □

Remark 4.26. In our proof of Theorem 4.16 above, our emulator $\mathcal{E}$ runs the Berlekamp–Welch decoder on the adversary-supplied word $f : S^{(0)} \to L$ (see its step 2). Most analyses of that algorithm (see e.g. [Gur06, Rem. 4]) assume inputs guaranteed to reside within the unique decoding radius, and implicitly leave undefined the algorithm's behavior on arbitrary words. The behavior of Algorithm 1 on a general word $f : S^{(0)} \to L$ is far from obvious. As far as our proof of Theorem 4.16 is concerned, we need merely the guarantee whereby, regardless of its input, Algorithm 1—and hence also $\mathcal{E}$—runs in strict polynomial time. (That guarantee follows straightforwardly from Algorithm 1's description.) Indeed, if $\mathcal{A}$ submits a word f outside of the unique decoding radius, then—as our Propositions 4.22 and 4.23 above show—$\mathcal{V}$ will reject with overwhelming probability in any case, so that $\mathcal{E}$'s output ultimately doesn't matter. As it happens, it's possible to show that, on input f outside of the unique decoding radius, Algorithm 1 will either return $\perp$ on line 5 or else will return a polynomial $P(X)$ of degree greater than or equal to 2^ℓ (and both of these outcomes can actually happen). We conclude in particular that $\mathcal{E}$'s test $\deg(P(X)) \stackrel{?}{<} 2^\ell$ above is necessary.

4.4 Efficiency

We discuss the efficiency of Construction 4.11. We measure L-operations throughout. Though the choices $\deg(L \,/\, \mathbb{F}_2) = \omega(\log \lambda)$ and $\gamma = \omega(\log \lambda)$ suffice to make that construction's soundness error negligible, we in fact set $\deg(L \,/\, \mathbb{F}_2) = \lambda$ and $\gamma = \lambda$. These latter choices make Construction 4.11 *exponentially secure*, and make its efficiency easier to analyze. As usual, we understand the positive integers $\mathcal{R}$ and ϑ as constants.

Prover cost. In its commitment phase 2, our prover must use Lin, Chung and Han's *additive NTT* [LCH14] to encode its length-2^ℓ vector $(t(w))_{w \in \mathcal{B}_\ell}$ onto $S^{(0)}$ (see also Algorithm 2 above). For this task, $\ell \cdot 2^{\ell+\mathcal{R}-1}$ L-multiplications and $\ell \cdot 2^{\ell+\mathcal{R}}$ L-additions suffice (see also Subsection 2.3).

To prepare its sumcheck 2, the prover must tensor-expand $(\widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell-1}))_{w \in \mathcal{B}_\ell}$; this task takes 2^ℓ L-additions and 2^ℓ L-multiplications (recall Subsection 2.1). Our prover can carry out that sumcheck itself also in $O(2^\ell)$ time (we refer to Thaler [Tha22, § 4.1]). Our prover is thus linear-time.

Verifier cost. To carry out the sumcheck 2, Construction 4.11's verifier must expend just $O(\ell)$ L-operations. It can compute $\widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1})$ in step 3 in again $O(\ell)$ L-operations. During its querying phase 4, the verifier must finally, for $\gamma = \lambda$ repetitions, make $2^\vartheta \cdot \frac{\ell}{\vartheta} = O(\ell)$ queries to the IOP oracle and perform $O\big(2^\vartheta \cdot \frac{\ell}{\vartheta}\big) = O(\ell)$ L-operations. Its total cost during that phase is thus $O(\lambda \cdot \ell)$ L-operations.

The BCS transform. In practice, we must use Ben-Sasson, Chiesa and Spooner's transformation [BCS16] to turn Construction 4.11 into an interactive protocol in the random oracle model. The resulting compiled protocol imposes further costs on both parties, as we now explain. First, its prover must moreover Merkle-hash both $f^{(0)}$ during its commitment phase and the positive-indexed oracles $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ during its evaluation phase; these commitments represent total work on the order of $O\big(2^{\ell+\mathcal{R}}\big) = O\big(2^\ell\big)$ hash invocations.

