Revisiting the IPA-sumcheck connection

Liam Eagen[1,2] and Ariel Gabizon[2]

[1]Alpen Labs
[2]Aztec Labs

September 28, 2026

Abstract

Inner Product Arguments (IPA)[BCC+16, BBB+17] are a family of proof systems with $O(\log n)$ sized proofs, $O(n)$ time verifiers, and transparent setup. Bootle, Chiesa and Sotiraki [BCS21] observed that an IPA can be viewed as a sumcheck protocol [LFKN92] *where the summed polynomial is allowed to have coefficients in a group rather than a field.* We leverage this viewpoint to improve the performance of multi-linear polynomial commitments based on IPA. Specifically,

1. We introduce a simplified variant of Halo-style accumulation that works for multilinear evaluation claims, rather than only univariate ones as in [BGH19, BCMS20].
2. We show that the size n MSM the IPA verifier performs can be replaced by a "group variant" of BaseFold [ZCF23]. This reduces the verifier complexity from $O(n)$ to $O_\lambda(\log^2 n)$ time at the expense of an additional $4n$ scalar multiplications for the IPA prover.

1 Introduction

Inner product arguments (IPA) [BCC+16, BBB+17][1] occupy a "middle ground" between hash-based and pairing-based proof systems. They enable proofs of logarithmic (measured in field elements) size rather than constant or *poly*logarithmic in the case of pairing-based and hash-based systems respectively. In terms of viable fields they can work over any large enough prime field - large enough in the sense of getting discrete log hardness when used as a scalar field of an elliptic curve. In contrast, hash-based SNARKs can work over much smaller prime fields and even small binary fields [HLP24, DP24]. On the other side, pairing-based SNARKs only work over special primes for which we

[1]We follow the current informal convention that the term IPA refers to inner product arguments based on discrete log hardness, although there are obviously other protocols for proving correctness of inner products.

have a pairing-friendly curve, and additionally, often this special prime is taken to be significantly larger than what is needed for IPA, to provide better protection against number-field sieve attacks.

Even when the main proof system is pairing-based, there is often a need to work with IPA to make certain recursive verifier operations "native" and thus more efficient using a "half-pairing curve cycle"[Wil23, KTW24].

The main hurdle for practical use of IPA is the *linear time in circuit/polynomial degree verification cost*. Halo [BGH19] gave an elegant solution to this hurdle, later formalized in [BCMS20]. [BGH19] observed that the IPA verification can be split into two parts such that

- The first part only requires *logarithmic* verifier work.
- The second part can be done *once for multiple instances* following an accumulation step also requiring only logarithmic verifier work.

This made using IPA much more practical. However, the linear verification work can be a bottleneck to do even once. For example, in instances where this verification is done in a recursive circuit as in the Aztec system [Wil23, KTW24]. Furthermore, [BGH19, BCMS20] presented their accumulation scheme only for claims about *univariate* polynomials, whereas systems based on multilinear polynomials are gaining more and more traction due to improved prover performance [Set19, CBBZ22, ZCF23, HJR$^+$25]. In this paper, we make two improvements on the state of IPA:

- We slightly simplify the accumulation step, and more importantly, show that it works for claims about mutlilinear evaluations.
- We present an alternative final deciding verifier that requires only polylogarithmic field and group work. Interestingly, we leverage techniques from the hash-based SNARK world! Notably, FRI and BaseFold [BBHR18, BSCI$^+$20, ZCF23].

Informally, our result is

Theorem 1.1 (Informal, see Theorem 4.4)**.** *Under DL, there is a* $\log n$*-variate multilinear polynomial commitment scheme where*

1. *Reducing and accumulating an evaluation claim requires* $O(n)$ *prover and* $O(\log n)$ *verifier scalar multiplications.*
2. *Deciding all claims requires* $4n$ *prover and* $O(\lambda \log n)$ *verifier scalar multiplications, and* $O(\lambda \log^2 n)$ *verifier hashes.*

Remark 1.2. *The deciding verifier complexity in Theorem 1.1 can always be achieved via recursion - proving the* n*-sized MSM the IPA verifier must check in a different proof system with a succinct verifier. But such a method would require both asymptotically and concretely higher deciding* prover *cost: We would need to prove a statement about a circuit doing an* n*-sized MSM. Using Pippenger this requires about* $8\lambda n/\log n$ *arithmetic*

gates, which is already asymptotically larger than n as $\lambda = \omega(\log n)$. And now this would be multiplied by a constant approximating how much scalar multiplications the prover does for a gate in the chosen proof system - e.g. 9 in [GWC19] giving $72\lambda n/\log n$, compared to our $4n$.

1.1 IPA as sumcheck

Our improvements rely on a viewpoint suggested by Bootle, Chiesa and Sotiraki[BCS21]. [BCS21] suggests applying the sumcheck protocol[LFKN92] for polynomials defined over groups or rings[2] instead of over a field. In particular, they showed that the IPA can be viewed as a sumcheck over a polynomial whose coefficients are in a discrete-log hard group $\mathbb{G}$.[3]

The main point we leverage here is as follows. When viewed as a sumcheck, the "heavy" linear-time operation of the IPA verifier is evaluating a polynomial G at some $r \in \mathbb{F}^k$ - where G is a pre-determined multilinear polynomial with values in $\mathbb{G}$. Specifically, G is the polynomial whose values on the boolean cube are the n randomly chosen generators used for the IPA.

Under this viewpoint, accumulation becomes easy. It is well-established that via sumcheck we can reduce multiple claims about evaluations of a multilinear G into one such claim. Usually this is used for *field-valued* polynomials; but in the spirit of [BCS21], there is no reason not to use the same method for a group-valued polynomial.

As for avoiding linear verifier work for the final decision, we look towards the hash-based SNARK world. Specifically, the recent BaseFold protocol[ZCF23] leverages FRI-style proximity testing[BBHR18, BSCI$^+$20] to prove multilinear evaluations. FRI and BaseFold rely solely on the properties of Reed-Solomon as an error-correcting code. We observe that we can look at the "group-based" Reed-Solomon code, defined by the same generator matrix, but now viewed as a function mapping $f \in \mathbb{G}^n$ to a longer vector over $\mathbb{G}$. Given this code over $\mathbb{G}$, we can use a "group version" of BaseFold to prove the evaluation $G(r)$ needed for final decision.

One may wonder for a moment - why not simply use BaseFold as your multilinear PCS and forget about all this group stuff? For proving a single evaluation that would probably be best. The advantage here is that, in a situation where we are verifying many evaluations from many parties, we pay the polylogarithmic overhead of BaseFold rather than logarithmic overhead of Halo-style IPA only once for all evaluations. While this is a verifier overhead it usually translates to a *prover* overhead when all these claims are verified recursively in-circuit as part of making a single proof for all of them. Additionally, the prover never needs to perform a quasilinear time Reed-Solomon encoding, as we are using BaseFold only for a *preprocessed* polynomial.

[2] More accurately, in their terminology - over bi-linear modules.

[3] We stress that this use of sumcheck is not totally straightforward, as it is needed to show that the "field part" of the final verifier evaluation can be taken as untrusted advice from the prover without breaking soundness (as all IPA arguments do). For details see the knowledge soundness proof in Section 5.

Overview of the paper

We go over preliminaries in Section 2. In Section 3 we discuss how components like polynomials and codes are used over a group rather than field (which for the most is quite straightforward). In Section 4 we give our definition of a multilinear PCS that supports accumulation. In Sections 5-7 we construct the components of the ml-PCS.

2 Preliminaries

2.1 Notation and conventions

We use throughout the paper positive integer parameters k and $n = 2^k$. We think of k as the number of variables of a multilinear function. We use bold $\mathbf{X}$ to denote a k-tuple of variables $X_1, \ldots, X_k$. Thus, for example, $A(\mathbf{X}) = A(X_1, \ldots, X_k)$ is a k-variate function.

For brevity, when we use the term "$f(X)$ has degree d" we mean $f(X)$ has degree *at most* d. We denote by $\mathscr{B}_s$ the set of binary strings of length s. We use the misnomer "order two subgroup" to refer to a subgroup of order 2^t for some integer $t > 0$. By e.w.p. p we mean with probability at least $1 - p$.

Fields and Groups We assume our field $\mathbb{F}$ is of prime order. We denote by $\mathbb{F}_{<d}[X]$ the set of univariate polynomials over $\mathbb{F}$ of degree smaller than d. We assume all algorithms described receive as an implicit parameter the security parameter λ.

Whenever we use the term *efficient*, we mean an algorithm running in time $\mathsf{poly}(\lambda)$. Furthermore, we assume an *object generator* $\mathcal{O}$ that is run with input λ before all protocols, and returns some parameters and all fields and groups used. Specifically, in our protocols $\mathcal{O}(\lambda) = (k, n, \mathbb{F}, \mathbb{G})$ where

- k, n are positive integers with $n = 2^k = \mathsf{poly}(\lambda)$.
- $\mathbb{F}$ is a prime field of super-polynomial size $p = \lambda^{\omega(1)}$.
- $\mathbb{G}$ is a group of size p.

We usually let the λ parameter be implicit, i.e. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. We write $\mathbb{G}$ additively.

Multilinear polynomials We define the well-known **eq** multilinear polynomial in $2k$ variables.

$$\mathbf{eq}(x, y) := \prod_{i=1}^{k} (x_i y_i + (1 - x_i)(1 - y_i))$$

We have for $x, y \in \mathscr{B}_k$, $\mathbf{eq}(x, y) = 1$ when $x = y$ and $\mathbf{eq}(x, y) = 0$ otherwise.

