Multivariate lookups based on logarithmic derivatives

Ulrich Haböck

Orbis Labs
`team@orbislabs.com`

November 4, 2022

Abstract

Logarithmic derivatives translate products of linear factors into sums of their reciprocals, turning zeroes into simple poles of same multiplicity. Based on this simple fact, we construct an interactive oracle proof for multi-column lookups over the boolean hypercube, which makes use of a single multiplicity function instead of working with a rearranged union of table and witnesses. For single-column lookups the performance is comparable to the well-known [GW20] strategy used by Hyperplonk+ [CBBZ22]. However, the real power of our argument unfolds in the case of batch lookups when multiple columns are subject to a single-table lookup: While the number of field operations is comparable to the Hyperplonk+ lookup (extended to multiple columns), the oracles provided by our prover are much less expensive. For example, for columns of length 2^{12}, paper-pencil operation counts indicate that the logarithmic derivative lookup is between 1.5 and 4 times faster, depending on the number of columns.

Contents

Introduction

Lookup arguments prove a sequence of values being member of an, often prediscribed, table. They are an essential tool for improving the efficiency of SNARKs for statements which are otherwise expensive to arithmetize. Main applications are lookups for relations of high algebraic complexity, and interval ranges which are extensively used by zero-knowledge virtual machines enforcing execution trace elements being valid machine words. Although closely related to permutation arguments [BG12, BSCGT13], a first explicit occurence of lookups dates back to [BCG⁺18]. The break-through was achieved by Plookup [GW20], a permutation argument based argument which improved over the one from [BCG⁺18] and provided the first solution for arbitrary tables. Since then Plookup (and slight variants of it) is *the* general purpose lookup argument used in many practical applications, for example [Azt, BLH⁺, Ark, GPR21, Mid].

In this paper we describe a lookup argument based which is based on logarithmic derivatives instead of permutation arguments. As in classical calculus, formal logarithmic derivatives turn products $\prod_{i=1}^{N}(X - z_i)$ into sums of their reciprocals,

$$\sum_{i=1}^{N} \frac{1}{X - z_i},$$

having poles with the same multiplicity as the zeros of the product. Working with poles instead of zeros is extremly useful for lookup arguments. While strategies for arguments about radicals of products are far from obvious, they turn trivial using logarithmic derivatives. Concretely, given a sequence of field elements $(a_i)_{i=1}^{N}$ and another sequence $(t_j)_{j=1}^{M}$, then $\{a_i : i = 1, \ldots, N\} \subseteq \{t_j : j = 1, \ldots, M\}$ as sets, if and only if there exists a sequence of field elements $(m_j)_{j=1}^{M}$ (the multiplicities) such that

$$\sum_{i=1}^{N} \frac{1}{X - a_i} = \sum_{j=1}^{M} \frac{m_j}{X - t_j}.$$

(This holds under quite mild conditions on the field, see Lemma 5 for details.) Based on this fractional identity we construct lookup protocols which are more efficient than the Plookup approach, which argues via a sorted union of witness and table sequence. This is particularly true in the case of *multi-column* lookups, where several sequences ("columns") are subject to the same table lookup. In our lookup the oracle costs, measuring the number and sizes of the oracles the prover needs to provide, are significantly lower than for a lookup based on the Plookup strategy. For large numbers of columns it is the half, while the arithmetic costs of the interactive oracle prover remain comparable.

We stress the fact that we are not the only ones who exploit fractional decompositions for lookups. Concurrently, improving the work of [ZBK⁺22] and [PK22], Gabizon et al. [GK22] use fractional decompositions for "large-table" lookups, the proving-time of which is independent of the table size. This use case is perpendicular to ours. We focus on multi-column lookups with respect to a single, in practice medium-sized table, a use case that is extensively used in execution trace proofs.

The document is organized as follows. In Section 2, we gather the preliminaries used in the sequel: The Lagrange kernel over the boolean hypercube, basic facts on the formal logarithmic derivative, and a summary of the multivariate sumcheck argument. Besides that, we introduce Lagrange interactive oracle proofs, an oracle model we consider the best fit for arguments which are based on the Lagrange representation of polynomials rather than their coefficients. In Section 3 we describe our multi-column lookup based on the logarithmic derivative. The protocol comes in two variants, one for a "small" number of columns, and another one which performs better for large numbers of columns (and is asymptotically

linear in the instance size). For comparison reasons, we add an extra section (Section 4) in which we sketch multi-column lookups using the Plookup strategy, adapted to the boolean hypercube. These rely on the time shift from Hyperplonk [CBBZ22], and we consider them state-of-the-art in the multivariate setting.

We finally point out, that although our protocols are written for the multilinear setting, their translation into univariate proofs is straight-forward. We expect these univariate arguments to improve similarly over multi-column lookups based on the Plookup strategy.

Preliminaries

The Lagrange kernel of the boolean hypercube

Let F denote a finite field, and F^* its multiplicative group. Throughout the document we regard the boolean hypercube $H = \{\pm 1\}^n$ as a multiplicative subgroup of $(F^*)^n$. For a multivariate function $f(X_1, \ldots, X_n)$, we will often use the vector notation $\vec{X} = (X_1, \ldots, X_n)$ for its arguments, writing $f(\vec{X}) := f(X_1, \ldots, X_n)$.

The *Lagrange kernel* of H is the multilinear polynomial

$$L_H(\vec{X}, \vec{Y}) = \frac{1}{2^n} \cdot \prod_{j=1}^{n} (1 + X_j \cdot Y_j). \tag{1}$$

Notice that $L_H(\vec{X}, \vec{Y})$ is symmetric in $\vec{X}$ and $\vec{Y}$, i.e. $L_H(\vec{X}, \vec{Y}) = L_H(\vec{Y}, \vec{X})$, and that (1) is evaluated within only $\mathcal{O}(\log |H|)$ field operations. Whenever $\vec{y} \in H$ we have that $L_H(\vec{X}, \vec{y})$ is the Lagrange polynomial on H, which is the unique multilinear polynomial which satisfies $L_H(\vec{x}, \vec{y}) = 1$ at $\vec{x} = \vec{y}$, and zero elsewhere on H. In particular for a function $f : H \to F$ the inner product evaluation formula

$$\langle f, L_H(\,.\,, \vec{y}) \rangle_H := \sum_{\vec{x} \in H} f(\vec{x}) \cdot L_H(\vec{x}, \vec{y}) = f(\vec{y}).$$

is valid for every $\vec{y} \in H$. This property extends beyond H, as the following Lemma shows.

Lemma 1. *Let $p(\vec{X})$ be the unique multilinear extension of $f : H \to F$. Then for every $\vec{y} \in F^n$,*

$$\langle f, L_H(\,.\,, \vec{y}) \rangle_H = \sum_{x \in H} f(\vec{x}) \cdot L_H(\vec{x}, \vec{y}) = p(\vec{y}). \tag{2}$$

Proof. Since $p(\vec{y}) = \sum_{\vec{x} \in H} f(\vec{z}) \cdot L_H(\vec{X}, \vec{z})$, it suffices to show the claim for $p(X) = L_H(\vec{X}, \vec{z})$, with $\vec{z} \in H$. By the property of $L_H(\vec{X}, \vec{z})$, we have $\langle L_H(\,.\,, \vec{z}), L_H(\,.\,, \vec{y}) \rangle_H = L_H(\vec{y}, \vec{z})$, which by symmetry is equal to $L_H(\vec{X}, \vec{y})$ at $\vec{X} = \vec{z}$. This completes the proof of the Lemma. $\square$

Note that for any $\vec{y} \in F^n$, the domain evaluation of $L_H(\vec{X}, \vec{y})$ over H can be computed in $\mathcal{O}(|H|)$ field operations, by recursively computing the domain evaluation of the partial products $p_k(X_1, \ldots, X_k, y_1, \ldots, y_k) = \frac{1}{2^n} \cdot \prod_{j=1}^{k} (1 + X_j \cdot y_j)$ over $H_k = \{\pm 1\}^k$ from the domain evaluation of p_{k-1}, where one starts with $f_0 = \frac{1}{2^n}$ over the single-point domain H_0. Each recursion step costs $|H_{k-1}|$ field multiplications, denoted by M, and the same number of additions, denoted by A, yielding overall

$$\sum_{k=1}^{n} |H_{k-1}| \cdot (\mathsf{M} + \mathsf{A}) < |H| \cdot (\mathsf{M} + \mathsf{A}). \tag{3}$$

The formal derivate

Given a univariate polynomial $p(X) = \sum_{k=0}^{d} c_k \cdot X^k$ over a general (possibly infinite) field F, its *derivative* is defined as

$$p'(X) := \sum_{k=1}^{d} k \cdot c_k \cdot X^{k-1}. \tag{4}$$

As in calculus, the derivative is linear, i.e. for every two polynomials $p_1(X), p_1(X) \in F[X]$, and coefficients $\lambda_1, \lambda_2 \in F$,

$$(\lambda_1 \cdot p_1(X) + \lambda_2 \cdot p_1(X))' = \lambda_1 \cdot p_1'(X) + \lambda_2 \cdot p_2'(X)$$

and we have the product rule

$$(p_1(X) \cdot p_2(X))' = p_1'(X) \cdot p_2(X) + p_1(X) \cdot p_2'(X).$$

For a function $\frac{p(X)}{q(X)}$ from the rational function field $F(X)$, the derivative is defined as the rational function

$$\left(\frac{p(X)}{q(X)}\right)' := \frac{p'(X) \cdot q(X) - p(X) \cdot q'(X)}{q(X)^2}. \tag{5}$$

By the product rule for polynomials, the definition does not depend on the representation of $\frac{p(X)}{q(X)}$. Both linearity as well as the product rule extend to rational functions.

