Zero-Knowledge Polynomial Commitment in Binary Fields

Benjamin E. Diamond

Irreducible

bdiamond@irreducible.com

Abstract

In recent work, Diamond and Posen ('24) introduce a polynomial commitment scheme for large binary fields, adapting *BaseFold* (CRYPTO '24). In this note, we devise a zero-knowledge variant of Diamond and Posen's scheme. Our construction reprises a few ideas from Aurora (EUROCRYPT '19). We carry through those ideas in characteristic 2, and moreover show that they're compatible with BaseFold.

1 Introduction

In two recent works, Diamond and Posen [DP25] [DP24] introduce multilinear polynomial commitment schemes for polynomials over tiny binary fields. That latter work's PCS proceeds in two steps. First, those authors introduce a reduction—called *ring-switching* [DP24, § 3]—from the problem of tiny-field commitment to the problem of large-field commitment. They then define a standalone, large-field scheme [DP24, § 4]. That standalone scheme adapts Zeilberger, Chen and Fisch's *BaseFold* [ZCF24] to the characteristic 2 setting.

In this work, we make the large-field multilinear polynomial commitment scheme [DP24, Cons. 4.11] zero-knowledge. Our adaptation adds a bit of constant, multiplicative overhead to [DP24, Cons. 4.11] (of about fourfold for the prover, to be precise). Reducing this overhead seems tricky; we discuss this problem further below. We only treat large-field commitment in this work, and not ring-switching (see Section 5).

Our ideas adapt ones already present in Zhang, Xie, Zhang and Song's *Virgo* [ZXZS20, Prot. 2]. In fact, Virgo itself calls these ideas "standard", and cites in turn Ames, Hazay, Ishai and Venkitasubramaniam's *Ligero* [AHIV23] and Ben-Sasson et al.'s *Aurora* [Ben+19]. They seem to be folklore. To set up the problem, we recall the basic structure of FRI. In that protocol, the prover encodes a sensitive polynomial onto a large Reed–Solomon domain, and further "folds" that encoding onto various successively smaller domains. Later in the protocol, the verifier queries all oracles—i.e., not just the initial one, but also the subsequent folded ones—at a smattering of points (roughly, at a number of points proportional to the security parameter).

We must take care that these evaluations learned by the verifier reveal nothing about the prover's sensitive polynomial. To this end, we apply two techniques, both of which appear (in some form) in the above works. First, we direct the prover to pad its input polynomial with random coefficients on the "high end". Provided that it uses enough random coefficients in this process—specifically, at least as many as the number of evaluations the verifier is liable to demand—the prover may ensure that its polynomial's evaluations become random-looking *outside* of a certain "fundamental domain". As is standard, we use a Reed–Solomon domain disjoint from this fundamental domain—a "disjoint coset" (or more properly for us, a *disjoint affine subspace*).

On the other hand, these high random coefficients are not enough. The issue is that as FRI proceeds, the prover's polynomial "shrinks", while the verifier continues to demand the same number of evaluations (i.e., on *each* smaller oracle). On these small domains, the verifier's fixed number of queries becomes liable to overpower the prover's randomness, since the prover's polynomial appears on them in "condensed" form.

To solve this issue, we use another technique from Aurora [Ben+19, § 5.1]. That is, we direct the prover to sample a second, purely random polynomial, just as long as its padded input. The parties run FRI not on the prover's padded initial input, but on a virtual linear combination between that padded input and the prover's second, purely random polynomial (with respect to a combination challenge sampled by the verifier). This procedure guarantees that, on the shrunken domains, the prover evaluates only this combination and its derivates, which are "purely random" (i.e., simulatable). The prover must evaluate its raw padded input only on the initial domain, where that polynomial is un-condensed, and its padding suffices to mask it.

1.1 Some Remarks

We mention various further aspects of our zero-knowledge construction.

Padding in the novel basis. High-end padding becomes "free" in Lin, Chung and Han's *novel polynomial basis*, a fact which Aurora [Ben+19] appears to have missed. We fix a large binary field L, equipped with an $\mathbb{F}_2$-basis $(\beta_0, \ldots, \beta_{r-1})$, as well as a statement size parameter ℓ. Lin, Chung and Han [LCH14] introduce a *novel polynomial basis* of the L-vector space $L[X]^{\prec 2^\ell}$ of L-polynomials of degree strictly less than 2^ℓ; we write $X_0(X), \ldots, X_{2^\ell-1}(X)$ for that basis. For each $j \in \{0, \ldots, 2^\ell - 1\}$, $X_j(X)$ is a polynomial of degree j. Lin, Chung and Han describe a quasilinear-time algorithm—the *additive NTT*—which, on input a coefficient vector $(a_0, \ldots, a_{2^\ell-1})$, outputs the evaluations of $P(X) \coloneqq \sum_{j=0}^{2^\ell-1} a_j \cdot X_j(X)$ on some affine subspace $S^{(0)} \subset L$. Here, the evaluation domain $S^{(0)}$ must be a union of cosets of $\langle \beta_0, \ldots, \beta_{\ell-1} \rangle$ (we refer to [DP24, § 2.3] for further background on the additive NTT).

We write $H \coloneqq \langle \beta_0, \ldots, \beta_{\ell-1} \rangle$. To pad the polynomial $f(X) \in L[X]^{\prec 2^\ell}$ with high, random coefficients—without disturbing $f(X)$'s values on H (since we can't change the prover's statement)—we begin as Aurora [Ben+19, § 5] does. That is, we replace $f(X)$ with $f'(X) \coloneqq f(X) + Z_H(X) \cdot r(X)$; here, $Z_H(X) \coloneqq \prod_{u \in H}(X - u)$ is H's vanishing polynomial, and $r(X)$ is random of appropriate degree. We note that $f(X)$ and $f'(X)$ have identical restrictions to $H \subset L$.