We use the convention that an integer $0 \leq i < n$ can be used as an input to **eq** by interpreting i as its binary representation. Namely, for $0 \leq i < n$, $u \in \mathbb{F}^k$, $\mathbf{eq}(i, u) := \mathbf{eq}(i_0, \ldots, i_{k-1}, u)$ where $i = \sum_{j<k} i_j 2^j$.

For $f = (f_0, \ldots, f_{n-1}) \in \mathbb{F}^n$, we define $\hat{f}$ to be the multilinear polynomial obtaining f's values on the boolean cube. Namely,

$$\hat{f}(X_1, \ldots, X_k) := \sum_{i<n} \mathbf{eq}(i, X_1, \ldots, X_k) \cdot f_{i+1}.$$

The Discrete log assumption We state the discrete log assumption in a multi-generator form that will be convenient. It is standard that this follows for the regular discrete-log assumption

Definition 2.1. *[DLA assumption for $\mathbb{G}$] For any efficient $\mathcal{A}$ the probability of winning the following game is* $\mathsf{negl}(\lambda)$.

1. *Uniformly chosen $G_1, \ldots, G_n \in \mathbb{G}$ are sent to $\mathcal{A}$.*
2. *$\mathcal{A}$ outputs a vector $f \in \mathbb{F}^n$.*
3. *$\mathcal{A}$ wins if and only if $f \neq 0^n$, and $\langle f, (G_1, \ldots, G_n) \rangle = 0$.*

2.2 IOPs and round-by-round soundness

We review some necessary definitions and results regarding *Interactive Oracle Proofs*(IOPs) [BCS16]. The definitions and results here are based on Section 3 of [BGK+23] which in turn are based on [BCS16, COS19].

A μ-round *Interactive Oracle Proof* (IOP) is a public coin interactive protocol between prover $\mathbf{P}$ and verifier $\mathbf{V}$ where the prover sends the first and last message. Thus, the transcript looks like $m_1, r_1, \ldots, m_\mu, r_\mu, m_{\mu+1}$ where the m_j are the messages of $\mathbf{P}$ and the $\{r_j\}$ are the random coins $\mathbf{V}$. We think of $\mathbf{P}$'s messages as *oracles* meaning $\mathbf{V}$ can query only certain indices of a message $\{m_j\}$ (but $\mathbf{P}$ cannot change the values of m_j at those indices after seeing the queries). At the end of the protocol $\mathbf{V}$ outputs either reject or accept.

We use the notion of *round-by-round soundness* of an IOP. Below by a *transcript* of an IOP Π we mean a sequence also containing the protocol input in addition to protocol messages.

Definition 2.2. *An IOP Π for language $\mathcal{L}$ has* round-by-round soundness error ε *if there exists a set $\mathcal{D}$ called the* doomed set *of partial and complete transcripts of Π such that*

1. *"Non-inputs are doomed": If $\phi \notin \mathcal{L}$ then $\mathfrak{t} = (\phi) \in \mathcal{D}$.*
2. *"Doomed transcripts are rejected": For a full transcript $\mathfrak{t}$, if $\mathfrak{t} \in \mathcal{D}$ then $\mathbf{V}(\mathfrak{t}) = \mathit{reject}$.*
3. *"Hard to get out of the doomed set": For a partial transcript $\mathfrak{t} = (\phi, m_1, r_1, \ldots, m_i, r_i) \in \mathcal{D}$, for any m_{i+1}, the probability over r_{i+1} that $(\mathfrak{t}, m_{i+1}, r_{i+1}) \notin \mathcal{D}$ is at most ε.*

Compiling to an argument in the random oracle model Given an IOP Π, and a random oracle $\mathcal{R} : \{0,1\}^* \to \{0,1\}^\lambda$, we define the non-interactive argument $\Pi^{\mathcal{R}}$ where each message m_i of $\mathbf{P}$ is replaced by a Merkle hash of m_i computed with $\mathcal{R}$; and each message r_i of $\mathbf{V}$ is replaced by an application of $\mathcal{R}$ on the transcript so far using the Fiat-Shamir transform. (See Section 6 of [BCS16] for exact details of the transformation.)

The following is a special case of Theorem 7.1 in [BCS16], but in the terminology of Theorem 3.15 in [BGK+23].

Theorem 2.3. *Suppose Π is an IOP for language $\mathcal{L}$ with $\varepsilon = \mathsf{negl}(\lambda)$ round-by-round soundness error, verifier query bound q, d rounds, and all oracles m_i sent have length at most n. Then $\Pi^{\mathcal{R}}$ is a non-interactive argument for $\mathcal{L}$ in the ROM with $\mathsf{negl}(\lambda)$ soundness error and proof length $O(\lambda(q \log n + d))$.*

3 Components for working over $\mathbb{G}$

We review several components that are usually used over a field $\mathbb{F}$, and explain how they are used over $\mathbb{G}$ instead. We crucially use that $|\mathbb{G}| = |\mathbb{F}| = p$ is prime. Thus, $\mathbb{G}$ is a cyclic group, and ab is defined for $a \in \mathbb{F}, b \in \mathbb{G}$ via scalar multiplication. In particular, for any $b \in \mathbb{G} \setminus \{0\}$, $\mathbb{G} = \{ab\}_{a \in \mathbb{F}}$.

Polynomials over $\mathbb{G}$ $\mathbb{G}[X]$ consists of elements of the form $G(X) = \sum_{i<n} a_i X^i$ for integer $n \geq 0$ and $a_i \in \mathbb{G}$. Such G can be *evaluated* at $x \in \mathbb{F}$ -

$$G(x) := \sum_{i<n} a_i x^i \in \mathbb{G},$$

where $a_i x^i$ is defined as scalar multiplication of $a_i \in \mathbb{G}$ by $x^i \in \mathbb{F}$.

It's easy to check that non-zero $G(X) \in \mathbb{G}[X]$ of degree at most d evaluates to zero on at most d $x \in \mathbb{F}$. In abstract algebra terms, $\mathbb{G}[X]$ is a *module* over $\mathbb{F}[X]$. This concretely means for us that

1. $f, g \in \mathbb{G}[X]$ can be *added:*

$$\sum_{i<n} a_i X^i + \sum_{i<n} b_i X^i := \sum_{i<n} (a_i + b_i) X^i$$

2. More interestingly, $f \in \mathbb{F}[X]$ and $g \in \mathbb{G}[X]$ can be *multiplied* to get

$$\left(\sum_{i<n_1} a_i X^i\right) \cdot \left(\sum_{i<n_2} b_i X^i\right) := \sum_{0 \leq \ell < n_1+n_2} \left(\sum_{i+j=\ell} a_i b_j\right) X^\ell \in \mathbb{G}[X]$$

Sumcheck over $\mathbb{G}$ We review the classic sumcheck protocol [LFKN92], highlighting it works fine for polynomials over $\mathbb{G}$.

We have a prover $\mathbf{P}$ and a verifier $\mathbf{V}$. Both have integer parameter d and target value $C \in \mathbb{G}$. $\mathbf{P}$ has a multivariate $A(X_1, \ldots, X_k) \in \mathbb{G}[X_1, \ldots, X_k]$ of individual degree d.

The sumcheck protocol between $\mathbf{P}$ and $\mathbf{V}$ proceeds as follows.

1. For $i = 1, \ldots, k$:
   (a) $\mathbf{P}$ computes and sends the univariate polynomial
   $$A_i(X) := \sum_{b \in \mathscr{B}_{k-i}} A(r_1, \ldots, r_{i-1}, X, b) \in \mathbb{G}[X].$$
   (b) $\mathbf{V}$ sends a random $r_i \in \mathbb{F}$.
2. $\mathbf{V}$ checks that
   (a) All polynomials sent have degree at most d.
   (b) $A_1(0) + A_1(1) = C$.
   (c) For $i \in [k-1]$, $A_i(r_i) = A_{i+1}(0) + A_{i+1}(1)$.
3. If one of the above checks failed, $\mathbf{V}$ outputs $\mathscr{reject}$. Otherwise, let $r = (r_1, \ldots, r_k)$. $\mathbf{V}$ computes $v = A_k(r_k)$ and outputs the pair $(r, v) \in \mathbb{F}^k \times \mathbb{G}$.

The main properties of the protocol are captured in the following lemma.

Lemma 3.1. *Fix any $A(X_1, \ldots, X_k) \in \mathbb{G}[X_1, \ldots, X_k]$ with individual degrees at most d and $C \in \mathbb{G}$.*

- ***Completeness:*** *Suppose $\sum_{b \in \mathscr{B}_k} A(b) = C$ and $\mathbf{P}$ correctly follows the protocol. Then, with probability one $\mathbf{V}$ outputs (r, v) with $A(r) = v$.*
- ***Soundness:*** *Suppose $\sum_{b \in \mathscr{B}_k} A(b) \neq C$. Then, for any prover $\mathbf{P}^*$ participating in the protocol, $\mathbf{V}$ outputs either $\mathscr{reject}$ or (r, v) with $A(r) \neq v$ e.w.p. $\varepsilon = \frac{dk}{p}$.*

Codes over $\mathbb{G}$ Note that $\mathbb{G}^n$ is an $\mathbb{F}$-vector subspace via coordinate-wise scalar multiplication and addition. We can define codes over $\mathbb{G}$ analagously to their definition over a field: An $[n, \hbar, \delta]$-linear code over $\mathbb{G}$ is an $\mathbb{F}$-linear subspace $V \subset \mathbb{G}^n$ of dimension $\hbar$ such that for all $u \neq v \in V$, $\Delta(u, v) \geq \delta/n$ where Δ denotes fractional Hamming distance - i.e., the fraction of coordinates $i \in [n]$ with $u_i \neq v_i$.