During the querying phase, both parties must handle Merkle paths. During each query repetition, the total length of all Merkle paths sent (measured in digests) is on the order of $O((\ell + \mathcal{R})^2) = O(\ell^2)$. Since there are $\gamma = \lambda$ total repetitions, the total cost for both parties during the querying phase is thus $O(\lambda \cdot \ell^2)$ hash operations.

5 Unrolled Small-Field IOPCS

In this section, we describe a "one-shot" small-field IOPCS construction. This construction inlines the large-field IOPCS of Section 4 into the ring-switching reduction of Section 3. We moreover streamline the resulting combination, by applying a few optimizations. That is, we unify Construction 4.11's sumcheck with that already required within Construction 3.1.

5.1 Combined Small-Field Protocol

We present our full combined protocol below. Our protocol directly instantiates the generic small-field template of Definition 2.9.

CONSTRUCTION 5.1 (Combined Small-Field IOPCS)**.**
We define $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ as follows.

1. $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell, \iota)$. On input 1^λ, ℓ, and ι, choose a constant, positive rate parameter $\mathcal{R} \in \mathbb{N}$ and a tower height $\tau \geq \log(\omega(\log \lambda))$ for which $\tau \geq \iota$ and $2^\tau \geq \ell - \tau + \iota + \mathcal{R}$. Write $\kappa \coloneqq \tau - \iota$ and $\ell' \coloneqq \ell - \kappa$. Initialize the vector oracle $\mathcal{F}_{\mathsf{Vec}}^{\mathcal{T}_\tau}$. Fix a folding factor $\vartheta \mid \ell'$ and a repetition parameter $\gamma = \omega(\log(\lambda))$. Write $(X_0(X), \ldots, X_{2^{\ell'}-1}(X))$ for the novel $\mathcal{T}_\tau$-basis of $\mathcal{T}_\tau[X]^{\prec 2^{\ell'}}$, and fix the domains $S^{(0)}, \ldots, S^{(\ell')}$ and the polynomials $q^{(0)}, \ldots, q^{(\ell'-1)}$ as in Subsection 4.1. Write $C^{(0)} \subset \mathcal{T}_\tau{}^{2^{\ell'+\mathcal{R}}}$ for the Reed–Solomon code $\mathsf{RS}_{\mathcal{T}_\tau, S^{(0)}}[2^{\ell'+\mathcal{R}}, 2^{\ell'}]$.

2. $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. On input $t(X_0, \ldots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$, construct as in Definition 2.1 the packed polynomial $t'(X_0, \ldots, X_{\ell'-1}) \in \mathcal{T}_\tau[X_0, \ldots, X_{\ell'-1}]^{\preceq 1}$. Write $P(X) \coloneqq \sum_{v \in \mathcal{B}_{\ell'}} t'(v) \cdot X_{\{v\}}(X)$ for its univariate flattening. Using Algorithm 2, compute the Reed–Solomon codeword $f : S^{(0)} \to \mathcal{T}_\tau$ defined by $f : x \mapsto P(x)$. Submit $(\texttt{submit}, \ell' + \mathcal{R}, f)$ to the vector oracle $\mathcal{F}_{\mathsf{Vec}}^{\mathcal{T}_\tau}$. Upon receiving $(\texttt{receipt}, \ell' + \mathcal{R}, [f])$ from the oracle, output the commitment $[f]$.

We define $(\mathcal{P}, \mathcal{V})$ as the following IOP, in which both parties have the common input $[f]$, $s \in \mathcal{T}_\tau$, and $(r_0, \ldots, r_{\ell-1}) \in \mathcal{T}_\tau^\ell$, and $\mathcal{P}$ has the further input $t(X_0, \ldots, X_{\ell-1}) \in \mathcal{T}_\iota[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.

