Flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size

Ariel Gabizon
Zeta Function Technologies

Dmitry Khovratovich
Ethereum Foundation

November 7, 2022

Abstract

We present a protocol for checking the values of a committed polynomial $\phi(X) \in \mathbb{F}_{<m}[X]$ over a multiplicative subgroup $\mathbb{H} \subset \mathbb{F}$ of size m are contained in a table $T \in \mathbb{F}^N$. After an $O(N \log^2 N)$ preprocessing step, the prover algorithm runs in *quasi-linear* time $O(m \log^2 m)$. We improve upon the recent breakthrough results Caulk [ZBK+22] and Caulk+ [PK22], which were the first to achieve the complexity sublinear in the full table size N with prover time being $O(m^2 + m \log N)$ and $O(m^2)$, respectively. We pose further improving this complexity to $O(m \log m)$ as the next important milestone for efficient zk-SNARK lookups.

1 Introduction

The *lookup problem* is fundamental to the efficiency of modern zk-SNARKs. Somewhat informally, it asks for a protocol to prove the values of a committed polynomial $\phi(X) \in \mathbb{F}_{<m}[X]$ are contained in a table T of size N of predefined legal values. When the table T corresponds to an operation without an efficient low-degree arithmetization in $\mathbb{F}$, such a protocol produces significant savings in proof construction time for programs containing the operation. Building on previous work of [BCG+18], plookup [GW20] was the first to explicitly describe a solution to this problem in the polynomial-IOP context. plookup described a protocol with prover complexity quasilinear in both m and N. This left the intriguing question of whether the dependence on N could be made *sublinear* after performing a preprocessing step for the table T. Caulk [ZBK+22] answered this question in the affirmative by leveraging bi-linear pairings, achieving a run time of $O(m^2 + m \log N)$. Caulk+ [PK22] improved this to $O(m^2)$ getting rid of the dependence on table size completely.

However, the quadratic dependence on m of these works makes them impractical for a circuit with many lookup gates. We resolve this issue by giving a protocol called Flookup that is quasi-linear in m and has no dependence on N after the preprocessing step.

1.1 Usefulness of the result

When is it worth it to use Flookup instead of plookup? The plookup prover runs in time $O(N \log N)$ and the Flookup prover requires time $O(m \log^2 m)$ with small constants in the $O()$. Hence, Flookup is worth it roughly when the table is larger than the number of lookups by a logarithmic factor; i.e. when $m << N/\log N$.

We write $<<$ instead of $<$ as Flookup entails other complications that make the tradeoff potentially less attractive. Notably, verification requires a pairing with a prover-defined $\mathbb{G}_2$ point (as do Caulk and Caulk+), which makes rescursive aggregation of proofs less smooth. Another inconvenience is that Flookup doesn't have the nice linearity properties of plookup or Caulk, and so reducing a tuple lookup to a single element lookup (cf. Section 4 of [GW20]), is less efficient. Because of these drawbacks, "simple" tables, like $T = \{0, \ldots, 2^t - 1\}$ for a range check, may not be the best use case for Flookup. As in such a case we can decompose into limbs and use a much smaller table; more generally, this is the case when T is a product set.

A better use case would be complex "SNARK unfriendly" operations on large ranges. For example, those arising inside SHA-256, like mapping a 32-bit input A into a 32-bit output B, via bitwise XOR of three different shifts of A. Given such a mapping f with 32-bit input and output, we can construct the table T containing all 2^{32} values $A + 2^{32} \cdot f(A)$.

To show witness values (w_1, w_2) satisfy $f(w_1) = w_2$, we check the corresponding combination $w_1 + 2^{32} \cdot w_2 \in T$ using Flookup. We additionaly range constrain each w_i to 32 bits. The need for the additional range constraints stems from Flookup not having nice reductions from vector to single-element lookup. Even with them, being able to represent an arbitrary 32-bit operation in one table (and use it inside a circuit of size $<< 2^{32}$), constitutes a significant simplification and potential efficiency boost over current use of lookups in zk-SNARKs.

