Multivariate lookups based on logarithmic derivatives

Ulrich Haböck

Orbis Labs, Polygon Zero
uhaboeck@polygon.technology

December 21, 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 lookups over the boolean hypercube, which makes use of a single multiplicity function instead of working with a rearranged union of table and witnesses, as Plookup [GW20] does. For single-column lookups the performance is comparable to Plookup, taken to the multivariate setting by Hyperplonk+ [CBBZ22]. However, the real power of our argument unfolds in the case of "batch-column" lookups, where multiple columns are subject to the same table lookup: While the number of field operations is comparable to the Plookup strategy, the oracles provided by our prover are significantly fewer. For example, given $M = 20$ columns of length between 2^{12} and 2^{18} , and assuming a commitment scheme over an elliptic curve with a 256 bit large base field, our paper-pencil operation counts indicate that the logarithmic derivative lookup is more than 3.9 times faster.

Contents

*This updated version clarifies the trade-off between algebraic degree and the number of commitments, previously expressed by two protocol variants (one for small and one for large numbers of columns). It further discusses Polygon Miden's bounded multiplicity encoding, applied to domain-sized table lookups.

1 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 arguments for statements which are otherwise quite expensive to arithmetize. Main applications are (1) lookups for relations of high algebraic complexity, and (2) lookups for interval ranges, the latter of which are extensively used by zero-knowledge virtual machines. Although closely related to permutation (or shuffle) arguments [BG12], a first explicit occurence of lookups dates back to Arya [BCG$^+$18], to the best of our knowledge. While their argument handles multiplicities directly in a quite costly manner, Plookup [GW20] greatly improved over [BCG$^+$18] using a rather geometric approach. Since then Plookup (and variants of it) is *the* general purpose lookup argument used in practical applications.

In this paper we describe a lookup argument based which is based on logarithmic derivatives. 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 the treatment of multiplicities is costly in the product approach, it turns cheap when using logarithmic derivatives: 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 a lookup proof which is 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 *batch-column* lookups, where several sequences ("columns") are subject to the same table lookup, a situation that often arises in zero-knowledge virtual machines, enforcing execution trace elements being valid machine words. For example, the arithmetic unit of Polygon's System Zero [Pol] has about 70 columns subject to one and the same range check, or the tinyRAM implementation of Orbis Labs [Lab] has 12 columns subject to one and the same lookup table. In our lookup the oracle costs, measuring the number and sizes of the oracles, are significantly lower than for a lookup based on the Plookup strategy.

We stress the fact that we are not the only ones who exploit fractional decompositions for lookups. Concurrently, building on the work of [ZBK$^+$22] and [PK22] on "large-table" lookups, Gabizon and Khovratovich [GK22] describe a bilinear argument, the proving-time of which is independent of the table size.

The main contribution of [GK22] is a univariate oracle proof for the radical of a witness sequence (i.e. the sequence with multiple occurences removed) which is almost identical to our approach[1].

This document focuses on batch-column lookups with asymptotically linear prover costs. It 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 informally introduce Lagrange interactive oracle proofs, an oracle model we consider suitable for arguments which are based on the Lagrange representation of polynomials rather than their coefficients. In Section 3 we describe our lookup argument based on the logarithmic derivative. For comparison reasons, we add an extra section (Section 4) in which we outline batch-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 protocol is written for the multilinear setting, its translation into a univariate proof is straight-forward. We expect these univariate arguments to improve similarly over multi-column lookups based on the Plookup strategy.

2 Preliminaries

2.1 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. □

[1] Almost simultaneously, G. Roh, W. Dai, M. Jabbour, and A. He published the same idea in a blog post [RDJH22]. To underpin that our work was not influenced by concurrent efforts we refer to the verified commit on our Github repository, `https://github.com/Orbis-Tertius/MVlookups/commit/4a8816a04e8107e05d4bb0897ee52e72210c84b8`

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

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

2.2 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[2] $c \in F$. 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. □

2.3 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[3]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. □

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

[3]At the time of writing we learned that also others are aware of Lemma 3. In a Delendum Frontiers session on multi-party computation (6. Oct. 2022) an audience member pointed out its usefulness for permutation arguments.

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. □

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 $i = 1, \ldots, N$.

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 $\{b_i\}$. □

2.4 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.

2.5 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

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

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

$$Q(Y_1, \ldots, Y_\nu) = \sum_{(i_1,\ldots,i_\nu)\in\{0,1\}^\nu} c_{i_1,\ldots,i_\nu} \cdot Y_1^{i_1} \cdots Y_\nu^{i_\nu}$$

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 and additions A to evaluate Q by $|Q|_\mathsf{M}$ and $|Q|_\mathsf{A}$, respectively. (For simplicity we count substractions as additions.) 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 $|Q|_{\mathsf{M}} \cdot \mathsf{M} + |Q|_{\mathsf{A}} \cdot \mathsf{A}$. In terms of field multiplications M and substractions/additions A, step i consumes

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

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

$$|H| \cdot \left(1 - \frac{1}{|H|}\right) \cdot \big((d \cdot \nu + (d+1) \cdot |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (d+1) \cdot (\nu + |Q|_{\mathsf{A}}) + d) \cdot \mathsf{A}\big). \tag{11}$$

We shall use the simplified formula

$$|H| \cdot (d+1) \cdot \big((\nu + |Q|_{\mathsf{M}}) \cdot \mathsf{M} + (\nu + |Q|_{\mathsf{A}}) \cdot \mathsf{A}\big) \tag{12}$$

for the operation counts of our lookup protocol.