Let A be the $n \times k$ generating matrix of an $[n, \hbar, \delta]$-linear code over $\mathbb{F}$. That is, $V_{\mathbb{F}} := \{A \cdot m\}_{m \in \mathbb{F}^{\hbar}}$ is an $[n, \hbar, \delta]$-code over $\mathbb{F}$. Then, if we look at $V := \{A \cdot m\}_{m \in \mathbb{G}^{\hbar}}$ this is an $[n, \hbar, \delta]$-code over $\mathbb{G}$.

In particular, we will use A corresponding to a rate $1/2$ $[2\hbar, \hbar, \hbar]$ Reed-Solomon code. Namely, given $m \in \mathbb{F}^{\hbar}$, $v = A \cdot m$ is the sequence of evaluations of $m(X)$ on a set D

(to be specified in Appendix A) of size $2\hbar$, where $m(X) := \sum_{i<\hbar} m_i X^i$. As explained, we can also apply A on $m \in \mathbb{G}^\hbar$. If we have an algorithm for computing $A \cdot m$ on $m \in \mathbb{F}^\hbar$ consisting only of products between *constants* from $\mathbb{F}$ and elements of $\mathbb{G}$, and additions, we can explain the same algorithm to compute $A \cdot m$ for $m \in \mathbb{G}^\hbar$. The number of $\mathbb{G}$-additions and $\mathbb{G}$-scalar multiplications in the modified algorithm for $m \in \mathbb{G}^\hbar$ will correspond to the number of $\mathbb{F}$-additions and $\mathbb{F}$-multiplications in the original algorithm for $m \in \mathbb{F}^\hbar$.

4 ml-PCS with accumulation

In this section we give a non-standard definition of a multilinear polynomial commitment scheme to facilitate our integration of the PCS with an accumulation scheme. Usually a PCS contains an open or eval procedure to check an evaluation where $\mathbf{V}$ accepts or rejects. Here instead we have a reduce procedure where $\mathbf{V}$ outputs ϕ that shoud be in a known language $\mathcal{L}$ if and only if the evaluation claim is correct. The ml-PCS additionally has an $\mathsf{accumulate}$ procedure to combine several such instances into one in a sound way - meaning that if one of the instances is not in $\mathcal{L}$ w.h.p. neither will the instance output by $\mathsf{accumulate}$ be. Finally, there is a decide protocol to determine if $\phi \in \mathcal{L}$. As in Halo[BGH19], the idea is that the reduce and $\mathsf{accumulate}$ protocols can be much cheaper than decide, and so we benefit from invoking decide only once for all evaluation claims.

We start with a definition of an accumulation scheme for a language.

Definition 4.1. *An accumulation scheme for a language $\mathcal{L}$ is a protocol between a prover $\mathbf{P}$ and verifier $\mathbf{V}$ that are both given parameter t, and input instances $\phi_1, \ldots, \phi_t$. At the end of the protocol $\mathbf{V}$ either outputs reject or an output instance ϕ such that*

1. ***Completeness:*** *If for all $i \in [t]$, $\phi_i \in \mathcal{L}$, and $\mathbf{P}$ follows the protocol, then with probability one $\mathbf{V}$ outputs $\phi \in \mathcal{L}$.*
2. ***Soundness:*** *If for some $i \in [t]$, $\phi_i \notin \mathcal{L}$, then for any prover $\mathbf{P}^*$, e.w.p. $\mathsf{negl}(\lambda)$ $\mathbf{V}$ either outputs reject or $\phi \notin \mathcal{L}$.*

Remark 4.2. *Our definition of accumulation schemes is simpler than [BCMS20] as it suffices to capture our construction. In particular, as opposed to [BCMS20]*

- *There is no distinction between accumulators and instances, as both have the same form and are supposed to be elements of $\mathcal{L}$.*
- *There is no explicit description of a decider, as again, the decision critera is being in the predetermined language $\mathcal{L}$.*

We move on to our definition of a PCS.

Definition 4.3 (ml-PCS with accumulation)**.** *Let $n = 2^k$. A k-variate multilinear polynomial commitment scheme with accumulation (ml-PCS) is a tuple*

$$\mathcal{P} = (\mathsf{gen}, \mathsf{com}, \mathsf{reduce}, \mathsf{accumulate}, \mathsf{decide})$$

such that

- $\mathsf{gen}(n)$ - *is a randomized algorithm that outputs an SRS* srs*, and an index* i *representing a choice of language* $\mathcal{L} = \mathcal{L}_i$*.*
- $\mathsf{com}(f, \mathsf{srs})$ *is a function that given a vector* $f \in \mathbb{F}^n$ *returns a commitment* cm *to* f*.*
- $\mathsf{reduce}(\mathsf{cm}, u, v; f)$ *is a public-coin protocol between parties* $\mathbf{P}$ *and* $\mathbf{V}$*.* $\mathbf{P}$ *is given* $f \in \mathbb{F}^n$*.* $\mathbf{P}$ *and* $\mathbf{V}$ *are both given* cm*- the purported commitment to* f*,* $u \in \mathbb{F}^k$ *and* $v \in \mathbb{F}$ *- the purported value* $\hat{f}(u)$*.*
- $\mathsf{accumulate}_i$ *is an accumulation scheme for* $\mathcal{L}_i$*.*
- decide_i *is an IOP for* $\mathcal{L}_i$ *with round-by-round soundness error* $\mathsf{negl}(\lambda)$ *(see Definition 2.2).*

such that

- ***Completeness:*** *Suppose that* $\mathsf{cm} = \mathsf{com}(f, \mathsf{srs})$*, and* $\mathcal{L} = \mathcal{L}_i$ *where* $(\mathsf{srs}, i) = \mathsf{gen}(n)$*. Then if* reduce *is run correctly with values* $\mathsf{cm}, u, v = \hat{f}(u)$*,* $\mathbf{V}$ *outputs* $\phi \in \mathcal{L}$ *with probability one.*
- ***Knowledge soundness:*** *There exists an expected* $\mathsf{poly}(\lambda)$*-time* $\mathcal{E}$ *such that for every efficient adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *the following holds.*
  Define ε_1 *to be the probability of the following event.*
  1. *Given* srs, i $\mathcal{A}_1$ *outputs* $n, \mathsf{cm}, u \in \mathbb{F}^k, v \in \mathbb{F}, \mathsf{state}$*.*
  2. $\mathcal{A}_2$*, given the above outputs takes the part of* $\mathbf{P}$ *in the protocol* reduce *with inputs* cm, u, v*.*
  3. *At the end of* reduce*,* $\mathbf{V}$ *outputs* $\phi \in \mathcal{L}_i$*.*

  Define ε_2 *to be the probability of the following event.*
  1. *Given* srs, i $\mathcal{A}_1$ *outputs* $\mathsf{cm}, u \in \mathbb{F}^k, v \in \mathbb{F}, \mathsf{state}$*.*
  2. $\mathcal{E}$ *given the above inputs and rewinding access to* $\mathcal{A}_2(\mathsf{state})$*, outputs* $f \in \mathbb{F}^n$*.*
  3. *We have* $\hat{f}(u) = v$*.*

  Then we have
  $$\varepsilon_2 \geq \varepsilon_1 - \mathsf{negl}(\lambda).$$

Under this terminology our main result is

Theorem 4.4. *Assuming DLA holds for* $\mathbb{G}$*, we can construct a* k*-variate ml-PCS* $\mathcal{P} = (\mathsf{gen}, \mathsf{com}, \mathsf{reduce}, \mathsf{accumulate}, \mathsf{decide})$ *such that*

1. *In* reduce*,* $\mathbf{V}$ *does* $2k$ *scalar multiplications, and* $\mathbf{P}$*'s communication consists of* $3k$ $\mathbb{G}$*-elements and one* $\mathbb{F}$*-element.*

2. *In* accumulate *with parameter* t, $\mathbf{V}$ *does* $t + 2k$ *scalar multiplications and* $O(tk)$ *field multiplications, and the proof consists of* $3k$ $\mathbb{G}$*-elements and one* $\mathbb{F}$*-element.*

3. *In the compiled non-interactive argument from* decide *(See Theorem 2.3)* $\mathbf{V}$ *does* $O(\lambda k)$ *scalar multiplications and* $O(\lambda k^2)$ *RO invocations; and the proof length is* $O(\lambda k^2)$ *elements of length* λ*, or* $O(\lambda^2 k^2)$ *bits.*

4. *In all protocols* $\mathbf{P}$ *does* $O(n)$ *scalar multiplications and* $O(n)$ $\mathbb{F}$*-operations.*

The rest of the paper's sections construct the elements of $\mathscr{P}$.

5 gen,com and reduce

We describe the first three components of $\mathscr{P}$ based on IPA and sumcheck [BCC$^+$16, BBB$^+$17, BCS21]. For $G \in \mathbb{F}^n$, we denote by $\mathcal{L}_G$ the set of pairs (r, V) such that $\hat{G}(r) = V$.

- $\mathsf{gen}(n)$ - Chooose random non-zero $G_0, \ldots, G_{n-1}, P \in \mathbb{G}$. Let $G := (G_0, \ldots, G_{n-1})$. Set $\mathsf{srs} = (G, P)$, $\mathit{i} = G$.

  Let g_0 be the encoding in RS_0 of G to be described in Section 7.1.(g_0 will only be used in Section 7.) Compute and set $\mathsf{srs} = (G, g_0)$

  Output $(\mathsf{srs}, \mathit{i})$.