1. $\mathcal{P}$ computes $\hat{s} \coloneqq \varphi_1(t')(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}))$ and sends $\mathcal{V}$ the A-element $\hat{s}$.

2. $\mathcal{V}$ decomposes $\hat{s} \eqqcolon \sum_{v \in \mathcal{B}_\kappa} \hat{s}_v \otimes \beta_v$. $\mathcal{V}$ requires $s \stackrel{?}{=} \sum_{v \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(v_0, \ldots, v_{\kappa-1}, r_0, \ldots, r_{\kappa-1}) \cdot \hat{s}_v$.

3. $\mathcal{V}$ samples batching scalars $(r''_0, \ldots, r''_{\kappa-1}) \leftarrow \mathcal{T}_\tau^\kappa$ and sends them to $\mathcal{P}$.

4. For each $w \in \mathcal{B}_{\ell'}$, $\mathcal{P}$ decomposes $\widetilde{\mathsf{eq}}(r_\kappa, \ldots, r_{\ell-1}, w_0, \ldots, w_{\ell'-1}) \eqqcolon \sum_{u \in \mathcal{B}_\kappa} A_{w,u} \cdot \beta_u$. $\mathcal{P}$ defines the function $A : w \mapsto \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot A_{w,u}$ on $\mathcal{B}_{\ell'}$ and writes $A(X_0, \ldots, X_{\ell'-1})$ for its multilinear extension. $\mathcal{P}$ defines $h(X_0, \ldots, X_{\ell'-1}) \coloneqq A(X_0, \ldots, X_{\ell'-1}) \cdot t'(X_0, \ldots, X_{\ell'-1})$.

5. $\mathcal{V}$ decomposes $\hat{s} \eqqcolon \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes \hat{s}_u$, and sets $s_0 \coloneqq \sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot \hat{s}_u$.

6. $\mathcal{P}$ and $\mathcal{V}$ both abbreviate $f^{(0)} \coloneqq f$, and execute the following loop:
   - **for** $i \in \{0, \ldots, \ell' - 1\}$ **do**
     - $\mathcal{P}$ sends $\mathcal{V}$ the polynomial $h_i(X) \coloneqq \sum_{w \in \mathcal{B}_{\ell'-i-1}} h\big(r'_0, \ldots, r'_{i-1}, X, w_0, \ldots, w_{\ell'-i-2}\big)$.
     - $\mathcal{V}$ requires $s_i \stackrel{?}{=} h_i(0) + h_i(1)$. $\mathcal{V}$ samples $r'_i \leftarrow \mathcal{T}_\tau$, sets $s_{i+1} \coloneqq h_i(r'_i)$, and sends $\mathcal{P}$ r'_i.
     - $\mathcal{P}$ defines $f^{(i+1)} : S^{(i+1)} \to \mathcal{T}_\tau$ as the function $\mathsf{fold}\big(f^{(i)}, r'_i\big)$ of Definition 4.6.
     - **if** $i + 1 = \ell'$ **then** $\mathcal{P}$ sends $c \coloneqq f^{(\ell')}(0, \ldots, 0)$ to $\mathcal{V}$.
     - **else if** $\vartheta \mid i + 1$ **then** $\mathcal{P}$ submits $(\texttt{submit}, \ell' + \mathcal{R} - i - 1, f^{(i+1)})$ to the oracle.

7. $\mathcal{V}$ sets $e \coloneqq \widetilde{\mathsf{eq}}\big(\varphi_0(r_\kappa), \ldots, \varphi_0(r_{\ell-1}), \varphi_1(r'_0), \ldots, \varphi_1(r'_{\ell'-1})\big)$ and decomposes $e \eqqcolon \sum_{u \in \mathcal{B}_\kappa} \beta_u \otimes e_u$.