1.2 Organization of the paper and recommended reading route

- In Section 2 we go over required preliminaries.
- In Section 3 we define the notion of a *bi-linear polynomial IOP* which enables us to model protocols that use pairings in addition to polynomial commitment schemes. *A reader deterred by the formality of this section might skip it on a first read; and simply keep in mind that the term "a bi-linear check" in the subsequent section translates to a pairing in the compiled protocol.*
- In section 4 we review a method of [PK22] to extract a commitment to the vanishing polynomial of a subtable using pairings. We extend it to work with arbitrary sets and not just subgroups.
- In Section 5 we give a lookup protocol given a commitment to the vanishing polynomial of the table.

- In Section 6 we combine the table extraction and subtable lookup protocols to give our final result.

2 Terminology and Conventions

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

Whenever we use the term "efficient", we mean an algorithm running in time $\mathsf{poly}(\lambda)$. Furthermore, we assume an "object generator" $\mathcal{O}$ that is run with input λ before all protocols, and returns all fields and groups used. Specifically, in our protocol $\mathcal{O}(\lambda) = (\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2, g_t)$ where

- $\mathbb{F}$ is a prime field of super-polynomial size $r = \lambda^{\omega(1)}$.
- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ are all groups of size r, and e is an efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$.
- g_1, g_2 are uniformly chosen generators such that $e(g_1, g_2) = g_t$.

We usually let the λ parameter be implicit, i.e. write $\mathbb{F}$ instead of $\mathbb{F}(\lambda)$. We write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively. We use the notations $[x]_1 := x \cdot g_1$ and $[x]_2 := x \cdot g_2$.

We often denote by $[n]$ the integers $\{1, \ldots, n\}$. We use the acronym e.w.p for "except with probability"; i.e. e.w.p γ means with probability *at least* $1 - \gamma$.

universal SRS-based public-coin protocols We describe public-coin (meaning the verifier messages are uniformly chosen) interactive protocols between a prover and verifier; when deriving results for non-interactive protocols, we implicitly assume we can get a proof length equal to the total communication of the prover, using the Fiat-Shamir transform/a random oracle. Using this reduction between interactive and non-interactive protocols, we can refer to the "proof length" of an interactive protocol.

We allow our protocols to have access to a structured reference string (SRS) that can be derived in deterministic $\mathsf{poly}(\lambda)$-time from an "SRS of monomials" of the form $\left\{[x^i]_1\right\}_{a \le i \le b}$, $\left\{[x^i]_2\right\}_{c \le i \le d}$, for uniform $x \in \mathbb{F}$, and some integers a, b, c, d with absolute value bounded by $\mathsf{poly}(\lambda)$. It then follows from Bowe et al. [BGM17] that the required SRS can be derived in a universal and updatable setup requiring only one honest participant; in the sense that an adversary controlling all but one of the participants in the setup does not gain more than a $\mathsf{negl}(\lambda)$ advantage in its probability of producing a proof of any statement.

For notational simplicity, we sometimes use the SRS srs as an implicit parameter in protocols, and do not explicitly write it.

2.1 Analysis in the AGM model

For security analysis we will use the Algebraic Group Model of Fuchsbauer, Kiltz and Loss[FKL18]. In our protocols, by an *algebraic adversary* $\mathcal{A}$ in an SRS-based protocol we mean a $\mathsf{poly}(\lambda)$-time algorithm which satisfies the following.

- For $i \in \{1,2\}$, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}_i$, it also outputs a vector v over $\mathbb{F}$ such that $A =< v, \mathsf{srs_i} >$.

Idealized verifier checks for algebraic adversaries We introduce some terminology to capture the advantage of analysis in the AGM.

First we say our srs *has degree* Q if all elements of $\mathsf{srs_i}$ are of the form $[f(x)]_i$ for $f \in \mathbb{F}_{<Q}[X]$ and uniform $x \in \mathbb{F}$. In the following discussion let us assume we are executing a protocol with a degree Q SRS, and denote by $f_{i,j}$ the corresponding polynomial for the j'th element of $\mathsf{srs_i}$.

Denote by a, b the vectors of $\mathbb{F}$-elements whose encodings in $\mathbb{G}_1, \mathbb{G}_2$ an algebraic adversary $\mathcal{A}$ outputs during a protocol execution; e.g., the j'th $\mathbb{G}_1$ element output by $\mathcal{A}$ is $[a_j]_1$.