- $\mathsf{com}(f, \mathsf{srs})$ - given $f \in \mathbb{F}^n$, output $\sum_{i<n} f_i G_i$.

- $\mathsf{reduce}(\mathsf{cm}, z, v; f)$. Below we identify G with its multilinear extension $\hat{G}$.

  1. $\mathbf{V}$ sends random $\alpha \in \mathbb{F}$.
  2. $\mathbf{P}$ and $\mathbf{V}$ compute $P' := \alpha P$.
  3. $\mathbf{P}$ and $\mathbf{V}$ run the sumcheck protocol as described in Section 3. They use polynomial
  $$A(\mathbf{X}) := \hat{f}(\mathbf{X})G(\mathbf{X}) + \mathbf{eq}(\mathbf{X}, z)\hat{f}(\mathbf{X})P',$$
  target value $C := \mathsf{cm} + vP'$, and individual degree bound $d = 2$.
  4. $\mathbf{V}$ outputs reject if it did in the sumcheck. Otherwise, let $(r, V) \in \mathbb{F}^k \times \mathbb{G}$ be $\mathbf{V}$'s output in the sumcheck.
  5. $\mathbf{P}$ sends $a := \hat{f}(r)$.
  6. $\mathbf{V}$ computes $b = \mathbf{eq}(r, z)$
  7. Note that checking $aG(r) + baP' = V$ is equivalent to checking $G(r) = (V - baP')/a$. Motivated by this, $\mathbf{V}$ outputs $(r, (V - baP')/a)$.

Theorem 5.1. *Under DLA (Assumption 2.1) for* $\mathbb{G}$, reduce *is knowledge sound.*

This was essentially shown in [BCS21] in a more general setting of so-called "sumcheck-friendly commitments". For dydactic purposes, we provide a similar self-contained proof in our simpler setting of Pedersen commitments over a prime order group.

Proof. During the proof, for brevity we use the term *collision* to mean a non-zero vector $f \in \mathbb{F}^{n+1}$ with $\langle f, (\mathsf{g}, P)\rangle = 0$. We describe the extractor $\mathcal{E}$. Define open to be the protocol consisting of running reduce to get output (C, r), with the additional step of $\mathbf{V}$ outputting accept iff $(C, r) \in \mathcal{L}_G$. Fix an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ as in the knowledge soundness definition in Definition 4.3. Let ε_1 be the probability defined there. That is, in terms of the protocol open, the probability that $\mathcal{A}_1$ outputs $\mathsf{cm}, u, v, \mathsf{state}$ and then $\mathcal{A}_2(\mathsf{state})$ causes $\mathbf{V}$ to accept in open with inputs cm, u, v.

As in [BCS21], $\mathcal{E}$ will invoke the algorithm tree, described in Lemma 5 of [ACK21]. tree runs in expected $\mathsf{poly}(\lambda)$ time and is guaranteed to return a tree of accepting transcripts with probability at least $\varepsilon_1 - 4^k/p = \varepsilon_1 - \mathsf{negl}(\lambda)$. (In a tree of transcripts as in Definition 9 of [ACK21] nodes correspond to prover messages and edges to verifier messages; so in our protocol the root corresponds to the empty message.) Given such a transcript tree, the following lemma shows an efficient procedure that either outputs $f \in \mathbb{F}^n$ with $\mathsf{com}(f) = \mathsf{cm}$ and $\hat{f}(u) = v$ or a collision. Under the DLA the latter option happens with probability $\mathsf{negl}(\lambda)$. In summary, if we define $\mathcal{E}$ to return the output of the procedure, we get $\varepsilon_2 \geq \varepsilon_1 - \mathsf{negl}(\lambda)$ as required for knowledge soundness. We proceed to formally state and prove the lemma.

Lemma 5.2. *Fix any input $I = (\mathsf{cm}, u, v)$ for open. There is an efficient procedure that given a 4-ary tree of accepting transcipts for open with input I returns with probability one either a collision or $a \in \mathbb{F}^n$ such that $\mathsf{com}(a) = \mathsf{cm}$ and $\hat{a}(u) = v$.*

Proof. Fix any (cm, u, v). Denote $t(\mathbf{X}) := \mathbf{eq}(\mathbf{X}, u)$. Suppose we are given a tree of accepting transcripts for $\mathsf{open}(\mathsf{cm}, z, v)$ of arity four. We introduce some notation to describe the contents of this tree. In fact, for now we fix a value $\alpha \in \mathbb{F}$ for the first verifier message, and look at the subtree of transcripts with this first message; we also use the notation $P' := \alpha P$. The root of the subtree contains the first sumcheck polynomial, which we denote $F_\emptyset(X)$ - as the path from the root to itself is empty. From the root we have four edges labeled by four distinct choices for $\mathbf{V}$'s randomness, say $s_1, \ldots, s_4 \in \mathbb{F}$. For $j \in [4]$, the j'th edge leads to a child node labeled by the second polynomial sent by $\mathbf{P}$ in response to s_j. We denote the polynomial by $F_{s_j}(X)$. Since the transcripts are accepting we have that for each $j \in [4]$, $F_\emptyset(s_j) = F_{s_j}(0) + F_{s_j}(1)$.

In general, for $1 \leq i < k$ a node at distance i from the root will be labeled by a polynomial F_r where $r = (r_1, \ldots, r_i) \in \mathbb{F}^i$ is the sequence of edge labes leading to the node - equivalenty, the random choices of $\mathbf{V}$ in that transcript up to that point. We will always have $F_{(r_1,\ldots,r_{i-1})}(r_i) = F_r(0) + F_r(1)$. Here $F_{(r_1,\ldots,r_{i-1})}(X)$ is the label of the node's father.

A path $r \in \mathbb{F}^k$ of length k will lead to a leaf labeled by constant $a_r \in \mathbb{F}$ such that (since the transcript is accepting):

$$F_{(r_1,\ldots,r_{k-1})}(r_k) = a_r \cdot G(r) + a_r t(r) P'.$$

For a path $r \in \mathbb{F}^i$ for $0 \leq i < k$, we say a mutlilinear function $a(X_{i+1}, \ldots, X_k)$ *satisfies* r if

$$F_r(0) + F_r(1) = \sum_{b \in \mathscr{B}_{k-i}} a(b)G(r,b) + a(b)t(r,b)P'.$$

Going from the leaves to the root, we'll efficiently construct a satisfying multilinear for all paths r of length $< k$. Afterwards, we'll show how to obtain $a \in \mathbb{F}^n$ as in the lemma from these satisfying multilinears.

This suffices to either find a collision or means that $a(z) = v$ and $\mathsf{com}(a) = \mathsf{cm}$.

We begin with paths of length $k-1$

Take a path $r \in \mathbb{F}^{k-1}$, with extensions $(r, s_1), \ldots, (r, s_4)$.

The nodes reached by these paths have labels $a_{r,s_1}, \ldots, a_{r,s_4} \in \mathbb{F}$. Interpolate $a(X) \in \mathbb{F}[X]$ of degree three with with $a(s_j) = a_{r,s_j}$ for $j \in [4]$. We have for $j \in [4]$

$$F_r(s_j) = a(s_j)G(r,s_j) + a(s_j)t(r,s_j)P'.$$

Define $H_0 := G(r,0), H_1 := G(r,1)$. Note that if either is zero we have found a collision. For example, $H_0 = \sum_{b \in \mathscr{B}_{k-1}} \mathbf{eq}(b,r)G_{b,0}$ so $f = (\mathbf{eq}(b,r))_{b \in \mathscr{B}_{k-1}}$ gives us a non-trivial combination of the $\{G_i\}$. In such a case we return the collision f. Otherwise, we have for each $j \in [4]$

$$F_r(s_j) = a(s_j)(1-s_j)H_0 + a(s_j)s_jH_1 + a(s_j)t(r,s_j)P'. \tag{1}$$

Interpolate degree two polynomials $b_0, b_1, b_2 \in \mathbb{F}[X]$ such that for each $j \in [3]$

$$b_0(s_j) = a(s_j)(1-s_j), b_1(s_j) = a(s_j)s_j, b_2(s_j) = a(s_j)t(r,s_j).$$

Thus, for each $j \in [3]$

$$F_r(s_j) = b_0(s_j)H_0 + b_1(s_j)H_1 + b_2(s_j)P'.$$

As the LHS and RHS are both degree two polynomials evaluated at s_j, we have as a polynomial identity

$$F_r(X) = b_0(X)H_0 + b_1(X)H_1 + b_2(X)P'.$$

Therefore, also for $j = 4$ we have

$$F_r(s_4) = b_0(s_4)H_0 + b_1(s_4)H_1 + b_2(s_4)P'$$

$$= a(s_4)(1-s_4)H_0 + a(s_4)s_4H_1 + a(s_4)t(r,s_4)P'. \tag{2}$$

Now look at two cases. The first is that one of the following three equalities doesn't hold:

1. $b_0(s_4) = a(s_4)(1-s_4)$.

2. $b_1(s_4) = a(s_4)s_4$.

3. $b_2(s_4) = a(s_4)t(r, s_4)$.

In such a case, we subtract the RHS from the LHS of Equation 2. The result is an equality $c_0H_0 + c_1H_1 + c_2P' = 0$ with c_0, c_1, c_2 not all zero. Again, writing out H_0 and H_1 in terms of the $\{G_i\}$ this translates to a collision we can output.

Otherwise, we have for all $j \in [4]$ that $b_0(s_j) + b_1(s_j) = a(s_j)$.