8. $\mathcal{V}$ requires $s_{\ell'} \stackrel{?}{=} \big(\sum_{u \in \mathcal{B}_\kappa} \widetilde{\mathsf{eq}}(u_0, \ldots, u_{\kappa-1}, r''_0, \ldots, r''_{\kappa-1}) \cdot e_u\big) \cdot c$.

9. $\mathcal{V}$ executes the following querying procedure:
   - **for** γ repetitions **do**
     - $\mathcal{V}$ samples $v \leftarrow \mathcal{B}_{\ell'+\mathcal{R}}$ randomly.
     - **for** $i \in \{0, \vartheta, \ldots, \ell' - \vartheta\}$ (i.e., taking ϑ-sized steps) **do**
       - for each $u \in \mathcal{B}_\vartheta$, $\mathcal{V}$ sends $\big(\texttt{query}, [f^{(i)}], (u_0, \ldots, u_{\vartheta-1}, v_{i+\vartheta}, \ldots, v_{\ell'+\mathcal{R}-1})\big)$ to the oracle.
       - **if** $i > 0$ **then** $\mathcal{V}$ requires $c_i \stackrel{?}{=} f^{(i)}(v_i, \ldots, v_{\ell'+\mathcal{R}-1})$.
       - $\mathcal{V}$ defines $c_{i+\vartheta} \coloneqq \mathsf{fold}\big(f^{(i)}, r'_i, \ldots, r'_{i+\vartheta-1}\big)(v_{i+\vartheta}, \ldots, v_{\ell'+\mathcal{R}-1})$.
     - $\mathcal{V}$ requires $c_{\ell'} \stackrel{?}{=} c$.

The completeness and security of Construction 5.1 follow directly from the techniques developed in Sections 3 and 4 above.

5.2 Efficiency

For matters of asymptotic efficiency, we refer to Subsections 3.2 and 4.4 above. Here, we discuss the concrete efficiency of Construction 5.1.

In our concrete proof size analyses below, we incorporate various optimizations. For example, for each oracle $i \in \{0, \vartheta, \ldots, \ell' - \vartheta\}$, we opt to send the entire j^{th} layer of the Merkle tree—as opposed to just its root—for some appropriately chosen constant $j \geq 0$. Increasing j exponentially increases the fixed size of each commitment, but also causes each among the γ subsequently sent paths to become shorter. The optimal truncation height turns out to be $j := \lceil \log_2(\gamma) \rceil$. Each path sent, of course, is of size $\ell' + \mathcal{R} - i - \vartheta - j$.

As soon as $i \in \{0, \vartheta, \ldots, \ell' - \vartheta\}$ becomes so large that $j > \ell' + \mathcal{R} - i - \vartheta$ holds, this convention becomes nonsensical; at this point, we instruct our prover rather to terminate FRI early and send its entire message to the verifier in the clear. (Of course, this measure moreover allows us to drop our requirement $\vartheta \mid \ell'$, which we instated only for notational convenience.) Interestingly, in certain parameter regimes, the parties stand to benefit by terminating FRI even earlier (i.e., even when $j \leq \ell' + \mathcal{R} - i - \vartheta$ still holds). Since this phenomenon doesn't appear in the particular problem instances we benchmark below, we have refrained from treating it more explicitly (say, by developing a criterion designed to predict when exactly FRI should be terminated). We leave for future work the establishment of closed-form formulae which serve to predict the optimal oracle-skipping parameter ϑ and the optimal early-termination threshold, given as input only ℓ', $\mathcal{R}$, γ and j (as well as global parameters like τ, ι, and the hash digest width).

