IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 25 pages.
https://doi.org/10.62056/abksdk5vt

A divide-and-conquer sumcheck protocol

Christophe Levrat[1], Tanguy Medevielle[2] and Jade Nardi[2]

[1] INRIA Saclay, Palaiseau, France
[2] Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000, Rennes, France

Abstract. We present a new sumcheck protocol called Fold-DCS (Fold-Divide-and-Conquer-Sumcheck) for multivariate polynomials based on a divide-and-conquer strategy. Its round complexity and soundness error are logarithmic in the number of variables, whereas they are linear in the classical sumcheck protocol. This drastic improvement in number of rounds and soundness comes at the expense of exchanging multivariate polynomials, which can be alleviated using polynomial commitment schemes. We first present Fold-DCS in the PIOP model, where the prover provides oracle access to a multivariate polynomial at each round. We then replace this oracle access in practice with a multivariate polynomial commitment scheme; we illustrate this with an adapted version of the recent commitment scheme Zeromorph [KT24], which allows us to replace most of the queries made by the verifier with a single batched evaluation check.

1 Introduction

The classical sumcheck protocol [LFKN92] is an interactive proof protocol used to verify the sum of the values of a given multivariate polynomial over a large domain, typically a hypercube. The protocol works by iteratively reducing a multivariate polynomial sum to a univariate case, allowing efficient verification without requiring the verifier to recompute the entire sum. At each round, the arity of the polynomial is reduced by one, meaning that there is one round per variable. It is highly efficient in terms of communication, as the prover only sends *univariate* polynomials to the verifier. Keeping the amount of data sent to the verifier this low alleviates the cost (in time and space) of computing cryptographic commitments to large vector in zero-knowledge proof systems and thus makes the sumcheck protocol a core component in several zk-SNARKs. For instance, Hyrax [WTS+18] calls for as many sumcheck invocations as the depth of the circuit, and Spartan [Set20] needs two sumcheck invocations for products of two multilinear polynomials.

The sumcheck protocol also plays a central role in Interactive Proofs (IPs). It is the main ingredient of the GKR interactive proof for circuit evaluation [GKR15]. Bootle et al. [BCS21] recently introduced a class of interactive protocols, called *sumcheck arguments*, which turn the knowledge proofs of openings for certain commitment schemes CM into sumcheck protocols for a function f_{CM} over a domain H. Such compatible commitment schemes are said *sumcheck-friendly*. Sumcheck arguments establish an elegant connection between the sumcheck protocol and several seemingly disparate works, such as folding techniques. This renews and reinforces the need for efficient sumcheck protocols.

In this work, we present a new polynomial interactive oracle proof (PIOP) to check the sum of a multivariate polynomial f of arity μ over a hypercube H^μ in $O(\log \mu)$ rounds.

All the authors are supported by the French National Research Agency through ANR *Barracuda* (ANR-21-CE39-0009). The second and the third authors are supported by the French government *Investissements d'Avenir* program ANR-11-LABX-0020-01.

E-mail: christophe.levrat@math.cnrs.fr (Christophe Levrat), tanguy.medevielle@univ-rennes.fr (Tanguy Medevielle), jade.nardi@univ-rennes.fr (Jade Nardi)

Received: 2025-01-07 Accepted: 2025-03-11

The strategy in the standard sumcheck protocol [LFKN92] is to reduce the problem at each round to another instance of the sumcheck protocol with a polynomial of lower arity. Here, instead of decreasing the arity by one at each round, we rely on the Divide-and-Conquer routine to turn one instance of the sumcheck into two instances of half the "size", here half the arity. The first instance still aims at verifying that the claimed sum is correct, while the second one allows to check the integrity of the function used in the first one. A classical trick (see [BSBHR18, BSBHR19]) to avoid doubling the instances at each turn is to *fold* them: instead of checking the sums of two polynomials f_0 and f_1, we check that a random linear combination of f_0 and f_1 has the expected sum, repeating the Divide-and-Conquer process described above.

Ultimately, the final check is a univariate sumcheck which can be performed either by the verifier herself (querying $|H|$ values of the last commit) or using an efficient interactive protocol, like the one in Aurora [BSCR⁺19] if H is structured. As a result, the round complexity is $O(\log \mu)$ for a μ-variate polynomial, compared to μ in the standard protocol.

Decreasing the number r of rounds is critical in the context of the Fiat-Shamir transform. For a $(2r+1)$-move interactive protocol in which the prover has a cheating probability of at most ϵ, the associated Fiat-Shamir-transformed protocol admits a cheating probability of at most $(Q+1)^r \cdot \epsilon$, where Q is the number of random-oracle queries. Attema et al. [AFK23] showed that this exponential security loss does not only occur for contrived examples, but also for some natural protocols such as the t-fold parallel repetition of protocols. It is worth noting that this critical loss of security does not happen when the interactive protocol satisfies a strengthened version of soundness, called *round-by-round soundness* [CCH⁺18].

Comparison with the standard sumcheck protocol In both the standard and our protocol, the soundness is linear in the number of rounds. However, the soundness of Fold-DCS depends on the total degree of the polynomial, not its individual degrees. Thanks to the exponential gain in round complexity, we thus also achieve a better soundness as long as the total degree of the polynomial is fixed and at most $\mu/\log(\mu)$ times its highest individual degree.

This significantly lower number of rounds comes at the expense of the exchange of *multivariate* polynomials between the prover and the verifier, which would make the proof size and the verifier complexity explode. Our PIOP for sumcheck thus requires a polynomial commitment scheme (PCS) for practical use. In our protocol, if the polynomials computed by the prover P were fully sent (without using commitment schemes), most of the verifier V's computational complexity would reside in evaluating multivariate polynomials sent by the prover P. We have chosen to first present the protocol in the PIOP model (see Section 3), in which V is not sent actual polynomials by P, but instead given oracle access to each one of them, allowing V to query evaluations of said polynomials at any point. Then, in Section 4, we present the protocol using a multivariate polynomial commitment scheme (PCS), in which P first sends commitments to the polynomials; later, P and V run an evaluation protocol in which the prover sends the values of a batch of polynomials at a given common point and a proof of convinces the verifier of the correctness of these values.

Complexities of the standard sumcheck protocol, the Fold-DCS in the PIOP model and its instantiated version with the commitment scheme Zeromorph [KT24] are gathered up in Table 1. Note that in the usual description of the standard sumcheck [LFKN92], the prover is given oracle access to the original polynomial. So the prover computations consist in querying $|H|^\mu$ values and summing them, hence a prover complexity of $O(|H|^\mu)$ $\mathbb{F}_q$-operations (see [Tha22, §4.1] for details). Handling the whole polynomial as in the PIOP-model, the prover can perform less operations (recall that $d < |H|$). For fair comparison, we give the prover complexity of the standard sumcheck protocol in the latter case. A similar computation to the one of §3.3 shows that the prover needs to perform at most $\mu^2 d^{\mu-1} + 2d\,|H|$ operations in $\mathbb{F}_q$, which is less than $|H|^\mu$ for μ large enough. As a result, our protocol with $\log(\mu)$ rounds also decreases the prover complexity in the PIOP model.

Choosing a multivariate polynomial commitment scheme We chose to instantiate Fold-DCS with the commitment scheme Zeromorph [KT24] based on KZG commitments [KZG10]. While Zeromorph is not transparent, it offers the following advantages:

- Since it does not use a sumcheck protocol as a subroutine, its evaluation protocol has constant round complexity;
- The verifier complexity of its evaluation protocol is linear in the number of variables;
- Its evaluation protocol allows for batching and shifting [KT24, §8].

Table 1: Comparison of protocol Fold-DCS (with the PCS Zeromorph [KT24]) with the standard sumcheck protocol for a μ-variate polynomial of partial degrees at most d and total degree at most D over a coset $H \subset \mathbb{F}$. The verifier and prover complexities are counted in terms of operations in the field $\mathbb{F}$. The communication complexity measures the number of elements of $\mathbb{F}$ exchanged during the protocols.

	Standard sumcheck	Fold-DCS	
		PIOP model	with Zeromorph
Number of rounds	μ	$O(\log \mu)$	$O(\log \mu)$
Randomness	μ	$\mu + \log \mu + 1$	$O(\mu \log d \log \mu)$
Query complexity		$2 \log \mu + 3$	
# of commitments (PIOP)		$\log \mu + 1$	
Communication complexity	$d \cdot \mu$		$O(\mu \log d \log \mu)$
Prover complexity	$\mu^2 d^{\mu-1} + 2d\,\|H\|$	$\log(\mu)\mu d^{\mu/2} + 2d\,\|H\|$	$O(d \log(\mu) 2^\mu)$
Verifier complexity	$\mu d \log d$	$O(\log \mu)$	$O(\mu \log d)$
Soundness	$\mu \cdot \frac{d}{\|\mathbb{F}\|}$	$(\log \mu + 1) \cdot \frac{D+1}{\|\mathbb{F}\|}$	

In particular, all the queries made by the verifier in our protocol may be summed up in a single batched evaluation when instantiating Fold-DCS with Zeromorph. Zeromorph is initially designed to commit to multilinear polynomials. We present an adapted version for multivariate polynomials with prescribed partial and total degrees. Since KZG and thus Zeromorph require bilinear pairings, this restricts their operation to large fields where pairing-friendly elliptic curves can be defined.

Recent works focused on building PCS that are *field-agnostic*, contrarily to aforementioned PCS based on elliptic curves over designated finite fields. Some examples of field-agnostic PCS are Brakedown [GLS+23], Basefold [ZCF24], BrakingBase [NST24]. Unfortunately, up to our knowledge, all field-agnostic PCS rely on the standard sumcheck protocol, so the number of rounds in the evaluation protocol equals the arity of the polynomial. Despite their efficiency with respect to time complexities, there is no point in using these PCS for Fold-DCS: this would annihilate our advantage in terms of rounds.

It is worth noting that with all currently known PCS with constant round complexity, the batched evaluation protocol between the prover and the verifier is the main bottleneck of Fold-DCS in terms of time complexities. This raises the natural question of the possibility of recursively invoking our sumcheck protocol in the state-of-the-art field-agnostic PCS cited above, which we leave for future works.