For any polynomial $p(X) \in F[X]$, if $p'(X) = 0$ then $p(X) = g(X^p)$ where p is the characteristic of the field F. In particular, if $\deg p(X) < p$, then the polynomial must be constant. As the analogous fact for fractions is not as commonly known, we give a proof of the following lemma.

Lemma 2. *Let F be a field of characteristic $p \neq 0$, and $\frac{p(X)}{q(X)}$ a rational function over F with both $\deg p(X) < p$ and $\deg q(X) < p$. If the formal derivative $\left(\frac{p(X)}{q(X)}\right)' = 0$, then $\frac{p(X)}{q(X)} = c$ for some constant $c \in F$.*

Proof. If $q(X)$ is a constant, then the assertion of the Lemma follows from the corresponding statement for polynomials. Hence we assume that $\deg q(X) > 0$. Use polynomial division to obtain the representation

$$\frac{p(X)}{q(X)} = m(X) + \frac{r(X)}{q(X)},$$

with $m(X), r(X) \in F[X]$, $\deg m(X) \leq \deg p(X)$, and $\deg r(X) < \deg q(X)$ whenever $r(X) \neq 0$. By linearity of the derivative, we have $0 = \left(\frac{p(X)}{q(X)}\right)' = m'(X) + \left(\frac{r(X)}{q(X)}\right)'$, and therefore

$$r'(X) \cdot q(X) - r(X) \cdot q'(X) = -m'(X) \cdot q(X)^2. \tag{6}$$

Comparing the degrees of left and right hand side in (6), we conclude that $m'(X) = 0$. Since $\deg m(X) \leq \deg p(X) < p$ we have $m(X) = c$ for some constant[1] $c \in F$. Furthermore, if we had $r(X) \neq 0$ then the leading term of the left hand side in (6) would be

$$(k - n) \cdot c_n \cdot d_k \cdot X^{n+k-1},$$

with $c_n \cdot X^n$, $n > 0$, being the leading term of $q(X)$, and $d_k \cdot X^k$, $0 \leq k < n$, the leading term of $r(X)$. As $0 < n - k < p$, and both $c_n \neq 0$ and $c_m \neq 0$, the leading term of the left hand side of (6) would not vanish. Therefore it must hold that $r(X) = 0$ and the proof of the lemma is complete. $\square$

---

[1] For general degrees of $p(X)$ we would only be able to conclude that $m(X) = g(X^p)$ for some polynomial $g(X)$.

The logarithmic derivative

The *logarithmic derivate* of a polynomial $p(X)$ over a (general) field F is the rational function

$$\frac{p'(X)}{p(X)}.$$

Note that the logarithmic derivative of the product $p_1(X) \cdot p_2(X)$ of two polynomials $p_1(X), p_2(X)$ equals the sum of their logarithmic derivatives, since by the product rule we have

$$\frac{(p_1(X) \cdot p_2(X))'}{p_1(X) \cdot p_2(X)} = \frac{p_1'(X) \cdot p_2(X) + p_1(X) \cdot p_2'(X)}{p_1(X) \cdot p_2(X)} = \frac{p_1'(X)}{p_1(X)} + \frac{p_2'(X)}{p_2(X)}.$$

In particular the logarithmic derivative of a product $p(X) = \prod_{i=1}^{n}(X + z_i)$, with each $z_i \in F$, is equal to the sum

$$\frac{p'(X)}{p(X)} = \sum_{i=1}^{n} \frac{1}{X + z_i}. \tag{7}$$

The following lemma is a simple consequence of Lemma 2 and essentially states that, under quite mild conditions on the field F, if two normalized polynomials have the same logarithmic derivative then they are equal. We state this fact for our use case of product representations.

Lemma 3. *Let $(a_i)_{i=1}^{n}$ and $(b_i)_{i=1}^{n}$ be sequences over a field F with characteristic $p > n$. Then $\prod_{i=1}^{n}(X + a_i) = \prod_{i=1}^{n}(X + b_i)$ in $F[X]$ if and only if*

$$\sum_{i=1}^{n} \frac{1}{X + a_i} = \sum_{i=1}^{n} \frac{1}{X + b_i}$$

in the rational function field $F(X)$.

Proof. If $p_a(X) = \prod_{i=1}^{n}(X + a_i)$ and $p_b(X) = \prod_{i=1}^{n}(X + b_i)$ coincide, so do their logarithmic derivatives. To show the other direction, assume that $\frac{p_a'(X)}{p_a(X)} = \frac{p_b'(X)}{p_b(X)}$. Then

$$\left(\frac{p_a(X)}{p_b(X)}\right)' = \frac{p_a'(X) \cdot p_b(X) - p_a(X) \cdot p_b'(X)}{p_b^2(X)} = 0.$$

Hence by Lemma 2 we have $\frac{p_a(X)}{p_b(X)} = c$ for some constant $c \in F$. As both $p_a(X)$ and $p_b(X)$ have leading coefficient equal to 1, we conclude that $c = 1$, and the proof of the Lemma is complete. □

Remark 1. We stress the fact that Lemma 3 also applies to the case where F is the function field $F_p(Y_1, \ldots, Y_k)$ over a finite field F_p of characteristic p. This observation will be useful when generalizing the permutation argument to the case where a_i and b_i are multilinear polynomials in $Y_1, \ldots, Y_n$.

Given a product $p(X) = \prod_{i=1}^{N}(X + a_i)$ we can gather the poles of its logarithmic derivative obtaining the fractional decomposition

$$\frac{p'(X)}{p(X)} = \sum_{a \in F} \frac{m(a)}{X + a},$$

where $m(a) \in \{1, \ldots, N\}$ is the multiplicity of the value a in $(a_i)_{i=1}^{N}$. Fractional decompositions are unique, as shown by the following lemma.

Lemma 4. *Let F be an arbitrary field and $m_1, m_2 : F \to F$ any functions. Then $\sum_{z \in F} \frac{m_1(z)}{X-z} = \sum_{z \in F} \frac{m_2(z)}{X-z}$ in the rational function field $F(X)$, if and only if $m_1(z) = m_2(z)$ for every $z \in F$.*

Proof. Suppose that the fractional decompositions are equal. Then $\sum_{z \in F} \frac{m_1(z) - m_2(z)}{X-z} = 0$, and therefore

$$p(X) = \prod_{w \in F} (X - w) \cdot \sum_{z \in F} \frac{m_1(z) - m_2(z)}{X - z} = \sum_{z \in F} (m_1(z) - m_2(z)) \cdot \prod_{w \in F \setminus \{z\}} (X - w) = 0.$$

In particular, $p(z) = (m_1(z) - m_2(z)) \cdot \prod_{w \in F \setminus \{z\}} (z - w) = 0$ for every $z \in F$. Since $\prod_{w \in F \setminus \{z\}} (z - w) \neq 0$ we must have $m_1(z) = m_2(z)$ for every $z \in F$. The other direction is obvious. $\qquad \square$

This leads to the following algebraic criterion for set membership, which is the key tool for our lookup arguments.

Lemma 5 (Set inclusion). *Let F be a field of characteristic $p > N$, and suppose that $(a_i)_{i=1}^N$, $(b_i)_{i=1}^N$ are arbitrary sequences of field elements. Then $\{a_i\} \subseteq \{b_i\}$ as sets (with multiples of values removed), if and only if there exists a sequence $(m_i)_{i=1}^N$ of field elements from $F_q \subseteq F$ such that*

$$\sum_{i=1}^N \frac{1}{X + a_i} = \sum_{i=1}^N \frac{m_i}{X + b_i} \tag{8}$$

in the function field $F(X)$. Moreover, we have equality of the sets $\{a_i\} = \{b_i\}$, if and only if $m_i \neq 0$, for every $i = 1, \dots, N$.

Proof. Let us denote by $m_a(z)$ the multiplicity of a field element z in the sequence $(a_i)_{i=1}^N$. Likewise, we do for $(b_i)_{i=1}^N$. Note that since $N < p$, the multiplicities can be regarded as non-zero elements from F_p as a subset of F. Suppose that $\{a_i\} \subseteq \{b_i\}$ as sets. Set (m_i) as the normalized multiplicities $m_i = \frac{m_a(b_i)}{m_b(b_i)}$. This choice of (m_i) obviously satisfies (8).

Conversely, suppose that (8) holds. Collecting fractions with the same denominator we obtain fractional representations for both sides of the equation (8),

$$\sum_{i=1}^N \frac{1}{X + a_i} = \sum_{z \in F} \frac{m_a(z)}{X + z},$$

$$\sum_{i=1}^N \frac{m_i}{X + b_i} = \sum_{z \in F} \frac{\mu(z)}{X + z}.$$

Note that since $N < p$, we know that for each $z \in \{a_i\}$ we have $m_a(z) \neq 0$. By the uniqueness of fractional representations, Lemma 4, $m_a(z) = \mu(z)$ for every $z \in \{a_i\}$, and therefore each $z \in \{a_i\}$ must occur also in $\{b_i\}$. $\qquad \square$

Lagrange interactive oracle proofs

The oracle proofs of many general purpose SNARKs such as Plonk [GWC19] or algebraic intermediate representations [BSBHR18] rely on witnesses that are given in Lagrange representation, i.e. by their values over a domain H. Their multivariate variants may completely avoid the usage of fast Fourier transforms whenever the polynomial commitment scheme can be turned into one that does not need to know the coefficients, neither when computing a commitment nor in an opening proof. Exactly this property is captured by *Lagrange oracle proofs*, rather than polynomial ones [BFS20].