3 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

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{X + f_i(\vec{x})} = \sum_{\vec{x} \in H} \frac{m(\vec{x})}{X + t(\vec{x})}, \tag{13}$$

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

$$m(\vec{x}) = \frac{m_f(t(\vec{x}))}{m_t(t(\vec{x}))} = \frac{\sum_{i=1}^{M} |\{\vec{y} \in H : f_i(\vec{y}) = t(\vec{x}))|}{|\{\vec{y} \in H : t(\vec{y}) = t(\vec{x}))|}. \tag{14}$$

Given a random challenge $x \leftarrow\!\!\$\, F$ from the verifier, the prover shows that the rational identity (13) holds at $X = x$, i.e.

$$\sum_{\vec{x} \in H} \sum_{i=1}^{M} \frac{1}{x + f_i(\vec{x})} - \frac{m(\vec{x})}{x + t(\vec{x})} = 0, \tag{15}$$

whenever evaluation is possible. However, in order to apply the sumcheck protocol we need to turn the fractional expression into a polynomial one. For that, the prover splits the sum into partial sums of (roughly) the same number of terms ℓ, and provides multilinear helper functions for each sum. These helper functions are subject to a domain identity of algebraic degree essentially equal to the number of reciprocal terms in the sum. In practice one chooses ℓ so that the prover time is minimal for the given number of columns M. That optimum depends on the used polynomial commitment scheme. The costlier the computation of a commitment, the higher the algebraic degree for the domain identity of the helper function can be.

In Section 3.1 we describe our batch-column lookup, Protocol 2, and we sketch its soundness analysis in Section 3.2. In the following Section 3.3 we give detailed operations counts for its oracle prover, which we then use to estimate the optimal choice of ℓ assuming a commitment scheme which uses an ordinary

sized elliptic curve[4]. Eventually, we point out the generalization of the lookup argument to vector-valued tables.

3.1 The protocol

Let ℓ be the chosen sum size, $[0, M] = \bigcup_{k=1}^{K} I_k$ the decomposition of $[0, M]$ into $K = \left\lceil \frac{M+1}{\ell} \right\rceil$ subintervals $I_k = [(k-1) \cdot \ell, k \cdot \ell) \cap [0, M]$, $k = 1, \ldots, K$. Let

$$h_k(\vec{x}) = \sum_{i \in I_k} \frac{m_i(\vec{x})}{\varphi_i(\vec{x})}, \quad k = 1, \ldots, K, \tag{16}$$

be the respective partial sum of consecutive terms in the overall expression $\frac{m(x)}{x+t(\vec{x})} - \frac{1}{x+f_1(\vec{x})} - \ldots - \frac{1}{x+f_M(\vec{x})}$, where we used the notation

$$\begin{aligned} m_0(\vec{x}) = m(\vec{x}), \quad & \varphi_0(\vec{x}) = x + t(\vec{x}), \\ m_i(\vec{x}) = -1, \quad & \varphi_i(\vec{x}) = x + f_i(\vec{x}), \quad i = 1, \ldots, M. \end{aligned}$$

The prover provides the oracles for $h_1, \ldots, h_k$, subject to

$$\sum_{\vec{x} \in H} h_1(\vec{x}) + \ldots + h_K(\vec{x}) = 0,$$

and the domain identities

$$h_k(\vec{x}) \cdot \prod_{i \in I_k} \varphi_i(\vec{x}) = \sum_{i \in I_k} m_i(\vec{x}) \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j(\vec{x}) \tag{17}$$

over H, the latter are reduced to sumchecks by applying the Lagrange kernel $L_H(\,.\,, \vec{z})$ at a randomly chosen $\vec{z} \leftarrow\!\!\$\, F^n$. All sumchecks are then combined into a single one, using random scalars $\lambda_1, \ldots, \lambda_K \leftarrow\!\!\$\, F$.

Protocol 2 (Batch-column lookup over $H = \{\pm 1\}^n$)**.** *Let M be an integer, and F a finite field with characteristic $p > M \cdot 2^n$. Fix any integer ℓ, $1 \leq \ell \leq M+1$, and let $K = \left\lceil \frac{M+1}{\ell} \right\rceil$. 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* (14)*, 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 values over H for the partial sums $h_1(\vec{x}), \ldots, h_K(\vec{x})$ as defined above, and sends their oracles to the verifier.*
3. *The verifier responds with a random vector $\vec{z} \leftarrow\!\!\$\, F^n$ and random batching scalars $\lambda_1, \ldots, \lambda_K \leftarrow\!\!\$\, F$. Now, both prover and verifier engage in the sumcheck protocol (Protocol 1) for*