Concrete soundness. We record proof sizes for both this work and [DP25, Cons. 3.11]. In order to appropriately select the query repetition parameter γ, we must examine the concrete security of our protocol (we refer to [DP25, § 3.5] for an analogous analysis). It follows essentially from the proofs of Theorems 3.5 and 4.16 that Construction 5.1's concrete soundness error is bounded from above by

$$\frac{\kappa + 2 \cdot \ell'}{|\mathcal{T}_\tau|} + \frac{2^{\ell' + \mathcal{R}}}{|\mathcal{T}_\tau|} + \left(\frac{1}{2} + \frac{1}{2 \cdot 2^{\mathcal{R}}} \right)^{\gamma}; \tag{43}$$

above, the first summand comes from ring-switching and the sumcheck, whereas the latter two reflect Propositions 4.22 and 4.23, respectively. For each desired *concrete security* level Ξ, we thus set γ minimally so that (43) becomes bounded from above by Ξ. (Clearly, this is possible only when τ is sufficiently large that $\Xi > \frac{\kappa+2\cdot\ell'}{|\mathcal{T}_\tau|} + \frac{2^{\ell'+\mathcal{R}}}{|\mathcal{T}_\tau|}$ holds.) We say in this case that Construction 5.1 attains $-\log_2(\Xi)$ *bits of security.*

Proof sizes. Our proof sizes appear in Table 1 below.

Total Data Size	Num. Variables ℓ	Coefficient Size ι	[DP25, Cons. 3.11]	Construction 5.1
32 MiB (2^{28} bits)	22	6	0.768 MiB	0.227 MiB
	25	3	1.018 MiB	0.227 MiB
	28	0	2.863 MiB	0.229 MiB
512 MiB (2^{32} bits)	26	6	4.546 MiB	0.334 MiB
	29	3	5.697 MiB	0.334 MiB
	32	0	11.314 MiB	0.336 MiB
8 GiB (2^{36} bits)	30	6	11.343 MiB	0.465 MiB
	33	3	22.586 MiB	0.465 MiB
	36	0	61.079 MiB	0.467 MiB

Table 1: Proof sizes, including oracle-skipping, Merkle caps, and early FRI termination.

In our proof size measurements above, we use a 128-bit field, and attain 96 bits of provable security. We set $\mathcal{R} := 2$ throughout, so that our code is of rate $\rho = \frac{1}{4}$. In Construction 5.1, we use between $\gamma = 142$ and $\gamma = 144$ queries, as the case may be. We use the Merkle tree truncation height $j := 8$. We fix the folding factor $\vartheta := 4$, which happens to yield the smallest proofs throughout. The previous work [DP25, Cons. 3.11] requires more queries—rather between $\gamma = 231$ and $\gamma = 232$, for the sizes we benchmark below—as [DP25, Rem. 3.18] explains.

We see that our Construction 5.1 beats [DP25, Cons. 3.11] by as much as a hundredfold.

Concrete performance. We concretely benchmark this work's Construction 5.1 above, as well as [DP25, Cons. 3.11] and the univariate-FRI-based scheme *Plonky3*. Our benchmarks of the first two schemes use *Binius*, an open-source implementation of both [DP25] and this work.

In our benchmarks below, we again use a 128-bit field and attain 96 bits of provable security. We work exclusively in the unique-decoding regime. We note that both [DP25, Cons. 3.11] and this work are proven secure solely in that regime (as of yet). As for Plonky3, we note that it's impossible to obtain 96 bits of provable security in the *list-decoding regime* over a field of merely 128 bits. Indeed, the best-available proximity gap in that regime—namely, [Ben+23, Thm. 5.1]—has a false witness probability [Ben+23, (5.3)] which grows quadratically in the problem size. We see that each reasonably-large instance stands to overwhelm that result's 128-bit denominator (yielding a vacuous bound). Our benchmarks below thus reflect the best-possible proof size attainable in Plonky3, conditioned on the 96-bit security level and the use of a 128-bit field.