A *Lagrange interactive oracle proof* (*Lagrange IOP*) over the boolean hypercube $H = \{\pm 1\}^n$ is an interactive protocol between two parties, the "prover" and the "verifier". In each round, the verifier sends a message (typically a random challenge) and the prover computes one or several functions over the boolean hypercube, and gives the verifier oracle access to them. From the moment on it is given access, the verifier is allowed to query the oracles for their inner products with the Lagrange kernel $L_H(\,.\,, \vec{y})$, associated with an arbitrary vector $\vec{y} \in F^n$.

The security notions for Lagrange IOPs, such as completeness, (knowledge) soundness, and zero-knowledge, are exactly the same as for other interactive oracle proofs. We assume that the reader is familiear with these, and refer to [BSCS16] or [BFS20] for their formal definitions.

Lagrange IOPs are turned into arguments by instantiating the Lagrange oracles by a *Lagrange commitment scheme*. A Lagrange commitment scheme is a commitment scheme for functions over H that comes with an evaluation proof for Lagrange queries. For example, inner product arguments [BCC+16] can be directly used to construct Lagrange commitment schemes, but also the multilinear variant [PST13] of the [KZG10] commitment scheme is easily modified to completely avoid dealing with coefficients. We suppose that this is well-known, and therefore we omit an explicit elaboration in this paper.

The sumcheck protocol

We give a concise summary on the multivariate sumcheck protocol [LFKN92]. Given a multivariate polynomial $p(X_1, \ldots, X_n) \in F[X_1, \ldots, X_n]$, a prover wants to convince a verifier upon that

$$s = \sum_{(x_1, \ldots, x_n) \in \{\pm 1\}^n} p(x_1, \ldots, x_n).$$

This is done by a random folding procedure which, starting with $H_0 = \{\pm 1\}^n$, which stepwise reduces a claim on the sum over $H_i = \{\pm 1\}^{n-i}$, $i = 0, \ldots, n-1$, to one over the hypercube H_{i+1} of half the size. Eventually, one ends up with a claim over a single-point sum, which is paraphrased as the value of $p(X_1, \ldots, X_n)$ at a random point $(r_1, \ldots, r_n) \in F^n$ sampled in the course of the reduction steps.

Protocol 1 (Sumcheck protocol, [LFKN92]). *Let $p(X_1, \ldots, X_n)$ be a multivariate polynomial over a finite field F. The sumcheck protocol, in which a prover wants to convince the verifier upon the sum $s = \sum_{(x_1, \ldots, x_n) \in \{\pm 1\}^n} p(x_1, \ldots, x_n)$, is as follows. We write $s_0(X)$ for the constant polynomial $s_0 = s$.*

- *In each round $i = 1, \ldots, n$, the prover sends the coefficients of the univariate polynomial*

$$s_i(X) = \sum_{(x_{i+1}, \ldots, x_n) \in \{\pm 1\}^{n-i}} p(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_n),$$

of degree $d_i \leq \deg_{X_i} p(X_1, \ldots, X_n)$, where $r_1, \ldots, r_{i-1}$ are the randomnesses received in the previous rounds. (In the first round $i = 1$ there are no previous randomnesses, and $p(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_n)$ is meant to denote $p(X, x_2, \ldots, x_n)$.) The prover sends the coefficients of $s_i(X)$ to the verifier, which checks whether the received polynomial $s_i(X)$ is in fact of the expected degree and that

$$s_{i-1}(r_{i-1}) = s_i(+1) + s_i(-1).$$

(Again, in the first round $i = 1$ there is no r_0, and the verifier checks wheather $s_0 = s_1(+1) + s_1(-1)$.) If so, the verifier samples random challenge $r_i \leftarrow_\$ F$ uniformly from F and sends it to the prover.

After these rounds the verifier checks that $s_n(r_n) = p(r_1, \ldots, r_n)$. If so, the verifier accepts (otherwise it rejects).

Soundness of the sumcheck protocol is proven by a repeated application of the Schwartz-Zippel lemma. We omit a proof, and refer to [LFKN92] or [Tha13].

Theorem 2 ([LFKN92]). *The sumcheck protocol (Protocol 1) has soundness error*

$$\varepsilon_{sumcheck} \leq \frac{1}{|F|} \cdot \sum_{i=1}^{n} \deg_{X_i} p(X_1, \ldots, X_n). \tag{9}$$

The sumcheck protocol is easily extended to the sumcheck for a batch of polynomials $p_i(X_1, \ldots, X_n)$, $i = 0, \ldots, L$, by letting the verifier sample a random vector $(\lambda_1, \ldots, \lambda_L) \leftarrow_\$ F^L$, and a subsequent sumcheck protocol for the random linear combination

$$\bar{p}(X_1, \ldots, X_n) = p_0(X_1, \ldots, X_n) + \sum_{i=1}^{L} \lambda_i \cdot p_i(X_1, \ldots, X_n).$$

The soundness error bound increases only slightly,

$$\varepsilon_{sumcheck} \leq \frac{1}{|F|} \cdot \left(1 + \sum_{i=1}^{n} \deg_{X_i} p(X_1, \ldots, X_n)\right). \tag{10}$$

Computational cost

Let us discuss the prover cost of the sumcheck protocol for the case that $p(\vec{X}) = p(X_1, \ldots, X_n)$ is of the form

$$p(\vec{X}) = Q(w_1(\vec{X}), \ldots, w_m(\vec{X})),$$

with each $w_i(\vec{X}) \in F[X_1, \ldots, X_n]$ being multilinear, and

$$Q(Y_1, \ldots, Y_m) = \sum_{(i_1, \ldots, i_m) \in \{0,1\}^m} c_{i_1, \ldots, i_m} \cdot Y_1^{i_1} \cdots Y_m^{i_m}$$

is a multivariate polynomial having (a typically low) absolute degree d. We denote the arithmetic complexity, i.e. the number of field multiplications M, substractions S and additions A to evaluate Q by $|Q|_\mathsf{M}$, $|Q_\mathsf{S}|$ and $|Q|_\mathsf{A}$, respectively. Each of the univariate polynomials $s_i(X)$, $i = 1, \ldots, n$, is of degree at most d the absolute degree of Q, and is computed from its values over a set $D \supseteq \{\pm 1\}$ of size $|D| = d + 1$. In each step $i = 1, \ldots, n$, the values of $s_i(z)$ for $z \in D$ are obtained by linear interpolation of the domain evaluations of each

$$w_j(r_1, \ldots, r_{i-1}, \pm 1, X_{i+1}, \ldots, X_n)$$

over $H_i = \{\pm 1\}^{n-i}$ as given from the previous step, to the domain evaluation

$$w_j(r_1, \ldots, r_{i-1}, z, X_{i+1}, \ldots, X_n),$$

the values of which are used for computing $s_i(z) = \sum_{(x_{i+1}, \ldots, x_n) \in H_i} Q(r_1, \ldots, r_{i-1}, z, x_{i+1}, \ldots, x_n)$. Given the random challenge r_i from the verifier, the domain evaluation of each

$$w_j(r_1, \ldots, r_{i-1}, r_i, X_{i+1}, \ldots, X_n)$$

is computed by another linear interpolation. Linear interpolation costs $|H_i|$ multiplications and the same number of additions/substractions for each multilinear polynomial, the values of Q are obtained within

8

$|Q|_{\mathsf{M}} \cdot \mathsf{M} + |Q|_{\mathsf{S}} \cdot \mathsf{S} + |Q|_{\mathsf{A}} \cdot \mathsf{A}$. In terms of field multiplications M, substractions S and additions A, step i consumes

$$m \cdot |H_i| \cdot \mathsf{S} + m \cdot (|D| - 1) \cdot |H_i| \cdot (\mathsf{M} + \mathsf{A}) + |D| \cdot |H_i| \cdot (|Q|_{\mathsf{M}} \cdot \mathsf{M} + |Q|_{\mathsf{S}} \cdot \mathsf{S} + |Q|_{\mathsf{A}} \cdot \mathsf{A}) + |D| \cdot |H_i| \cdot \mathsf{A},$$

where the last term is for the domain sums. Since $\sum_{i=1}^{n} |H_i| = |H| - 1$, the overall cost for the prover is bounded by

$$|H| \cdot \left(1 - \frac{1}{|H|}\right) \cdot ((d \cdot m + (d+1) \cdot |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (m + (d+1) \cdot |Q|_{\mathsf{S}}) \cdot \mathsf{S} + (d \cdot m + (d+1) \cdot (|Q|_{\mathsf{A}} + 1)) \cdot \mathsf{A}). \quad (11)$$

We shall use this formula for the operation counts of our lookup protocol.