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{z}), m(\vec{x}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), h_1(\vec{x}), \ldots, h_K(\vec{x})) = 0,$$

where

$$Q(L, m, \varphi_0, \ldots, \varphi_M, h_1, \ldots, h_K) = \sum_{k=1}^{K} h_k + L \cdot \lambda_k \cdot \left(h_k \cdot \prod_{i \in I_k} \varphi_i - \sum_{i \in I_k} m_i \cdot \prod_{j \in I_k \setminus \{i\}} \varphi_j \right), \tag{18}$$

[4]By ordinary sized we mean a 128 bit secure curve over a 256 bit large base field. This covers both KZG-like commitments in Barreto-Naehrig curves, and IPA commitments in an ordinary elliptic curve.

with $m_0 = m$, and all other $m_i = -1$, $i = 1, \ldots, M$. The sumcheck protocol outputs the expected value v for the multivariate polynomial

$$Q(L_H(\vec{X}, \vec{z}), m(\vec{X}), \varphi_0(\vec{X}), \ldots, \varphi_M(\vec{X}), h_1(\vec{X}), \ldots, h_K(\vec{X})) \tag{19}$$

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

4. *The verifier queries $[m], [t], [f_1], \ldots, [f_M], [h_1], \ldots, [h_K]$ for their inner product with $L_H(\,.\,, \vec{r})$, and uses the answers to check whether (19) 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_0 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 for h_0 to

$$\left(h_0 \cdot \prod_{i \in I_0} \varphi_i - \sum_{i \in I_0} m_k \cdot \prod_{j \neq i} \varphi_j \right) \cdot \varphi_0 = 0 \tag{20}$$

over H, which imposes no condition on h_0 whenever $\varphi_0(\vec{x}) = 0$. However, this approach comes at the cost of a slight increase in the degree of Q with respect to φ_0.

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_0 needs to be provided by the prover. More interestingly, Protocol 2 is easily extended to a proof of range equality, showing that $\bigcup_{i=1}^{M} \{f_i(\vec{x})\}_{\vec{x} \in H} = \{t(\vec{x})\}_{\vec{x} \in H}$ as sets. For this the prover additionally shows that $m \neq 0$ over H, which is done by providing another auxiliary function $g : H \to F$ subject to $g \cdot m = 1$ over H. However, we are not aware of any application of this fact.

3.2 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 (13) is turned into a polynomial identity of degree at most $|H| \cdot (M + 1) - 1$ by multiplying it with the common denominator

$$\prod_{\vec{x} \in H} (X + t(\vec{x})) \cdot \prod_{i=1}^{M} (X + f_i(\vec{x})). \tag{21}$$

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

$$\varepsilon_1 \leq \frac{(M + 1) \cdot |H| - 1}{|F| - |H|}. \tag{22}$$

The soundness error due to the reduction of the domain identities (17) to the Lagrange kernel based sumcheck is altogether

$$\varepsilon_2 \leq \frac{K}{|F|},$$

as scalar products with the Lagrange kernel translate to point evaluation of the multilinear extension. Finally, the batching step of the $K+1$ sumchecks has soundness error $\varepsilon_3 = \frac{1}{|F|}$. This yields the following theorem.

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

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

where $\varepsilon_{sumcheck}$ is the soundness error of the sumcheck argument (9) *over H for a multivariate polynomial in $M+4$ variables with maximum individual degree $M+3$.*

3.3 Computational cost and optimal sum size

The polynomial Q from (18) has $\nu = M+K+3$ variables, and absolute degree $d = \ell+2$. Let us describe a domain evaluation strategy for Q which uses of batch inversion. This strategy allows us to evaluate Q much more efficiently than using (18), but demands a modification of the sumcheck operation count formula (12). Assume that in each group I_k the inverses of φ_i are given, except for one distinct $i_k \in I_k$. Then we may evaluate the domain identity terms in Q by the fractional representation

$$h_k \cdot \prod_{i\in I_k} \varphi_i - \sum_{i\in I_k} m_i \cdot \prod_{j\in I_k\setminus\{i\}} \varphi_j = \prod_{i\in I_k\setminus\{i_k\}} \varphi_j \cdot \left(\varphi_{i_k}\cdot\left(h_k - \sum_{i\in I_k\setminus\{i_k\}} \frac{m_i}{\varphi_i}\right) - m_{i_k}\right)$$

For $k \geq 2$, all involved $m_i = -1$, and we have

$$\prod_{i\in I_k\setminus\{i_k\}} \varphi_j \cdot \left(\varphi_0\cdot\left(h_k + \sum_{i\in I_k\setminus\{i_k\}} \frac{1}{\varphi_i}\right) + 1\right), \tag{23}$$

regardless of the choice of i_k. For the first group $k=1$, we chose $i_1 = 0$, so that

$$\prod_{i\in I_1\setminus\{0\}} \varphi_j \cdot \left(\varphi_0\cdot\left(h_k + \sum_{i\in I_1\setminus\{0\}} \frac{1}{\varphi_i}\right) - m\right). \tag{24}$$

In both cases (24) and (23) the evaluation costs are $\ell\cdot\mathsf{M} + \ell\cdot\mathsf{A}$, counting substractions as additions. Henceforth Q is evaluated in overall

$$K\cdot(\ell+2)\cdot\mathsf{M} + K\cdot(\ell+1)\cdot\mathsf{A}. \tag{25}$$

Now, to attribute the inverses in formula (12), we increase the multiplicative complexity by $3\cdot(M-K)$, which represents the fractional cost of the batch inversion[5]of all the φ_i except one in each of the groups. This yields the following equivalent complexities

$$|Q_{\mathsf{M}}| = 3\cdot M + K\cdot(\ell-1), \quad |Q_{\mathsf{A}}| = K\cdot(\ell+1),$$

which we may plug into formula (12).