Mixing the standard and Fold-DCS approaches With this new efficient sumcheck protocol Fold-DCS in hand, one can reasonably suggest to mix both the standard protocol and Fold-DCS. A natural idea to lower the round complexity of the standard sumcheck protocol is to send at each round a polynomial with a fixed arity $k = k(\mu)$ (which may depend on the total number μ of variables). This reduces a μ-variate sumcheck to μ/k sumchecks of arity k, which can be merged into one sumcheck of arity k using the folding technique described in §3.2.1. However, both the communication and the verifier complexity are now higher due to the fact that k-variate polynomials are exchanged and used for sumchecks. This may, like in Fold-DCS, be alleviated by having P provide oracle access to the polynomials; for each of the sumchecks, V then needs to make $|H|^k$ queries.

For the sake of the query complexity, this k-variate sumcheck should again be handled using the standard sumcheck protocol, or using Fold-DCS. In the former case, the total round number is $\mu/k + k$, which is minimal when $k = \sqrt{\mu}$. This means that the soundness error is $O(\sqrt{\mu}d/q)$: this is much worse than Fold-DCS in terms of round complexity as well as soundness. In the latter case, the total number of rounds is $\mu/k + \log(k)$. The round number is minimal when $k = \mu$, as will be the soundness error; this corresponds to the case where Fold-DCS is directly performed on the initial polynomial. However, the query complexity is $1 + \log(k)$, and is minimal when $k = 1$, which corresponds to the standard protocol.

2 Preliminaries

2.1 Interactive proofs and sumcheck protocols

Interactive proofs (IPs) were introduced by Goldwasser, Micali, and Rackoff [GMR89]: in an rn-round interactive proof for a language $\mathcal{L}$, a probabilistic polynomial-time verifier V exchanges rn messages with an unbounded prover P, and then accepts or rejects. The goal is that V accepts when the inputs belong to $\mathcal{L}$, and rejects with high probability when they do not. *Interactive oracle protocols* (IOP) were introduced by Ben-Sasson, Chiesa and Spooner [BSCS16] and differ from IPs by the way the verifier accesses the prover's messages. At each round, the verifier sends a message to the prover which he reads in full, whereas the prover replies with a message to the verifier, which she can *query* (via random access) in the given round and all later rounds. In both cases, we denote by $\langle \mathsf{P} \leftrightarrow \mathsf{V} \rangle \in \{\mathsf{accept}, \mathsf{reject}\}$ the output of V after interacting with P. Certain inputs of V can also only be given via oracle access. Traditionally, this difference is highlighted by writing $\mathsf{V}^{i_o}(i_f)$ where i_o is the set of inputs which V accesses via oracles, and i_f the one she can fully read.

Definition 1 (Perfect completeness)**.** An interactive (oracle) proof for a language $\mathcal{L}$ is said to be *perfectly complete* if

$$\Pr\left[\langle \mathsf{P}(i_o, i_f) \leftrightarrow \mathsf{V}^{i_o}(i_f)\rangle = \mathsf{accept} \;\middle|\; (i_o, i_f) \in \mathcal{L}\right] = 1.$$

Definition 2 (Soundness)**.** Let $\mathcal{L} = (\mathcal{L}_\rho)_{\rho\in\mathcal{P}}$ be a family of languages which depend on an element ρ of some parameter space $\mathcal{P}$. An interactive (oracle) protocol for $\mathcal{L}$ is said to have *soundness error* $s\colon \mathcal{P} \to \mathbb{R}$ if for any parameter $\rho \in \mathcal{P}$, any *unbounded* malicious prover $\widetilde{\mathsf{P}}$, and any inputs (i_0, i_f),

$$\Pr\left[\langle \widetilde{\mathsf{P}}(i_o, i_f) \leftrightarrow \mathsf{V}^{i_o}(i_f)\rangle = \mathsf{accept} \;\middle|\; (i_o, i_f) \notin \mathcal{L}_\rho\right] \leqslant s(\rho).$$

Definition 3. Let μ, d, D be nonnegative integers. Let $\mathbb{F}$ be a finite field. We denote by $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ the $\mathbb{F}$-vector space of μ-variate polynomials coefficients in $\mathbb{F}$ with individual degrees at most d and total degree at most D, *i.e.* generated by the set of monomials

$$\left\{ x_1^{i_1} \dots x_\mu^{i_\mu} \;\middle|\; \sum_{j=1}^{\mu} i_j \leqslant D \text{ and } \forall j \in \{1, \dots, \mu\}, \, i_j \leqslant d \right\}.$$

Definition 4. Let μ, d, D be nonnegative integers. Let $\mathbb{F}$ be a finite field, and H be a subset of $\mathbb{F}$. A *sumcheck protocol* for μ-variate polynomials with coefficients in $\mathbb{F}$, partial degrees $\leqslant d$ and total degree $\leqslant D$ for the summation set H is an interactive (oracle) protocol for the language

$$\mathcal{L}_{\mu,d,D,\mathbb{F},H} = \left\{ (f, S) \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D} \times \mathbb{F} \;\middle|\; \sum_{\boldsymbol{a} \in H^\mu} f(\boldsymbol{a}) = S \right\}.$$

Remark 1. Using the low-degree extension [BFLS91, Proposition 4.1], we can assume w.l.o.g. that $d \leqslant |H| - 1$. Moreover, the degrees and the arity satisfy $D \leqslant \mu d$.

We are going to study sumcheck protocols in the Polynomial IOP model, as introduced in [BFS20, Definition 5]. We give here a slightly modified definition that is more suitable for the sumcheck in terms of degree bounds and arity.

Definition 5 ((μ, d, D)-Polynomial IOP)**.** Let $\mathcal{L}$ be a language, $\mathbb{F}$ some finite field, and $\mu, d, D \in \mathbb{N}$. A (μ, d, D)-*Polynomial IOP* for $\mathcal{L}$ with partial degree bound d and total degree bound D over $\mathbb{F}$ is a pair of interactive machines (P, V), satisfying the following description.

- (P, V) is an interactive proof for $\mathcal{L}$;
- P sends polynomials $f_i(\boldsymbol{x}) \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ to V;
- V is an oracle machine with access to a list of oracles, which contains one oracle for each polynomial it has received from the prover.
- When an oracle associated with a polynomial f_i is queried on a point $\boldsymbol{z}_j \in \mathbb{F}^\mu$, the oracle responds with the value $f_i(\boldsymbol{z}_j)$.

The computation of the soundness error of sumcheck protocols relies on the well-known Schwartz-Zippel lemma [DL78, Zip79, Sch80].

Lemma 1 (Schwartz-Zippel)**.** *Let $f \in \mathbb{F}[\boldsymbol{x}]$ be a nonzero μ-variate polynomial of total degree D. For uniformly picked $\boldsymbol{a} \in H^\mu$,*

$$\Pr_{\boldsymbol{a}}[f(\boldsymbol{a}) = 0] \leqslant \frac{D}{|H|}.$$

2.2 The standard sumcheck protocol

The standard sumcheck protocol [LFKN92] is described in Protocol 1. In this protocol, the verifier V checks at each round the sum of a univariate polynomial sent by the prover P. In the end, V queries one evaluation of the initial function to ensure consistency. The univariate sumchecks at each round are usually presented as being carried out by hand; however, V may also run a univariate sumcheck protocol to do this.

Its soundness is usually computed by considering a union over all rounds of the protocol, resulting in an upper bound of $\mu d / |\mathbb{F}|$. This result can be refined as follows.

Protocol 1: The standard sumcheck protocol [LFKN92]

Parameters: integer μ, field $\mathbb{F}$, and $H \subseteq \mathbb{F}$.
Inputs: $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ and $S \in \mathbb{F}$.

$P(f,S)$		$V^f(S)$
Compute $f_1(x_1) = \sum_{\boldsymbol{a} \in H^{\mu-1}} f(x_1, a_2, \ldots, a_\mu)$		
	$\xrightarrow{f_1}$	
		$\sum_{a \in H} f_1(a) \stackrel{?}{=} S$
	$\xleftarrow{\alpha_1 \xleftarrow{\$} \mathbb{F}}$	
Compute $f_2(x_2) = \sum_{\boldsymbol{a} \in H^{\mu-2}} f(\alpha_1, x_2, a_3, \ldots, a_\mu)$		
	$\xrightarrow{f_2}$	
		$\sum_{a \in H} f_2(a) \stackrel{?}{=} f_1(\alpha_1)$
	$\vdots$	
	$\xleftarrow{\alpha_{i-1} \xleftarrow{\$} \mathbb{F}}$	
Compute $f_i(x_i) = \sum_{\boldsymbol{a} \in H^{\mu-i}} f(\alpha_1, \ldots, \alpha_{i-1}, x_i, a_{i+1}, \ldots, a_\mu)$		
	$\xrightarrow{f_i}$	
		$\sum_{a \in H} f_i(a) \stackrel{?}{=} f_{i-1}(\alpha_{i-1})$
	$\vdots$	
Compute $f_\mu(x_\mu) = f(\alpha_1, \ldots, \alpha_{\mu-1}, x_\mu)$		
	$\xrightarrow{f_\mu}$	
		$\sum_{a \in H} f_\mu(a) \stackrel{?}{=} f_\mu(\alpha_{\mu-1})$
		$f_\mu(\alpha_\mu) \stackrel{?}{=} f(\alpha_1, \ldots, \alpha_\mu)$ with $\alpha_\mu \xleftarrow{\$} \mathbb{F}$

Proposition 1 (Soundness)**.** *Let p be the soundness error of the univariate sumcheck protocol used at each round to check if the sum of f_i over H equals $f_{i-1}(\alpha_{i-1})$. The number*

$$s_{d,\mathbb{F},p}(\mu) \coloneqq \Pr_{\alpha_1,\dots,\alpha_\mu}\left[\langle \widetilde{\mathsf{P}}_{\mu,d,\mathbb{F},H}(f,S) \leftrightarrow \mathsf{V}^f_{\mu,d,\mathbb{F},H}(S)\rangle = \mathsf{accept} \;\middle|\; \sum_{\boldsymbol{a}\in H^\mu} f(\boldsymbol{a}) \neq S\right]$$

satisfies

$$s_{d,\mathbb{F},p}(\mu) \leqslant 1 - \left(1 - \frac{d}{|\mathbb{F}|}\right)^{\mu-1}\left(1 - \max\left(p, \frac{d}{|\mathbb{F}|}\right)\right).$$

When $p \leqslant d/|\mathbb{F}| \leqslant 1$, this is bounded from above by $\mu d/|\mathbb{F}|$.