Thus $b_0(X) + b_1(X) = a(X)$ as a polynomial, and so $a(X)$ has degree at most two. Furthermore, since for $j \in [4]$,

$$b_0(s_j) = a(s_j)(1 - s_j),$$

and both sides of the equation have degree at most three, we have

$$b_0(X) = a(X)(1 - X)$$

as a polynomial identity. Hence, $a(X)$ has degree at most one.

Now, from equation 1 holding for all $j \in [4]$ we have the polynomial identity

$$F_r(X) = a(X)(1 - X)H_0 + a(X)XH_1 + a(X)t(r, X)P'$$

Unrolling the definition of H_0, H_1 and using this identity at $b \in \{0, 1\}$ we have

$$F_r(0) + F_r(1) = \sum_{b\in\{0,1\}} a(b)G(r, b) + a(b)t(r, b)P'.$$

This implies that a satisfies r.

Now fix $0 \le i < k-1$ and a path $r \in \mathbb{F}^i$. Denote the extensions of r by $(r, s_1), \ldots, (r, s_4) \in \mathbb{F}^{i+1}$. Suppose that for each $j \in [k]$ we have a satisfying $a_j \in \mathbb{F}[X_{i+2}, \ldots, X_k]$ for (r, s_j). We construct a satisfying $a \in \mathbb{F}[X_{i+1}, \ldots, X_k]$ for r. The construction is almost identical to the previous one for $r \in \mathbb{F}^{k-1}$ if we work with polynomials over $K := \mathbb{F}[X_{i+2}, \ldots, X_k]$ rather than over $\mathbb{F}$, and define the product between $f, g \in K$ to be the inner product over values on the boolean cube. Namely,

$$f \cdot g := \sum_{b\in\mathscr{B}_{n-(i+1)}} f(b)g(b).$$

Now note that for $a(X_1, \ldots, X_k)$ satisfying the empty path we have

$$\sum_{b\in\mathscr{B}_k} (a(b)G(b) + a(b)t(b)\alpha P) = F_\emptyset(0) + F_\emptyset(1) = \mathsf{cm} + v\alpha P.$$

We do this procedure for two subtrees with different initial randomness $\alpha_1 \neq \alpha_2$, to obtain $a_1, a_2 \in \mathbb{F}^n$ such that for $i = 1, 2$

$$\sum_{b\in\mathscr{B}_k} (a_i(b)G(b) + a_i(b)t(b)\alpha_i P) = \mathsf{cm} + v\alpha_i P.$$

Subtracting the second equation from the first we get

$$\sum_{b\in\mathscr{B}_k}((a_2(b)-a_1(b))G(b)+(\alpha_2 a_2(b)-\alpha_1 a_1(b))t(b)P = v(\alpha_2-\alpha_1)P.$$

If $a_1 \neq a_2$, some $G(b)$ has non-zero coefficient and this gives us a collision we can output. Otherwise, setting $a := a_1$, we have for $i = 1, 2$

$$\sum_{b\in\mathscr{B}_k} a(b)G(b) + a(b)t(b)\alpha_i P = \mathsf{cm} + v\alpha_i P.$$

which implies $\mathsf{com}(a) = \sum_{b\in\mathscr{B}_k} a(b)G(b) = \mathsf{cm}$ and $\hat{a}(u) = \sum_{b\in\mathscr{B}_k} a(b)t(b) = v$. □

□

(of Theorem 5.1)

6 The accumulate protocol

6.1 An accumulation scheme for $\mathcal{L}_G$

Inputs to protocol: $t, \phi_1 = (r_1, C_1), \ldots, \phi_t = (r_t, C_t)$.

1. **V** sends a random $\gamma \in \mathbb{F}$.
2. **P** and **V** compute $C = \sum_{i\in[t]} \gamma^i C_i$.
3. Let $A(\mathbf{X}) := G(\mathbf{X}) \cdot e(\mathbf{X})$, where $e(\mathbf{X}) := \sum_{i\in[t]} \gamma^i \mathbf{eq}(\mathbf{X}, r_i)$.
4. **P** and **V** run the sumcheck protocol with polynomial A, target value C, and degree bound $d = 2$.
5. If **V** outputted reject in the sumcheck it outputs reject.
6. Otherwise, let (r, V) be the sumcheck output. Note that V is the purported value

   $$A(r) = G(r) \cdot e(r),$$

   and checking $V = A(r)$ is equivalent to checking $(1/e(r)) \cdot V = G(r)$.
7. **V** computes $c := e(r)$ and outputs $(r, (1/c)V)$.

Lemma 6.1. *The above is protocol is an accumulation scheme for $\mathcal{L}_G$.*

Proof. We need to show
Completeness: Suppose $\phi_1, \ldots, \phi_t \in \mathcal{L}_G$. Then for each $i \in [t]$

$$G(r_i) = \sum_{b\in\mathscr{B}_k} G(b)\mathbf{eq}(b, r_i) = C_i$$

Thus for any $\gamma \in \mathbb{F}$

$$\sum_{b\in\mathscr{B}_k} A(b) = \sum_{i\in[t]} \gamma^i G(b)\mathbf{eq}(b, r_i) = \sum_{i\in[t]} \gamma^i C_i = C.$$

And so completeness follows from that of the sumcheck protocol (Lemma 3.1).
Soundness: If for some $i \in [t]$

$$G(r_i) = \sum_{b\in\mathscr{B}_k} G(b)\mathbf{eq}(b, r_i) \neq C_i$$

As

$$\sum_{b\in\mathscr{B}_k} G(b)e(b) = \sum_{i\in[t]} \gamma^i G(b)\mathbf{eq}(b, r_i) = \sum_{i\in[t]} \gamma^i G(r_i)$$

Thus we have at most t γ such that $\sum_{b\in\mathscr{B}_k} A(b) = \sum_{i\in[t]} \gamma^i G(r_i)$. Hence from Lemma 3.1 the probability that $(r, (1/c)v) \in \mathcal{L}_G$ is at most $\frac{n+dk}{p} = \mathsf{negl}(\lambda)$ □

7 The decide protocol

Before describing the decide procedure we review some needed components related to Reed-Solomon codes. As remark 7.1 states, one may think of the special case of order two subgroups for simplicity. Our motivation for describing the more general framework from [BSCKL21, BSCKL22] is enabling working with curves without a large order two subgroup, like the Grumpkin curve used by Aztec. We make small alterations from [BSCKL21] to enable working with polynomials over $\mathbb{G}$.

7.1 Folding Reed Solomon codewords

We assume the existence of sets $D_0, D_1, \ldots, D_k \subset \mathbb{F}$ and degree two rational functions $\psi_1, \ldots, \psi_k$ such that $n_0 := 2n = |D_0|$ and for $i \in [k]$,

- $|D_i| = n_i := \frac{2n}{2^i}$.
- ψ_i is $2-1$ from D_{i-1} to D_i.
- Denote $d_i := n_i/2$. For any $f \in \mathbb{G}_{<2d_i}[X]$ we have unique $f_0, f_1 \in \mathbb{G}_{<d_i}[X]$ such that
$$f(X) = (f_0(\psi_i(X)) + Xf_1(\psi_i(X)))v_i(X)^{d_i-1}$$

where $u_i, v_i \in \mathbb{F}_{<3}[X]$ are[4] such that $\psi_i(X) = \frac{u_i(X)}{v_i(X)}$.

In Appendix A we give some details on how the sets $\{D_i\}$ and maps $\{\psi_i\}$ are constructed using ECFFT[BSCKL22].

[4]Since u_i, v_i are over $\mathbb{F}$ they can be composed and multiplied with polynomials over $\mathbb{G}$.

Remark 7.1. *It is instructive to keep in mind the usual FFT on order two subgroup case. There we have for all $i \in [k]$,*

1. *D_i is an order two subgroup consisting of the squares of elements of D_{i-1}*
2. *$\psi_i(X) = X^2$.*

One may think solely of this case for simplicity in the rest of the section.

For $i \in \{0, \ldots, k\}$ we denote by $\mathsf{RS}_i \subset \mathbb{G}^{n_i}$ the (group) Reed-Solomon code of vectors $v \in \mathbb{G}^{n_i}$ consisting of the n_i evaluations of a polynomial $f \in \mathbb{G}_{<d_i}[X]$ on D_i. Note that this is a $[n_i, d_i, d_i]$-code. In particular, the relative distance is $1/2$. For $x \in \mathbb{F}$ we define

$$\mathsf{fold}_x(f)(X) := (1-x)f_0(X) + xf_1(X).$$

Given $s \in D_i$, let its preimages under ψ_i by $s_0, s_1 \in D_{i-1}$. As discussed in Appendix B.2 of [BSCKL22] $\mathsf{fold}_z(f)(s)$ can be computed as a function $H_z[f](s)$ whose value at s depends only on $z, f(s_0), f(s_1)$. Moreover, it is a *degree one polynomial* in $f(s_0), f(s_1)$ and thus still well-defined when $f(s_0), f(s_1)$ are in $\mathbb{G}$ rather than $\mathbb{F}$ as in [BSCKL22].

Thus, we can define $\mathsf{fold}_z(f) : D_i \to \mathbb{G}$ for an arbitrary function $f : D_{i-1} \to \mathbb{G}$ (i.e. not just $f \in \mathbb{G}_{<d_{i-1}}[X]$) by

$$\mathsf{fold}_z(f)(s) := H_z[f](s).$$

Choosing the encoding: We must work with an encoding of $H \in \mathbb{G}^n$ in RS_0 such that folding corresponds to multilinear evaluation. For this purpose, we define bases B_i of $\mathbb{F}_{<d_i}[X]$ inductively over decreasing $0 \le i \le k$.