On the other hand, we note that $Z_H(X) = W_\ell(X)$, the Lin–Chung–Han "subspace vanishing polynomial" attached to $\langle \beta_0, \ldots, \beta_{\ell-1} \rangle$. Because

$$X_{2^\ell + j}(X) = \widehat{W}_\ell(X) \cdot X_j(X)$$

holds for each $j \in \{0, \ldots, 2^\ell - 1\}$, the coordinate representation of $Z_H(X) \cdot r(X)$ in the novel basis is just the upward shift by 2^ℓ positions of $r(X)$'s (technically we need to normalize $Z_H(X)$, but this is immaterial). Equivalently, to obtain $f'(X)$, the prover can just take $f(X)$'s coefficient vector—again in the novel basis—and tack on some purely random entries to its high end. It can thus randomize $f(X)$ "without doing any arithmetic".

Interaction with BaseFold. Aurora [Ben+19] and Virgo [ZXZS20] develop zero-knowledge variants of FRI [BBHR18]. In our case, we have a sumcheck too to deal with. Indeed, BaseFold PCS [ZCF24, § 5] *interleaves* FRI-folding with a sumcheck. For further detail on BaseFold PCS, we refer to Zeilberger, Chen and Fisch [ZCF24, § 5] (see also [DP24, § 4]).

A priori, BaseFold's sumcheck presents a problem for us, since it could reveal something about the prover's input multilinear. We note that Virgo [ZXZS20, Prot. 4] develops a zero-knowledge sumcheck variant; on the other hand, that protocol *assumes* access to a zero-knowledge multilinear polynomial commitment scheme, an object which we're in the process of trying to create.

In fact, we adopt a simpler solution. As we show below, the recourse used by Aurora to protect its prover's positive-indexed FRI oracles serves also to save our sumcheck. Indeed, the message upon which the parties run FRI is the linear combination between the prover's padded input and its further, purely random input. Since that combination is simulatable, the prover's sumcheck on it is too. In particular, our simulator—having generated the prover's combination in full—may simply run the sumcheck "honestly" on it. We explain this remedy in detail below.

2 Background and Notation

We import all notation from [DP24, § 2]. In particular, we recall multilinear polynomials (see [DP24, § 2.1]), the *novel polynomial basis* [LCH14] polynomials $X_0(X), \ldots, X_{2^\ell-1}(X)$ (see also [DP24, § 2.3]), the additive NTT (see also [DP24, Alg. 2]), the Reed–Solomon code $\mathsf{RS}_{L,S^{(0)}}[2^{\ell+\mathcal{R}}, 2^\ell]$ (see also [DP24, § 2.2]), and FRI [BBHR18] (see also [DP24, § 2.4]).

3 Security Definitions

We record security definitions.

3.1 Multilinear Polynomial Commitment Schemes

We begin by reviewing various definitions from [DP24].

FUNCTIONALITY 3.1 ($\mathcal{F}_{\mathsf{Vec}}^{L}$—vector oracle)**.**
An arbitrary alphabet L is given.

- Upon receiving $(\texttt{submit}, m, f)$ from $\mathcal{P}$, where $m \in \mathbb{N}$ and $f : \mathcal{B}_m \to L$, 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 $\mathcal{V}$ $(\texttt{result}, f(v))$.

Definition 3.2. 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 $\mathcal{F}_{\mathsf{Vec}}^{L}$. 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 define the *soundness* of IOPCSs exactly as in [DP24, Def. 2.8].

3.2 Zero-Knowledge

We now define *zero-knowledge* for IOPCSs. Our definition of below adapts Virgo's [ZXZS20, Def. 3], and also incorporates the IOP model.

Definition 3.3. For each interactive oracle polynomial commitment scheme Π, security parameter $\lambda \in \mathbb{N}$, number-of-variables parameter $\ell \in \mathbb{N}$, and PPT simulator $\mathcal{S}$, we define the following experiment:

- The experimenter samples $\mathsf{params} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \ell)$, and gives params, including L, to $\mathcal{P}$ and $\mathcal{S}$.
- $\mathcal{P}$ outputs a multilinear $t(X_0, \ldots, X_{\ell-1}) \in L[X_0, \ldots, X_{\ell-1}]^{\preceq 1}$.
- $\mathcal{P}$ interacts with the vector oracle as prescribed by $\Pi.\mathsf{Commit}(\mathsf{params}, t)$, and so produces $[f]$.
- $\mathcal{V}$, given $[f]$, outputs an evaluation point $r \in L^\ell$. The experimenter gives $\mathcal{V}$ $s := t(r)$.
- The experimenter defines two distributions:
  - $\mathsf{Real}^{\Pi,\ell}(\lambda)$: $\mathcal{P}$ and $\mathcal{V}$ interact honestly on $\langle \mathcal{P}([f], s, r; t), \mathcal{V}([f], s, r) \rangle$. Output $\mathcal{V}$'s view.
  - $\mathsf{Ideal}_{\mathcal{S}}^{\Pi,\ell}(\lambda)$: $\mathcal{S}$, given params, r and s (but not t), initiates an internal simulation to the honest verifier $\mathcal{V}$, in which $\mathcal{S}$ simulates the existence both of $\mathcal{P}$ and of the vector oracle. Output $\mathcal{V}$'s view.

The IOPCS Π is said to be *zero-knowledge* if there exists a PPT simulator $\mathcal{S}$ such that the distributions $\left\{\mathsf{Real}^{\Pi,\ell}(\lambda)\right\}_{\ell,\lambda}$ and $\left\{\mathsf{Ideal}_{\mathcal{S}}^{\Pi,\ell}(\lambda)\right\}_{\ell,\lambda}$ are identical.

4 Our Protocol

We now give our protocol, adapting [DP24, Cons. 4.11].