Proof. We are going to provide a recurrence relation bounding $s_{d,\mathbb{F},p}(\mu)$ in terms of $s_{d,\mathbb{F},p}(\mu-1)$. We assume that the sum of f over H^μ is different from S. First, suppose $\mu \geqslant 2$. During the first round of the protocol, $\widetilde{\mathsf{P}}$ sends a function $\widetilde{f_0}$ which may or may not be equal to the function f_0 defined in the protocol.

- If $\widetilde{f_1} = f_1$, then the sum of $\widetilde{f_1}$ over H^μ is not S, hence the univariate sumcheck of the first round passes with probability at most p.
- If $\widetilde{f_1} \neq f_1$, then
  - either $\widetilde{f_1}(\alpha_1) = f_1(\alpha_1)$, which happens with probability $u \leqslant d/|\mathbb{F}|$ since $\deg(f_1) \leqslant d$,
  - or $\widetilde{f_1}(\alpha_1) \neq f_1(\alpha_1)$, which happens with probability $1-u$. In this case, V accepts with probability at most $s_{d,\mathbb{F},p}(\mu-1)$, since the remainder of the protocol is just a sumcheck for the $(\mu-1)$-variate function $f(\alpha_1, x_2, \dots, x_\mu)$ with an incorrect claimed sum $\widetilde{f_1}(\alpha_1)$.

  Hence when $\widetilde{f_1} \neq f_1$, the probability that V accepts is smaller than or equal to $u \cdot 1 + (1-u)\cdot s_{d,\mathbb{F},p}(\mu-1)$. Since $s_{d,\mathbb{F},p}(\mu-1) \leqslant 1$ and $u \leqslant d/|\mathbb{F}|$, this is bounded from above by
  $$d/|\mathbb{F}| + (1 - d/|\mathbb{F}|)s_{d,\mathbb{F},p}(\mu-1).$$

Taking both of these cases into account, we obtain

$$s_{d,\mathbb{F},p}(\mu) \leqslant \max\left(p, \frac{d}{|\mathbb{F}|} + \left(1 - \frac{d}{|\mathbb{F}|}\right)s_{d,\mathbb{F},p}(\mu-1)\right).$$

When $\mu = 1$, we may consider the same two cases; the probability of the second case is just $d/|\mathbb{F}|$ since V never accepts if $\widetilde{f_1}(\alpha_1) \neq f_1(\alpha_1)$. Hence

$$s_{d,\mathbb{F},p}(1) \leqslant \max\left(p, \frac{d}{|\mathbb{F}|}\right).$$

Consider the sequence $(t_\mu)_{\mu\geqslant 1}$ defined by

$$t_1 = \max(p, d/|\mathbb{F}|)$$

and for all $\mu \geqslant 1$, $t_{\mu+1} = \max(p, d/|\mathbb{F}| + (1-d/|\mathbb{F}|)t_\mu)$. Then $s_{d,\mathbb{F},p}(\mu) \leqslant t_\mu$ for all $\mu \geqslant 1$. Using the fact that for all $x \in [0,1]$, $d/|\mathbb{F}| + (1-d/|\mathbb{F}|)x \geqslant x$, one can easily show that:

- If $p \leqslant d/|\mathbb{F}|$ then $t_1 = d/|\mathbb{F}|$, and for all $\mu \geqslant 1$,
  $$t_\mu = 1 - \left(1 - \frac{d}{|\mathbb{F}|}\right)^\mu.$$

- If $p > d/|\mathbb{F}|$ then $t_1 = p$ and for all $\mu \geqslant 1$,

$$t_\mu = 1 - \left(1 - \frac{d}{|\mathbb{F}|}\right)^{\mu-1}(1-p).$$

The result follows immediately. □

3 A sumcheck protocol with logarithmic round complexity

Consider a finite field $\mathbb{F}$, and a subset H of $\mathbb{F}$. In this section, we describe a sumcheck protocol for polynomials in $\mu = 2^m$ variables. We still denote by $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ the space of μ-variate polynomials with coefficients in $\mathbb{F}$, of partial degree in each variable bounded by d and total degree bounded by $D \leqslant d^\mu$. Let $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$, and $S \in \mathbb{F}$ be the claimed value of the sum of all evaluations of f over H^μ. We first describe a somewhat crude but easily understandable version of the protocol. After that, we present the genuine protocol.

3.1 A simplified version of the protocol

The simple protocol DCS_μ described below showcases the core idea of our construction. It takes as inputs a μ-variate polynomial $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ and a value $S \in \mathbb{F}$, and it recursively checks the assertion

$$\sum_{\boldsymbol{a} \in H^\mu} f(\boldsymbol{a}) = S.$$

We will denote by $\mathsf{DCS}_\mu[f, S]$ the execution of the protocol DCS_μ on the inputs f, S, which will be refined later in order to achieve a better communication complexity.

Base case For $\mu = 1$ (*i.e.* $m = 0$), the polynomial f is univariate. In that case, $\mathsf{DCS}_1[f, S]$ is just the verifier checking by hand that $\sum_{a \in H} f(a) = S$. If H has a particular structure, this may be replaced with another univariate sumcheck protocol (see §3.3.2).

General case For $\mu \geqslant 2$, $\mathsf{DCS}_\mu[f, S]$ recursively calls $\mathsf{DCS}_{\mu/2}$ as described below.

A few observations At each round, the number of parallel executions of the protocol doubles, but the number of variables of the functions involved is halved. So after i rounds of DCS_μ, there are 2^i parallel instances of $\mathsf{DCS}_{\mu/2^i}$, which is a sumcheck protocol for 2^{m-i}-variate polynomials. Thus, protocol DCS_μ has $\log_2(\mu)$ rounds, and ends with μ univariate sumchecks. In order to reduce the randomness and communication complexity, the verifier may use the same randomness $\boldsymbol{\alpha}$ for every parallel execution of the protocol.

The relations between the different functions appearing in a full execution of DCS_μ can be represented by the tree in Figure 1. The solid edges lead to functions which are computed and sent by the prover, while the dashed edges lead to functions which are implicitly defined during the protocol but not actually computed, and on which the prover has no influence.

Let us now provide an intuitive explanation of the soundness of DCS_μ, as well as an example.

Protocol 2: DCS_μ

Parameters: field $\mathbb{F}$, arity $\mu = 2^m$, degrees d and D and $H \subseteq \mathbb{F}$.
Inputs: $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ and $S \in \mathbb{F}$.

$\mathsf{P}_{\mathbb{F},H}(f, S)$ $\qquad\qquad$ $\mathsf{V}^f_{\mathbb{F},H}(S)$

Compute for $\boldsymbol{x} = \boldsymbol{x}_{1:\mu/2}$

$$f_0(\boldsymbol{x}) = \sum_{\boldsymbol{a} \in H^{\mu/2}} f(\boldsymbol{x}, \boldsymbol{a})$$

$\xrightarrow{\ f_0\ }$

$\xleftarrow{\ \boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}^{\mu/2}\ }$

Both set

- $f_1(\boldsymbol{x}) = f(\boldsymbol{\alpha}, \boldsymbol{x})$ (which can be accessed by V as a virtual oracle when needed),
- $S_1 = f_0(\boldsymbol{\alpha})$ (which can be requested by V when needed),

and then perform in parallel $\mathsf{DCS}_{\mu/2}[f_0, S]$ and $\mathsf{DCS}_{\mu/2}[f_1, S_1]$.

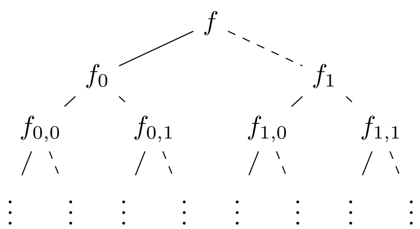

Figure 1: The tree of functions involved in DCS_μ for $\mu \in \{8, 4, 2\}$. The children with dashed line from their parents are not computed by P and are dealt as virtual oracles in the protocol.

Intuition behind the soundness of DCS_μ In the standard sumcheck protocol, the verifier checks one univariate sum at each round, which ties the sums of the functions sent by the prover to the claimed sum of the function f. With only these checks however, the prover could send any functions which have the right sum. This is why the verifier performs one final evaluation check which ties the functions sent by the prover to the function f itself. In our protocol, these goals are achieved in a different way: the sum of the function f_0 sent by the prover is that of f, while the sum of f_1 (a function which is not sent by the prover, but computed directly from f) ties f_0 to f. The soundness error of DCS_μ is computed in a similar way to that of the classical sumcheck protocol: at every round, there is a probability $D/|\mathbb{F}|$ that the function sent by the prover accidentally has the same evaluation as the function required by the protocol. The total soundness is $O(\log(\mu)D/|\mathbb{F}|)$. A precise proof of this will be given later for the refined protocol $\mathsf{Fold\text{-}DCS}$. The following example illustrates the soundness in a simple case.