We define $\mathsf{B}_k := \{1\}$.

Now assume B_i is defined.

We define

$$\mathsf{B}_{i-1} := \left\{ P(\psi_i(X)) \cdot v_i(X)^{d_i - 1}, X \cdot P(\psi_i(X)) v_i(X)^{d_i - 1} \right\}_{P \in \mathsf{B}_i}.$$

Now denote $\mathsf{B}_0 := \{P_0, \ldots, P_{n-1}\}$. Given $H \in \mathbb{G}^n$, we define its encoding in RS_0 to be the evaluation of the polynomial $f_0(X) := \sum_{j<n} H_j P_j(X) \in \mathbb{G}_{<n}[X]$ on D_0.

The crucial property of this encoding is that we have for any $r \in \mathbb{F}^k$, when defining $f_i := \mathsf{fold}_{r_i}(f_{i-1})$ for $i \in [k]$, that $f_k \equiv \hat{H}(r_1, \ldots, r_k)$.

Proximity gaps: [BSCI$^+$20] showed that

Lemma 7.2. *Fix $0 \le \delta \le 1/4$, $i \in [k]$ and $f : D_{i-1} \to \mathbb{G}$. If $\Delta(f, \mathsf{RS}_{i-1}) \ge \delta$ then e.w.p. n/p over $x \in \mathbb{F}$, $\Delta(\mathsf{fold}_x(f), \mathsf{RS}_i) \ge \delta$.*

As the above statement does not appear in [BSCI$^+$20] in this precise form, we provide a proof - heavily relying on the correlated agreement theorem for the unique distance range (Theorem 4.1 from [BSCI$^+$20]).

Proof. We prove the contrapositive. Assume $\Delta(\mathsf{fold}_x(f), \mathsf{RS}_i) < \delta$ for more than $n \geq |D_i|$ $x \in \mathbb{F}$. Then according to Theorem 4.1 in [BSCI$^+$20] taking there $u_0 = f_0, u_1 = (f_1 - f_0)$, there exist $g_0, g_1 \in \mathsf{RS}_i$ and a set $S \subset D_i$ of size at least $(1-\delta)|D_i|$ such that $g_0(a) = f_0(a), g_1(a) = (f_1 - f_0)(a)$ for all $a \in S$. Then $g_1'(a) := (g_0 - g_1)(a) = f_1(a)$ for all $a \in S$. Let $S' \subset D_{i-1}$ be the set $S' := \{a \in D_{i-1} : \psi_i(a) \in S\}$.

We have $|S'| = 2|S|$. For any $a \in S'$

$$f(a) = (f_0(\psi_i(a)) + af_1(\psi_i(a))v_i(a)^{d/2-1} = (g_0(\psi_i(a)) + ag_1'(\psi_i(a))v_i(a)^{d/2-1} = g(a).$$

Where $g \in \mathsf{RS}_{i-1}$ is defined as the encoding in RS_{i-1} of the polynomial

$$g(X) = (g_0(\psi_i(X)) + Xg_1'(\psi_i(X)))v_i(X)^{d/2-1}.$$

Hence $\Delta(f, \mathsf{RS}_{i-1}) < \delta$. □

FRI consistency checks: Fix a sequence of functions $g = (g_0, \ldots, g_k)$, with $g_i : D_i \to \mathbb{G}$, constants $r = (r_1, \ldots, r_k) \in \mathbb{F}^k$, and $s \in D_1 \times \ldots \times D_k$. We define the predicate $\mathsf{consistent}_{g,r}(s)$ based on the well-known FRI test [BBHR18].
For simplicity of analysis[5], at the expense of larger proof length, we present a variant of FRI where the query at each layer is chosen *independently* (as used in Section 4.2 of [EA23]).

$\mathsf{consistent}_{g,r}(s)$:

1. For each $i \in [k]$, let $\mathsf{l}_i, \mathsf{r}_i \in D_{i-1}$ be the two preimages of s_i under ψ_i; and compute $a_i := H_{r_i}[g_{i-1}](s_i)$ from $g_{i-1}(\mathsf{l}_i)$ and $g_{i-1}(\mathsf{r}_i)$. Note that $a_i = \mathsf{fold}_{r_i}(g_{i-1})(s_i)$.
2. Output *accept* if and only if for each $i \in [k]$:

$$a_i = g_i(s_i).$$

7.2 The BaseFold IOP

The BaseFold IOP[ZCF23] is an interleaved sumcheck and FRI-style[BBHR18] proximity check. Here, we describe it specifically for a preprocessed group-valued multilinear polynomial $G \in \mathbb{G}[\mathbf{X}]$. Identifying G with the vector of its values on the boolean cube, we denote as before $\mathcal{L}_G = \{(r, C) | G(r) = C\}$.

$\mathsf{BaseFold}_{\mathbb{G}}(r, C)$ for checking $(r, C) \in \mathcal{L}_G$

Setup: Compute the sets $D_0, \ldots, D_k$ and $\psi_1, \ldots, \psi_k$ as described in Section 7.1. Given multilinear $G \in \mathbb{G}[X]$, compute $g_0 = \mathsf{RS}_0(G)$.

Main protocol:

[5]More specifically, we are able to use "regular" correlated agreement for soundness analysis rather than weighted or mutual correlated agreement as in [BSCI$^+$20] and [GMW25] respectively.

1. Let $A(\mathbf{X}) := G(\mathbf{X})\mathbf{eq}(r, \mathbf{X})$.
2. For $i = 1, \ldots, k$:
   (a) $\mathbf{P}$ computes and sends the sumcheck univariate
   $$A_i(X) := \sum_{b \in \mathscr{B}_{k-i}} A(r_1, \ldots, r_{i-1}, X, b).$$
   (b) $\mathbf{V}$ chooses random $r_i \in \mathbb{F}$.
   (c) $\mathbf{P}$ sends as an oracle the folding $g_i := \mathsf{fold}_{r_i}(g_{i-1})$.
3. After the above rounds, $\mathbf{V}$ performs the consistency checks
   - $C = A_1(0) + A_1(1)$.
   - For $i \in [k-1]$, $A_i(r_i) = A_{i+1}(0) + A_{i+1}(1)$.
   - Setting $r = (r_1, \ldots, r_k)$, $g_k \equiv c$ and $c \cdot \mathbf{eq}(r, v) = A_k(r_k)$.
   - Let $c := e^{-1/8}$, and $\ell := \log_c(2^{-\lambda})$. For $i = 1 \ldots, \ell$, $\mathbf{V}$ chooses uniform query $q_i \in D_1 \times \ldots \times D_k$ and checks the predicate $\mathsf{consistent}_{g,r}(q_i)$.
   - $\mathbf{V}$ outputs reject if one of the above checks failed, and outputs accept otherwise.

7.3 RBR soundness:

Theorem 7.3. *The protocol* $\mathsf{BaseFold}_{\mathbb{G}}$ *has round-by-round soundness error* $\varepsilon := \max\left\{(n+2)/p, 2^{-\lambda}\right\}$.

Note that given the theorem we can set $\mathsf{decide} = \mathsf{BaseFold}_{\mathbb{G}}$ in our PCS $\mathscr{P}$.

Proof. The proof is analogous to the one in [ZCF23] using [BKS18]. As [ZCF23] do not formally show round-by-round soundness we give a self-contained proof that does (occasionally referencing [ZCF23]). We need to define the elements of the doomed set $\mathcal{D}$. As in the protocol description, below we denote by A_i for $i \in [k]$ the correctly computed univariate
$$A_i(X) := \sum_{b \in \mathscr{B}_{k-i}} A(r_1, \ldots, r_{i-1}, X, b),$$
and we will use A'_i to denote univariates contained in the transcript (which may or may not equal A_i).

1. If $\phi \notin \mathcal{L}_G$ then $\phi \in \mathcal{D}$.
2. $\mathfrak{t} = (\phi, A'_1, r_1) \in \mathcal{D}$ if $\phi \notin \mathcal{L}_G$ and
   (a) $\Delta(\mathsf{fold}_{r_1}(g_0), \mathsf{RS}_1) \geq \min\{1/4, \Delta(g_0, \mathsf{RS}_0)\}$
   (b) One of the following holds
      - $A'_1 \neq A_1$ and $A'_1(r_1) \neq A(r_1)$, or

- $A'_1(0) + A'_1(1) \neq C$.

3. More generally, $\mathfrak{t} = (\phi, A'_1, r_1, g_1, \ldots, g_{i-1}, A'_i, r_i) \in \mathcal{D}$ if $\phi \notin \mathcal{L}_G$ and
   (a) For each $j < i$, $\Delta(\mathsf{fold}_{r_j}(g_{j-1}), \mathsf{RS}_j) \geq \min\{1/4, \Delta(g_{j-1}, \mathsf{RS}_{j-1})\}$.
   (b) One of the following holds
   - $A'_i \neq A_i$ and $A'_i(r_i) \neq A_i(r_i)$, or
   - $A'_1(0) + A'_1(1) \neq C$, or
   - For some $1 < j \leq i$, $A'_j(0) + A'_j(1) \neq A'_{j-1}(r_{j-1})$.
4. $\mathfrak{t} = (\phi, \ldots, A'_k, r_k, g_k, q_1, \ldots, q_k) \in \mathcal{D}$ if $\mathbf{V}$ rejects $\mathfrak{t}$.