CONSTRUCTION 4.1 (Zero-Knowledge Binary BaseFold)**.**
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, run the setup algorithm of [DP24, § 4.11] on the inputs 1^λ and $\ell+1$. In this way, obtain $\mathcal{R}$, a binary field $L/\mathbb{F}_2$, a repetition parameter $\gamma = \omega(\log \lambda)$, an $\mathbb{F}_2$-basis $(\beta_0, \ldots, \beta_{r-1})$ of L, and the novel basis $(X_0(X), \ldots, X_{2^{\ell+1}-1}(X))$ of $L[X]^{\prec 2^{\ell+1}}$. Fix a folding factor $\vartheta \mid \ell$. Write $S^{(0)} \coloneqq \beta_{\ell+\mathcal{R}+1} + \langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}} \rangle$; for each $i \in \{1, \ldots, \ell\}$, write $S^{(i)} \coloneqq \widehat{W}_i\big(S^{(0)}\big)$. Write $C^{(0)} \subset L^{2^{\ell+\mathcal{R}+1}}$ for the Reed–Solomon code $\mathsf{RS}_{L,S^{(0)}}[2^{\ell+1+\mathcal{R}}, 2^{\ell+1}]$. Write $\kappa \coloneqq \gamma \cdot 2^\vartheta$ for the number of points of each FRI oracle that $\mathcal{V}$ opens.

2. $[f] \leftarrow \Pi.\mathsf{Commit}(\mathsf{params}, t)$. Write $(t_i)_{i=0}^{2^\ell-1}$ for $t(X_0, \ldots, X_{\ell-1})$'s Lagrange coefficient vector. Moreover, for each $j \in \{0, \ldots, \kappa-1\}$, sample $t_{2^\ell+j} \leftarrow L$. Write $P(X) \coloneqq \sum_{j=0}^{2^\ell+\kappa-1} t_j \cdot X_j(X)$. Using the additive NTT [DP24, Alg. 2], compute the codeword $f : S^{(0)} \to L$ defined by $f : x \mapsto P(x)$. Send $(\mathtt{submit}, \ell+\mathcal{R}+1, f)$ to $\mathcal{F}_{\mathsf{Vec}}^L$. Upon getting $(\mathtt{receipt}, \ell+\mathcal{R}+1, [f])$ from $\mathcal{F}_{\mathsf{Vec}}^L$, output $[f]$.

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. $\mathcal{P}$ samples $2^\ell + \kappa$ further fresh random coefficients, say $t'_j \leftarrow L$, for $j \in \{0, \ldots, 2^\ell + \kappa - 1\}$. $\mathcal{P}$ writes $P'(X) \coloneqq \sum_{j=0}^{2^\ell+\kappa-1} t'_j \cdot X_j(X)$, and encodes it just as in the commitment procedure above, so obtaining $f' : S^{(0)} \to L$. $\mathcal{P}$ sends $(\mathtt{submit}, \ell+\mathcal{R}+1, f')$ too to the oracle.

2. $\mathcal{P}$'s new random coefficients $\big(t'_j\big)_{j=0}^{2^\ell+\kappa-1}$ lexicographically define an $\ell+1$-variate multilinear, say $t'(X_0, \ldots, X_\ell)$. $\mathcal{P}$ calculates $s' \coloneqq t'(r_0, \ldots, r_{\ell-1}, 0)$ and sends s' to $\mathcal{V}$ in the clear.

3. Upon receiving $[f']$ and s', $\mathcal{V}$ samples $\alpha \leftarrow L$ and sends α to $\mathcal{P}$.

4. $\mathcal{P}$ and $\mathcal{V}$ define the virtual combination oracle $f^{(0)} \coloneqq \alpha \cdot f + f'$. Moreover, they write $s_0 \coloneqq \alpha \cdot s + s'$. $\mathcal{P}$ defines
$$t''(X_0, \ldots, X_{\ell-1}) \coloneqq \alpha \cdot t(X_0, \ldots, X_{\ell-1}) + t'(X_0, \ldots, X_{\ell-1}, 0),$$
and sets $h(X_0, \ldots, X_{\ell-1}) \coloneqq \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, X_0, \ldots, X_{\ell-1}) \cdot t''(X_0, \ldots, X_{\ell-1})$.

5. Exactly as in [DP24, § 4.11], $\mathcal{P}$ and $\mathcal{V}$ proceed for ℓ rounds, interleaving a sumcheck on $h(X_0, \ldots, X_{\ell-1})$—with the initial statement s_0—with FRI folding on the (virtual) initial oracle $f^{(0)}$. In this way, if the verifier doesn't reject, they wind up a final sumcheck claim s_ℓ, positive oracles $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$, and a final FRI message (c_0, c_1) sent in the clear. Here, $c_0 + c_1 \cdot X_1^{(\ell)}(X)$ is the ℓ^{th}-order basis representation of $\mathcal{P}$'s final FRI oracle $f^{(\ell)} : S^{(\ell)} \to L$. Note here that $S^{(\ell)} \subset L$ is affine-linear of dimension $\mathcal{R}+1$, and $f^{(\ell)}$ is the encoding of a polynomial of degree at most 1.

6. $\mathcal{V}$ checks $s_\ell \overset{?}{=} \widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}) \cdot c_0$.

7. $\mathcal{V}$ executes the usual FRI querying procedure, for γ repetitions, on the words $f^{(0)}, f^{(\vartheta)}, \ldots, f^{(\ell)}$. Here, again, $f^{(0)} = \alpha \cdot f + f'$ is virtual, the oracles $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ are committed, and $f^{(\ell)}$ is given fully in the clear to the verifier, as the encoding of $c_0 + c_1 \cdot X_1^{(\ell)}(X)$ onto the domain $S^{(\ell)}$.