Lookups based on the logarithmic derivative

Assume that F is a finite field, and that $f_1, \ldots, f_M$ and $t : H \to F$ are functions over the Boolean hypercube $H = \{\pm 1\}^n$. By Lemma 5, it holds that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ as sets, if and only if there exists a function $m : H \to F$ such that

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{X + f_i(\vec{x})} = \sum_{\vec{x} \in H} \frac{m(\vec{x})}{X + t(\vec{x})}, \quad (12)$$

assuming that the characteristic of F is larger than M times the size of the hypercube. If t is injective (which is typically the case for lookup tables) then m is the multiplicity function, counting the number of occurences for each value $t(\vec{x})$ in $f_1, \ldots, f_M$ altogether, i.e. $m(\vec{x}) = m_f(t(\vec{x})) = \sum_{i=1}^{M} |\{\vec{y} \in H : f_i(\vec{y}) = t(\vec{x})\}|$. If t is not one-to-one, we set m as the *normalized* multiplicity function

$$m(\vec{x}) = \frac{m_f(t(\vec{x}))}{m_t(t(\vec{x}))} = \frac{\sum_{i=1}^{M} |\{\vec{y} \in H : f_i(\vec{y}) = t(\vec{x})\}|}{|\{\vec{y} \in H : t(\vec{y}) = t(\vec{x})\}|}. \quad (13)$$

The plot for proving that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ is as follows. Given a random challenge $x \leftarrow_{\$} F$, the prover shows that the rational identity (12) holds at $X = x$, whenever evaluation is possible. However, in order to make (12) applicable to the sumcheck argument, the prover needs to provide multilinear helper functions for the rational expressions. We shall discuss two different different approaches for doing that. In the first one, explained in Section 3.1, we use a single multilinear function for the entire fractional expression in (12), which is subject to a domain identity over H which has $\mathcal{O}(M)$ variables and absolute degree $\mathcal{O}(M)$. This will lead to a protocol with a $\mathcal{O}(M^2)$ prover. However, if M is not too large this approach will be more performant than the second one, discussed in Section 3.2, in which we essentially use helper functions for each of the reciprocals $\frac{1}{x + f_i(\vec{x})}$, and $\frac{1}{x + t(\vec{x})}$. This second variant has a sumcheck polynomial in $\mathcal{O}(M)$ many variables, but the absolute degree is bounded, henceforth having a $\mathcal{O}(M)$ prover.

An argument for not too many columns

In this variant we provide a single helper function

$$h(\vec{x}) = \sum_{i=1}^{M} \frac{1}{x + f_i(\vec{x})} - \frac{m(\vec{x})}{x + t(\vec{x})}, \quad (14)$$

subject $\sum_{\vec{x} \in H} h(\vec{x}) = 0$. Correctness of h is ensured by the domain identity

$$(h(\vec{x}) \cdot (x + t(\vec{x})) + m(\vec{x})) \cdot \prod_{i=1}^{M} (x + f_i(\vec{x})) = (x + t(\vec{x})) \cdot \sum_{i=1}^{M} \prod_{j \neq i} (x + f_j(\vec{x})) \quad (15)$$

9

over H, and we apply the Lagrange kernel $L_H(\,.\,,\vec{z})$ at a randomly chosen $\vec{z} \leftarrow_\$ F^n$ to reduce the domain identity to another sumcheck over H. Both sumchecks, the one for h and the one for the domain identity, are then combined into a single one, using another randomness $\lambda \leftarrow_\$ F$.

Protocol 2 (Multi-column lookup over $H = \{\pm 1\}^n$). *Let $M \geq 1$ be an integer, and F a finite field with characteristic $p > M \cdot 2^n$. Given any functions $f_1, \ldots, f_M, t : H \to F$ on the boolean hypercube $H = \{\pm 1\}^n$, the Lagrange IOP for that $\bigcup_{i=1}^M \{f_i(\vec{x}) : \vec{x} \in H\} \subseteq \{t(\vec{x}) : \vec{x} \in H\}$ as sets is as follows.*

1. *The prover determines the (normalized) multiplicity function $m : H \to F$ as defined in (13), and sends the oracle for m to the verifier. The verifier answers with a random sample $x \leftarrow_\$ F \setminus \{-t(\vec{x}) : \vec{x} \in H\}$.*

2. *Given the challenge x from the verifier, the prover computes the randomized functions $\varphi_i(\vec{x}) = x + f_i(\vec{x})$, $i = 1, \ldots, M$, and $\tau(\vec{x}) = x + t(\vec{x})$. It determines the values for*

$$h(\vec{x}) = \sum_{i=1}^M \frac{1}{\varphi_i(\vec{x})} - \frac{m(\vec{x})}{\tau(\vec{x})}, \tag{16}$$

   over H, and sends the oracle for h to the verifier.

3. *The verifier responds with a random vector $\vec{z} \leftarrow_\$ F^n$ and a batching randomness $\lambda \leftarrow_\$ F$. Now, both prover and verifier engage in the sumcheck protocol (Protocol 1) for*

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), h(\vec{x}), m(\vec{x}), \varphi_1(\vec{x}), \ldots, \varphi_M(\vec{x}), \tau(\vec{x})) = 0,$$

   where

$$Q(L, h, m, \varphi_1, \ldots, \varphi_M, \tau) = L \cdot \left((h \cdot \tau + m) \cdot \prod_{i=1}^M \varphi_i - \tau \cdot \sum_{i=1}^M \prod_{j \neq i} \varphi_j \right) + \lambda \cdot h. \tag{17}$$

   The sumcheck protocol outputs the expected value v for the multivariate polynomial

$$Q(L_H(\vec{X}, \vec{z}), h(\vec{X}), m(\vec{X}), \varphi_1(\vec{X}), \ldots, \varphi_M(\vec{X}), \tau(\vec{X})) \tag{18}$$

   at $\vec{X} = \vec{r}$ sampled by the verifier in the course of the protocol.

4. *The verifier queries $[f_1], \ldots, [f_M], [t], [m], [h]$ for their inner product with $L_H(\,.\,,\vec{r})$, and uses the answers to check whether (18) equals the expected value v at $\vec{X} = \vec{r}$. (The value $L_H(\vec{r}, \vec{z})$ is computed by the verifier.)*

Remark 3. We imposed the condition $x \notin \{-t(\vec{x})\}_{\vec{x} \in H}$ merely for completeness. However in some applications it may be not be desirable, or even not possible, to sample x from outside the range of t. There are several ways to handle this. One can simply omit the constraint on x, letting the verifier sample $x \leftarrow_\$ F$ and the prover set h arbitrary whenever (16) is not defined. This comes at no extra cost, but the obtained protocol is only overwhelmingly complete. That is, with a probability of at most $\frac{|H|}{|F|}$ in the verifier randomness x, the honest prover does not succeed. In practice this is often considered acceptable, and many lookup implementations have a non-zero completeness error. Whenever this is not acceptable, one may modify the domain identity (15) to

$$\left((h \cdot \tau + m) \cdot \prod_{i=1}^M \varphi_i - \tau \cdot \sum_{i=1}^M \prod_{j \neq i} \varphi_j \right) \cdot \tau \cdot \prod_{i=1}^M \varphi_i = 0 \tag{19}$$

over H, which imposes no condition on $h(\vec{x})$ whenever $\tau(\vec{x}) = 0$. However, this approach comes at the cost of almost doubling the absolute degree of Q.

Let us point out two variations of Protocol 2. In the single-column case $M = 1$ the lookup argument can be turned into a multiset check for the ranges of f_1 and t, by setting m as the constant function $m(\vec{x}) = 1$. In this case only h needs to be provided by the prover. More interestingly, Protocol 2 is easily extended to a proof of range equality, showing that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} = \{\tau(\vec{x})\}_{\vec{x} \in H}$ as sets. For this the prover additionally shows that $m \neq 0$ over H, which is done by providing another auxiliary function $h_m : H \to F$ subject to $h_m \cdot m = 1$ over H. However, we are not aware of any application of this fact.

A variant for a large number of columns

Assume that $M = 2^m$, so that we can index the columns to be looked up by $f_{\vec{z}}$, where $\vec{z} \in \{\pm 1\}^m$. We patch these columns into a single function f over the extended hypercube $\bar{H} = \{\pm 1\}^m \times H$ by

$$f(\vec{y}, \vec{x}) = \sum_{\vec{z} \in \{\pm 1\}^m} L_m(\vec{y}, \vec{z}) \cdot f_{\vec{z}}(\vec{x}),$$

where $L_m(\vec{y}, \vec{z})$ is the Lagrange polynomial for $\{\pm 1\}^m$. Given the random challange $x \leftarrow_\$ F \setminus \{-t(\vec{x}) : \vec{x} \in H\}$ from the verifier, the prover supplies an oracle for the values of

$$h(\vec{y}, \vec{x}) = \frac{1}{x + f(\vec{y}, \vec{x})} - \frac{\tilde{m}(\vec{x})}{x + t(\vec{x})} \tag{20}$$

over the extended hypercube, where $\tilde{m}(\vec{x}) = \frac{1}{2^m} \cdot m(\vec{x})$. The supplementary function h is subject to the domain identity

$$\big(h(\vec{y}, \vec{x}) \cdot (x + t(\vec{x})) + \tilde{m}(\vec{x})\big) \cdot (x + f(\vec{y}, \vec{x})) - (x + t(\vec{x})) = 0$$

over $\bar{H}$, and

$$\sum_{\vec{y} \in \{\pm 1\}^m} \sum_{\vec{x} \in H} h(\vec{y}, \vec{x}) = 0.$$

Again, the domain identity is turned into a sumcheck over $\bar{H}$ by applying the Lagrange kernel $L_{\bar{H}}(\,.\,, \vec{z})$, where $\vec{z}$ is now sampled from F^{m+n}. Combining the two sumchecks using a random $\lambda \leftarrow_\$ F$ leads to the overall sumcheck

$$\sum_{\vec{y} \in \{\pm 1\}^m} \sum_{\vec{x} \in H} Q(L_{\bar{H}}((\vec{y}, \vec{x}), \vec{z}), h(\vec{y}, \vec{x}), \tilde{m}(\vec{x}), \varphi(\vec{y}, \vec{x}), \tau(\vec{x})) = 0,$$

over $\bar{H} = \{\pm 1\}^m \times H$, with $\varphi(\vec{y}, \vec{x}) = x + f(\vec{y}, \vec{x})$ and $\tau(\vec{x}) = x + t(\vec{x})$, and where Q is

$$Q(L, h, \tilde{m}, \varphi, \tau) = L \cdot ((h \cdot \tau + \tilde{m}) \cdot \varphi - \tau) + \lambda \cdot h. \tag{21}$$

In this variant, providing h amounts to M oracles over H, yielding an overall equivalent of $M + 1$ oracles of size $|H|$. However, Q has only $\nu = 5$ variables and its absolute degree is independent of the number of columns.