By a "real pairing check" we mean a check of the form

$$(a \cdot T_1) \cdot (T_2 \cdot b) = 0$$

for some matrices T_1, T_2 over $\mathbb{F}$. Note that such a check can indeed be done efficiently given the encoded elements and the pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$.

Given such a "real pairing check", and the adversary $\mathcal{A}$ and protocol execution during which the elements were output, define the corresponding "ideal check" as follows. Since $\mathcal{A}$ is algebraic when he outputs $[a_j]_i$ he also outputs a vector v such that, from linearity, $a_j = \sum v_\ell f_{i,\ell}(x) = R_{i,j}(x)$ for $R_{i,j}(X) := \sum v_\ell f_{i,\ell}(X)$. Denote, for $i \in \{1,2\}$ the vector of polynomials $R_i = (R_{i,j})_j$. The corresponding ideal check, checks as a polynomial identity whether

$$(R_1 \cdot T_1) \cdot (T_2 \cdot R_2) \equiv 0$$

The following lemma from [GWC19] is inspired by [FKL18]'s analysis of [Gro16]. It tells us that for soundness analysis against algebraic adversaries it suffices to look at ideal checks. Before stating the lemma we define the Q-DLOG assumption similarly to [FKL18].

Definition 2.1. *Fix integer Q. The* Q-DLOG assumption for $(\mathbb{G}_1, \mathbb{G}_2)$ *states that given*

$$[1]_1, [x]_1, \ldots, \left[x^Q\right]_1, [1]_2, [x]_2, \ldots, \left[x^Q\right]_2$$

for uniformly chosen $x \in \mathbb{F}$, the probability of an efficient $\mathcal{A}$ outputting x is $\mathsf{negl}(\lambda)$.

Lemma 2.2. *Assume the Q-DLOG for $(\mathbb{G}_1, \mathbb{G}_2)$. Given an algebraic adversary $\mathcal{A}$ participating in a protocol with a degree Q SRS, the probability of any real pairing check passing is larger by at most an additive $\mathsf{negl}(\lambda)$ factor than the probability the corresponding ideal check holds.*

Proof. Let γ be the difference between the satisfiability of the real and ideal check. We describe an adversary $\mathcal{A}^*$ for the Q-DLOG problem that succeeds with probability γ; this implies $\gamma = \mathsf{negl}(\lambda)$. $\mathcal{A}^*$ receives the challenge

$$[1]_1, [x]_1, \ldots, \left[x^Q\right]_1, [1]_2, [x]_2, \ldots, \left[x^Q\right]_2$$

and constructs using group operations the correct SRS for the protocol. Now $\mathcal{A}^*$ runs the protocol with $\mathcal{A}$, simulating the verifier role. Note that as $\mathcal{A}^*$ receives from $\mathcal{A}$ the vectors of coefficients v, he can compute the polynomials $\{R_{i,j}\}$ and check if we are in the case that the real check passed but ideal check failed. In case we are in this event, $\mathcal{A}^*$ computes

$$R := (R_1 \cdot T_1)(T_2 \cdot R_2).$$

We have that $R \in \mathbb{F}_{<2Q}[X]$ is a non-zero polynomial for which $R(x) = 0$. Thus $\mathcal{A}^*$ can factor R and find x. □

Knowlege soundness in the Algebraic Group Model We say a protocol $\mathscr{P}$ between a prover $\mathbf{P}$ and verifier $\mathbf{V}$ for a relation $\mathcal{R}$ has *Knowledge Soundness in the Algebraic Group Model* if there exists an efficient E such that the probability of any algebraic adversary $\mathcal{A}$ winning the following game is $\mathsf{negl}(\lambda)$.

1. $\mathcal{A}$ chooses input x and plays the role of $\mathbf{P}$ in $\mathscr{P}$ with input x.
2. E given access to all of $\mathcal{A}$'s messages during the protocol (including the coefficients of the linear combinations) outputs ω.
3. $\mathcal{A}$ wins if
   (a) $\mathbf{V}$ outputs acc at the end of the protocol, and
   (b) $(\mathsf{x}, \omega) \notin \mathcal{R}$.

2.2 KZG-like Polynomial commitment schemes

We define a polynomial commitment scheme where we force the commitment procedure to be consistent with that of [KZG10]. This will be useful in the next section when we define bi-linear polynomial IOPs.