Example 1. Consider the function $f(x, y) = x + y \in \mathbb{F}_3[x, y]$, and the set $H = \{0, 1\} \subset \mathbb{F}_3$. We have

$$\sum_{a,b \in H} f(a, b) = 1.$$

Consider a claimed sum $S = 0 \neq 1$. The protocol $\mathsf{DCS}_1[f, S]$ asks the prover P to send one linear function $\widetilde{f}_0(x) = rx + t$ with $r,\, t \in \mathbb{F}_3$. Let us find the couples $(r, t) \in (\mathbb{F}_3)^2$ which maximize the probability that V accepts. The verifier picks $\alpha \in \mathbb{F}_3$ and checks that

$$\begin{cases} \widetilde{f}_0(0) + \widetilde{f}_0(1) & = S \\ f(\alpha, 0) + f(\alpha, 1) & = \widetilde{f}_0(\alpha) \end{cases}$$

These two verifications amount to the following linear system in the variables r, t over $\mathbb{F}_3$.

$$\begin{cases} r + 2t & = S \\ \alpha + \alpha + 1 & = r\alpha + t \end{cases} \iff \begin{cases} r - t & = S \\ r\alpha + t & = -\alpha + 1 \end{cases}$$

which since $S = 0$, is equivalent to the following

$$\begin{cases} r & = t \\ (1 + \alpha)t & = 1 - \alpha \end{cases}$$

If $\alpha = -1$, this system has no solution, the second equation being "$0 = 2$". If $\alpha = 1$, the only solution is $(r, t) = (0, 0)$. If $\alpha = 0$, the only solution is $(r, t) = (1, 1)$. Hence the best possible strategy for the prover P is to pick $t \in \{0, 1\}$ and send $\widetilde{f}_0^{(1)} = tx + t$. In this case, the verifier V accepts if and only if $\alpha = 1 - t$. So the probability of V accepting is $1/3$ when α is uniformly random in $\mathbb{F}_3$.

3.2 The protocol Fold-DCS

Let us set the notations for this section: $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ is the tested function, $H \subset \mathbb{F}$ is the evaluation set, and $S \in \mathbb{F}$ is the claimed sum. We describe Fold-DCS in Protocol 3.

3.2.1 Folding for better complexity

One of the drawbacks of both the standard and our sumcheck protocol DCS is the fact that the verifier needs to perform as many univariate sumchecks as there are variables. The protocol DCS may be improved in order to require the verifier to perform only a single univariate sumcheck $\mathsf{US}_{d,H}$ of a degree-d polynomial over H at the end. This is done using a folding technique. Each step of protocol DCS consists in splitting one 2^m-variate sumcheck into two 2^{m-1}-variate sumchecks; replacing these two sumchecks with a linear combination of the two allows to keep just one function at each step of the protocol (see Figure 2).

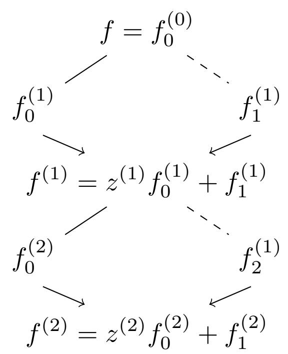

Figure 2: Tree of functions involved in the first two rounds of the protocol Fold-DCS.

Remark 2. This folding technique slightly affects the soundness of our protocol compared to the simplified version presented in the previous section. Indeed, even if the function $\widetilde{f}_0^{(1)}$ sent by the prover either does not have the claimed sum or does not have the right evaluation at the random point chosen by the verifier, the random linear combination $f^{(1)}$ might still have the correct sum. This happens with probability $1/|\mathbb{F}|$. We will see in the proof of Proposition 3 that, at each round, this quantity is added to the probability that the resulting function has the claimed sum. This roughly implies adding $\log(\mu)/|\mathbb{F}|$ to the overall soundness error of the protocol.

Protocol 3: Fold-DCS between $\mathsf{P} = \mathsf{P}_{\mu,\mathbb{F},H}(f, S)$ and $\mathsf{V} = \mathsf{V}^f_{\mu,\mathbb{F},H}(S)$

Parameters: field $\mathbb{F}$, arity $\mu = 2^m$ with $m \geqslant 1$, degrees d and D and $H \subseteq \mathbb{F}$.
Inputs: $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ and $S \in \mathbb{F}$.
Commit phase:
Initialisation: $f^{(0)} = f$ and $S^{(0)} = S$.

1. for $i \in \{1, \dots, m\}$:
   (a) P computes $f_0^{(i)} = \sum_{\boldsymbol{a} \in H^{2^{m-i}}} f^{(i-1)}(\,\cdot\,, \boldsymbol{a})$.
   (b) P gives oracle access to the 2^{m-i}-variate polynomial $f_0^{(i)}$.
   (c) V picks $\boldsymbol{\alpha}^{(i)} \xleftarrow{\$} \mathbb{F}^{[m-i]}$ and $z^{(i)} \xleftarrow{\$} \mathbb{F}$ and sends them to P.
   (d) Set the polynomials $f_1^{(i)} = f^{(i-1)}(\boldsymbol{\alpha}^{(i)}, \,\cdot\,)$ and $f^{(i)} = z^{(i)} f_0^{(i)} + f_1^{(i)}$, and the value $S^{(i)} = z^{(i)} S^{(i-1)} + f_0^{(i)}(\boldsymbol{\alpha}^{(i)})$.
2. P gives oracle access to the univariate polynomial $f^{(m)}$.

Query phase:

1. V computes $S^{(m)}$ by
   - querying $f_0^{(j)}(\boldsymbol{\alpha}^{(j)})$ for $j \in \{1, \dots, m\}$,
   - using the formula $S^{(m)} = \prod_{j=1}^{m} z^{(j)} S + \sum_{j=1}^{m} \left(\prod_{\ell=j+1}^{m} z^{(\ell)} \right) f_0^{(j)}(\boldsymbol{\alpha}^{(j)})$.
2. V checks the consistency of $f^{(m)}$ by
   - picking $\beta \xleftarrow{\$} \mathbb{F}$,
   - querying $f^{(m)}(\beta)$, $f(\boldsymbol{\alpha}^{(1)}, \dots, \boldsymbol{\alpha}^{(m)}, \beta)$ and $f_0^{(j)}(\boldsymbol{\alpha}^{(j+1)}, \dots, \boldsymbol{\alpha}^{(m)}, \beta)$ for $j \in \{1, \dots, m\}$, and
   - verifying $f^{(m)}(\beta) = f(\boldsymbol{\alpha}^{(1)}, \dots, \boldsymbol{\alpha}^{(m)}, \beta) + \sum_{j=1}^{m} z^{(j)} f_0^{(j)}(\boldsymbol{\alpha}^{(j+1)}, \dots, \boldsymbol{\alpha}^{(m)}, \beta)$.
3. V checks $\sum_{a \in H} f^{(m)}(a) \stackrel{?}{=} S^{(m)}$ via the univariate sumcheck $\mathsf{US}_{d,H}(f^{(m)}, S^{(m)})$.

3.2.2 Completeness and soundness

In this section, we prove that our protocol Fold-DCS is perfectly complete, and that is soundness error is logarithmic in the number of variables. We recall the notations: f is a $\mu = 2^m$-variate polynomial with coefficients in a field $\mathbb{F}$. For $i \in \{0, \dots, m\}$, we set $\mu_i = 2^{m-i}$. The subset of $\mathbb{F}$ over which the sums are computed is denoted by H.

Proposition 2 (Completeness)**.** *We suppose that the univariate sumcheck protocol used at the last round of* Fold-DCS *is perfectly complete. If* $\sum_{\boldsymbol{a} \in H^\mu} f(\boldsymbol{a}) = S$ *then, given an honest prover* P*,*

$$\Pr_{\boldsymbol{\alpha}^{(1)}, \dots, \boldsymbol{\alpha}^{(m)}} \left[\langle \mathsf{P}_{\mu,d,\mathbb{F},H}(f, S) \leftrightarrow \mathsf{V}^f_{\mu,d,\mathbb{F},H}(S) \rangle = \mathsf{accept} \right] = 1.$$

Proof. We prove the result by induction on $m = \log_2(\mu)$. The base case $m = 0$ is true, since we suppose that the univariate sumcheck protocol is perfectly complete. For $m > 0$, it is enough to prove that for every $i \in \{0, \dots, m-1\}$, if the sum of $f^{(i)}$ over H^{μ_i} is $S^{(i)}$, then the sum of $f^{(i+1)}$ over $H^{\mu_i/2}$ is $S^{(i+1)}$. We have

$$\begin{aligned}
\sum_{\boldsymbol{a} \in H^{\mu_i/2}} f^{(i+1)}(\boldsymbol{a}) &= z^{(i+1)} \sum_{\boldsymbol{a} \in H^{\mu_i/2}} f_0^{(i+1)}(\boldsymbol{a}) + \sum_{\boldsymbol{a} \in H^{\mu_i/2}} f_1^{(i+1)}(\boldsymbol{a}) \\
&= z^{(i+1)} \sum_{\boldsymbol{a} \in H^{\mu_i/2}} \sum_{\boldsymbol{b} \in H^{\mu_i/2}} f^{(i)}(\boldsymbol{a}, \boldsymbol{b}) + \sum_{\boldsymbol{a} \in H^{\mu_i/2}} f^{(i)}(\boldsymbol{\alpha}, \boldsymbol{a}) \\
&= z^{(i+1)} \sum_{\boldsymbol{a} \in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) + \sum_{\boldsymbol{a} \in H^{\mu_i/2}} f^{(i)}(\boldsymbol{\alpha}, \boldsymbol{a}) \\
&= z^{(i+1)} S^{(i)} + f_0^{(i+1)}(\boldsymbol{\alpha}) \\
&= S^{(i+1)}.
\end{aligned}$$

□

Next, we study the soundness error of our protocol. We recall that the soundness error of the classical protocol is $\mu d / |\mathbb{F}|$, where d is a bound on the partial degrees of the given polynomial. That of Fold-DCS, however, is bounded by $\log(\mu) D / |\mathbb{F}|$, where D is the total degree of the polynomial. Hence, Fold-DCS offers a better soundness as long as the total degree of the polynomial does not far exceed its partial degrees.

Proposition 3 (Soundness)**.** *Denote by p the soundness error of the univariate sumcheck protocol executed at the end of protocol* Fold-DCS*. Let $\mu = 2^m$ for a positive integer m. The soundness error of* Fold-DCS *for μ-variate polynomials with coefficients in $\mathbb{F}$ of total degree $\leqslant D$ is bounded above by*

$$1 - \left(1 - \left(\frac{D+1}{|\mathbb{F}|} - \frac{D}{|\mathbb{F}|^2}\right)\right)^m \left(1 - \max\left(p, \frac{D}{|\mathbb{F}|}\right)\right).$$

When $p \leqslant (D+1)/|\mathbb{F}| \leqslant 1$, this is bounded from above by $(m+1)(D+1)/|\mathbb{F}|$.