Construction 4.1 is complete. The idea is straightforward, once you understand [DP24, Thm. 4.12]. First, $h(X_0, \ldots, X_{\ell-1})$ is defined in such a way that $\sum_{w \in \mathcal{B}_\ell} h(w) = t''(r_0, \ldots, r_{\ell-1}) = \alpha \cdot t(r_0, \ldots, r_{\ell-1}) + t'(r_0, \ldots, r_{\ell-1}, 0)$, which itself equals $\alpha \cdot s + s' = s_0$ if $\mathcal{P}$ is honest. $\mathcal{P}$'s sumcheck claim will thus be true. On the other hand, by [DP24, Lem. 4.13], c_0 will be $\alpha \cdot t(r'_0, \ldots, r'_{\ell-1}) + t'(r_0, \ldots, r'_{\ell-1}, 0)$. The product of c_0 with $\widetilde{\mathsf{eq}}(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1})$ will thus give $\mathcal{V}$ exactly what it needs at the very end of the sumcheck.

Theorem 4.2. *The IOPCS* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *of Construction 4.1 is sound.*

Proof (sketch). $\mathcal{E}$, on input $\mathcal{A}$'s message f to the oracle, runs the Berlekamp–Welch decoder (see e.g. [DP24, Alg. 1]), as usual. If $\mathcal{E}$ successfully obtains a polynomial $P(X) = \sum_{j=0}^{2^{\ell+1}-1} t_j \cdot X_j(X)$, then $\mathcal{E}$ writes $t(X_0, \ldots, X_{\ell-1})$ for the multilinear whose Lagrange coefficients are $(t_j)_{j=0}^{2^\ell-1}$, and outputs $t(X_0, \ldots, X_{\ell-1})$.

If either $f : S^{(0)} \to L$ or $f' : S^{(0)} \to L$ fails to reside within $C^{(0)}$'s unique decoding radius, then so too will $f^{(0)}$, with high probability over $\mathcal{V}$'s choice of α, by the basic Reed–Solomon proximity gap [Ben+23, Thm. 4.1] (see also [DP24, Thm. 2.2]). Moreover, if $f^{(0)}$ isn't in $C^{(0)}$'s unique decoding radius, then $\mathcal{V}$ will reject [DP24, Prop. 4.23]. We can thus safely treat just the case in which f and f' are close to the code, so that $t(X_0, \ldots, X_{\ell-1})$ and $t'(X_0, \ldots, X_\ell)$ are both well-defined.

Again by [DP24, Prop. 4.23], we can likewise assume that $\mathcal{A}$'s positive oracles $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ too are close to the code, as well as that these oracles' close codewords are consistent in the sense of [DP24, Def. 4.17]. Under these conditions, [DP24, Lem. 4.13] gives us the guarantee $c_0 = \alpha \cdot t(r'_0, \ldots, r'_{\ell-1}) + t'(r'_0, \ldots, r'_{\ell-1}, 0)$.

Now if $s \neq t(r_0, \ldots, r_{\ell-1})$ holds, then so too will

$$\alpha \cdot s + s' \neq \alpha \cdot t(r_0, \ldots, r_{\ell-1}) + t'(r_0, \ldots, r_{\ell-1}, 0) = t''(r_0, \ldots, r_{\ell-1}),$$

except with low probability over $\mathcal{V}$'s choice of $\alpha \leftarrow L$. We thus further assume that

$$s^{(0)} = \alpha \cdot s + s' \neq \alpha \cdot t(r_0, \ldots, r_{\ell-1}) + t'(r_0, \ldots, r_{\ell-1}, 0) = t''(r_0, \ldots, r_{\ell-1}) = \sum_{w \in \mathcal{B}_\ell} h(w).$$

Invoking the soundness of the sumcheck, we conclude that, except with low probability,

$$s_\ell \neq h(r'_0, \ldots, r'_{\ell-1}) = \widetilde{\mathsf{eq}}\big(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}\big) \cdot t''(r'_0, \ldots, r'_{\ell-1})$$

too will hold as of that protocol's end. On the other hand, we just justified assuming that $c_0 = \alpha \cdot t(r'_0, \ldots, r'_{\ell-1}) + t'(r'_0, \ldots, r'_{\ell-1}, 0) = t''(r'_0, \ldots, r'_{\ell-1})$. Substituting that last equality into the inequality just above, we obtain

$$s_\ell \neq \widetilde{\mathsf{eq}}\big(r_0, \ldots, r_{\ell-1}, r'_0, \ldots, r'_{\ell-1}\big) \cdot c_0,$$

so that the verifier will reject. □

Theorem 4.3. *The IOPCS* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathcal{P}, \mathcal{V})$ *of Construction 4.1 is zero-knowledge.*

Proof. We first write down our simulator $\mathcal{S}$. On input params, r and s, $\mathcal{S}$ operates as follows:

1. $\mathcal{S}$ simulates the vector oracle's sending to $\mathcal{V}$ the receipt $[f']$, as if $\mathcal{P}$ had just committed to $f' : S^{(0)} \to L$.
2. $\mathcal{S}$ simulates $\mathcal{P}$'s supposed evaluation $s' \leftarrow L$ uniformly randomly, and sends s' to $\mathcal{V}$ as if from $\mathcal{P}$.
3. $\mathcal{S}$ receives in turn $\mathcal{V}$'s message α, intended for $\mathcal{P}$.
4. $\mathcal{S}$ samples coefficients $(t''_j)_{j=0}^{2^\ell+\kappa-1}$ uniformly randomly *subject* to the condition $t''(r_0, \ldots, r_{\ell-1}) = \alpha \cdot s + s'$. Here, we write
$$t''(X_0, \ldots, X_{\ell-1}) \coloneqq \sum_{w \in \mathcal{B}_\ell} t''_{\{w\}} \cdot \widetilde{\mathsf{eq}}(X_0, \ldots, X_{\ell-1}, w_0, \ldots, w_{\ell-1}).$$
$\mathcal{S}$ locally encodes $(t''_j)_{j=0}^{2^\ell+\kappa-1}$ in the usual way; that is, it evaluates $P''(X) \coloneqq \sum_{j=0}^{2^\ell+\kappa-1} t''_j \cdot X_j(X)$ on $S^{(0)}$, and so obtains a function $f'' : S^{(0)} \to L$.
5. $\mathcal{S}$ performs the entire sumcheck "honestly" with $\mathcal{V}$, using $t''(X_0, \ldots, X_{\ell-1})$. Moreover, $\mathcal{S}$ FRI-folds "honestly" using $f^{(0)} \coloneqq f''$. That is, it honestly locally computes, just as $\mathcal{P}$ would, the positive oracles $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ (and simulates to $\mathcal{V}$ their commitment receipts), as well as the final message (c_0, c_1).
6. $\mathcal{S}$ writes $(u_0, \ldots, u_{\kappa-1})$ for the points of $S^{(0)}$ opened by $\mathcal{V}$. For each such u_i, $\mathcal{S}$ simulates $f(u_i) \leftarrow L$ uniformly randomly. Moreover, it sets $f'(u_i) \coloneqq f''(u_i) - \alpha \cdot f(u_i)$. $\mathcal{S}$ simulates all of $\mathcal{V}$'s oracle queries to f and f' in this way; it answers all of $\mathcal{V}$'s queries to $f^{(\vartheta)}, \ldots, f^{(\ell-\vartheta)}$ "honestly".

In step 4, the condition $t''(r_0, \ldots, r_{\ell-1}) = \alpha \cdot s + s'$ is linear on the space of coefficients $(t''_j)_{j=0}^{2^\ell-1}$, so that $\mathcal{S}$ can perform that step's sampling procedure efficiently. In step 6 above, as well as henceforth, we assume for convenience that the query points $(u_0, \ldots, u_{\kappa-1})$ in $S^{(0)}$ are *distinct*; the case in which they're not is similar.

We argue that this simulator $\mathcal{S}$ fulfills the requirements of Definition 3.3.

We begin with a linear-algebraic lemma. This lemma shows that, for low terms $(t_j)_{j=0}^{2^\ell-1}$ and evaluation points $(u_i)_{i=0}^{\kappa-1}$ fixed, choosing high terms $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$ is *equivalent* to choosing the evaluations $(f(u_i))_{i=0}^{\kappa-1}$. The key is that $S^{(0)}$ is disjoint from $\langle \beta_0, \ldots, \beta_{\ell-1} \rangle$.

Lemma 4.4. *For each list $(t_j)_{j=0}^{2^\ell-1}$ and each list $(u_0, \ldots, u_{\kappa-1})$ of distinct $S^{(0)}$-elements, the affine-linear map $L^\kappa \to L^\kappa$ which sends*

$$\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1} \mapsto (P(u_i))_{i=0}^{\kappa-1}, \tag{1}$$

where $P(X) \coloneqq \sum_{j=0}^{2^\ell+\kappa-1} t_j \cdot X_j(X)$, is bijective.

Proof. For each list of low coefficients $(t_j)_{j=0}^{2^\ell-1}$ as in the hypothesis of the theorem, we write $P_{\text{low}}(X) \coloneqq \sum_{j=0}^{2^\ell-1} t_j \cdot X_j(X)$; moreover, for each high list $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$, we write $P_{\text{high}}(X) \coloneqq \sum_{j=0}^{\kappa-1} t_{2^\ell+j} \cdot X_{2^\ell+j}(X)$. Since $P(X) = P_{\text{low}}(X) + P_{\text{high}}(X)$, we may freely, after subtracting off the constant vector $(P_{\text{low}}(u_i))_{i=0}^{\kappa-1}$ from the image of (1), consider instead the *linear* map

$$\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1} \mapsto (P_{\text{high}}(u_i))_{i=0}^{\kappa-1}. \tag{2}$$

Indeed, if (2) is bijective, then (1) is too, since they differ by the additive affine offset $(P_{\text{low}}(u_i))_{i=0}^{\kappa-1}$.

We re-express (2) using the following matrix identity:

$$\left.\begin{aligned} \begin{bmatrix} \\ P_{\text{high}}(u_i) \\ \\ \end{bmatrix} &= \begin{bmatrix} & & \\ X_{2^\ell}(u_i) & \cdots & X_{2^\ell+\kappa-1}(u_i) \\ & & \end{bmatrix} \cdot \begin{bmatrix} & \\ t_{2^\ell+j} & \\ & \end{bmatrix} \\ &= \begin{bmatrix} X_{2^\ell}(u_0) & & \\ & \ddots & \\ & & X_{2^\ell}(u_{\kappa-1}) \end{bmatrix} \cdot \begin{bmatrix} & & \\ X_0(u_i) & \cdots & X_{\kappa-1}(u_i) \\ & & \end{bmatrix} \cdot \begin{bmatrix} & \\ t_{2^\ell+j} & \\ & \end{bmatrix} \end{aligned} \right\} \begin{matrix} i \text{ varying} \\ \\ j \text{ varying.} \end{matrix}$$

The first equality above is just the definition of $P_{\text{high}}(X)$. That first matrix contains a "Vandermonde" of the novel basis polynomials $X_{2^\ell}(X), \ldots, X_{2^\ell+\kappa-1}(X)$, evaluated respectively at the points $(u_0, \ldots, u_{\kappa-1})$. That is, the j^{th} column of that matrix gives the respective evaluations at $(u_0, \ldots, u_{\kappa-1})$ of the novel basis polynomial $X_{2^\ell+j}(X)$.