[5]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.

Table 1: 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	84.78 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	$N \cdot 351 \cdot \mathsf{M}$

The prover cost of Protocol 2 is as follows: Given the values of t and $f_1, \ldots, f_M$ over H, computing $\varphi_0 = x + t$ and $\varphi_1 = x + f_1, \ldots, \varphi_M = x + f_M$ costs $|H| \cdot (M+1) \cdot \mathsf{A}$, and their reciprocals $\frac{1}{\varphi_0}, \ldots, \frac{1}{\varphi_M}$ are obtained within $3 \cdot |H| \cdot (M+1) \cdot \mathsf{M}$, using batch inversion. With these reciprocals we obtain the values for $h_1, \ldots, h_K$ within overall $|H| \cdot (\mathsf{M} + K \cdot (\ell - 1) \cdot \mathsf{A})$. By the remark following Lemma 1, the values for $L_H(\vec{X}, \vec{y})$ over H are computed in $|H| \cdot (\mathsf{M} + \mathsf{A})$ operations. Hence the total cost of the preparation phase is

$$|H| \cdot ((3 \cdot M + 5) \cdot \mathsf{M} + (M + K \cdot (\ell - 1) + 2) \cdot \mathsf{A}.$$

According to (12) the sumcheck costs

$$|H| \cdot (\ell + 3) \cdot \big((4 \cdot M + 3 + \ell \cdot K) \cdot \mathsf{M} + (M + 3 + (\ell + 2) \cdot K) \cdot \mathsf{A}\big).$$

However, as we may reuse the reciprocals of $\varphi_0, \ldots, \varphi_M$ in the first step of the sumcheck, we may correct the sumcheck cost by substracting $|H| \cdot 3 \cdot (M - K)$. The total costs of the oracle prover are

- arithmetic costs of
$$|H| \cdot (K + 5 + (\ell + 3) \cdot (4 \cdot M + 3 + \ell \cdot K)) \cdot \mathsf{M}, \tag{26}$$
neglecting field additions and substractions, and
- oracle costs of
$$K + 1 = \left\lceil \frac{M+1}{\ell} \right\rceil + 1 \tag{27}$$
oracles of size $|H|$.

The optimal choice of ℓ

Let us discuss the concrete impact of the partial sum size ℓ on the overall prover cost. If $\ell = 1$, then each reciprocal is given a helper function, resulting in $M + 1$ additional commitments (besided the one for m) but a very low degree of Q, $d = 3$. Increasing ℓ reduces the number of commitments, but comes at the cost of a growing degree. The most extreme case is $\ell = M + 1$, where the prover provides a single additional commitment besides m, but needs to cope with a degree of $d = M + 3$, resulting in sumcheck costs which are quadratic in M. The optimal choice for ℓ is between these two cases, but depends on the concrete polynomial commitment scheme. We consider elliptic curve based commitments. To take the commitment costs into account we rely on the benchmarks from Table 1 which measure the equivalent number of field multiplications for a multi-scalar multiplication in an ordinary-sized elliptic curve of 128 bit security.

Table 2: An estimate of the partial sum size ℓ in Protocol 2, which minimizes the prover cost. The numbers are based on the operation counts (33) for the oracle prover and the benchmarks for a multi-scalar multiplication in the Pallas curve, Table 1.

$\log N$	ℓ				field mult. per col.			
	$M = 10$	$M = 20$	$M = 70$	$M = \infty$	$M = 10$	$M = 20$	$M = 70$	$M = \infty$
12	11	11	11	11	$N \cdot 188$	$N \cdot 158$	$N \cdot 135$	$N \cdot 120$
14	11	11	9	9	$N \cdot 170$	$N \cdot 144$	$N \cdot 120$	$N \cdot 112$
16	11	11	9	9	$N \cdot 155$	$N \cdot 132$	$N \cdot 112$	$N \cdot 104$
18	11	11	8	8	$N \cdot 147$	$N \cdot 126$	$N \cdot 106$	$N \cdot 99$

3.4 Vector-valued lookups

As Plookup, Protocol 2 is easily generalized to functions with multilinear values,

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

$i = 1, \dots, M$, without changing the soundness error bound from Theorem 4. As $F[X, Y_1, \dots, Y_k]$ is a unique factorization domain, and polynomials of the form $X - \sum_{(i_1,\dots,i_k)\in\{0,1\}^k} c_{i_1,\dots,i_k} \cdot Y_1^{i_1} \cdots Y_k^{i_k}$ 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 $F(X, Y_1, \dots, Y_k)$, if and only if there exists a function $m : H \to F$ such that

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

The only change to Protocol 2 is that the verifier now samples x from F and $\vec{y} = (y_1, \dots, y_k)$ 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 $x - t(\vec{x})(\vec{y})$.