Soundness

The soundness analysis of Protocol 2 is a straight-forward application of the Schwartz-Zippel lemma and the Lagrange-query to point-query correspondence stated by Lemma 1. We merely sketch it. The univariate rational lookup identity (12) is turned into a polynomial identity of degree at most $|H| \cdot (M + 1) - 1$ by multiplying it with the common denominator

$$p(X) = \prod_{\vec{x} \in H} (X + t(\vec{x})) \cdot \prod_{i=1}^{M} (X + f_i(\vec{x})). \tag{22}$$

Since we sample x from a set of size at least $|F| - |H|$, the soundness error of Step 2 of the protocol is at most

$$\varepsilon_1 \le \frac{(M+1) \cdot |H| - 1}{|F| - |H|}. \tag{23}$$

The soundness error due to the reduction of the domain identity (15) to the Lagrange kernel based sumcheck is

$$\varepsilon_2 \le \frac{1}{|F|},$$

as scalar products with the Lagrange kernel translate to point evaluation of the multilinear extension. This yields the following theorem.

Theorem 4. *The interactive oracle proof described Protocol 2 has soundness error*

$$\varepsilon < \frac{(M+1) \cdot |H| - 1}{|F| - |H|} + \varepsilon_{sumcheck},$$

where $\varepsilon_{sumcheck}$ is the soundness error of the sumcheck argument (10) over H for a multivariate polynomial in $M + 4$ variables with maximum individual degree $M + 3$.

Remark 5. The $\mathcal{O}(M)$-variant described in Section 2 has same soundness error, with $\varepsilon_{sumcheck}$ being the soundness error of the sumcheck argument over the extended hypercube of size $M \cdot |H|$ for a multivariate polynomial in $\nu = 5$ variables and maximum individual degree 4.

Remark 6. Protocol 2 and its variant from Section 3.2 are easily generalized to functions with multilinear values,

$$t(\vec{x}) = \sum_{(j_1,\ldots,j_k) \in \{0,1\}^k} t_{j_1,\ldots,j_k}(\vec{x}) \cdot Y_1^{i_1} \cdots Y_k^{j_k},$$

$$f_i(\vec{x}) = \sum_{(j_1,\ldots,j_k) \in \{0,1\}^k} f_{i,j_1,\ldots,j_k}(\vec{x}) \cdot Y_1^{i_1} \cdots Y_k^{j_k},$$

$i = 1, \ldots, M$, without changing the soundness error bound from Theorem 4. As $F[X, Y_1, \ldots, Y_k]$ is a unique factorization domain, and polynomials of the form $X - \sum_{(i_1,\ldots,i_k) \in \{0,1\}^k} c_{i_1,\ldots,i_k} \cdot Y_1^{i_1} \cdots Y_k^{i_k}$ are irreducible, we may apply Lemma 5 to see that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} \subseteq \{t(\vec{x})\}_{\vec{x} \in H}$ as sets in the rational function field $F(X, Y_1, \ldots, Y_k)$, if and only if there exists a function $m : H \to F$ such that

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{X + f_i(\vec{x})(\vec{Y})} = \sum_{\vec{x} \in H} \frac{m(\vec{x})}{X + t(\vec{x})(\vec{Y})}. \tag{24}$$

The only change to Protocol 2 is that the verifier now samples x from F and $\vec{y} = (y_1, \ldots, y_k)$ from F^k, and continues with $x - f_i(\vec{x})$ and $x - t(\vec{x})$ replaced by $x - f_i(\vec{x})(\vec{y})$ and $x - t(\vec{x})(\vec{y})$.

Computational cost

The polynomial Q from (17) has $\nu = M + 4$ variables, and absolute degree $d = M + 3$. Let us discuss an domain evaluation strategy for the values of Q, which makes use of batch inversion. This strategy allows us to evaluate Q much more efficiently than using (17), but demands a modification of the sumcheck operation count formula (11). Assume that the inverses of $\varphi_1, \ldots, \varphi_{M-1}$ are given. Then we may evaluate Q by the fractional representation

$$Q = L \cdot \prod_{i=1}^{M-1} \varphi_i \cdot \left(\varphi_M \cdot m + \tau \cdot \left(\varphi_M \cdot h - \left(\sum_{i=1}^{M-1} \frac{1}{\varphi_i} + 1 \right) \right) \right) + \lambda \cdot h.$$

12

This costs $M + 4$ multiplications, one substraction, and $M + 1$ additions, and hence the arithmetic complexities are $|Q_M| = M + 4$, $|Q_S| = 1$, $|Q_A| = M + 1$. Now, to attribute the inverses in formula (11), we increase the multiplicative complexity by $3 \cdot (M - 1)$, which represents the fractional cost of the batch inversion[2] of $\varphi_1, \ldots, \varphi_{M-1}$. This yields the following equivalent complexities

$$|Q_M| = 4 \cdot M + 1, |Q_S| = 1, |Q_A| = M + 1,$$

which we may plug into formula (11).

Therefore the prover cost of Protocol 2 is as follows: Given the values of $f_1, \ldots, f_M$ and t over H, computing $\varphi_1 = x + f_1, \ldots, \phi_M = x + f_M$, $\tau = x + t$ costs $|H| \cdot (M+1) \cdot \mathsf{A}$, and their reciprocals $\frac{1}{\varphi_1}, \ldots, \frac{1}{\varphi_M}$, $\frac{1}{\tau}$ are obtained within $3 \cdot |H| \cdot (M + 1) \cdot \mathsf{M}$, using batch inversion. With these reciprocals we obtain the values for

$$h = \sum_{i=1}^{M} \frac{1}{\varphi_i} - \frac{m}{\tau}$$

by $|H| \cdot (1 \cdot \mathsf{S} + (M-1) \cdot \mathsf{A})$. By the remark following Lemma 1, the values for $L_H(\vec{X}, \vec{y})$ over H are obtained within $|H| \cdot (\mathsf{M} + \mathsf{A})$ operations. Hence the total cost of the preparation phase is

$$|H| \cdot ((3 \cdot M + 4) \cdot \mathsf{M} + 1 \cdot \mathsf{S} + (2 \cdot M + 1) \cdot \mathsf{A}.$$

According to (11) the sumcheck costs

$$|H| \cdot \left(1 - \frac{1}{2^n}\right) \cdot ((5 \cdot M^2 + 24 \cdot M + 16) \cdot \mathsf{M} + (2 \cdot M + 8) \cdot \mathsf{S} + (2 \cdot M^2 + 13 \cdot M + 20) \cdot \mathsf{A}).$$

However, as we may reuse the reciprocals of $\varphi_1, \ldots, \varphi_{M-1}$ in the first step of the sumcheck, we correct the sumcheck cost by substracting $|H| \cdot (3 \cdot (M - 1))$. Neglecting the $1/2^n$-term, the overall cost of the prover is

$$|H| \cdot ((5 \cdot M^2 + 24 \cdot M + 23) \cdot \mathsf{M} + (2 \cdot M + 9) \cdot \mathsf{S} + (2 \cdot M^2 + 15 \cdot M + 21) \cdot \mathsf{A}), \qquad (25)$$

whereas it provides two H-sized oracles. The cost is $\mathcal{O}(|H|)$ but depends quadratically in M the number of columns to be looked up. This quadratic occurence is due to the fact that both, the number of function as well as the degree of Q grow linearly in M.

The cost for the $\mathcal{O}(M)$ strategy from Section 3.2 is as follows. There, the prover provides h over the extended hypercube of size $|\bar{H}| = M \times |H|$, and Q from (21) has $\nu = 5$ variables, absolute degree $d = 4$, and arithmetic complexities $|Q_M| = 4$, $|Q_S| = 1$, $|Q_A| = 2$. Computing the values of h over $\bar{H}$ using batch inversion costs

$$M \cdot |H| \cdot (3 \cdot \mathsf{M} + \mathsf{A}),$$

and the values for $L_{\bar{H}}(\ ,\ ,\vec{z})$ over $\bar{H}$ are determined in $M \cdot |H| \cdot (\mathsf{M} + \mathsf{A})$. The overall sumcheck costs

$$M \cdot |H| \cdot \left(1 - \frac{1}{M \cdot |H|}\right) \cdot (40 \cdot \mathsf{M} + 10 \cdot \mathsf{S} + 35 \cdot \mathsf{A}).$$

Neglecting the $\frac{1}{M \cdot |H|}$-term, the overall cost for the prover is

$$M \cdot |H| \cdot (44 \cdot \mathsf{M} + 10 \cdot \mathsf{S} + 37 \cdot \mathsf{A}), \qquad (26)$$

---

[2]Batch, or Montgomery inversion, of a sequence $(a_i)_{i=1}^N$ computes the inverses of a_i^{-1} by recursively computing the cumulative products $p_i = a_1 \cdot \ldots \cdot a_n$, $i = 0, \ldots, n$, then calculating their inverses $q_i = \frac{1}{p_i}$ in a reverse manner starting with $q_n = \frac{1}{p_n}$, and putting $q_{i-1} = q_i \cdot a_i$, where i goes from n down to 1. The inverses are then derived via $a_i^{-1} = p_{i-1} \cdot q_i$, where $p_0 := 1$. The overall cost of the batch inversion is $3 \cdot (N - 1)$ multiplications and a single inversion.