In [DP25] and this work, we work over the 128-bit tower field $\mathcal{T}_7$. In Plonky3, we use the quartic extension $\mathbb{F}_p[X]/\left(X^4 - 11\right)$ of the *Baby Bear* prime field $\mathbb{F}_p$, where $p := 2^{31} - 2^{27} + 1$. Throughout, we use the code rate $\rho = \frac{1}{4}$. We benchmark [DP25] and Construction 5.1 on ℓ-variate multilinear polynomials, for ℓ equal to 20, 24, and 28. In each case, we consider polynomials over $\mathcal{T}_\iota$, for ι equal in turn to 0, 3 and 5 (i.e., with coefficients of 1 bit, 8 bits and 32 bits). As far as Plonky3, we benchmark size-16 batches of polynomials comprising total data size 2^ℓ equal to 2^{20}, 2^{24} and 2^{28}. In that setting, we consider only polynomials over the 31-bit Baby Bear field $\mathbb{F}_p$; indeed, that scheme would not perform any better upon being given as input a polynomial whose coefficients were "smaller" (albeit still $\mathbb{F}_p$-elements).

In our concrete benchmarks both of this work and of Plonky3 below, we omit throughout the Merkle-caps, oracle-skipping, and early-termination optimizations. (That is, in this work, we set $\vartheta := 1$ and $j := 0$, and moreover proceed analogously in Plonky3.) These omissions make our proofs become significantly larger (and our prover and verifier slower to boot); we refer to Table 1 above for our protocol's "true" proof sizes. On the other hand, they make our comparison to Plonky3 below more direct, since that work also neglects to include these optimizations, as currently written.

We explain our use of batching in our Plonky3 benchmarks. The most natural benchmark would have compared our scheme's performance on ℓ-variate multilinear polynomials to Plonky3's on single, degree-2^ℓ univariate polynomials. We note, however, that Plonky3's FRI-PCS implementation is heavily optimized towards the case of batched polynomial commitments. In order to compare our works more fairly, we instead run Plonky3 in the batched setting; that is, we benchmark it on batches of 2^4 univariate polynomials, each of degree $2^{\ell-4}$, for each problem size ℓ. Separately, in our own, non-batched scheme, we incorporate a straightforward optimization which serves to reduce by 4 the number of butterfly stages which our commitment phase must compute. In sum, both our scheme (operating on single multilinear polynomials) and theirs (operating on size-16 batches of univariate polynomials) must perform NTTs of essentially the same shape and size. This fact makes our works naturally comparable.

In our CPU benchmarks below, we use throughout a Google Cloud machine of type `c3-standard-22` with an Intel *Xeon Scalable* ("Sapphire Rapids") processor and 22 virtual cores. Both the Binius and Plonky3 implementations leverage AVX-512 accelerated instructions; Binius moreover uses the Intel GFNI instruction set extension. We benchmark Plonky3 using both the Poseidon2 and Keccak-256 hashes (the former hash is "recursion-friendly" in that work's prime-field setting). We present singlethreaded and multithreaded results in Tables 2 and 3 below, respectively.

Commit. Scheme	Prob. Size ℓ	Coef. Bits	Proof Size (MiB)	Commit (s)	Prove (s)	Verify (s)
Plonky3	20	31	0.593	0.3624	0.3048	0.02462
Baby Bear	24	31	0.866	6.019	4.993	0.03576
Poseidon	28	31	1.200	100.7	82.35	0.04847
Plonky3	20	31	0.842	0.2620	0.2870	0.01273
Baby Bear	24	31	1.200	4.487	4.780	0.01811
Keccak-256	28	31	1.700	77.07	79.14	0.02420
[DP25, Cons. 3.11]	20	1	0.195	0.00372	0.00274	0.00587
		8	0.217	0.02961	0.00420	0.00318
		32	0.288	0.1161	0.01672	0.00432
	24	1	0.750	0.04471	0.06061	0.01752
		8	0.772	0.3971	0.08984	0.00954
		32	1.213	1.600	0.4436	0.01287
	28	1	2.927	0.7548	1.932	0.06346
		8	4.324	6.281	2.452	0.03707
		32	14.806	28.32	17.16	0.1088
Construction 5.1	20	1	0.510	0.003212	0.01278	0.004076
		8	0.729	0.02835	0.1236	0.005783
		32	0.898	0.1175	0.3479	0.007369
	24	1	0.813	0.0551	0.1922	0.006882
		8	1.085	0.5081	1.980	0.009196
		32	1.288	2.223	5.613	0.01117
	28	1	1.186	1.097	3.331	0.01052
		8	1.509	9.874	31.80	0.01335
		32	1.748	41.93	90.32	0.01696

Table 2: Singlethreaded benchmarks.