4 Multivariate Plookup

In this section we sketch batch-column lookups using the Plookup strategy and the multivariate time shift introduced by Hyperplonk [CBBZ22]. The time shift $T : H \to H$ on the boolean hypercube $H = \{\pm 1\}^n$ 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[6]on the punctuated hypercube

$$H' = H \setminus \{\vec{1}\},$$

and more importantly, evaluations of a shifted function $f(T(\vec{x}))$ can be simulated from two evaluations of f by

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

4.1 Batch-column Plookup

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 union of the f_i together with t in a piecewise manner, via the additional functions $s_i : H' \to F$, $i = 1, \ldots, M + 1$. The *Plookup identity* (in the [Gab22] style) is then

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

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

$$\prod_{\vec{x} \in H'} h(\vec{x}) = 1, \tag{28}$$

where

$$h(\vec{x}) = \frac{\alpha + s_{M+1}(\vec{x}) + s_1(T(\vec{x})) \cdot \beta}{\alpha + t(\vec{x}) + t(T(\vec{x})) \cdot \beta} \cdot \prod_{i=1}^{M} \frac{(\alpha + s_i(\vec{x}) + s_{i+1}(\vec{x}) \cdot \beta)}{(\alpha + f_i(\vec{x}) + f_i(\vec{x}) \cdot \beta)}. \tag{29}$$

As in Protocol 2, we control the algebraic degree of the resulting identity for (28) by splitting the product (29) into $K = \left\lceil \frac{M+1}{\ell} \right\rceil$ partial products of size ℓ, where $1 \leq \ell \leq M + 1$. For this we use the notation

$$\varphi_0(\vec{x}) = \alpha + t(\vec{x}) + \beta \cdot t(T(\vec{x})),$$
$$\sigma_0(\vec{x}) = \alpha + s_{M+1}(\vec{x}) + \beta \cdot s_1(T(\vec{x})),$$

[6]As a group automorphism it has $\vec{1} = (1, 1, \ldots, 1)$ as a fixed point.

and for $i = 1, \ldots, M$,

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

Then

$$h(\vec{x}) = \prod_{k=1}^{K} h_k(\vec{x}),$$

with

$$h_k(\vec{x}) = \prod_{i \in I_k} \frac{\sigma_i(\vec{x})}{\varphi_i(\vec{x})},$$

where $I_k = [(k-1) \cdot \ell, k \cdot \ell) \cap [0, M]$. For each $k = 1, \ldots, K$, the prover computes the cumulative products of the values $h_k(\vec{x})$ along the orbit of the time shift T on H', starting with $\phi_k(-\vec{1}) = 1$, and setting

$$\phi_k(T^j(-\vec{1})) = \phi_k(T^{j-1}(-\vec{1})) \cdot h(T^{j-1}(-\vec{1})),$$

for $j = 1, \ldots, |H'| - 1$. At the remaining point $\vec{x} = \vec{1}$ outside H', the prover sets ϕ_k to zero. Correctness of the grand product (28) is proven by the domain identities

$$\phi_k(T(\vec{x})) \cdot \prod_{i \in I_k} \varphi_i(\vec{x}) - \phi_k(\vec{x}) \cdot \prod_{i \in I_k} \sigma_i(\vec{x}) = 0,$$

and the point identities

$$\phi_1(-1) = 1,$$
$$\phi_k(T^{-1}(-\vec{1})) = \phi_{(k \bmod K)+1}(-\vec{1}),$$

where $k = 1, \ldots, K$. These identities 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 random batching scalars $\lambda_1, \ldots, \lambda_K \leftarrow\!\!\$\, F$. The resulting overall sumcheck is

$$\sum_{\vec{x} \in H} Q(L_H(\vec{x}, \vec{y}), L_H(\vec{x}, -\vec{1}), \varphi_0(\vec{x}), \ldots, \varphi_M(\vec{x}), \sigma_0(\vec{x}), \ldots, \sigma_M(\vec{x}), \phi_1(\vec{x}), \ldots, \phi_K(\vec{x}), \phi_1(T(\vec{x})), \ldots \phi_K(T(\vec{x}))) = 0,$$

where

$$\begin{aligned} &Q(L_H, L, L_T, \varphi_0, \ldots, \varphi_M, \sigma_0, \ldots, \sigma_M, \phi_1, \ldots, \phi_K, \phi_{1,T}, \ldots, \phi_{K,T}) = \\ &\qquad L \cdot (\phi - 1) + \sum_{k=1}^{K} \lambda_k \cdot L_H \cdot \left(\phi_{k,T} \cdot \prod_{i \in I_k} \varphi_i - \phi_k \cdot \prod_{i \in I_k} \sigma_i \right) + \mu_k \cdot \left(L_T \cdot \phi_k - L \cdot \phi_{(k \bmod K)+1} \right). \end{aligned} \tag{30}$$

The sumcheck polynomial Q has absolute degree $d = \ell + 2$, which is the same as in Protocol 2, but about the double of variables, $\nu = 2 \cdot (M + 1 + K) + 3$. Its arithmetic complexities are $|Q_{\mathsf{M}}| = K \cdot (2 \cdot \ell + 3) + 2$, $|Q_{\mathsf{A}}| = 4 \cdot K - 1$.