Definition 2.3. *A d-polynomial commitment scheme (d-PCS) over a field $\mathbb{F}$ consists of*

- $\mathsf{gen}(d)$ - *a randomized algorithm that outputs an SRS* srs *that contains as a substring* $[1]_1, [x]_1, \ldots, \left[x^{d-1}\right]_1$ *for uniformly chosen* $x \in \mathbb{F}$ *and no other* $\mathbb{G}_1$ *elements.*
- $\mathsf{com}(f, \mathsf{srs})$ - *that given a polynomial* $f \in \mathbb{F}_{<d}[X]$ *returns the commitment* cm *to* f *defined as* $\mathsf{com}(f) := [f(x)]_1$.
- *A public coin protocol* open *between parties* P_{PC} *and* V_{PC}. P_{PC} *is given* $f_1, \ldots, f_t \in \mathbb{F}_{<d}[X]$. P_{PC} *and* V_{PC} *are both given integer* $t = \mathsf{poly}(\lambda)$, $\mathsf{cm}_1, \ldots, \mathsf{cm}_t$ - *the alleged commitments to* $f_1, \ldots, f_t$, $z_1, \ldots, z_t \in \mathbb{F}$ *and* $s_1, \ldots, s_t \in \mathbb{F}$ - *the alleged correct openings* $f_1(z_1), \ldots, f_t(z_t)$. *At the end of the protocol* V_{PC} *outputs* acc *or* rej.

such that

- ***Completeness:*** *Fix integer* t, $z_1, \ldots, z_t \in \mathbb{F}$, $f_1, \ldots, f_t \in \mathbb{F}_{<d}[X]$. *Suppose that for each* $i \in [t]$, $\mathsf{cm}_i = \mathsf{com}(f_i, \mathsf{srs})$. *Then if* open *is run correctly with values* $t, \{\mathsf{cm}_i, z_i, s_i = f_i(z_i)\}_{i\in[t]}$, V_{PC} *outputs* acc *with probability one.*
- ***Binding Knowledge soundness in the algebraic group model:*** *For any algebraic adversary* $\mathcal{A}$ *the probability of* $\mathcal{A}$ *winning the following game is* $\mathsf{negl}(\lambda)$ *over the randomness of* $\mathcal{A}$ *and* gen.
  1. *Given* srs, $\mathcal{A}$ *outputs* $t, \mathsf{cm}_1, \ldots, \mathsf{cm}_t$.
  2. *Note that as* $\mathcal{A}$ *is algebraic, in the step above it also outputs polynomials* $f_1, \ldots, f_t \in \mathbb{F}_{<d}[X]$ *such that* $\mathsf{cm}_i = [f_i(x)]_1$.
  3. $\mathcal{A}$ *outputs* $z_1, \ldots, z_t \in \mathbb{F}$, $s_1, \ldots, s_t \in \mathbb{F}$.
  4. $\mathcal{A}$ *takes the part of* P_{PC} *in the protocol* open *with common inputs* $\mathsf{cm}_1, \ldots, \mathsf{cm}_t, z_1, \ldots, z_t, s_1, \ldots, s_t$
  5. $\mathcal{A}$ *wins if*
     - V_{PC} *outputs* acc *at the end of the protocol.*
     - *For some* $i \in [t]$, $s_i \neq f_i(z_i)$.

2.3 Other notational conventions

Given a polynomial $f \in \mathbb{F}[X]$ and a subset $I \subset \mathbb{F}$ we define $f|_I$ to be the set $\{f(v)\}_{v\in I}$. Given a set $T \subset \mathbb{F}$ we denote by $Z_T(X) \in \mathbb{F}[X]$ the *vanishing polynomial* of T:

$$Z_T(X) := \prod_{i\in T}(X - i).$$

3 Bi-linear polynomial IOPs

While most recent works on zk-SNARKs have leveraged the power of polynomial commitment schemes [KZG10], [ZBK+22] has additionaly leveraged the power of pairings to essentially take products of commitments. This enables checking degree two identites between polynomials without needing to compute the polyomials themselves, but only their commitments - which can be much faster when they are a small linear combination of preprocessed polynomials. We formalize a framework to capture protocols using pairings in addition to polynomial openings.