Commit. Scheme	Prob. Size ℓ	Coef. Bits	Proof Size (MiB)	Commit (s)	Prove (s)	Verify (s)
Plonky3	20	31	0.593	0.04214	0.05521	0.02475
Baby Bear	24	31	0.866	0.6071	0.7995	0.03566
Poseidon	28	31	1.200	10.26	15.17	0.04850
Plonky3	20	31	0.842	0.0385	0.05559	0.01306
Baby Bear	24	31	1.200	0.4633	0.8384	0.01862
Keccak-256	28	31	1.700	8.606	15.67	0.02481
[DP25, Cons. 3.11]	20	1	0.195	0.001763	0.0008533	0.004823
		8	0.217	0.005349	0.001007	0.003085
		32	0.288	0.01297	0.002191	0.003182
	24	1	0.750	0.006406	0.005559	0.01278
		8	0.772	0.07380	0.01205	0.006936
		32	1.213	0.3092	0.04194	0.007944
	28	1	2.927	0.1591	0.1750	0.04020
		8	4.324	1.389	0.2881	0.02706
		32	14.806	5.421	1.392	0.07108
Construction 5.1	20	1	0.510	0.001172	0.006706	0.004194
		8	0.729	0.005159	0.02021	0.005891
		32	0.898	0.01718	0.05467	0.007490
	24	1	0.813	0.008364	0.02688	0.006973
		8	1.085	0.07555	0.2201	0.009278
		32	1.288	0.3341	0.6982	0.01123
	28	1	1.186	0.1528	0.3303	0.01059
		8	1.509	1.426	3.210	0.01353
		32	1.748	6.029	10.29	0.01582

Table 3: Multithreaded benchmarks.

References

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. *WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification.* Cryptology ePrint Archive, Paper 2024/1586. 2024. URL: `https://eprint.iacr.org/2024/1586`.

[AER24] Guillermo Angeris, Alex Evans, and Gyumin Roh. *A Note on Ligero and Logarithmic Randomness.* Cryptology ePrint Archive, Paper 2024/1399. 2024. URL: `https://eprint.iacr.org/2024/1399`.

[AHIV23] 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.

[BBHR18a] 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. DOI: 10.4230/LIPIcs.ICALP.2018.14.

[BBHR18b] 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.

[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.

[Ben+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. “Aurora: Transparent Succinct Arguments for R1CS”. In: *Advances in Cryptology – EUROCRYPT 2019*. Berlin, Heidelberg: Springer-Verlag, 2019, pp. 103–128. ISBN: 978-3-030-17652-5. DOI: 10.1007/978-3-030-17653-2_4.

[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). DOI: 10.1145/3614423.

[BGKS19] 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.

[Bre+25] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. “Blaze: Fast SNARKs from Interleaved RAA Codes”. In: *Advances in Cryptology – EUROCRYPT 2025*. Ed. by Serge Fehr and Pierre-Alain Fouque. Cham: Springer Nature Switzerland, 2025, pp. 123–152. ISBN: 978-3-031-91134-7.

[Can89] David G Cantor. “On arithmetical algorithms over finite fields”. In: *Journal of Combinatorial Theory, Series A* 50.2 (1989), pp. 285–300. DOI: https://doi.org/10.1016/0097-3165(89)90020-4.

[CBBZ23] 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.