4.2 Computational cost and optimal product size

The prover cost for the Plookup strategy is as follows. Computing the values for all φ_i, σ_i over H consumes overall

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

the quotients $h_k(\vec{x})$ are obtained within $|H| \cdot K \cdot (2 \cdot (\ell - 1) + 4) \cdot \mathsf{M}$, using batch inversion. From these values of the $\phi_k(x)$ over H are derived by another $K \cdot |H| \cdot \mathsf{M}$. The domain evaluation for $L_H(\vec{X}, \vec{y})$ is obtained within $|H| \cdot (\mathsf{M} + \mathsf{A})$ operations, resulting in the overall preparation cost

$$|H| \cdot ((2 \cdot (M+1) + K \cdot (2 \cdot \ell + 3) + 1) \cdot \mathsf{M} + (3 \cdot M + 5) \cdot \mathsf{A}). \tag{31}$$

According to (12) the sumcheck costs

$$|H| \cdot (\ell + 3) \cdot \big((2 \cdot M + 7 + (2 \cdot \ell + 5) \cdot K) \cdot \mathsf{M} + (2 \cdot \ell + 6) \cdot (M + 2 + 3 \cdot K) \cdot \mathsf{A}\big). \tag{32}$$

The total costs of the oracle prover are

- arithmetic costs of

$$|H| \cdot \left(\left(2 \cdot \ell^2 + 13 \cdot \ell + 18\right) \left\lceil \frac{M+1}{\ell} \right\rceil + \ell \cdot (2 \cdot M + 7) + 8 \cdot (M + 3)\right) \cdot \mathsf{M}, \tag{33}$$

  neglecting field additions and substractions, and

- oracle costs of

$$M + K + 1 = M + \left\lceil \frac{M+1}{\ell} \right\rceil + 1 \tag{34}$$

  functions of size $|H|$.

The optimal choice of ℓ

As for Protocol 2, the partial product size ℓ determines the trade-off between algebraic degree and the number of commitments. Based on the multi-scalar multiplication benchmarks from Table 1, and the operation counts (31) and (32) we obtain the following estimates for the optimal choice of ℓ, see Table 3.

Table 3: An estimate of the partial sum size ℓ which minimizes the prover cost. The numbers are based on the operation counts (31) and (32) for the oracle prover and the benchmarks for a multi-scalar multiplication in the Pallas curve, Table 1.

$\log N$	ℓ				field mult. per col.			
	$M = 10$	$M = 20$	$M = 70$	$M = \infty$	$M = 10$	$M = 20$	$M = 70$	$M = \infty$
12	11	11	12	12	$N \cdot 750$	$N \cdot 717$	$N \cdot 687$	$N \cdot 675$
14	11	11	12	11	$N \cdot 637$	$N \cdot 609$	$N \cdot 584$	$N \cdot 572$
16	11	11	9	10	$N \cdot 553$	$N \cdot 528$	$N \cdot 506$	$N \cdot 496$
18	11	11	9	10	$N \cdot 503$	$N \cdot 479$	$N \cdot 458$	$N \cdot 450$

4.3 Bounded multiplicity encoding

The Polygon MidenVM [Mid] uses an improvement of Plookup, which reduces the number of commitments for the sorted union of $f_1, \ldots, f_M$ and t by arranging it as runs of at most $B = 2^b$ occurences, where $b \geq 1$. The length m of each run (reduced by one, as explained below) is then given bitwise,

$$m - 1 = m_0 + m_1 \cdot 2 + \ldots + m_{b-1} \cdot 2^{b-1},$$

and the power of the corresponding term Z in the lookup product is selected by the expression

$$V(Z, m_0, \ldots, m_{b-1}) = \prod_{j=0}^{b-1} (m_j \cdot Z^{2^j} + (1 - m_j) \cdot 1) = Z^{m_0 + m_1 \cdot 2 + \ldots + m_{b-1} \cdot 2^{b-1}}. \tag{35}$$

From this point of view it can be seen as an application of the Arya [BCG$^+$18] multiplicity technique to Plookup. Although [Mid] restricts to range proofs (optimized for the cases of small ranges, the size of which is a fraction of the domain), we quickly outline its generalization to lookups of domain-sized tables.