Definition 3.1. *Fix positive integer* d *and field* $\mathbb{F}$. *A* d-bi-linear polynomial IOP over $\mathbb{F}$ *(*d-BLIOP*) is a multiround protocol between a prover* $\mathrm{P}_{\mathsf{poly}}$, *verifier* $\mathrm{V}_{\mathsf{poly}}$ *and trusted party* $\mathcal{I}$ *that proceeds as follows.*

1. *The protocol definition includes two sets of* preprocessed polynomials $P_1, P_2 \subset \mathbb{F}_{<d}[X]$.

2. *The messages of* $\mathrm{P}_{\mathsf{poly}}$ *are sent to* $\mathcal{I}$ *and are of the form* (f, i) *for* $f \in \mathbb{F}_{<d}[X]$ *and* $i \in \{1, 2\}$. *If* $\mathrm{P}_{\mathsf{poly}}$ *sends a message not of this form, the protocol is aborted.* $\mathrm{P}_{\mathsf{poly}}$ *may also send other messages directly to* $\mathrm{V}_{\mathsf{poly}}$.

3. *The messages of* $\mathrm{V}_{\mathsf{poly}}$ *to* $\mathrm{P}_{\mathsf{poly}}$ *are always random coins.*

4. *At the end of the protocol,*

   - *For* $i \in \{1, 2\}$, *let* F_i *denote the set of polynomials* f *that were sent from* $\mathrm{P}_{\mathsf{poly}}$ *to* $\mathcal{I}$ *as part of a message* (f, i). *And denote* $A_i := F_i \cup P_i$.
   - $\mathrm{V}_{\mathsf{poly}}$ *may ask* $\mathcal{I}$
     (a) evaluation queries *of the form* (f, x) *for* $f \in A_1$ *and* $x \in \mathbb{F}$. $\mathcal{I}$ *responds with the value* $f(x)$.
     (b) bi-linear identity queries *of the form* $\sum_{j \in [k]} c_j f_j(X) h_j(X) \stackrel{?}{\equiv} 0$, *where* k *is some positive integer,* $c_j \in \mathbb{F}, f_j \in A_1, h_j \in A_2$ *for each* $j \in [k]$. $\mathcal{I}$ *responds with* true *or* false *according to whether the identity holds.*
   - *After concluding her queries* $\mathrm{V}_{\mathsf{poly}}$ *outputs* acc *or* rej *by a deterministic procedure depending only on the query results.*

We define bi-linear polynomial iops for relations and languages in the natural way.

Definition 3.2. *Given a relation* $\mathcal{R}$, *a* d-BLIOP for $\mathcal{R}$ *is a* d-BLIOP *with the following additional properties.*

1. *At the beginning of the protocol,* $\mathrm{P}_{\mathsf{poly}}$ *and* $\mathrm{V}_{\mathsf{poly}}$ *are both given - in addition to the preprocessed polynomial sets* P_1, P_2 *- an input* x. *The description of* $\mathrm{P}_{\mathsf{poly}}$ *assumes possession of* ω *such that* $(\mathsf{x}, \omega) \in \mathcal{R}$.

2. ***Completeness:*** *If* $\mathrm{P}_{\mathsf{poly}}$ *follows the protocol correctly using a witness* ω *for* x, $\mathrm{V}_{\mathsf{poly}}$ *accepts with probability one.*

3. ***Knowledge Soundness:*** *There exists an efficient* E, *that given access to the messages of* $\mathrm{P}_{\mathsf{poly}}$ *to* $\mathcal{I}$, *and the random coins of* $\mathrm{V}_{\mathsf{poly}}$ *outputs* ω *such that, for any strategy of* $\mathrm{P}_{\mathsf{poly}}$, *the probability of the following event is* $\mathsf{negl}(\lambda)$.
   (a) $\mathrm{V}_{\mathsf{poly}}$ *outputs* acc *at the end of the protocol, and*
   (b) $(\mathsf{x}, \omega) \notin \mathcal{R}$.

Definition 3.3. *Given a language* $\mathcal{L}$, *a* d-BLIOP for $\mathcal{L}$ *is a* d-BLIOP *with the following additional properties.*

1. *At the beginning of the protocol,* $\mathrm{P}_{\mathsf{poly}}$ *and* $\mathrm{V}_{\mathsf{poly}}$ *are both given an input* x.