In the second equality, we use the recursive substructure of the novel basis polynomials. Indeed, for each $j \in \{0, \ldots, \kappa-1\}$, $X_{2^\ell+j}(X) = X_{2^\ell}(X) \cdot X_j(X)$ as polynomials (we assume that $2^\ell \geq \kappa$ here). We may thus replace the high-end Vandermonde associated with $X_{2^\ell}(X), \ldots, X_{2^\ell+\kappa-1}(X)$ with the standard Vandermonde associated instead with $X_0(X), \ldots, X_{\kappa-1}(X)$, provided that we scale the rows of that latter Vandermonde by the scalars $X_{2^\ell}(u_0), \ldots, X_{2^\ell}(u_{\kappa-1})$.

The diagonal matrix $\mathsf{diag}(X_{2^\ell}(u_0), \ldots, X_{2^\ell}(u_{\kappa-1}))$ is nonsingular. Indeed, $X_{2^\ell}(X)$ vanishes exactly on $\langle \beta_0, \ldots, \beta_{\ell-1} \rangle$; moreover, we chose $S^{(0)}$ specifically to be disjoint from that subspace. Thus the entries of that diagonal are all nonzero. (This is where we use the disjointness of $S^{(0)}$ from $U_\ell = \langle \beta_0, \ldots, \beta_{\ell-1} \rangle$.)

Finally, the Vandermonde associated with $X_0(X), \ldots, X_{\kappa-1}(X)$ is also nonsingular. This fact is standard; to prove it, we note that that Vandermonde in turn can be expressed as the product between the standard univariate Vandermonde, with respect to the points $(u_0, \ldots, u_{\kappa-1})$, and the change-of-basis matrix between the monomial basis and the novel basis. Both of these matrices are nonsingular. □

We now define a number of hybrid distributions, all of which will be identical to each other.

1. Defined to be $\mathsf{Real}^{\Pi,\ell}$.
2. Same as above, except for the following differences. $\mathcal{V}$'s receipt of $[f']$ is replaced by a simulation of that receipt. $\mathcal{P}$'s value $s' \coloneqq t'(r_0, \ldots, r_{\ell-1}, 0)$ is replaced with a freshly sampled random scalar $s' \leftarrow L$. The oracle $f^{(0)}$ upon which FRI folding is conducted is defined not as $\alpha \cdot f + f'$, but rather as the encoding f'' of a vector $\left(t''_j\right)_{j=0}^{2^\ell+\kappa-1}$ sampled exactly as in 4 above (that is, uniformly randomly subject to the linear condition $t''(r_0, \ldots, r_{\ell-1}) = \alpha \cdot s + s'$). Finally, $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1}$ is defined according to the backfilling rule $t'_j \coloneqq t''_j - \alpha \cdot t_j$ for each $j \in \{0, \ldots, 2^\ell + \kappa - 1\}$, and f' is defined to be its encoding.
3. Same as above, except instead of defining f as the encoding of $(t_j)_{j=0}^{2^\ell+\kappa-1}$, the simulator instead samples the evaluations $f(u_i) \leftarrow L$ directly as random scalars for each $i \in \{0, \ldots, \kappa - 1\}$. Using these values and the prover's secret message $(t_j)_{j=0}^{2^\ell-1}$, the simulator backfills the high values $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$. This prescription is well-defined by Lemma 4.4. Finally, the simulator uses the resulting list $(t_j)_{j=0}^{2^\ell+\kappa-1}$ to perform the backfilling $t'_j \coloneqq t''_j - \alpha \cdot t_j$ for $j \in \{0, \ldots, 2^\ell + \kappa - 1\}$, and to define f', just as it did above.
4. Defined to be $\mathsf{Ideal}_{\mathcal{S}}^{\Pi,\ell}$.

Lemma 4.5. *The distributions 1 and 2 are identical.*

Proof. Since, for each $(r_0, \ldots, r_{\ell-1})$, the map $L^{2^\ell+\kappa} \to L$ which sends $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1} \mapsto t'(r_0, \ldots, r_{\ell-1}, 0)$ is linear and surjective, the honest prover might as well, as opposed to choosing $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1}$ uniformly, instead first sample the image point $s' \leftarrow L$, and then pick a uniform element $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1}$ in the hyperplane $H' \subset L^{2^\ell+\kappa}$ for which $t'(r_0, \ldots, r_{\ell-1}, 0) = s'$ holds. Thus 2's selection procedure for s' is valid. Furthermore, adding $(\alpha \cdot t_j)_{j=0}^{2^\ell+\kappa-1}$ translates H' identically to the further hyperplane $H'' \subset L^{2^\ell+\kappa}$ consisting of coefficient vectors $\left(t''_j\right)_{j=0}^{2^\ell+\kappa-1}$ for which $t''(r_0, \ldots, r_{\ell-1}) = \alpha \cdot s + s'$ holds. Instead of choosing a uniform element of H', therefore, the simulator might as well choose a uniform element of H'' and backfill the corresponding element of H'. □

Lemma 4.6. *The distributions 2 and 3 are identical.*

Proof. As of the point at which $\mathcal{V}$ first reveals its query locations, the prover's high coefficients $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$ remain purely random and independent of everything else in the transcript. $\mathcal{V}$'s choice of query positions thus amounts to a choice of isomorphism $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1} \mapsto (P(u_i))_{i=0}^{\kappa-1}$. Instead of choosing the preimage $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$ uniformly as $\mathcal{P}$ would, $\mathcal{S}$ might as well uniformly choose the image $(f(u_i))_{i=0}^{\kappa-1}$ and backfill $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$. □

Lemma 4.7. *The distributions 3 and 4 are identical.*

Proof. The only difference between these distributions is how the evaluations $(f'(u_i))_{i=0}^{\kappa-1}$ are generated. In 3, they come from an encoding of two backfills: first, from $(f(u_i))_{i=0}^{\kappa-1}$ to $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1}$, which uses the prover's secret vector $(t_j)_{j=0}^{2^\ell-1}$, and next from $\left(t''_j\right)_{j=0}^{2^\ell+\kappa-1}$ to $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1}$, which proceeds by subtraction. In 4, they come instead from the backfilling rule $f'(u_i) \coloneqq f''(u_i) - \alpha \cdot f(u_i)$ in the evaluation domain. That latter backfilling doesn't use $(t_j)_{j=0}^{2^\ell-1}$, and can be carried out by the simulator. We show that these rules respectively yield identical distributions.