Let N be the size of the punctuated hypercube H'. Instead of committing to the full-length union $(s_i)_{i=1}^{(M+1)\cdot N}$, the prover provides the compressed sequence

$$(r_j, (m_{j,0}, \ldots, m_{j,b-1}))_{j=1}^{n}$$

of values r_j and their B-bounded run lengths m_j (minus one), represented bitwise by $m_{j,0}, \ldots, m_{j,b-1} \in \{0, 1\}$. Depending on the distribution of the values, the size n of the compressed sequence is bounded by

$$\frac{M+1}{B} \cdot N \leq n \leq \left(\frac{M}{B} + 1\right) \cdot N.$$

(The left bound corrsponds to uniform distribution, and the right bound to the singleton case.) The lookup product over s, written in terms of the compressed sequence is then

$$\prod_{i=1}^{(M+1)\cdot N} (X + s_i + s_{(i \bmod N)+1} \cdot Y) = \prod_{j=1}^{n} V(X + (1+Y) \cdot r_j, \underbrace{m_0, \ldots, m_{b-1}}_{\text{encoding } m_j - 1}) \cdot (X + r_j + Y \cdot r_{(j \bmod n)+1}).$$

As before, the compressed sequence is provided piecewise, splitted into functions over H'. This results in

$$R = \left\lceil \frac{M}{B} + 1 \right\rceil$$

"blocks" of functions, $r_k(\vec{x})$ and $(m_{k,0}(\vec{x}), \ldots, m_{k,b-1}(\vec{x}))$, $k = 1, \ldots, R$, subject to the lookup identity

$$\begin{aligned}
&\prod_{\vec{x} \in H'} V(X + r_K(\vec{x}) \cdot (1+Y), m_{R,0}(\vec{x}), \ldots, m_{R,b-1}(\vec{x})) \cdot (X + r_R(\vec{x}) + r_1(T(\vec{x})) \cdot Y) \\
&\qquad \cdot \prod_{k=1}^{R-1} V(X + r_k(\vec{x}) \cdot (1+Y), m_{k,0}(\vec{x}), \ldots, m_{k,b-1}(\vec{x})) \cdot (X + r_k(\vec{x}) + r_{k+1}(\vec{x}) \cdot Y) \\
&\qquad\qquad = \prod_{\vec{x} \in H'} (X + t(\vec{x}) + t(T(\vec{x})) \cdot Y) \cdot \prod_{i=1}^{M} (X + f_i(\vec{x}) \cdot (1+Y)),
\end{aligned}$$

with V as above, and the boolean domain identities $m_{k,i}(\vec{x}) \cdot (1 - m_{k,i}(\vec{x}))^2 = 0$, over H, for all k, i. Note that the commitment costs for s are reduced by the factor

$$\frac{R \cdot (1 + \log_2 B)}{M+1} \approx \frac{1 + \log_2 B}{B},$$

whereas the absolute degree of the lookup identity is almost invariant for large B, increasing only by the factor

$$\frac{R}{M+1} \cdot (B + \log_2 B) \approx 1 + \frac{\log_2 B}{B}.$$

As the protocol is along the lines of plookup, we omit further details.

5 Comparison

The significant advantage of the logarithmic derivative lookup over Plookup is the lower oracle costs. While the Plookup strategy demands $M+1$ oracles for committing to the sorted union of the M witness column and the table, the logarithmic derivative approach demands only a single oracle for the multiplicities. We compare the two strategies by their computational cost per column, when running them with optimal sum/product size ℓ as given in Table 2 and 3.

Table 4: The estimated performance advantage of logarithmic derivative lookups (Protocol 2) over the multivariate Plookup strategy (Section 4.1), as the ratio of their number of field multiplications per column. The numbers are based on the equivalent number of field multiplications for a multi-scalar multiplication over the Pallas curve, given in Table 1.

$\log \lvert H \rvert$	ratio Plookup / log. derivative					
	$M = 1$	$M = 5$	$M = 10$	$M = 20$	$M = 70$	$M = \infty$
12	1.5	3.1	4.0	4.5	5.2	5.6
14	1.5	3.0	3.8	4.2	4.8	5.1
16	1.5	2.9	3.6	4.0	4.5	4.8
18	1.5	2.9	3.4	3.9	4.3	4.5

Let us give a rough comparison with the bounded multiplicity improvement of Plookup described in Section 4.3. For that we simply count the number of oracles needed for obtaining a *quadratic* overall domain identity (and hence a cubic sumcheck polynomial Q). Under this assumption,

- The logarithmic derivative approach needs overall $M+1$ oracles: one for the multiplicity function, and M helper functions.
- Plookup demands overall $2 \cdot (M+1)$ oracles: $M+1$ for the sorted union of table and witnesses, and $M+1$ for the cumulative products.
- With multiplicity encoding using b bits, Plookup is improved down to

  $$M + 1 + \left\lceil \frac{M}{2^b} + 1 \right\rceil \cdot 3 \cdot b$$

  oracles: Each of the $\left\lceil \frac{M}{2^b} + 1 \right\rceil$ blocks of the compressed representation consists of $b+1$ domain-sized functions (the multiplicity bits and the value), and to linearize each factor in the power selection expression (35), one additionally needs $b-1$ functions for the two-adic powers, and another b functions for the products with the bits. The grand product argument uses $M+1$ cumulative product functions.