Proof. We consider an instance where $\sum_{\boldsymbol{a} \in H^\mu} f(\boldsymbol{a}) \neq S$.

Notations. Set $f^{(0)} = f$ and $S^{(0)} = S$. For $i \geqslant 1$, denote by $\widetilde{f}_0^{(i)}$ the function $\widetilde{\mathsf{P}}$ actually sends during round i. Denote by $f_0^{(i)}$ and $f_1^{(i)}$ the functions as defined in the protocol computed from $f^{(i-1)}$, set $S^{(i)} = z^{(i)} S^{(i-1)} + \widetilde{f}_0^{(i)}(\boldsymbol{\alpha}^{(i)})$, and $f^{(i)} = z^{(i)} \widetilde{f}_0^{(i)} + f_i^{(1)}$ the function used in the next rounds. Write $\mu_i = 2^{m-i}$ for the arity of the functions superscripted by (i).

This soundness proof is divided into two steps.

1. We first deal with the commit phase. At each round, we give an upper bound on the probability that the sum of the function $f^{(i)}$ considered in this round has the claimed value $S^{(i)}$. This yields an upper bound on the probability that the sum of the last function $f^{(m)}$ considered in the protocol has the sum $S^{(m)}$.

2. We then consider what happens in the query phase of the protocol.

Commit phase. We begin by proving by induction that for all $i \in \{0 \dots m\}$, the number

$$s^{(i)} = \Pr_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(i)}} \left[\sum_{\boldsymbol{a} \in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) \neq S^{(i)} \right]$$

satisfies

$$s^{(i)} \geqslant \left(1 - \left(\frac{D+1}{|\mathbb{F}|} - \frac{D}{|\mathbb{F}|^2} \right) \right)^i .$$

We know that $s^{(0)} = \Pr[\sum_{\boldsymbol{a} \in H^{\mu}} f(\boldsymbol{a}) \neq S] = 1$. Let $i \geqslant 1$. Let us compute the probability

$$\Pr \left[\sum_{\boldsymbol{a} \in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) = S^{(i)} \,\middle|\, \sum_{\boldsymbol{a} \in H^{\mu_{i-1}}} f^{(i-1)}(\boldsymbol{a}) \neq S^{(i-1)} \right]$$

using the law of total probability with respect to the event "$\widetilde{f}_0^{(i)} = f_0^{(i)}$" and its complement.

(A) In case $\widetilde{f}_0^{(i)} = f_0^{(i)}$, its sum over H^{μ_i} is not $S^{(i-1)}$. Then the sum of $f^{(i)} = z^{(i)} \widetilde{f}_0^{(i)} + f_1^{(i)}$ over H^{μ_i} is $S^{(i)} = z^{(i)} S^{(i-1)} + \widetilde{f}_0^{(i)}(\boldsymbol{\alpha}^{(i)})$ with probability $1/|\mathbb{F}|$.

(B) In case $\widetilde{f}_0^{(i)} \neq f_0^{(i)}$,

- $\widetilde{f}_0^{(i)}(\boldsymbol{\alpha}^{(i)})$ coincides with $f_0(\boldsymbol{\alpha}^{(i)})$ with probability say $v \leqslant D/|\mathbb{F}|$ by the Schwartz-Zippel Lemma (see Lemma 1);
- if it does not, the sum of $f^{(i)} = z^{(i)} \widetilde{f}_0^{(i)} + f_1^{(i)}$ over H^{μ_i} coincides with $S^{(i)} = z^{(i)} S^{(i-1)} + \widetilde{f}_0^{(i)}(\boldsymbol{\alpha}^{(i)})$ with probability $1/|\mathbb{F}|$.

Hence, setting

$$w = \Pr \left[\sum_{\boldsymbol{a}} f^{(i)}(\boldsymbol{a}) = S^{(i)} \,\middle|\, \left(\sum_{\boldsymbol{a}} f^{(i-1)}(\boldsymbol{a}) \neq S^{(i-1)} \right) \wedge \left(\widetilde{f}_0^{(i)} \neq f_0^{(i)} \right) \right], \quad (1)$$

we get that

$$w = v + (1-v)/|\mathbb{F}| \leqslant \frac{D+1}{|\mathbb{F}|} - \frac{D}{|\mathbb{F}|^2} =: A.$$

Since $w \geqslant 1/|\mathbb{F}|$, the sum of $f^{(i)}$ equals $S^{(i)}$ with probability less than w in each of these two cases so

$$\Pr \left[\sum_{\boldsymbol{a} \in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) = S^{(i)} \,\middle|\, \sum_{\boldsymbol{a} \in H^{\mu_{i-1}}} f^{(i-1)}(\boldsymbol{a}) \neq S^{(i-1)} \right] \leqslant w.$$

Hence

$$\begin{aligned} 1 - s^{(i)} &= \Pr_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(i)}}\left[\sum_{\boldsymbol{a}\in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) = S^{(i)}\right] \\ &\leqslant (1 - s^{(i-1)})\cdot 1 + s^{(i-1)}\cdot w && \text{(by the law of total probability)} \\ &\leqslant 1 - (1-A)^{i-1} + (1-A)^{i-1} w && \text{(since } A \leqslant 1\text{)} \\ &= 1 - (1-A)^{i} && \text{(since } w \leqslant A\text{)} \end{aligned}$$

from which we deduce that

$$s^{(i)} \geqslant (1-A)^i.$$

Query phase. Now, let us come to the last steps of the protocol. With probability $s^{(m)}$, we have $\sum_{a\in H} f^{(m)}(a) \neq S^{(m)}$. Denote by $\widetilde{f}^{(m)}$ the function sent by $\widetilde{\mathsf{P}}$, which would be equal to $f^{(m)}$ if the prover were honest.

(A) If $\widetilde{f}^{(m)} = f^{(m)}$, then $\sum_{a\in H} \widetilde{f}^{(m)} \neq S^{(m)}$, and V accepts if and only if the univariate sumcheck on $\widetilde{f}^{(m)}$ (Step 3) passes, which happens with probability p.

(B) If $\widetilde{f}^{(m)} \neq f^{(m)}$, then for V to accept, the evaluations of $\widetilde{f}^{(m)}$ and $f^{(m)}$ at β need to coincide (Step 2), which happens with probability at most $D/|\mathbb{F}|$.

In total,

$$\begin{aligned} \Pr_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(m)}}\left[\langle \widetilde{\mathsf{P}}_{\mu,D,\mathbb{F},H}(f,S) \leftrightarrow \mathsf{V}^{f}_{\mu,D,\mathbb{F},H}(S)\rangle = \mathsf{accept}\right] &\leqslant \left(1 - s^{(m)}\right) + s^{(m)} \max\left(p, \frac{D}{|\mathbb{F}|}\right) \\ &\leqslant 1 - (1-A)^m\left(1 - \max\left(p, \frac{D}{|\mathbb{F}|}\right)\right). \end{aligned}$$

□

Remark 3. In most cases, this upper bound on the soundness is tight. The best strategy for a malicious prover can be deduced from the proof, and is similar to that used in the standard protocol: at each step, send a function which has the claimed sum. However, there are a few rare instances in which this strategy is not possible. Consider the following example. The field $\mathbb{F}$ has characteristic 2, the set H has even cardinality, and f is a linear polynomial in 4 variables. Then the sum of f over H^4 is necessarily 0. During the first round of our protocol, the prover sends a linear function in 2 variables: such a function always sums to 0 over H^2. Hence, if they want to convince a verifier that the sum is anything but 0, they cannot implement the optimal strategy at the first round, and they actually have at best a chance of 1/2 of convincing the verifier.

3.2.3 A detailed example

Let us write out the protocol for a polynomial $f \in \mathbb{F}[\boldsymbol{x}_{1:4}]$. Here, $m = 2$ so the protocol has two rounds.

- Round 1: The prover P computes

$$f_0^{(1)}(x_1,x_2) = \sum_{a_3,a_4\in H} f(x_1,x_2,a_3,a_4)$$

and sends it to V. The verifier V picks $\alpha_1^{(1)}, \alpha_2^{(1)}, z^{(1)} \in \mathbb{F}$ at random and sends them to P. Both P and V implicitly define the function

$$f_1^{(1)}(x_3, x_4) = f\left(\alpha_1^{(1)}, \alpha_2^{(1)}, x_3, x_4\right)$$

which V can access via f, knowing $\alpha_1^{(1)}, \alpha_2^{(1)}$, as well as

$$\begin{aligned} f^{(1)} &= z^{(1)} f_0^{(1)} + f_1^{(1)} \\ S^{(1)} &= z^{(1)} S + f_0^{(1)}\left(\alpha_1^{(1)}, \alpha_2^{(1)}\right). \end{aligned}$$

- Round 2: The prover P computes

$$f_0^{(2)}(x) = \sum_{a \in H} f^{(1)}(x, a)$$

and sends it to V, who then chooses $\alpha_1^{(2)}, z^{(2)} \in \mathbb{F}$ at random and implicitly defines

$$f_1^{(2)}(x) = f^{(1)}\left(\alpha_1^{(2)}, x\right)$$

as well as

$$\begin{aligned} f^{(2)}(x) &= z^{(2)} f_0^{(2)}(x) + f_1^{(2)}(x) \\ &= z^{(2)} f_0^{(2)}(x) + z^{(1)} f_0^{(1)}\left(\alpha_1^{(2)}, x\right) + f\left(\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha_1^{(2)}, x\right) \end{aligned}$$

and

$$\begin{aligned} S^{(2)} &= z^{(2)} S^{(1)} + f_0^{(2)}\left(\alpha_1^{(2)}\right) \\ &= z^{(2)}\left(z^{(1)} S + f_0^{(1)}\left(\alpha_1^{(1)}, \alpha_2^{(1)}\right)\right) + f_0^{(2)}\left(\alpha_1^{(2)}\right). \end{aligned}$$

- Final sumcheck: The verifier V checks whether

$$\sum_{a \in H} f^{(2)}(a) = S^{(2)}.$$

This last sumcheck requires computing $S^{(2)}$ as described by the formula above, using oracle queries to $f_0^{(1)}, f_0^{(2)}$. The actual computation of the sum is facilitated by requiring P to give oracle access to $f^{(2)}$ to V, who checks its correctness by choosing $\beta \in \mathbb{F}$ and verifying the equality

$$f^{(2)}(\beta) = z^{(2)} f_0^{(2)}(\beta) + z^{(1)} f_0^{(1)}\left(\alpha_1^{(2)}, \beta\right) + f\left(\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha_1^{(2)}, \beta\right).$$

This in turn requires oracle queries to $f, f_0^{(1)}, f_0^{(2)}, f^{(2)}$.