This turns into a calculation. To prepare for it, we introduce some notation. We write $V : L^{2^\ell+\kappa} \to L^\kappa$ for the function $(t_j)_{j=0}^{2^\ell+\kappa-1} \mapsto (P(u_i))_{i=0}^{\kappa-1}$ (we reuse the notation of Lemma 4.4 here). Similarly, we write $V_{\text{low}} : L^{2^\ell} \to L^\kappa$ for the map $(t_j)_{j=0}^{2^\ell-1} \mapsto (P_{\text{low}}(u_i))_{i=0}^{\kappa-1}$ and $V_{\text{high}} : L^\kappa \to L^\kappa$ for $\left(t_{2^\ell+j}\right)_{j=0}^{\kappa-1} \mapsto (P_{\text{high}}(u_i))_{i=0}^{\kappa-1}$. Clearly, for each $(t_{\text{low}}, t_{\text{high}})$ in $L^{2^\ell+\kappa}$:

$$V(t_{\text{low}}, t_{\text{high}}) = V_{\text{low}}(t_{\text{low}}) + V_{\text{high}}(t_{\text{high}}). \tag{3}$$

We now have the main calculation. Below, we give meaning to all symbols with reference to the distribution 3.

$$
\begin{aligned}
(f'(u_i))_{i=0}^{\kappa-1} &= V\Big((t''_{\text{low}}, t''_{\text{high}}) - \alpha \cdot \Big(t_{\text{low}}, V_{\text{high}}^{-1}\big((f(u_i))_{i=0}^{\kappa-1} - V_{\text{low}}(t_{\text{low}})\big)\Big)\Big) && \text{(this is the definition of 3.)} \\
&= V(t''_{\text{low}}, t''_{\text{high}}) - \alpha \cdot V\Big(t_{\text{low}}, V_{\text{high}}^{-1}\big((f(u_i))_{i=0}^{\kappa-1} - V_{\text{low}}(t_{\text{low}})\big)\Big) && \text{(by the linearity of } V.) \\
&= V(t''_{\text{low}}, t''_{\text{high}}) - \alpha \cdot V_{\text{low}}(t_{\text{low}}) - \alpha \cdot \big((f(u_i))_{i=0}^{\kappa-1} - V_{\text{low}}(t_{\text{low}})\big) && \text{(use (3); cancel } V_{\text{high}}^{-1} \circ V_{\text{high}}.) \\
&= V(t''_{\text{low}}, t''_{\text{high}}) - \alpha \cdot (f(u_i))_{i=0}^{\kappa-1} && \text{(cancel plus and minus } \alpha \cdot V_{\text{low}}(t_{\text{low}}).) \\
&= (f''(u_i))_{i=0}^{\kappa-1} - \alpha \cdot (f(u_i))_{i=0}^{\kappa-1}. && \text{(by definition of } f''(u_i))_{i=0}^{\kappa-1}.)
\end{aligned}
$$

This is exactly how $\mathcal{S}$ simulates $(f'(u_i))_{i=0}^{\kappa-1}$ in 4. We conclude that these distributions are identical. □

This completes the proof of the theorem. □

4.1 The BCS transform

In applications, we want to work in the random oracle model, not in the IOP model. For this reason, we must use the Ben-Sasson–Chiesa–Spooner transformation [BCS16]. Importantly, we must use that transformation's zero-knowledge variant, given in [BCS16, § 3.2]. In that variant, each "tag" in the Merkle tree is salted with a purely random string before it is hashed into a leaf. We explain this fact here.

To begin, we must first explain what zero-knowledge *means* in the random oracle model (see also Chiesa and Yogev [CY25, § 5.2]). Instead of demanding that the two distributions of Definition 3.3 be identical, we must merely ask that they be computationally indistinguishable. Moreover, we need to be careful about the random oracle. Firstly, we need to allow $\mathcal{S}$ to "program" the oracle—that is, to manually inscribe various input–output pairs into its table—during its execution. Moreover—and this is the crucial point—when the distinguisher runs on a transcript (whether from the real or ideal world), the distinguisher must *inherit* the state of the oracle. That state includes any input–output pairs sampled by the oracle itself in response to queries made by the various parties, as well as any pairs which $\mathcal{S}$ programmed. This model is called the *explicitly programmable random oracle model*. We must finally give the distinguisher D access to this oracle *and* to the honest prover's witness before it runs.

In the explicitly programmable model, the standard, *non-zero-knowledge* BCS transformation fails to bootstrap our zero-knowledge IOPCS above into a zero-knowledge PCS. Indeed, the distinguisher, given access to the prover's secret $(t_j)_{j=0}^{2^\ell-1}$ *and* to the transcript, may easily use the further evaluations $(f(u_i))_{i=0}^{\kappa-1}$ to backfill $\big(t_{2^\ell+j}\big)_{j=0}^{\kappa-1}$, exactly as in Lemma 4.4. Given the full list $(t_j)_{j=0}^{2^\ell+\kappa-1}$, the distinguisher could further independently recompute the full Merkle tree over $(t_j)_{j=0}^{2^\ell+\kappa-1}$, using the random oracle it inherited. It could finally check whether the resulting root matched that present in the transcript or not. If the standard BCS transformation were used, then the simulator $\mathcal{S}$ would have no way to simulate the Merkle root in such a way as to forestall this attack by the distinguisher (after all, $\mathcal{S}$ doesn't get $(t_j)_{j=0}^{2^\ell-1}$; this is the whole point).