In its most extreme setting, $2^b = M$, and therefore only $R = 2$ blocks, the number of oracles for the multiplicity improvement is $M + 1 + 6 \cdot \log_2(M)$. We use this configuration in our comparison, Table 5.

Table 5: A rough comparison of the three lookup protocols in their degree $d = 2$ variant. The table presents the overall number of commitments for a batch of $M = 2^b$ columns. (bPlookup denotes Plookup using multiplicity encoding with b bits.)

	$M = 4$	$M = 8$	$M = 16$	$M = 32$	$M = 64$	$M = 128$
log. derivative	5	9	17	33	65	129
Plookup	10	18	34	66	130	258
bPlookup ($b = \log_2(M)$)	17	27	41	63	101	171

Multiplicity encoding seems to be beneficial only for quite large batches of columns. Below $M = 32$ columns it is expected to perform even worse than Plookup. However, for $M = 64$ and $M = 128$ that picture clearly changes. For these batch sizes it consumes only 78% and 66% of Plookups' number of commitments, respectively, which is still about 55% and 30% more than with the logarithmic derivative approach.

6 Acknowledgements

The author would like to thank Rayan Matovu and Morgan Thomas for giving me the space and time to dwell on batch-column lookups. Special thanks to Marcin Bugaj for helping out with the Pippenger benchmarks. Furthermore, I would like to thank Ariel Gabizon for his feedback and useful discussions.

References

[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.

[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`.

[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.

[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.

[Lab] Avi Dessauer (Orbis Labs). tiny-ram-halo2. https://github.com/Orbis-Tertius/tiny-ram-halo2.

[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.

[Pol] Polygon Zero Team: plonky2. https://github.com/mir-protocol/plonky2.

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

[RDJH22] Gyumin Roh, Wei Dai, Maria Jabbour, and Andrew He. New lookup argument. zkResearch, 2022. https://zkresear.ch/t/new-lookup-argument/32.

[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.

A The flookup proof of radical

Section 5 of [GK22] describes a polynomial IOP for lookups, which is almost identical to the logarithmic derivative approach. We present its generalization to batch-lookups. Let F be an FFT-friendly finite field, having a multiplicative subgroup $H = \{x \in F : x^n = 1\}$ of order n, and denote by g a generator of it. For showing that the ranges of witness function $f_i : H \to F$, $i = 0, \ldots, M-1$, are contained in the range of a table $t : H \to F$, the fractional logaritmic derivative identity is turned into the polynomial identity

$$\sum_{x \in H} \sum_{i=0}^{M-1} \frac{v_T(X)}{X + f_i(x)} = \sum_{x \in H} m(x) \cdot \frac{v_T(X)}{X + t(x)},$$

by multiplying with the precomputed table polynomial $v_T(X) = \prod_{x \in H}(X + t(x))$. Instead of the multiplicity function m, the prover explicitly provides the polynomial[7]

$$R_T(X) = \sum_{x \in H} m(x) \cdot \frac{v_T(X)}{X - t(x)},$$

and engages with the verifier in an IOP for showing that

$$\sum_{x \in H} \sum_{i=0}^{M-1} \frac{v_T(X)}{X + f_i(x)} = R_T(X). \tag{36}$$

The verifier queries $v_T(X)$ and $R_T(X)$ at a random challenge $\alpha \leftarrow\!\!\$\, F$, and both prover and verifier run a sumcheck argument for

$$\sum_{x \in H} \sum_{i=0}^{M-1} \frac{1}{\varphi_i(x)} = \frac{R_T(\alpha)}{v_T(\alpha)},$$

where $\varphi_i(x) = \alpha + f_i(x)$. For this the prover provides the Lagrange representation of the sumcheck polynomial $\phi(X)$ subject to the domain identity

$$\phi(g \cdot x) - \phi(x) + \frac{R_T(\alpha)}{|H| \cdot v_T(\alpha)} = \sum_{i=0}^{M-1} \frac{1}{\varphi_i(x)}$$

for all $x \in H$, or

$$\left(\phi(g \cdot X) - \phi(X) + \frac{R_T(\alpha)}{|H| \cdot v_T(\alpha)}\right) \cdot \prod_{i=0}^{M-1} \varphi_i(X) = \prod_{i=0}^{M-1} \varphi_i(X) \cdot \sum_{i=0}^{M-1} \frac{1}{\varphi_i(x)} \mod v_H(X). \tag{37}$$

The overall identity is of the form

$$Q(\phi_0(X), \ldots, \phi_{M-1}(X), \phi(X), \phi(g \cdot X)) = 0 \mod v_H(X),$$

with Q having $\nu = M + 2$ variables and absolute degree $d = M + 1$.

[7] In a variant of the protocol, communicated by A. Gabizon, the prover provides the Lagrange representation of $R_T(X)$ with respect to the Lagrange basis of the table set $\{t(x) : x \in H\}$. As Lagrange IOPs with respect to several Lagrange bases have different impacts on the Lagrange commitment scheme (e.g., it leads to a separate commitment in a KZG-like scheme, or a separate inner product argument for an IPA-like scheme), we do not compare with this variant.