3.3 Complexities of Fold-DCS in the ROM

Here, we consider our protocol Fold-DCS for $\mu = 2^m$-variate polynomials of partial degree at most d and total degree at most D over a finite field $\mathbb{F}$.

3.3.1 Complexities without the last univariate sumcheck

We first compute the complexities without taking the last univariate sumcheck into account.

Round complexity Each loop of Fold-DCS runs in one round. There are thus $\log(\mu)+1$ rounds.

Randomness At the i^{th} loop of Fold-DCS, the verifier V picks a random 2^{m-i}-tuple $\boldsymbol{\alpha}^{(i)}$ of elements of $\mathbb{F}$, as well as a random element $z \in \mathbb{F}$. This amounts to $\mu + \log\mu$ random elements during the commitment phase. In addition, at Step 2, V picks an element of $\mathbb{F}$. The total randomness is $\mu + \log\mu + 1$.

Communication complexity The messages sent to P by V are exactly the $\mu+\log\mu+1$ random elements she picks along the loops. The prover's messages will be commitments of the $\log\mu + 1$ polynomials he sends.

Queries During the query phase, the verifier V queries $\log\mu$ evaluations at Step 1 to compute $S^{(m)}$ and $\log\mu + 2$ evaluations at Step 2 check the value of $f^{(m)}(\beta)$. In total V makes $\mathsf{q} = 2(\log(\mu)+1)$ queries.

Remark 4. Note that the evaluations queried for computing $S^{(m)}$ and $f^{(m)}(\beta)$ at Steps 1 and 2 can be batched using the sole evaluation point $(\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(m)}, \beta) \in \mathbb{F}^{\mu}$. For $S^{(m)}$, we evaluate the polynomials $f_0^{(1)}(x_1, \ldots, x_{\mu/2})$, $f_0^{(2)}(x_{\mu/2+1}, \ldots, x_{3\mu/4})$, …, and $f_0^{(m)}(x_{\mu-1})$, whereas for $f^{(m)}(\beta)$, we evaluate $f_0^{(i)}$ as polynomials in the last $\mu/(2^i)$ variables (i.e. $f_0^{(i)}(x_{\mu-\mu/2^i+1}, \ldots, x_\mu)$).

Prover complexity The predominant computations on the prover's side are the ones performed in the loops. At the i^{th} loop, P compute sums over $H^{2^{m-i}} = H^{\mu_i}$.

Write

$$f^{(i)}(\boldsymbol{x}) = \sum_{j_1,\ldots,j_{\mu_i}} \lambda_{j_1,\ldots,j_{\mu_i}} \prod_{k=1}^{\mu_i} x_k^{j_k}.$$

Then

$$\sum_{\boldsymbol{a}\in H^{2^{m-i}}} f^{(i)}(\boldsymbol{a}) = \sum_{j_1,\ldots,j_{\mu_i}} \lambda_{j_1,\ldots,j_{\mu_i}} \prod_{k=1}^{\mu_i} \sigma_{j_k} \tag{2}$$

where

$$\sigma_j = \sum_{a\in H} a^j.$$

All the σ_j can be simultaneously computed in $d\,|H|$ additions and $d\,|H|$ multiplications in $\mathbb{F}_q$. As the polynomial $f^{(i)}$ has partial degree d in each variable, the number of terms in (2) is bounded from above by d^{μ_i}. Each term can be computed using μ_i multiplications in $\mathbb{F}_q$. Knowing the sums σ_j, the total number of $\mathbb{F}_q$-operations to compute the sum of $f^{(i)}$ over H^{μ_i} is $\mu_i d^{\mu_i}$. Summing over the m rounds and bounding each term by the largest one, we get

$$\sum_{i=1}^{m} \mu_i d^{\mu_i} \leqslant m\frac{\mu}{2} d^{\mu/2}$$

and the overall complexity is $O(m\mu d^{\mu/2} + d\,|H|)$ $\mathbb{F}_q$-operations.

Verifier complexity The verifier V computes, in the end, two linear combinations of these evaluations. The coefficients of this linear combination are products of field elements; in total, there are $\log\mu$ products to compute the products of the $z^{(i)}$ as well as $2\log\mu$ sums and $2\log\mu+1$ products to compute the linear combinations. This amounts to $2\log\mu$ sums and $3\log\mu+1$ products in the field $\mathbb{F}$.

3.3.2 Total complexities

Table 2: Total complexities of Fold-DCS including the ones of $\mathsf{US}_{d,H}$. For unstructured H, $\mathsf{US}_{d,H}$ is performed by V without the prover's help. For H coset of $(\mathbb{F}^\times, \times)$ or $(\mathbb{F}, +)$, V and P perform Aurora's sumcheck protocol [BSCR+19].

	Fold-DCS without US_d	Fold-DCS + $\mathsf{US}_{d,H}$ Unstructured H	Fold-DCS + $\mathsf{US}_{d,H}$ H coset
Round	$\log \mu + 1$	$\log \mu + 1$	$\log \mu + 2$
Randomness in $\mathbb{F}$	$\mu + \log \mu + 1$	$\mu + \log \mu + 1$	$\mu + \log \mu + 2$
Communication	$\mu + \log \mu$	$\mu + \log \mu + \|H\|$	$\mu + \log d$
Number of commitments	$\log \mu + 1$	$\log \mu + 1$	$\log \mu + 3$
Queries	$2(\log \mu + 1)$	$2(\log \mu + 1) + \|H\|$	$2(\log(\mu) + 2)$
Prover's time (op. in $\mathbb{F}$)	$O(\log(\mu)\mu d^{\mu/2} + d\,\|H\|)$	$O(\log(\mu)\mu d^{\mu/2} + d\,\|H\|)$	$O(\log(\mu)\mu d^{\mu/2} + d\,\|H\|)$
Verifier's time (op. in $\mathbb{F}$)	$5\log \mu + 1$	$5\log \mu + \|H\|$	$O(\log(\mu d) + \log \|H\|)$

Univariate sumcheck protocols Univariate sumcheck protocols are protocols for the language

$$\mathcal{L}_{d,\mathbb{F},H} = \left\{ (f, S) \in \mathbb{F}[x]_d \times \mathbb{F} \;\middle|\; \sum_{a \in H} f(a) = S \right\}$$

in which the verifier V has oracle access to f. By default, the verifier may just query $|H|$ values of f and compute the sum. However, in order to reduce the number of queries, there are better options in specific cases. In particular, when H is a coset modulo a subgroup of either $(\mathbb{F}, +)$ of $(\mathbb{F}^\times, \times)$, such protocols may be found in Aurora [BSCR+19, §5]. The resulting PIOP for the sumcheck relation, described in detail in [ACY23, §6.1] runs in one round. The prover gives access to two polynomials, which the verifier queries at a random element of $\mathbb{F}$. The verifier performs $O(\log |H|)$ field operations.

4 Instantiating Fold-DCS with a polynomial commitment scheme

In order to instantiate the oracle accesses in Fold-DCS, we may use Fold-DCS with a polynomial commitment scheme (PCS) for μ-variate polynomials.

4.1 Polynomial commitment schemes

Let us define a PCS as needed for Fold-DCS.

Definition 6. A μ-variate (d, D)-degree polynomial commitment scheme (PCS) is a quadruple (Setup, Commit, Open, Eval) that satisfies the following properties.

- Setup $\left(1^\lambda, \mu, d, D\right)$ generates public parameters pp (a structured reference string) suitable to commit to polynomials in $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$.
- Commit (pp, f) outputs a commitment C to the polynomial $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$, using pp.
- Open (pp, f, C) checks if the commitment C is correctly computed from the polynomial $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ using pp.
- Eval is a (public-coin) protocol between two parties, a prover P_{PC} and a verifier V_{PC} that either accepts or rejects. The prover is given a polynomial $f \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$. Both parties receive the following:

– the security parameter λ, the arity μ and the degrees d and D,
– the public parameters pp , where $\mathsf{pp} = \mathsf{Setup}\left(1^\lambda, \mu, d, D\right)$,
– an evaluation point x and the alleged opening y,
– the alleged commitment C for the polynomial f.

The protocol Fold-DCS for $\mu = 2^m$-variate polynomials in the polynomial IOP model requires V to query $2\log\mu + 2$ polynomial evaluations. As the partial and total degrees do not increase through the protocol, the same commitment scheme for μ-variate polynomials may be used throughout the protocol. In particular, if the PCS in question requires a trusted setup, this may be dealt with beforehand. Moreover, as noted in Remark 4, it is possible for V to get all theses evaluations at one by interacting with P via a batched-evaluation protocol, which we will recall here.

Definition 7. A μ-variate (d, D)-degree PCS as in Definition 6 allows *batched evaluation* if for every positive integer ℓ, there exists a two-party protocol ℓ-Eval which takes as input an ℓ-tuple $(f_1, \ldots, f_\ell)$ of polynomials and provides both parties with the following:

- the security parameter λ, the arity μ and the degrees d and D,
- the public parameters pp , where $\mathsf{pp} = \mathsf{Setup}\left(1^\lambda, \mu, d, D\right)$.
- An evaluation point x and the alleged openings $y_1, \ldots, y_\ell$,
- the alleged commitments $C_1, \ldots, C_\ell$ for the polynomials $f_1, \ldots f_\ell$.

Definition 8. A function $f\colon \mathbb{N} \to \mathbb{N}$ is said to be negligible if for any positive integer c, there is an integer λ_c such that for any $\lambda \geqslant \lambda_c,\ \ f(\lambda) < \lambda^{-c}$. In that case, we write $f(\lambda) = \mathsf{negl}(\lambda)$.