We recall the three properties of $\mathcal{D}$ we need to show hold from Definition 2.2:

1. If $\phi \notin \mathcal{L}$ then $\mathfrak{t} = (\phi) \in \mathcal{D}$.
2. For a full transcript $\mathfrak{t}$, if $\mathfrak{t} \in \mathcal{D}$ then $\mathbf{V}(\mathfrak{t}) = \mathit{reject}$.
3. For a partial transcript $\mathfrak{t} = (\phi, m_1, r_1, \ldots, m_i, r_i) \in \mathcal{D}$, for any $m_{i+1} = (g_i, A'_{i+1})$, the probability over r_{i+1} that $(\mathfrak{t}, m_{i+1}, r_{i+1}) \notin \mathcal{D}$ is at most ε.

The first and second are immediate from definition. We focus on the third.

Fix $1 \leq i < k$, and any partial transcript $\mathfrak{t} = (\phi, A'_1, r_1, g_1, \ldots, g_{i-1}, A'_i, r_i) \in \mathcal{D}$. We show that for any g_i, A'_{i+1} e.w.p. $(n+2)/p$ over $r_{i+1} \in \mathbb{F}$, $\mathfrak{t}^* := (\mathfrak{t}, g_i, A'_{i+1}, r_{i+1}) \in \mathcal{D}$. This essentially follows from Lemma 7.2 and the soundness of the sumcheck protocol. In more detail,

- Lemma 7.2 tells us that e.w.p. n/p

$$\Delta(\mathsf{fold}_{r_{i+1}}(g_i), \mathsf{RS}_{i+1}) \geq \min\{1/4, \Delta(g_i, \mathsf{RS}_i)\}.$$

- We must show one of the three conditions in Item 3b holds for $\mathfrak{t}^*$: If one of the latter two held for $\mathfrak{t}$ it also holds for $\mathfrak{t}^*$. If $A'_{i+1} \neq A_{i+1}$ then $A'_{i+1}(r_{i+1}) \neq A_{i+1}(r_{i+1})$ e.w.p. $2/p$ and the first condition holds. In the remaining cases $A'_i(r_i) \neq A(r_i)$ and $A'_{i+1} = A_{i+1}$; so $A'_{i+1}(0) + A'_{i+1}(1) = A_i(r_i) \neq A'_i(r_i)$ which means the third condition is satisfied.

Now it's left to prove that for partial transcript $\mathfrak{t} = (\phi, \ldots, A'_k, r_k) \in \mathcal{D}$ and any g_k, e.w.p. ε over $q = (q_1, \ldots, q_\ell)$, $(\mathfrak{t}, g_k, q) \in \mathcal{D}$; or equivalently $\mathbf{V}$ rejects $(\mathfrak{t}, g_k, q)$.

We know that $\mathbf{V}$ outputs reject (with probability one) if $g_k \not\equiv c$ for some constant c. We also know from the definition of $\mathcal{D}$ that $\mathbf{V}$ outputs reject if $c = A(r)/\mathbf{eq}(r, z) = G(r)$ - as if $\mathfrak{t} \in \mathcal{D}$ but the sumcheck consistency checks pass (the first two items in Step 3 of main protocol), then $A'_k(r_k) \neq A_k(r_k) = A(r)$, so the check $c \cdot \mathbf{eq}(r, v) = A'_k(r_k)$ will fail.

Hence it suffices, given a choice $g_k \equiv c$, to bound the probability over q that $\mathbf{V}$ accepts $(\mathfrak{t}, g_k, q)$, and we can focus on the case $c \neq G(r)$. We'll show that for any such choice

of g_k, a query $\mathsf{consistent}_{g,r}(s)$ for uniform $s \in D_1 \times \ldots \times D_k$ accepts with probability at most c. As q consists of ℓ independent such queries $\mathbf{V}$ accepts with probability at most $c^\ell = 2^{-\lambda}$.

For this purpose we look at two cases:

Case one: *For all* $i \in [k-1]$, $\Delta(g_i, \mathsf{RS}_i) \leq 1/8$.

Recall that $g_0 \in \mathsf{RS}_0$ and we know that in $\mathfrak{t}$ $g_k \in \mathsf{RS}_k$. For $i \in \{0, \ldots, k\}$, let $f_i \in \mathsf{RS}_i$ be the (unique) closest codeword to g_i. Note that $f_0 = g_0$ and $f_k = g_k$. Let $h_1, \ldots, h_k$ be the correct foldings of g_0. That is, $h_1 := \mathsf{fold}_{r_1}(g_0) \in \mathsf{RS}_1$, and $h_i := \mathsf{fold}_{x_i}(h_{i-1}) \in \mathsf{RS}_i$ for $2 \leq i \leq k$. Set also $h_0 := g_0$. As the correct folding process leads to multilinear evaluation, we have that $h_k \equiv G(r)$. Hence, f_k and h_k are different elements of RS_k. On the other hand we have $f_0 = h_0$. Thus, there must be $i \in [k]$ such that $f_{i-1} = h_{i-1}$ and $f_i \neq h_i$.

We show for this i that

$$\Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1})) \geq 1/8. \tag{3}$$

Note that this implies $\mathbf{V}$ accepts each $\mathsf{consistent}_{g,r}(q_j)$ with probability at most $7/8 = (1 - 1/8) \leq e^{-1/8} = c$ over q_j - because it will in particular check $g_i(s_i) = \mathsf{fold}_{r_i}(g_{i-1})(s_i)$ at a random $s_i \in D_i$.

We proceed to prove equation 3. We have using the triangle inequality

$$\Delta(g_i, h_i) \leq \Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1})) + \Delta(\mathsf{fold}_{r_i}(g_{i-1}), h_i).$$

Using the fact that folding increases distance by at most a factor of two, we get

$$\leq \Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1})) + 2\Delta(g_{i-1}, h_{i-1}) \leq \Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1})) + 1/4.$$

So, we have

$$\Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1})) \geq \Delta(g_i, h_i) - 1/4 \geq 1/2 - 1/8 - 1/4 = 1/8,$$

where we used the fact that $\Delta(g_i, f_i) \leq 1/8$ implies $\Delta(g_i, h_i) \geq 1/2 - 1/8$ by the distance of the code RS_i.

Case two: For some $j \in [k-1]$, $\Delta(g_j, \mathsf{RS}_j) > 1/8$. We use calculations similar to Section 7 of [BKS18]. All claims below are under the assumption we are in case two.

Define for each $i \in [k]$, $\beta_i := \Delta(g_i, \mathsf{fold}_{r_i}(g_{i-1}))$. Define their average as $\mathcal{b} := \frac{1}{k}\sum_{i \in [k]} \beta_i$. We have that

Claim 7.4. *For uniform* $s \in D_1 \times \ldots \times D_k$, $\mathsf{consistent}_{g,r}(s)$ *accepts with probability at most* $(1 - \mathcal{b})^k$.

Proof. It's immediate that the acceptance probability is at most $\prod_{i \in [k]}(1 - \beta_i)$, as choosing $s_i \in D_i$ with $g_i(s) \neq \mathsf{fold}_{r_i}(g_{i-1})(s)$ at any layer i leads to rejection. Now using the AM-GM inequality on the elements $\{1 - \beta_i\}$ with arithmetic mean $1 - \mathcal{b}$, we have

$$\prod_{i \in [k]} (1 - \beta_i) \leq (1 - \mathcal{b})^k.$$

□

Now define for each $0 \leq i \leq k$, $\delta_i := \min\{1/4, \Delta(g_i, \mathsf{RS}_i)\}$. As we are working with "good" r_i from $\mathfrak{t} \in \mathcal{D}$, we have $\delta_{i-1} \leq \Delta(\mathsf{fold}_{r_i}(g_{i-1}), \mathsf{RS}_i)$ for all $i \in [k]$. Recall also that $\delta_k = \Delta(g_k, \mathsf{RS}_k) = 0$. We have

Claim 7.5. *For each $i \in [k]$,*

$$\beta_i \geq \delta_{i-1} - \delta_i.$$

Proof. Fix $i \in [k]$. If $\delta_i \geq \delta_{i-1}$, as $\beta_i \geq 0$ the claim is trivial. Assume now that $\delta_i < \delta_{i-1}$. Recalling the definition of the δ's, this means that $\delta_i < 1/4$ and so $\delta_i = \Delta(g_i, \mathsf{RS}_i)$. We have using the triangle inequality

$$\delta_{i-1} \leq \Delta(\mathsf{fold}_{r_i}(g_{i-1}), \mathsf{RS}_i) \leq \Delta(\mathsf{fold}_{r_i}(g_{i-1}), g_i) + \Delta(g_i, \mathsf{RS}_i) = \beta_i + \delta_i.$$

Rearranging terms we have $\beta_i \geq \delta_{i-1} - \delta_i$ as required. □

Using Claim 7.5 we have

$$\mathcal{b} = \frac{1}{k}\sum_{i=1}^{k} \beta_i \geq \frac{1}{k}\sum_{i=j+1}^{k} \beta_i \geq \frac{1}{k}\sum_{i=j+1}^{k} (\delta_{i-1} - \delta_i) = (\delta_j - \delta_k)/k = \delta_j/k \geq 1/(8k).$$

Hence, from Claim 7.4 the acceptance probability is at most $(1 - 1/(8k))^k \leq e^{-1/8}$ as required.