13

but the prover needs to provide the oracles for one function over a domain of size $M \cdot |H|$, and one over H.

Let us estimate the range for M where Protocol 2 is more efficient than the $\mathcal{O}(M)$ variant. For this we use the benchmarks from Table 3 which measure the equivalent number of field multiplication for a multi-scalar multiplication in an elliptic curve over 256 bit large field. Based on this equivalent, and our operation counts for the oracle prover, we obtain the following break even points.

Table 1: The estimated number of columns M where the $\mathcal{O}(M)$ strategy starts to perform better than Protocol 2. The numbers are based on the operation counts (25) and (26) for the oracle prover, and the benchmarks for a multi-scalar multiplication over the Pallas curve, see Table 3.

$\log|H|$	12	14	16	18
M	114	95	81	73

Lookups based on the Hyperplonk shift

In this section we informally discuss multi-column lookups based on the Plookup strategy and the time shift from Hyperplonk+ [CBBZ22]. We give detailed operation counts and compare with our lookup arguments from Section 3.

On of the main contributions of [CBBZ22] is the introduction of an (almost) transitive time shift $T : H \to H$ which translates to multilinear extensions in a tame manner. The shift is derived from the multiplication by a primitive root in $GF(2^n)$,

$$T(x_1, \ldots, x_n) = \frac{1 + x_n}{2} \cdot (1, x_1, \ldots, x_{n-1}) + \frac{1 - x_n}{2} \cdot (-1, (-1)^{1-c_1} \cdot x_1, \ldots, (-1)^{1-c_{n-1}} \cdot x_{n-1}),$$

where the $c_i \in \{0, 1\}$ are the coefficients of a primitive polynomial $1 + \sum_{i=1}^{n-1} c_i \cdot X^i + X^n$ over $GF(2)$. The time shift acts transitively on the punctuated hypercube $H' = H \setminus \{\vec{1}\}$ (as a group automorphism it has $\vec{1}$ as a fixed point), and more importantly, evaluations of a shifted function $f(T(\vec{x}))$ can be simulated from two evaluations of f by

$$f(T(x_1, \ldots, x_n)) = \frac{1 + x_n}{2} \cdot f(1, x_1, \ldots, x_{n-1}) + \frac{1 - x_n}{2} \cdot f(-1, (-1)^{1-c_1} \cdot x_1, \ldots, (-1)^{1-c_{n-1}} \cdot x_{n-1}). \quad (27)$$

Using the time shift T allows for the same strategy for the univariate Plookup argument. (See Appendix A.1 for a recap.) The argument is based on the fact that, given two sequences of field elements $(a_i)_{i=0}^{N-1}$ and $(t_i)_{i=0}^{N-1}$, then $\{a_i : j = 0, \ldots, N - 1\} \subseteq \{t_i : i = 0, \ldots, N - 1\}$ as sets, if and only if there exists a sequence $(s_i)_{i=0}^{2N-1}$ of double the size, which satisfies the lookup identity

$$\prod_{i=0}^{N-1} (X + s_i + s_{i+1 \bmod N} \cdot Y) = \prod_{i=0}^{N-1} (X + a_i + a_i \cdot Y) \cdot (X + t_i + t_{i+1 \bmod N} \cdot Y). \quad (28)$$

(The sequence $(s_i)_{i=0}^{2N-1}$ is the concatenation of the (a_i) and (t_i), ordered by value in the same way as given by t.) We again discuss two approaches[3] for dealing with the grand product obtained from (28), when sampling random $(\alpha, \beta) \leftarrow_\$ F^2$ for (X, Y). The first one applies a batched grand product argument over H, independent of M the number of columns, leading to a $\mathcal{O}(M^2)$ prover for similar reasons as Protocol 2 does. The second one proves the grand product argument over an extended hypercube $\bar{H}$ of size $M \cdot |H|$, which on the one hand leads to larger functions to be provided, but on the other hand has a $\mathcal{O}(M)$ prover which outperforms the first approach at a high number of columns.

---

[3]We point out that the presented strategies slightly differ from the one in [CBBZ22], which uses the more expensive grand product argument from [SL20].

The argument for not too many columns

Let $t : H' \to F$ be the lookup table, and $f_i : H' \to F$, $i = 1, \ldots, M$, the functions subject to the lookup. Although the functions are defined over the punctuated hypercube H', we assume arbitrary values at $\vec{1}$. (These will be ignored by the lookup argument.) The prover provides the ordered merge of the f_i together with t via additional functions $s_i : H' \to F$, $i = 1, \ldots, M + 1$, subject to the generalized lookup identity

$$\prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot Y) \cdot (X + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot Y)$$

$$= \prod_{\vec{x} \in H'} \prod_{i=1}^{M} (X + f_i(\vec{x}) + f_i(\vec{x}) \cdot Y) \cdot (X + t(\vec{x}) + t(T(\vec{x})) \cdot Y).$$

The identity is reduced to a grand product over H' using random samples $\alpha, \beta \leftarrow_\$ F$ for X and Y, yielding

$$\prod_{\vec{x} \in H'} h(\vec{x}) = 1,$$

where

$$h(\vec{x}) = \frac{\alpha + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot \beta}{\alpha + t(\vec{x}) + t(T(\vec{x})) \cdot \beta} \cdot \prod_{i=1}^{M} \frac{(\alpha + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot \beta)}{(\alpha + f_i(\vec{x}) + f_i(\vec{x}) \cdot \beta)}.$$

The prover computes the cumulative products of the values $h(\vec{x})$ along the orbit of the time shift T on H', starting with $\phi(-\vec{1}) = 1$, and setting

$$\phi(T^k(-\vec{1})) = \phi(T^{k-1}(-\vec{1})) \cdot h(T^{k-1}(-\vec{1})),$$

for $k = 1, \ldots, |H| - 1$. At the remaining point $\vec{x} = \vec{1}$ outside H', the prover sets $\phi(\vec{x})$ to zero. Correctness of the grand product is proven by the constraint on its initial value $\phi(-\vec{1}) = 1$, and the domain identity

$$\phi(T(\vec{x})) \cdot \tau(\vec{x}) \cdot \prod_{i=1}^{M} \varphi_i(\vec{x}) - \phi(\vec{x}) \cdot \sigma_{M+1}(\vec{x}) \cdot \prod_{i=1}^{M} \sigma_i(\vec{x}) = 0, \tag{29}$$

for all $\vec{x} \in H$, where $\tau(\vec{x}) = \alpha + t(\vec{x}) + \beta \cdot t(T(\vec{x}))$ and

$$\varphi_i(\vec{x}) = \alpha + (1 + \beta) \cdot f_i(\vec{x}),$$
$$\sigma_i(\vec{x}) = \alpha + s_i(\vec{x}) + \beta \cdot s_{i+1}(\vec{x}),$$

for $i = 1, \ldots, M$, except $\sigma_{M+1}(\vec{x}) = \alpha + s_{M+1}(\vec{x}) + \beta \cdot s_1(T(\vec{x}))$. As in Protocol 2, both constraints on ϕ, the initial value condition and the domain identity, are reduced to sumchecks over H by help of the Lagrange polynomials $L_H(\,.\,, -\vec{1})$ and $L_H(\,.\,, \vec{y})$, where $\vec{y} \leftarrow_\$ F^n$, and then combined into a single one by a batching randomness $\lambda \leftarrow_\$ F$. The resulting overall sumcheck is

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{y}), L_H(\vec{x}, -\vec{1}), \tau(\vec{x}), \varphi_1(\vec{x}), \ldots, \varphi_M(\vec{x}), \sigma_1(\vec{x}), \ldots, \sigma_{M+1}(\vec{x}), \phi(\vec{x}), \phi(T(\vec{x}))) = 0,$$

where

$$Q(L_H, L, \tau, \varphi_1, \ldots, \varphi_M, \sigma_1, \ldots, \sigma_{M+1}, \phi, \phi_T) = L_H \cdot \left(\phi_T \cdot \tau \cdot \prod_{i=1}^{M} \varphi_i - \phi \cdot \prod_{i=1}^{M+1} \sigma_i \right) + \lambda \cdot L \cdot (\phi - 1). \tag{30}$$

Note that Q has absolute degree $d = M + 3$, which is the same as in Protocol 2, but about the double of variables, $\nu = 2 \cdot M + 6$. Its arithmetic complexities are $|Q_\mathsf{M}| = 2 \cdot (M + 1) + 3$, $|Q_\mathsf{S}| = 2$, $|Q_\mathsf{A}| = 1$.