Definition 9. A μ-variate (d, D)-degree PCS as in Definition 6 is said to be

- *extractable* if for any PPT adversary that computes a valid commitment C, there is a PPT extractor algorithm which, given C, produces a function f that opens C with overwhelming probability. Formally, for any PPT adversary $\widetilde{P}$, there exists a PPT algorithm $E_{\widetilde{P}}$ such that

$$\Pr\left[\begin{array}{cc|l} & \exists g\colon C = \mathsf{Commit}(\mathsf{pp}, g) & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mu, d, D) \\ \wedge & \mathsf{Open}(\mathsf{pp}, f, C) = \mathsf{reject} & C \leftarrow \widetilde{P}(\mathsf{pp}) \\ & & f \leftarrow E_{\widetilde{P}}(C, \mathsf{pp}) \end{array}\right] = \mathsf{negl}(\lambda),$$

- *computationally binding* if for any probabilistic polynomial-time (PPT) algorithm A,

$$\Pr\left[\begin{array}{cc|l} & f \neq g & \\ \wedge & \mathsf{Open}(\mathsf{pp}, f, C) = \mathsf{accept} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mu, d, D) \\ \wedge & \mathsf{Open}(\mathsf{pp}, g, C) = \mathsf{accept} & f, g, C \leftarrow A(\mathsf{pp}) \end{array}\right] = \mathsf{negl}(\lambda),$$

- *computationally evaluation-binding* if for any PPT algorithm A and PPT prover $\widetilde{\mathsf{P}}$,

$$\Pr\left[\begin{array}{cc|l} & y \neq y' & \\ \wedge & \langle \widetilde{\mathsf{P}}(C, x, y) \overset{\mathsf{Eval}}{\longleftrightarrow} \mathsf{V}_{\mathsf{PC}}(C, x, y)\rangle = \mathsf{accept} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mu, d, D) \\ \wedge & \langle \widetilde{\mathsf{P}}(C, x, y') \overset{\mathsf{Eval}}{\longleftrightarrow} \mathsf{V}_{\mathsf{PC}}(C, x, y')\rangle = \mathsf{accept} & C, x, y, y' \leftarrow A(\mathsf{pp}) \end{array}\right] = \mathsf{negl}(\lambda).$$

The evaluation-binding property for PCS with batched evaluation of ℓ polynomials is similar: the top line $y \neq y'$ in the probability is replaced by $\exists i \in \{1, \ldots \ell\}, y_i \neq y'_i$.

Remark 5. The extractability condition defined above is strong, and may require working in a model with additional assumptions. For instance, the PCS used in Section 4.2.1 is extractable in the AGM model.

4.2 Soundness and complexity of our protocol with a PCS allowing batching evaluation

In the following, we will use a PCS that allows batched evaluation. We write $\mathsf{t}(\mathsf{P}_{\ell-\mathsf{PC}})$ (resp. $\mathsf{t}(\mathsf{V}_{\ell-\mathsf{PC}})$) for the prover's (resp. verifier's) time complexity for ℓ-Eval. We denote by $\mathsf{rn}(\ell\text{-}\mathsf{Eval})$ the number of rounds of the ℓ-batched evaluation protocol, and by $\mathsf{rand}(\mathsf{Commit})$ and $\mathsf{rand}(\ell\text{-}\mathsf{Eval})$ the amount of random field elements required in Commit and ℓ-Eval. The notation $\mathsf{cc}(\ell\text{-}\mathsf{Eval})$ stands for the communication complexity of the ℓ-batched evaluation protocol.

When instantiating Fold-DCS, we set $\ell = 2(\log \mu + 1)$. We suppose that P begins by sending a commitment $\mathsf{Commit}(f)$ of the initial polynomial f to V. Each time P is supposed to send a polynomial $f^{(i)}$, he now sends $\mathsf{Commit}(f^{(i)})$. At the end of the protocol, V and P engage in the protocol ℓ-Eval for V to get the evaluations she needs to compute $S^{(m)}$ and $f^{(m)}(\beta)$, as explained in Remark 4.

The complexities of the instantiated version of Fold-DCS are thus the sum of the complexities of the IOP protocol and the ones of ℓ-Eval, taking into account the last univariate sumcheck.

In this setting, the soundness of our protocol is no longer statistical soundness, but computational soundness: polynomial commitment schemes usually have a computational evaluation-binding property, meaning that for P to convince V of a false evaluation value, P would have to solve a computationally hard problem.

A simple adaption of the soundness of the polynomial IOP protocol (Proposition 3) gives the soundness of the instantiated version depending on the soundness of the PCS involved.

Corollary 1 (Computational soundness)**.** *Let μ, d, D be positive integers. Let λ be a security parameter. Consider protocol* Fold-DCS *for μ-variate polynomials with coefficients in $\mathbb{F}$ of total degree $\leqslant D$ and partial degree $\leqslant d$, instantiated with a PCS allowing batch-evaluation. This PCS is supposed to be*

- *extractable,*
- *computationally binding,*
- *computationally ℓ-batch evaluation binding, where $\ell = 2\log(\mu) + 1$.*

Denote by p the soundness error of the univariate sumcheck protocol executed at the end of Fold-DCS*. Then, for any probabilistic polynomial-time prover $\widetilde{\mathsf{P}}$, the probability*

$$\Pr_{\alpha_1,\ldots,\alpha_\mu}\left[\langle \widetilde{\mathsf{P}}_{\mu,d,\mathbb{F},H}(f,S) \leftrightarrow \mathsf{V}^f_{\mu,d,\mathbb{F},H}(S)\rangle = \mathsf{accept} \;\middle|\; \sum_{\boldsymbol{a}\in H^\mu} f(\boldsymbol{a}) \neq S\right]$$

is bounded from above by

$$(m+1)\varepsilon(\lambda) + 1 - \left(1 - \left(\frac{D+1}{|\mathbb{F}|} - \frac{D}{|\mathbb{F}|^2}\right)\right)^m \left(1 - \max\left(p, \frac{D}{|\mathbb{F}|}\right) + \sigma(\lambda)\right)$$

where ε and σ are negligible functions. When $p \leqslant D/|\mathbb{F}|$, this is bounded from above by

$$(m+1)\left(\varepsilon(\lambda) + \frac{(D+1)}{|\mathbb{F}|}(1 + \sigma(\lambda))\right).$$

Proof. We need to adapt the proof of Proposition 3.

Commit phase. During the i^{th} round, $\widetilde{\mathsf{P}}$ sends a commitment $\widetilde{C}_i$. As the PCS is extractable and computationally binding, with probability $1-\mathsf{negl}(\lambda)$, exactly one function which opens $\widetilde{C}_i$ can be extracted from $\widetilde{C}_i$. The hybrid argument [MF21, Theorem 3.8] now ensures that there exists a negligible function $\varepsilon(\lambda)$ such that, with probability $1-m\cdot\varepsilon(\lambda)$, for every $i\in\{1\dots m\}$, exactly one function which opens $\widetilde{C}_i$ can be extracted from $\widetilde{C}_i$. We will denote this function by $\widetilde{f}_0^{(i)}$.

In this case, we may still define $f^{(i)}, S^{(i)}$ using $\widetilde{f}_0^{(i)}$ as in the proof of Proposition 3. Then the lower bound for

$$s^{(i)} = \Pr_{\boldsymbol{\alpha}^{(1)},\dots,\boldsymbol{\alpha}^{(i)}}\left[\sum_{\boldsymbol{a}\in H^{\mu_i}} f^{(i)}(\boldsymbol{a}) \neq S^{(i)}\right]$$

does not change, since the two cases corresponding to (A) and (B) are completely unchanged. Note that the definition of $s^{(i)}$ still depends on the actual values of $\widetilde{f}_0^{(i)}$, and not some claimed evaluations.

Query phase. In the present case, $\widetilde{\mathsf{P}}$ sends a commitment $\widetilde{C}$ at the beginning of the query phase, as well as m alleged evaluations y_i at $\boldsymbol{\alpha}^{(i)}$ of the commitments C_i. With probability $1-\varepsilon'(\lambda)$, where ε' is negligible,a unique function $\widetilde{f}^{(m)}$ which opens $\widetilde{C}$ can be extracted from $\widetilde{C}$. Replacing ε with $\max(\varepsilon,\varepsilon')$ if needed, we may suppose that $\varepsilon'\leqslant\varepsilon$. We set

$$\widetilde{S}^{(m)} = \prod_{j=1}^{m} z^{(j)} S + \sum_{j=1}^{m}\left(\prod_{\ell=j+1}^{m} z^{(\ell)}\right) y_j,$$

the value computed by V at Step 1 of the query phase using the alleged evaluations y_i. Since the PCS is computationally ℓ-batch evaluating binding, there is a negligible function σ such that $\Pr(\widetilde{S}^{(m)}\neq S^{(m)})\leqslant\sigma(\lambda)$. Recall that with probability $s^{(m)}$, we have $\sum_{a\in H} f^{(m)}(a)\neq S^{(m)}$.

(A') If $\widetilde{f}^{(m)} = f^{(m)}$, then

- either $\widetilde{S}^{(m)} = S^{(m)}$, and then $\sum_{a\in H}\widetilde{f}^{(m)}\neq S^{(m)}$ so V accepts if and only if the univariate sumcheck on $\widetilde{f}^{(m)}$ passes, which happens with probability p,
- or $\widetilde{S}^{(m)}\neq S^{(m)}$ and then $\sum_{a\in H}\widetilde{f}^{(m)} = \widetilde{S}^{(m)}$ with probability $1/|\mathbb{F}|$, in which case V accepts. And otherwise V accepts if and only if the univariate sumcheck on $(\widetilde{f}^{(m)},\widetilde{S}^{(m)})$ passes.