This concludes the second case. Thus we have shown that $\mathsf{BaseFold}_{\mathbb{G}}$ has RBR soundness error $\max\left\{(n+2)/p, 2^{-\lambda}\right\}$. □

Remark 7.6. *It seems that*

1. *A more nuanced analysis using* coset relative distance *as in [ZCF23] can improve the constant* $1/8$ *describing the rejection probability of a single* FRI *query to* $1/6$*. This in turn reduces the number ℓ of required queries.*
2. *The rejection probability may be significantly improved further using analysis of* BaseFold *in the list-decoding regime as done by Haböck[Hab24].*

Acknowledgements

We thank Jonathan Bootle for answering questions about [BCS21]. We thank Ulrich Haböck for numerous discussions on proximity testing, FRI and BaseFold. We thank Nicolas Mohnblatt for discussions on FRI that helped us realize the existence of an error in a previous version of the paper. We thank Giacomo Fenzi and Alan Szepieniec for helpful comments and discussions.

References

[ACK21] T. Attema, R. Cramer, and L. Kohl. A compressed σ-protocol theory for lattices. Cryptology ePrint Archive, Paper 2021/307, 2021.

[BBB+17] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066, 2017.

[BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, 2018.

[BCC+16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. pages 327–357, 2016.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2020.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016.

[BCS21] J. Bootle, A. Chiesa, and K. Sotiraki. Sumcheck arguments and their applications. Cryptology ePrint Archive, Paper 2021/333, 2021.

[BGH19] S. Bowe, J. Grigg, and D. Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, page 1021, 2019.

[BGK+23] A. R. Block, A. Garreta, J. Katz, J. Thaler, P. Ranjan Tiwari, and M. Zajac. Fiat-shamir security of FRI and related snarks. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part II*, volume 14439 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2023.

[BKS18] Eli Ben-Sasson, Swastik Kopparty, and Shubhangi Saraf. Worst-case to average case reductions for the distance to a code. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24,*

2018, San Diego, CA, USA, volume 102 of *LIPIcs*, pages 24:1–24:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[BSCI+20] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf. Proximity gaps for reed-solomon codes. Cryptology ePrint Archive, Paper 2020/654, 2020.

[BSCKL21] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit. Elliptic curve fast fourier transform (ecfft) part i: Fast polynomial algorithms over all finite fields. *arXiv preprint arXiv:2107.08473*, Jul 2021.

[BSCKL22] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit. Scalable and transparent proofs over all large fields, via elliptic curves (ECFFT part II). Cryptology ePrint Archive, Paper 2022/1542, 2022.

[CBBZ22] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. *IACR Cryptol. ePrint Arch.*, page 1355, 2022.

[COS19] A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. *IACR Cryptology ePrint Archive*, 2019:1076, 2019.

[DP24] B. E. Diamond and J. Posen. Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, Paper 2024/504, 2024.

[EA23] A. Evans and G. Angeris. Succinct proofs and linear algebra. Cryptology ePrint Archive, Paper 2023/1478, 2023.

[GMW25] A. Garreta, N. Mohnblatt, and B. Wagner. A simplified round-by-round soundness proof of FRI. Cryptology ePrint Archive, Paper 2025/1993, 2025.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptology ePrint Archive*, 2019:953, 2019.

[Hab24] U. Haböck. Basefold in the list decoding regime. Cryptology ePrint Archive, Paper 2024/1571, 2024.

[HJR+25] Tamir Hemo, Kevin Jue, Eugene Rabinovich, Gyumin Roh, and Ron D. Rothblum. Jagged polynomial commitments (or: How to stack multilinears). Cryptology ePrint Archive, Paper 2025/917, 2025.

[HLP24] U. Haböck, D. Levit, and S. Papini. Circle STARKs. Cryptology ePrint Archive, Paper 2024/278, 2024.

[KTW24] Tohru Kohrita, Patrick Towa, and Zachary J. Williamson. One-shot native proofs of non-native operations in incrementally verifiable computations. *IACR Cryptol. ePrint Arch.*, page 1651, 2024.

[LFKN92] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Set19] S. T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. *IACR Cryptol. ePrint Arch.*, page 550, 2019.

[Wil23] Z. J. Williamson. Goblin plonk: lazy recursive proof composition, 2023. `https://hackmd.io/@aztec-network/B19AA8812`.

[ZCF23] H. Zeilberger, B. Chen, and B. Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Cryptology ePrint Archive, Paper 2023/1705, 2023.

A ECFFT

When working over a field with a large power of 2 subgroup, there is a natural choice for the sequence of domains and quadratic maps. Assuming that $n \mid p-1$, we can find an n primitive root of unity ω in the field and let $D_i = \{\omega^{j2^i} : j \in [n/2^i]\}$ and $\psi_i(x) = x^2$. These maps are always 2-to-1 over the field, since all the domains are closed under inverse except the last, and we have $\psi(D_i) = D_{i+1}$ by construction. Unfortunately, many fields of interest in practice do not have this property, including the scalar field of the Grumpkin curve.

In these cases, we need to consider other families of domains. To understand how to generalize the roots of unity domains, it is helpful to understand "why" they exist in the first place. The roots of unity of order n form an algebraic, cyclic group under field multiplication. The squaring maps take this group to a subgroup of half the size. In the same way, generalizations of this construction also start with an algebraic cyclic group of order n and use some "doubling" homomorphism in this group to derive a sequence of domains. If this doubling map is quadratic, then we have the necessary ψ

One possible generalization is the construction used by CircleSTARKs. Here, the group arises via a group action of the multiplicative group of the quadratic extension. This is equivalent to the "circle group" of the projective line. If this group has a subgroup of order n, then it is possible to derive a sequence of quadratic maps that amount of squaring in larger multiplicative group. Unfortunately, this requires that $n \mid p+1$, which is also not the case in general.

The more general approach of ECFFT [BSCKL21] works over any field. Elliptic curves with arbitrary order within the Hasse bound are known to exist, so we can always construct an algebraic group with a subgroup of order n when n is small compared to the characteristic. Checking a curve has a subgroup of order n is easy, and the ECFFT authors show that finding such a curve via brute force search takes $O(n)$ time, so it is feasible to find such curves.

Endomorphisms of curves are algebraic, but they are defined for curve points not elements of the field. However, it is easy to see that for any isogeny $\phi : E \to E'$ there must exist some $\psi : \mathbb{F} \to \mathbb{F}$ such that $x(\phi(P)) = \psi(x(P))$ where x is the x coordinate of

a Weierstrass formula for E and E'. This is because an isogeny must respect the inverse map, so $\phi(-P) = -\phi(P)$ and in particular $x(\phi(P)) = x(\phi(-P))$. Thus, $x(\phi(P))$ must be an even function of $y(P)$, and $y(P)^2$ is a function of $x(P)$ by assumption. So, we can in principle define a sequence of 2-to-1 maps on the x coordinates of elliptic curves.

There is another issue though: the doubling map in an elliptic curve is not a quadratic but a quartic map. This is because the torsion subgroup of an elliptic curve in a suitable extension is always isomorphic to two cyclic groups. So, there are 4 points in the kernel of the multiplication by 2 map. ECFFT avoids this issue by instead using a sequence of degree 2 isogenies between different elliptic curves. In general this is necessary, since degree 2 endomorphisms only exist for certain elliptic curves with particular kinds of complex multiplication.

Fixing an elliptic curve E_0 with a subgroup of order $2n$ generated by G, there must be a degree 2 cyclic subgroup of the curve generated by $H = nG$. Using Velu's formulas, we can compute an isogeny $\phi_1 : E_0 \to E_1$ whose kernel is $\langle H \rangle$ and whose image is E_1. Since they are isogenous, $E_0 \simeq E_1$ as groups so it also has a degree n subgroup, and we can repeat the process to construct a sequence of elliptic curves and subgroups $G_0 = \langle G \rangle$ and $G_i = \phi_i(G_{i-1})$. Then, we can define the maps ϕ_i as the isogeny applied to the x coordinate so $x(\phi_i(P)) = \psi_i(x(P))$ and the domains $D_i = \{x(P) : P \in G_i\}$. These maps must be quadratic, as their kernel is fixed, and thus we have a sequence of domains and quadratic maps.

A.0.1 Computing Twiddle Factors

To instantiate a protocol using these groups, and in particular to have an efficient verifier, we must be able to compute the so-called "twiddle factors" efficiently. These are used to relate the evaluations of $f(X)$ to its $f_0(X)$ and $f_1(X)$ parts. For roots of unity domains, these twiddle factors are easy to compute, but for ECFFT twiddle factors the process is more complex. Recall that

$$f(X) = (f_0(\psi_i(X)) + X f_1(\psi_i(X)))v_i(X)^{d_i - 1}$$

Where v_i is the denominator of ψ_i. We can compute the twiddle factors by evaluating this expression at points in domain D_i. We know that ψ_i is the x component of the isogeny between elliptic curves E_i and E_{i+1}, and further that v_i vanishes precisely when this isogeny vanishes in the elliptic curve group. Therefore, we have that v_i vanishes on the subgroup of order 2 that we used to construct the isogeny. These functions can easily be precomputed and stored in $O(\log n)$ space.

The domain D_i is always the x coordinates of a cyclic group of $G_i \subseteq E_i$, and we can exploit this structure to efficiently compute elements of the domain. All the generators are the result of successive application of the isogenies, and all elements of the groups are integer multiples of their respective generators. Thus, all elements of the group can be computed in linear time and specific elements can be computed in $O(\log n)$ time using double and add. This means we can both compute all the twiddle factors and compute specific twiddle factors efficiently. In the case of STARK verification, this can

be performed even more efficiently since we only need twiddle factors that lie in successive images of the isogenies, which can be computed even more efficiently.

In all these cases, we must perform division operations to compute the twiddle factors. Since division is expensive to compute, it may be more efficient for the prover to precompute and store all the factors ahead of time. However, this is not possible for an efficient verifier and divisions are unavoidable. In circuit, divisions are free so this is not an issue for our application.