Under the zero-knowledge BCS variant, this attack wouldn't work: the distinguisher would have the full list of messages $(t_j)_{j=0}^{2^\ell+\kappa-1}$, but would lack the salts used by the honest prover while it computed its Merkle tree. The distinguisher would thus have no way to independently redo what the prover (supposedly) did during its execution of the protocol. Equivalently, $\mathcal{S}$, in this setting, could get away with simulating a purely random Merkle root in the transcript, and D would be none the wiser.

This attack only works when D inherits the oracle's state. In that hypothetical, alternative framework which chose to define "zero-knowledge" in such a way as to withhold this oracle state from the distinguisher, even the *standard* BCS transform would yield a "secure" IP. That model, called the *fully programmable random oracle model*, turns out to be pathological (it too readily deems things "secure"). In that model, $\mathcal{S}$ itself simulates the existence of the random oracle internally to $\mathcal{V}$. Its very existence is a figment of $\mathcal{S}$'s simulation. In particular, not only may $\mathcal{S}$ program it arbitrarily (subject to generating an appropriate view to $\mathcal{V}$), but moreover, before D arrives, the state of that oracle becomes "wiped" or destroyed. Thus, in that model, D could again attempt to reconstruct $\mathcal{P}$'s Merkle root, as it did above, but that endeavor would tell D nothing (its oracle would be independent of $\mathcal{P}$'s). In any case, it's interesting that Construction 4.1, compiled using the *standard* BCS transform, exhibits a separation between the EPROM and the FPROM.

4.2 Efficiency

We discuss the efficiency of Construction 4.1, and compare it to [DP24, Cons. 4.11]'s. Our construction is about four times more costly than [DP24, Cons. 4.11] is. Indeed, our prover must additive-NTT the vectors $(t_j)_{j=0}^{2^\ell+\kappa-1}$ and $\left(t'_j\right)_{j=0}^{2^\ell+\kappa-1}$ onto $S^{(0)}$, which is of size $2^{\ell+\mathcal{R}+1}$. These vectors are inconveniently sized: they are just larger than 2^ℓ. By zero-padding, the prover can carry out each NTT in no worse than $(\ell+1)\cdot 2^{\ell+\mathcal{R}}$ L-multiplications, for a total of $4\cdot(\ell+1)\cdot 2^{\ell+\mathcal{R}-1}$ L-multiplications. In [DP24, Cons. 4.11], the prover must NTT just a single vector of length 2^ℓ onto a domain of size $2^{\ell+\mathcal{R}}$; this task takes $\ell\cdot 2^{\ell+\mathcal{R}-1}$ L-multiplications.

We are not sure if there is a way to exploit the fact $(t_j)_{j=0}^{2^{\ell+1}-1}$ and $\left(t'_j\right)_{j=0}^{2^{\ell+1}-1}$'s respective high halves are mostly zero (i.e., to encode them faster). We note that our protocol would still be correct and secure if the prover sampled these upper halves fully randomly; this would be less efficient though (i.e., merely to generate and write the random values). As our security proof shows, κ random values are enough.

Avenues for improvement. In one possible variant of our above encoding algorithm, the parties would instead define $S^{(0)} \coloneqq \beta_{\ell+\mathcal{R}} + \langle \beta_0, \ldots, \beta_{\ell+\mathcal{R}-1}\rangle$—i.e., as an $\ell+\mathcal{R}$-dimensional, as opposed to an $\ell+\mathcal{R}+1$-dimensional, subspace—and encode $P(X)$ on $S^{(0)}$. It is tempting to hope that the parties might carry out FRI rather in the code $\mathsf{RS}_{L,S^{(0)}}[2^{\ell+\mathcal{R}}, 2^\ell+\kappa]$, whose rate is just slightly worse than ours's. In this way, they would reduce by half the prover's cost, while paying just a small soundness penalty.

Unfortunately, this idea seems not to work, at least in any obvious way. The problem is that FRI seems not to work for non-power-of-2-sized domains. The "excess" κ coefficients in the prover's message would "leak" as the protocol proceeded, and cause the relative rates of the prover's higher-indexed codewords to saturate to 1. This fact would apparently break the soundness analysis of FRI, leave aside its implications for BaseFold. (In particular, the verifier would have to stop at least one round early, in order to retrieve the prover's folded message.)

Joseph Johnston has suggested to us an interesting idea whereby the prover might start with a *fully random* upper half $(t_j)_{j=2^\ell}^{2^{\ell+1}-1}$, and then delay its introduction of the blinding codeword f' until the FRI oracle $f^{(i)}$ becomes so small that $\kappa \approx 2^{\ell-i}$. This approach would reduce the multiplicative cost overhead of Construction 4.1 over [DP24, Cons. 4.11] from $4+o(1)$ to just $2+o(1)$. In fact, if just FRI were at stake, this approach could be made to work. Unfortunately, we have the sumcheck too to deal with. It's hard to imagine how the prover might handle the early rounds of its sumcheck—i.e., before f' gets introduced—without leaking something about $(t_j)_{j=0}^{2^\ell-1}$. The point for us is that our prover's $(t''_j)_{j=0}^{2^\ell+\kappa-1}$ is already fully random, and can be simulated. Nothing about it needs to be hidden (either in the sumcheck or its encodings).

5 Future Work

This note makes zero-knowledge *just* the large-field scheme [DP24, Cons. 4.11]; it doesn't handle ring-switching or small-field commitment. We leave for follow-up work the task of making ring-switching zero-knowledge, as well as that of designing a zero-knowledge, higher-level PIOP.

References

[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.

[BBHR18] 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.

[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.

[CY25] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2025.

[DP24] Benjamin E. Diamond and Jim Posen. *Polylogarithmic Proofs for Multilinears over Binary Towers*. Cryptology ePrint Archive, Paper 2024/504. 2024. URL: https://eprint.iacr.org/2024/504.

[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.

[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.

[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.