[DG25] Benjamin E. Diamond and Angus Gruen. “Proximity Gaps in Interleaved Codes”. In: *IACR Communications in Cryptology* 1.4 (Jan. 13, 2025). ISSN: 3006-5496. DOI: 10.62056/a0ljbkrz.

[DP24] Benjamin E. Diamond and Jim Posen. “Proximity Testing with Logarithmic Randomness”. In: *IACR Communications in Cryptology* 1.1 (2024). ISSN: 3006-5496. DOI: 10.62056/aksdkp10.

[DP25] Benjamin E. Diamond and Jim Posen. "Succinct Arguments over Towers of Binary Fields". In: *Advances in Cryptology – EUROCRYPT 2025*. Ed. by Serge Fehr and Pierre-Alain Fouque. Cham: Springer Nature Switzerland, 2025, pp. 93–122. ISBN: 978-3-031-91134-7.

[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.

[Fs24] Matteo Frigo and abhi shelat. *Anonymous credentials from ECDSA*. Cryptology ePrint Archive, Paper 2024/2010. 2024. URL: `https://eprint.iacr.org/2024/2010`.

[GG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 3rd Edition. Cambridge University Press, 2013.

[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. DOI: `10.1007/978-3-031-38545-2_7`.

[Gur06] Venkatesan Guruswami. *Algorithmic Results in List Decoding*. Vol. 2. Foundations and Trends in Theoretical Computer Science 2. now publishers, 2006. DOI: `10.1561/0400000007`.

[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`.

[HLP24] Ulrich Haböck, David Levit, and Shahar Papini. *Circle STARKs*. Cryptology ePrint Archive, Paper 2024/278. 2024. URL: `https://eprint.iacr.org/2024/278`.

[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. DOI: `10.1007/978-3-642-17373-8`.

[Lan02] Serge Lang. *Algebra*. Revised Third Edition. Vol. 211. Graduate Texts in Mathematics. Springer, 2002.

[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`.

[Li+18] Wen-Ding Li, Ming-Shing Chen, Po-Chun Kuo, Chen-Mou Cheng, and Bo-Yin Yang. "Frobenius Additive Fast Fourier Transform". In: *ACM International Symposium on Symbolic and Algebraic Computation*. 2018. ISBN: 9781450355506. DOI: `10.1145/3208976.3208998`.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Ed. by G.-C. Rota. 2nd. Vol. 20. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1996. DOI: `10.1017/CBO9780511525926`.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2111–2128. ISBN: 9781450367479. DOI: `10.1145/3319535.3339817`.

[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`.

[RR24] Noga Ron-Zewi and Ron Rothblum. "Local Proofs Approaching the Witness Length". In: *Journal of the ACM* 71.3 (June 2024). DOI: `10.1145/3661483`. URL: `https://doi.org/10.1145/3661483`.

[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`.

[Sou24] Lev Soukhanov. "Hashcaster". Unpublished report. Sept. 2024. URL: https://hackmd.io/@levs57/SJ4fuZMD0#Hashcaster.

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

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

[Wat79] William C. Waterhouse. "The Normal Basis Theorem". In: *The American Mathematical Monthly* 86.3 (1979), pp. 212–212. URL: http://www.jstor.org/stable/2321527.

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

[Xie+19] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation". In: *Advances in Cryptology – CRYPTO 2019.* Berlin, Heidelberg: Springer-Verlag, 2019, pp. 733–764. ISBN: 978-3-030-26953-1. DOI: 10.1007/978-3-030-26954-8_24.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes". In: *Advances in Cryptology – CRYPTO 2024.* Berlin, Heidelberg: Springer-Verlag, 2024, pp. 138–169. ISBN: 978-3-031-68402-9. DOI: 10.1007/978-3-031-68403-6_5.

[ZXZS20] J. Zhang, T. Xie, Y. Zhang, and D. Song. "Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof". In: *IEEE Symposium on Security and Privacy.* 2020, pp. 859–876. ISBN: 2375-1207. DOI: 10.1109/SP40000.2020.00052.