As a result, in this case, the probability that V accepts is at most

$$\rho = (1-\sigma(\lambda))p + \sigma(\lambda)\left(\frac{1}{|\mathbb{F}|} + \left(1-\frac{1}{|\mathbb{F}|}\right)p\right) = p + \frac{\sigma(\lambda)}{|\mathbb{F}|}(1-p).$$

(B') If $\widetilde{f}^{(m)}\neq f^{(m)}$, then for V to accept, the openings of $\widetilde{f}^{(m)}$ and the evaluations of $f^{(m)}$ at β need to coincide, which happens with probability at most $D/|\mathbb{F}| + \sigma(\lambda)$.

Using the inequality

$$p + \frac{\sigma(\lambda)}{|\mathbb{F}|}(1-p) \leqslant p + \sigma(\lambda)$$

we obtain that, when no two different functions can be extracted for the same commitment, V accepts with probability at most

$$(1 - s^{(m)}) + s^{(m)} \left(\max\left(p, \frac{D}{|\mathbb{F}|}\right) + \sigma(\lambda) \right).$$

The result now follows from the expression of $s^{(m)}$ computed in Proposition 3. □

4.2.1 Instantiation with Zeromorph (tweaked for $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$)

In 2024, Kohrita and Towa [KT24] built a multilinear commitment scheme, *i.e.* for $d = 1$, from any additively homomorphic PCS for *univariate* polynomials, as well as any protocol to check degree bounds on committed polynomials. The construction relies on bilinear pairings. They also instantiate their scheme using the KZG univariate PCS [KZG10] – in a hiding version to ensure zero knowledge, which we do not require here. This instantiated version is computationally binding, ℓ-batch evaluation binding and extractable in the algebraic group model under the DLOG assumption in the bilinear group [KT24, §4, §6]. We propose a tweaked version of Zeromorph, to get a (d, D)-degree PCS, which preserves these properties.

First, (see [Lee21, §2.5] for instance), any μ-variate polynomial of partial degrees $d_1, \ldots, d_\mu$ can be reformulated as a multilinear polynomial in $\sum_{1 \leqslant i \leqslant \mu} \lceil \log_2(d_i + 1) \rceil$ variables. Concretely, in our case, we set $\delta = \lceil \log_2(d+1) \rceil$ and define the linear isomorphism MultiLin between the space $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{\leqslant d}$ of polynomials with partial degrees $\leqslant d$ and the space $\mathbb{F}[y_{i,\ell} \mid 1 \leqslant i \leqslant \mu, 0 \leqslant j < \delta]_{\leqslant 1}$ of multilinear polynomials by

$$\textsc{MultiLin}(x_i^{\alpha_i}) = \prod_{j=0}^{\delta-1} y_{i,j}^{\alpha_{i,j}} \tag{3}$$

using the binary decomposition of the exponent $\alpha_i = \sum_{j=0}^{\delta-1} \alpha_{i,j} 2^j$. This maps the space of polynomials $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ into the set of multilinear polynomials of arity $n = \mu\delta$, which enables us to use the mutilinear PCS Zeromorph to commit to polynomials in $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$. However, for the soundness of Fold-DCS, we need to make sure that the prover can only commit to polynomials of total degree at most D. To achieve this, we shall modify the setup of Zeromorph.

We follow the exposition of [KT24, §2.5]. For any integer n, there is a linear isomorphism $\mathcal{U}_n$ between the vector space of multilinear polynomials $\mathbb{F}[y_0, \ldots, y_{n-1}]_{\leqslant 1}$ in n variables and the space $\mathbb{F}[t]_{<2^n}$ of univariate polynomials of degree less than 2^n defined as

$$\mathcal{U}_n : \begin{cases} \mathbb{F}[y_0, \ldots, y_{n-1}]_{\leqslant 1} & \to \quad \mathbb{F}[t]_{<2^n} \\ \prod_{j=0}^{n-1} (b_j \cdot y_j + (1 - b_j) \cdot (1 - y_j)) & \mapsto \quad \left(t^{2^0}\right)^{b_0} \cdots \left(t^{2^{n-1}}\right)^{b_{n-1}} \end{cases}$$

for any bits $b_j \in \{0, 1\}$. In other words, given an n-variate multilinear polynomial g, we have

$$\mathcal{U}_n(g) = \sum_{(b_0, \ldots, b_{n-1}) \in \{0,1\}^n} g(b_0, \ldots, b_{n-1}) t^{b_0 + 2b_1 + \ldots b_{n-1} 2^{n-1}}$$

Let $\mathcal{F}_{d,D}$ be the image of the monomial basis of $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$ under the composition of the isomorphisms MultiLin and $\mathcal{U}_n$ for $n = \mu \lceil \log_2(d+1) \rceil = \mu\delta$. Given a monomial

$\boldsymbol{x}^{\boldsymbol{\alpha}} = \prod_{i=1}^{\mu} x_i^{\alpha_i} \in \mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$, we have

$$\mathcal{U}_n(\textsc{MultiLin}(\boldsymbol{x}^{\boldsymbol{\alpha}})) = \sum_{\boldsymbol{b}\in\{0,1\}^{\mu\delta}} \prod_{i=1}^{\mu}\prod_{j=0}^{\delta-1} b_{i,j}^{\alpha_{i,j}} t^{2^{(i-1)\delta+j}}.$$

Then

$$\mathcal{F}_{d,D} = \left\{ \sum_{\boldsymbol{b}\in\{0,1\}^{\mu\delta}} \prod_{j=0}^{\delta-1} b_{i,j}^{\alpha_{i,j}} t^{2^{(i-1)\delta+j}} \;\middle|\; \forall i,\ \alpha_i \leqslant d \text{ and } \alpha_1 + \cdots + \alpha_\mu \leqslant D \right\}. \tag{4}$$

Every polynomial encountered in the protocol has total degree $\leqslant D$. To ensure that the prover can only commit to polynomials of total degree at most D, he is given a constrained structured reference string.

Protocol 4: Zeromorph adapted for $\mathbb{F}[\boldsymbol{x}_{1:\mu}]_{d,D}$

$\mathsf{Setup}(1^\lambda, \mu, d, D)$:

- $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathrm{GEN}\left(1^\lambda\right)$
- $\tau, \xi \leftarrow \mathbb{F}^*$
- $srs \leftarrow \left(([g(\tau)]_1)_{g\in\mathcal{F}_{d,D}}, [\xi]_1, ([g(\tau)]_2)_{g\in\mathcal{F}_{d,D}}, [\xi]_2\right)$
- Return $\mathsf{pp} \leftarrow (\mathbb{G}, srs)$.

In Commit, Open and Eval, every instance of KZG.$\mathsf{Commit}(\mathcal{U}_n(\cdot))$ is replaced by KZG.$\mathsf{Commit}(\mathcal{U}_n(\textsc{MultiLin}(\cdot)))$.

Let us study the complexities of this tweaked version.

Each commitment requires $\mathsf{rand}(\mathsf{Commit}) = \mu\delta = \mu(\log(d) + O(1))$ random field elements and $O(d2^\mu)$ field operations on the prover's side. Note that the transformation of a multivariate polynomials into a univariate one via $\mathcal{U}_n(\textsc{MultiLin}(\cdot))$, done on the prover's side, has a negligible computational cost in comparison. The ℓ-batched evaluation protocol ℓ-Eval with $\ell = 2(\log\mu + 1)$ runs in $\mathsf{rn}(\ell\text{-}\mathsf{Eval}) = 3$ rounds (6 moves, where Eval requires 5 moves) and calls for 2 random elements on the prover's side, and 4 on the verifier's one, so $\mathsf{rand}(\ell\text{-}\mathsf{Eval}) = 6$. The prover performs $\mathsf{t}(\mathsf{P}_{\ell-\mathsf{PC}}) = O(d2^\mu)$ field operations, whereas the verifier complexity is $\mathsf{t}(\mathsf{V}_{\ell-\mathsf{PC}}) = O(\mu\log(d))$ in $\mathbb{F}$ since $\ell = o(\mu\log(d))$.

The evaluation protocol is computationally sound: a dishonest prover capable of forging a proof of a false evaluation would be able to solve the discrete logarithm problem in a group where it is hard. The complexities are summed up in Table 1 in the introduction.

Acknowledgement

The authors warmly thank the anonymous referees, whose comments and questions drove them to improve the paper.

References

[ACY23] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. Iops with inverse polynomial soundness error. In *2023 IEEE 64th Annual Symposium on Foundations of*

Computer Science (FOCS), pages 752–761. IEEE, 2023. doi:10.1109/FOCS57990.2023.00050.

[AFK23] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat–Shamir Transformation of Multi-Round Interactive Proofs (Extended Version). *Journal of Cryptology*, 36(4):36, 2023. doi:10.1007/s00145-023-09478-y.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 742–773. Springer, 2021. doi:10.1007/978-3-030-84242-0_26.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 21–32, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103428.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 677–706. Springer, 2020. doi:10.1007/978-3-030-45721-1_24.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael 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)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2018.14.

[BSBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 701–732, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-26954-8_23.

[BSCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for r1cs. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 103–128, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-17653-2_4.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14*, pages 31–60. Springer, 2016. doi:10.1007/978-3-662-53644-5_2.

[CCH+18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-shamir from simpler assumptions. Cryptology ePrint Archive, Paper 2018/1004, 2018. URL: https://eprint.iacr.org/2018/1004.

[DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015. URL: 10.1145/2699436.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for r1cs. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 193–226, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-38545-2_7.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. doi:10.1137/0218012.

[KT24] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. *Journal of Cryptology*, 37(4):38, 2024. doi:10.1007/s00145-024-09519-0.

[KZG10] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 1–34, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-90453-1_1.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992. doi:10.1145/146585.14660.

[MF21] Arno Mittelbach and Marc Fischlin. The theory of hash functions and random oracles. *An Approach to Modern Cryptography, Cham: Springer Nature*, 2021. doi:10.1007/978-3-030-63287-8.

[NST24] Vineet Nair, Ashish Sharma, and Bhargav Thankey. Brakingbase-a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme. *Cryptology ePrint Archive*, 2024. URL: https://eprint.iacr.org/2024/1825.

[Sch80] J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020. doi:10.1007/978-3-030-56877-1_25.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022. doi:10.1561/3300000030.

[WTS+18] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018. doi:10.1109/SP.2018.00060.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 138–169, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-68403-6_5.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg. doi:10.1007/3-540-09519-5_73.