2. ***Completeness:*** *If* $\mathsf{x} \in \mathcal{L}$, *and* $\mathrm{P}_{\mathsf{poly}}$ *follows the protocol correctly using* x, $\mathrm{V}_{\mathsf{poly}}$ *accepts with probability one.*

3. ***Soundness:*** *If* $\mathsf{x} \notin \mathcal{L}$ *then any strategy of* $\mathrm{P}_{\mathsf{poly}}$ *will result in* $\mathrm{V}_{\mathsf{poly}}$ *rejecting e.w.p* $\mathsf{negl}(\lambda)$.

3.1 From bi-linear polynomial IOPs to protocols against algebraic adversaries

We wish to "compile" BLIOPs to protocols against algebraic adversaries. We will define a few terms to enable us to track the compilation efficency in terms of the resultant prover and verifier efficiency.

Note that given a polynomial protocol $\mathscr{P}$ and fixed input (x, ω) for the protocol. We have some distribution over the sets of polynomials A_1, A_2 sent during the protocol.

Thus, we can define , $D_1(\mathscr{P}, \mathsf{x}, \omega) := \sum_{f \in F_1} (\deg(f) + 1)$. And $D_2(\mathscr{P}, \mathsf{x}, \omega)$ as the number of $\mathbb{G}_2$ scalar multiplications required to compute $[f]_2$ for all $f \in F_2$. Also, there will be some distribution over the set of evaluation queries (f, z) asked during the protocol.

Define $\mathcal{O}$ to be the set of tuples $([f]_1, z, f(z); f)$ when iterating over all evaluation queries asked by $\mathrm{V}_{\mathsf{poly}}$.

Finally, define $E(\mathscr{P}, \mathsf{x}, \omega)$ to be the total number of summands in all bi-linear queries asked by $\mathrm{V}_{\mathsf{poly}}$ during protocol execution.

Lemma 3.4. *Assume the d-DLOG assumption holds for $(\mathbb{G}_1, \mathbb{G}_2)$. Given a* d-BLIOP *$\mathscr{P}$ over $\mathbb{F}$ and a d-PCS $\mathscr{S}$ we can construct a protocol $\mathscr{P}^*$ for $\mathcal{R}$ with knowledge soundness against algebraic adversaries such that*

1. *Preprocessing time: For $i = 1, 2$, $C_i(\mathscr{P})$ $\mathbb{G}_i$ scalar multiplications, where $C_i(\mathscr{P})$ is the number of $\mathbb{G}_i$ scalar multiplications required to compute $[f]_i$ for all $f \in P_i$.*

2. *Prover efficiency: The prover $\mathbf{P}$ in $\mathscr{P}^*$ consists of running $\mathrm{P}_{\mathsf{poly}}$ on the same inputs; $D_i(\mathscr{P}, \mathsf{x}, \omega)$ $\mathbb{G}_i$ scalar multiplications for $i \in \{1, 2\}$ and running the prover of $\mathscr{S}$ with input $\mathcal{O}$.*

3. *Verifier efficiency: The verifier $\mathbf{V}$ in $\mathscr{P}^*$ consists running $\mathrm{V}_{\mathsf{poly}}$ on the same inputs; $E(\mathscr{P}, \mathsf{x}, \omega)$ pairings and $\mathbb{G}_t$ exponentiations, and running the verifer of $\mathscr{S}$ with input $\mathcal{O}$.*

4. *Proof size: For $i \in \{1, 2\}$, Let $B_i(\mathscr{P})$ be the number of messaged (f, i) sent during a protocol execution by an honest $\mathrm{P}_{\mathsf{poly}}$; and assume this number doesn't depend on (x, ω). The final proof consists of $B_i(\mathscr{P})$ $\mathbb{G}_i$-elements, and a proof of $\mathscr{S}$ with input $\mathcal{O}$.*

Proof. Let $\mathscr{S} = (\mathsf{gen}, \mathsf{com}, \mathsf{open})$. Let P_1, P_2 be the sets of preprocessd polynomials in the definition of $\mathscr{P}$. The SRS of $\mathscr{P}^*$ consists of

- $\mathsf{srs} = [1]_1, \ldots, \left[x^{d-1}\right]_1, [1]_2, \ldots, \left[x^{d-1}\right]_2$,
- $\{[f(x)]_1\}_{f \in P_1}, \{[h(x)]_2\}_{h \in P_2}$

Given $\mathscr{P}$ we describe $\mathscr{P}^*$. $\mathbf{P}$ and $\mathbf{V}$ behave identically to $\mathrm{P}_{\mathsf{poly}}$ and $\mathrm{V}_{\mathsf{poly}}$, except in the following two cases.