Prover costs

The prover cost for the multi-colum Plookup is as follows. Computing the values for τ and all φ_i, σ_i over H consumes overall

$$|H| \cdot (2 \cdot (M+1) \cdot \mathsf{M} + (3M+4) \cdot \mathsf{A}),$$

the quotient $h(\vec{x})$ is obtained within $|H| \cdot (2 \cdot M + 4) \cdot \mathsf{M}$, using batch inversion. From these values $\phi(x)$ over H is derived by another $|H| \cdot \mathsf{M}$. The domain evaluation for $L_H(\vec{X}, \vec{y})$ is obtained within $|H| \cdot (\mathsf{M} + \mathsf{A})$ operations, and the sumcheck costs

$$|H| \cdot \left(1 - \frac{1}{|H|}\right) \cdot ((4 \cdot M^2 + 25 \cdot M + 38) \cdot \mathsf{M} + (6 \cdot M + 20) \cdot \mathsf{S} + (2 \cdot M^2 + 14 \cdot M + 26) \cdot \mathsf{A}).$$

Neglecting the $1/|H|$-term, the overall cost of the prover is

$$|H| \cdot ((4 \cdot M^2 + 29 \cdot M + 46) \cdot \mathsf{M} + (6 \cdot M + 20) \cdot \mathsf{S} + (2 \cdot M^2 + 17 \cdot M + 31) \cdot \mathsf{A}). \tag{31}$$

This is comparable with (25), but does not take into account that the prover needs to supply a total of $M + 1$ functions over H instead of two.

A variant for a large number of columns

Suppose that we have a table function $t : H' \to F$ defined over the punctuated hypercube $H' = H \setminus \{\vec{1}\}$, and M column functions $f_1, \dots, f_M : H \to F$ defined on the *entire* hypercube. We assume that $M + 1 = 2^m$ for an integer $m \geq 1$, so that we can index the column functions as $f_{\vec{z}}$, with $\vec{z}$ from $\{\pm 1\}^m \setminus \{\vec{1}\}$. The ordered concatenation of the column functions and the table function has now $2^m \cdot |H| - 1$ entries, which we arrange in a single function s over $\bar{H} \setminus \{\vec{1}\}$ along the orbit of the time shift $T_{\bar{H}}$ on $\bar{H}$, where $\bar{H} = \{\pm 1\}^m \times H$. The value of s at $\vec{1}$ can be set arbitrary (it will be ignored by the lookup argument). Using the patched function

$$f(\vec{y}, \vec{x}) = \sum_{\vec{z} \in \{\pm 1\}^m \setminus \{\vec{1}\}} L_m(\vec{y}, \vec{z}) \cdot f_{\vec{z}}(\vec{x}),$$

over $\bar{H}$, where L_m is the Lagrange kernel for $\{\pm 1\}^m$, the lookup identity at random $(\alpha, \beta) \leftarrow_\$ F^2$ reads now as

$$\prod_{(\vec{y}, \vec{x}) \in \bar{H}'} (\alpha + s(\vec{y}, \vec{x}) + \beta \cdot s(T_{\bar{H}}(\vec{y}, \vec{x}))) = \prod_{(\vec{y}, \vec{x}) \in \bar{H}'} (\alpha + (1 + \beta) \cdot f(\vec{y}, \vec{x}) + L_m(\vec{y}, \vec{1}) \cdot (t(\vec{x}) + \beta \cdot t(T_H(\vec{x})))),$$

where $\bar{H}' = \bar{H} \setminus \{\vec{1}\}$. (Notice that the α-term for t on the right hand side is combined with the α-term for f.) In other words the prover needs to show that

$$\prod_{(\vec{y}, \vec{x}) \in \bar{H}'} \frac{\varphi(\vec{y}, \vec{x})}{\sigma(\vec{y}, \vec{x})} = 1,$$

where

$$\varphi(\vec{y}, \vec{x}) = \alpha + (1 + \beta) \cdot f(\vec{y}, \vec{x}) + L_m(\vec{y}, \vec{1}) \cdot (t(\vec{x}) + \beta \cdot t(T_H(\vec{x}))),$$
$$\sigma(\vec{y}, \vec{x}) = \alpha + s(\vec{y}, \vec{x}) + \beta \cdot s(T_{\bar{H}}(\vec{y}, \vec{x})).$$

As before, the prover provides a function ϕ over $\bar{H}'$ for the cumulative products of the values of $h(\vec{x}) = \frac{\varphi(\vec{y}, \vec{x})}{\sigma(\vec{y}, \vec{x})}$ along the orbit of $T_{\bar{H}}$, starting with $\phi(-\vec{1}) = 1$, and setting

$$\phi(T_{\bar{H}}^k(-\vec{1})) = \phi(T_{\bar{H}}^{k-1}(-\vec{1})) \cdot h(T_{\bar{H}}^{k-1}(-\vec{1})),$$

16

for $k = 1, \ldots, |\bar{H}| - 2$. At the remaining point $\vec{1}$ we set $\phi(\vec{1}) = 0$. Correctness of ϕ is ensured by $\phi(-\vec{1}) = 1$, and the domain identity

$$\phi(T_{\bar{H}}(\vec{y}, \vec{x})) \cdot \varphi(\vec{y}, \vec{x}) - \phi(\vec{y}, \vec{x}) \cdot \sigma(\vec{y}, \vec{x}) = 0$$

over $\bar{H}$. Applying the Lagrange polynomials $L_{\bar{H}}(. , \vec{z})$, $\vec{z} \leftarrow_{\$} F^{m+n}$, and $L_{\bar{H}}(. , -\vec{1})$, the constraints on ϕ are reduced to sumchecks over $\bar{H}$, which are then combined into a single one using a random $\lambda \leftarrow_{\$} F$. The overall sumcheck is

$$\sum_{(\vec{y}, \vec{x}) \in \bar{H}} Q(L_{\bar{H}}((\vec{y}, \vec{x}), \vec{z}), L_{\bar{H}}((\vec{y}, \vec{x}), -\vec{1}), \varphi(\vec{y}, \vec{x}), \sigma(\vec{y}, \vec{x}), \phi(\vec{y}, \vec{x}), \phi(T_{\bar{H}}(\vec{y}, \vec{x}))) = 0,$$

where

$$Q(L_H, L, \varphi, \sigma, \phi, \phi_T) = L_H \cdot (\phi_T \cdot \varphi - \phi \cdot \sigma) + \lambda \cdot L \cdot (\phi - 1). \tag{32}$$

The polynomial Q has $\nu = 6$ variables, absolute degree $d = 3$, and its arithmetic complexities are $|Q_\mathsf{M}| = 5$, $|Q_\mathsf{S}| = 2$, $|Q_\mathsf{A}| = 1$.

Prover cost

Computing the values of the patched function φ over $\bar{H}$ takes $|\bar{H}| \cdot (1 \cdot \mathsf{M} + 2 \cdot \mathsf{A})$, and the same number of operations are needed for σ. The quotient h over $\bar{H}$ is obtained by batch inversion, consuming

$$|\bar{H}| \cdot (3 + 1) \cdot \mathsf{M},$$

and the values of ϕ are computed within another $|\bar{H}|$ multiplications. Last but not least, computing $L_{\bar{H}}(. , \vec{z})$ over $\bar{H}$ costs $|\bar{H}| \cdot (\mathsf{M} + \mathsf{A})$, and the sumcheck

$$|\bar{H}| \cdot \left(1 + \frac{1}{|\bar{H}|}\right) \cdot (38 \cdot \mathsf{M} + 20 \cdot \mathsf{S} + 26 \cdot \mathsf{A}).$$

Neglecting the $\frac{1}{|\bar{H}|}$-term, the overall cost of the prover is

$$(M + 1) \cdot |H| \cdot (45 \cdot \mathsf{M} + 20 \cdot \mathsf{S} + 30 \cdot \mathsf{A}), \tag{33}$$

whereas it needs to provide the functions s and ϕ over $\bar{H}$, which amounts the equivalent of $2 \cdot (M + 1)$ functions over $|H|$. Based on our benchmark-backed equivalent number of field operations for a multi-scalar multiplications (Table 3), we obtain the following break even points.

Table 2: The estimated number of columns M where the $\mathcal{O}(M)$ strategy starts to perform better than the protocol from Section 4.1. The numbers are based on the operation counts (31) and (33) for the oracle prover, and the benchmarks for a multi-scalar multiplication over the Pallas curve, see Section **??**.

$\log|H|$	12	14	16	18
M	143	120	103	92

Table 3: Benchmark of Halo2's Pippenger multi-scalar multiplication in the Pallas curve, varying the number N of scalars. The benchmarks where done on a AMD Ryzen 7 PRO 4750U, 32GB RAM DDR4, restricting to a single core.

$\log N$	Pippenger of size N	$2^8 \cdot N$ field mult.	equivalent field mult.
12	46.010 ms	21.11 ms	$N \cdot 557 \cdot \mathsf{M}$
14	153.67 ms	42.315 ms	$N \cdot 464 \cdot \mathsf{M}$
16	522.13 ms	169.70 ms	$N \cdot 394 \cdot \mathsf{M}$
18	1.869 ms	679.27 ms	$N \cdot 351 \cdot \mathsf{M}$

Comparison with logarithmic derivative lookups

The main advantage of logarithmic derivative lookups over the ones from the current section is not the arithmetic cost as reflected by the operation counts (25), (26) and (25), (26), but the number and sizes of oracle functions the prover needs to provide. That oracle costs depend on the used commitment scheme. To give an estimate for these costs in the case of an elliptic curve based Lagrange commitment scheme (such as the IPA), we rely on a benchmark-backed equivalent of field multiplications for the multi-scalar multiplication in the Pallas curve. These are found in Table 3.