- Whenever $\mathrm{P}_{\mathsf{poly}}$ sends a message (f, i), for $f \in \mathbb{F}_{<d}[X]$ and $i \in \{1, 2\}$ to $\mathcal{I}$ in $\mathscr{P}$; $\mathbf{P}$ instead sends $[f]_i$ to $\mathbf{V}$.

- Instead of making evaluation queries to $\mathcal{I}$, $\mathbf{V}$ does the following.
  1. For each evaluation query (f, z) made by $\mathrm{V}_{\mathsf{poly}}$ to $\mathcal{I}$, $\mathbf{V}$ instead sends the query directly to $\mathbf{P}$ which responds with the alleged value $s = f(z)$. Let $\mathcal{O}$ be the set of tuples $([f]_1, z, s; f)$ obtained by all evaluation queries.
  2. $\mathbf{P}$ and $\mathbf{V}$ engage in the open protocol with input $\mathcal{O}$
  3. If $\mathbf{V}$ outputs rej in this execution of open, it also outputs rej in $\mathscr{P}^*$.
- When $\mathrm{V}_{\mathsf{poly}}$ makes a bi-linear query $\sum_{i\in[k]} c_i f_i(X) h_i(X) \stackrel{?}{\equiv} 0$, $\mathbf{V}$ instead checks the pairing equation
$$\prod_{i\in[k]} e([f_i(x)]_1, [h_i(x)]_2)^{c_i} = 1$$
and proceeds as if the query reply was true if and only if the pairing equation held.
- Finally, $\mathbf{V}$ outputs acc or rej according to whether $\mathrm{V}_{\mathsf{poly}}$ did given the query replies it has obtained.

To prove the claim about knowledge soundness in the AGM we must describe the extractor E for the protocol $\mathscr{P}^*$. For this purpose, let $E_{\mathscr{P}}$ be the extractor of the protocol $\mathscr{P}$ as guaranteed to exist from Definition 3.2, and $E_{\mathscr{S}}$ be the extractor for the Knowledge Soundness game of $\mathscr{S}$ as in Definition 2.3.

Now assume an algebraic adversary $\mathcal{A}$ is taking the role of $\mathbf{P}$ in $\mathscr{P}^*$.

1. When $\mathcal{A}$ sends a message $[f]_i$ to $\mathbf{V}$ then E receives the coefficients of f from $\mathcal{A}$ and adds f to a set A_i.
2. At the end of the protocol E sends A_1, A_2 and the random coins of $\mathbf{V}$ to $E_{\mathscr{P}}$, and receives ω in return.
3. E returns ω.

Note that we can think of A_1, A_2 as random variables of the randomness of $\mathbf{V}, \mathcal{A}$ and gen.

Now let us define three events (also over the randomness of $\mathbf{V}$,$\mathcal{A}$ and gen):

1. We let A be the event that for some $(\mathsf{cm}, z, s; f) \in \mathcal{O}$ $f(z) \neq s$, and at the same time V_{PC} has output acc when open was run by $\mathbf{P}$ and $\mathbf{V}$. By the KS of $\mathscr{S}$, $\Pr(A) = \mathsf{negl}(\lambda)$.
2. Let B be the event that for one of the bi-linear queries, we had $\sum c_i f_i(X) h_i(X) \not\equiv 0$; but $\prod_{i\in[k]} e([f_i(x)]_1, [h_i(x)]_2)^{c_i} = 1$. The latter is equivalent to $\sum c_i f_i(x) h_i(x) = 0$, where x is the "secret" in srs. We show the probability of this event is $\mathsf{negl}(\lambda)$: Note that in the above event we have that x is a root of $P(X) := \sum c_i f_i(X) h_i(X)$. Thus, we can define an algorithm $\mathscr{A}$ for finding x given srs that runs $\mathscr{P}^*$ between $\mathcal{A}$ and $\mathbf{V}$, and for each bi-linear query of $\mathbf{V}$ attempts to factor the corresponding P, and checks for each of its roots x' if it's equal to x. The success probability of $\mathscr{A}$ is at least the probability of the event C, and thus must be $\mathsf{negl}(\lambda)$ as otherwise we would contradict the d-DLOG assumption for $(\mathbb{G}_1, \mathbb{G}_2)$.