With these equivalence measure and the operation counts from (25), (26) and (31), (33), we obtain the following ratios of field multiplications.

Table 4: The estimated performance advantage of the logarithmic derivative lookups over the ones from this section, as the ratio of their number of field multiplications $r = \mathsf{M}(Plookup)/\mathsf{M}(logD)$. The numbers are based on the equivalent number of field multiplication from Table 3. For hypercube sizes $|H|$ ranging from 2^{12}–2^{18} we describe the maximum ratio r_{max} over the number of columns M, as well as the ranges for M over which r is larger than 2 and 3, respectively. The minimum ratio is throughout $r = 1.5$ and obtained in the single-column setting $M = 1$.

$\log|H|$	$r \geq 3$	$r \geq 2$	r_{max}
12	$M \in [5, 41]$	$M \in [3, 87]$	4.1 (at $M = 15$)
14	$M \in [6, 32]$	$M \in [3, 71]$	3.8 (at $M = 13$)
16	$M \in [6, 26]$	$M \in [3, 59]$	3.5 (at $M = 12$)
18	$M \in [6, 21]$	$M \in [3, 52]$	3.2 (at $M = 12$)

References

[Ark]   Arkworks: An ecosystem for developing and programming with zkSNARKs. `https://github.com/arkworks-rs/`.

[Azt]   Aztek Protocol. `https://github.com/AztecProtocol/`.

[BCC+16]   Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In M. Fischlin and J.S. Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*. Springer, 2016. Full paper: `https://eprint.iacr.org/2016/263`.

[BCG+18]   Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *ASIACRYPT 2018*, volume 11272 of *LNCS*, 2018. Full paper: https://eprint.iacr.org/2018/380.

[BFS20]   Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT 2020*, 2020. Full paper: https://eprint.iacr.org/2019/1229.

[BG12]   Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT 2012*, volume 7237 of *LNCS*. Springer, 2012.

[BLH+]   Sean Bowe, Ying Tong Lai, Daira Hopwood, Jack Grigg, and Steven Smith. Halo 2. https://github.com/zcash/halo2.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. In *IACR ePrint Archive 2018/046*, 2018. https://eprint.iacr.org/2018/046.

[BSCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems. In *ITCS'13*, 2013. Full paper: https://eprint.iacr.org/2012/071.

[BSCS16]   Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC 2016*, pages 31–60, 2016.

[CBBZ22]   Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: PLONK with a linear-time prover and high-degree costum gates. In *IACR ePrint Archive 2020/1355*, 2022. https://eprint.iacr.org/2022/1355.

[Gab22]   Ariel Gabizon. Multiset checks in PLONK and Plookup. hackmd.io, 2022. https://hackmd.io/@arielg/ByFgSDA7D.

[GK22]   Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of the table size. In *IACR ePrint Archive 2022/1447*, 2022. https://eprint.iacr.org/2022/1447.

[GPR21]   Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo– a turing-complete STARK-friendly cpu architecture. In *IACR ePrint Archive 2021/1063*, 2021. https://eprint.iacr.org/2021/1063.

[GW20]   Ariel Gabizon and Zachary J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. In *IACR ePrint Archive 2020/315*, 2020. https://eprint.iacr.org/2020/315.

[GWC19]   Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical non-interactive arguments of knowledge. In *IACR ePrint Archive 2019/953*, 2019. https://eprint.iacr.org/2019/953.

[KZG10]   Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commimtments to polynomials and their applications. In Abe M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*. Springer, 2010.

[LFKN92]   Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *Journal of the Association for Computing Machinery*, volume 39, pages 859–868, 1992. Full paper: https://eprint.iacr.org/2017/1066.

[Mid]   Polygon Miden: A STARK-based virtual machine. `https://github.com/maticnetwork/miden`.

[PK22]   Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. In *IACR ePrint Archive 2022/957*, 2022. `https://eprint.iacr.org/2022/957`.

[PST13]   Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC 3*, volume 7785 of *LNCS*. Springer, 2013.

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

[Tha13]   Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO 2013*, volume 8043 of *LNCS*, 2013. Full paper: `https://eprint.iacr.org/2013/351`.

[ZBK⁺22]   Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *IACR ePrint Archive 2022/621*, 2022. `https://eprint.iacr.org/2022/621`.

Appendix

Univariate Lookups

Let us sketch Plookup variant from [GW20] described in the blog post [Gab22], generalized to the case of multi-column lookups. Here, the set H denotes the univariate domain $H = \{x \in F : x^n = 1\}$ with generator g. Suppose that $f_i : H \to F$, $i = 0, \ldots, M - 1$, are functions for which we want to prove that its range is contained in that of $t : H \to F$, i.e. $\bigcup_{i=0}^{M-1} \{f_i(x)\}_{x \in H} \subseteq \{t(x)\}_{x \in H}$ as sets. Let $(f_{i,k})_{k=0}^{n-1}$ and $(t_k)_{k=0}^{n-1}$ denote the sequences of values given by the index with respect to g, i.e. $f_{i,k} = f(g^k)$ and $t_k = f(g^j)$, and let m_k be the overall multiplicity of t_k in the union (or, concatenation) of the sequences $(f_{i,k})_i$, $i = 1, \ldots, M$. Consider the sequence

$$\bar{s} = (\bar{s}_i)_{i=0}^{(M+1) \cdot n - 1} = (\underbrace{t_0, \ldots, t_0}_{1+m_0 \text{ times}}, \underbrace{t_1, \ldots, t_1}_{1+m_1 \text{ times}}, \ldots, \underbrace{t_{n-1}, \ldots, t_{n-1}}_{1+m_{n-1} \text{ times}}),$$

and split it into $(M + 1)$ sequences of length n,

$$s_i = (\bar{s}_{i+k \cdot (M+1)})_{k=0}^{n-1},$$

with $i = 0, \ldots, M$. We regard these sequences again as functions on H. Then

$$\bigcup_{i=0}^{M-1} \{s_i(x), s_{i+1}(x)\}_{x \in H} \cup \{(s_M(x), s_0(g \cdot x))\}_{x \in H} = \bigcup_{i=0}^{M-1} \{(f_i(x), f_i(x))\}_{x \in H} \cup \{(t(x), t(g \cdot x))\}_{x \in H} \quad (34)$$

as multisets. Moreover, this multiset equality is in fact equivalent to $\bigcup_{i=0}^{M-1} \{f_i(x)\}_{x \in H} \subseteq \{t(x)\}_{x \in H}$. The set inclusion holds if and only if there exists functions $s_0, \ldots, s_M : H \to F$ satisfying (34). The multiset equation (34) is equivalent to the formal identity

$$\prod_{x \in H} \prod_{i=0}^{M-1} (X + s_i(x) + s_{i+1}(x) \cdot Y) \cdot (X + s_M(x) + s_0(g \cdot x) \cdot Y)$$

$$= \prod_{x \in H} \prod_{i=0}^{M-1} (X + f_i(x) + f_i(x) \cdot Y) \cdot (X + t(x) + t(g \cdot x) \cdot Y),$$

which is reduced by random sampling $\alpha, \beta \leftarrow_{\$} F$ for X and Y, to the grand product

$$\prod_{x \in H} \frac{\sigma_M(x)}{\tau(x)} \cdot \prod_{i=0}^{M-1} \frac{\sigma_i(x)}{\varphi_i(x)} = 1, \tag{35}$$

where $\tau(x) = \alpha + t(x) + \beta \cdot t(g \cdot x)$, and

$$\varphi_i(x) = \alpha + (1 + \beta) \cdot f_i(x),$$
$$\sigma_i(x) = \alpha + s_i(x) + \beta \cdot s_{i+1}(x),$$

for $i = 0, \ldots, M - 1$, except for $\sigma_M(x) = \alpha + s_M(x) + \beta \cdot s_0(g \cdot x)$. To prove the grand product over the quotients $h(x) = \frac{\sigma_M(x)}{\tau(x)} \cdot \prod_{i=0}^{M-1} \frac{\sigma_i(x)}{\varphi_i(x)}$, the prover sets $\phi(1) = 1$, and computes recursively $\phi(g^i) = \phi(g^{i-1}) \cdot h(g^{i-1})$, for $i = 1, \ldots, n - 1$. (Recall that the grand product is equal to 1 if and only if $\phi(g^n) = \phi(1) = 1$.) This leads to the following domain identities over H,

$$\phi(g \cdot x) \cdot \tau(x) \cdot \prod_{i=0}^{M-1} \varphi_i(x) - \phi(x) \cdot \prod_{i=0}^{M} \sigma_i(x) = 0,$$

and

$$L_H(x, 1) \cdot (\phi(x) - 1) = 0,$$

where $L_H(X, Y) = \frac{1}{n} \cdot \frac{Y \cdot v_H(X) - X \cdot v_H(Y)}{X - Y}$ is the Lagrange kernel for the univariate domain H. The indentities are combined into a single one using a random $\lambda \leftarrow_{\$} F$, yielding the overall polynomial identity is

$$\phi(g \cdot x) \cdot \tau(x) \cdot \prod_{i=0}^{M-1} \varphi_i(x) - \phi(x) \cdot \prod_{i=0}^{M} \sigma_i(x) + \lambda \cdot L_H(X, 1) \cdot (\phi(X) - 1) = 0 \mod v_H(X),$$

where $v_H(X) = X^n - 1$ is the vanishing polynomial of H.