3. We think of an adversary $\mathcal{A}_{\mathscr{P}}$ participating in $\mathscr{P}$, where $\mathrm{V}_{\mathsf{poly}}$ is using the same randomness as $\mathbf{V}$, and using the polynomials A_1, A_2 as their messages to $\mathcal{I}$. We define C to be the event that $\mathrm{V}_{\mathsf{poly}}$ outputs acc but $(\mathsf{x}, \omega) \notin \mathcal{R}$. By the KS of $\mathscr{P}$, $\Pr(C) = \mathsf{negl}(\lambda)$.

Now look at the event D that $\mathbf{V}$ outputs acc, but E failed in the sense that $(\mathsf{x}, \omega) \notin \mathcal{R}$. This is the event we need to show has probability $\mathsf{negl}(\lambda)$. We claim that $D \subset A \cup B \cup C$. If A, B didn't happen, it means that $\mathbf{V}$ and $\mathrm{V}_{\mathsf{poly}}$ have received exactly the same answers to their queries, and thus will have the same output. In particular, if we are outside of the event $A \cup B$, $\mathbf{V}$ will output acc only when $\mathrm{V}_{\mathsf{poly}}$ does. In other words, $D \backslash (A \cup B) \subset C$. □

Remark 3.5. *The above also implies a similar transformation for protocols for languages rather than relations: Given a language $\mathcal{L}$ we can define a relation $\mathcal{R} = \{(\mathsf{x}, \omega) | \mathsf{x} \in \mathcal{L}\}$. A sound protocol for $\mathcal{L}$ will be knowledge sound for $\mathcal{R}$ (e.g. by defining an extractor that always outputs $\omega = 0$), and vice versa.*

3.2 Conventions for describing BLIOPs and PIOPs

1. When a d-BLIOP doesn't include any bi-linear checks, and accordingly A_2 is empty, we call it a *d-polynomial IOP* or *d*-PIOP. In this case we abbreviate "$\mathrm{P}_{\mathsf{poly}}$ sends $(f, 1)$" to "$\mathrm{P}_{\mathsf{poly}}$ sends f".

2. When we say $\mathrm{V}_{\mathsf{poly}}$ "checks the identity $P(f_1(X), \ldots, f_k(X))$", for $f_i \in A_1$, we mean that $\mathrm{V}_{\mathsf{poly}}$ chooses a random $\alpha \in \mathbb{F}$, queries $f_1(\alpha), \ldots, f_k(\alpha)$, computes the value $z = P(f_1(\alpha), \ldots, f_k(\alpha))$ and outputs rej if $z \neq 0$. Note that when analyzing soundness or knowledge-soundness of a d-BLIOP, we can assume the event that $g(X) := P(f_1(X), \ldots, f_k(X))$ is not the zero-polynomial but $g(x) = 0$ didn't happen as it has $\mathsf{negl}(\lambda)$ probability.

3. When we say $\mathrm{V}_{\mathsf{poly}}$ "checks the identity $P(f_1(X), ..., f_k(X))$ on H", for $f_i \in A_1$ and a set $H \subset \mathbb{F}$, we mean that $\mathrm{P}_{\mathsf{poly}}$ sends the quotient $T(X) := P(f_1(X), ..., f_k(X))/Z_H(X)$ and that $\mathrm{V}_{\mathsf{poly}}$ checks the identity $P(f_1(X), \ldots, f_k(X)) = Z_H(X)T(X)$.

4. When describing the efficiency of specific PIOPs and BLIOPs in the rest of the paper, we implicitly use the compilation lemma above, and actually describe the efficiency of the resultant protocl against algebraic adversaries. For example, when we state a BLIOP "requires t $\mathbb{G}_1$-scalar multiplications on input x", we are implicitly claiming $D_1(\mathscr{P}, \mathsf{x}) = t$ and therefore this is the number of scalar multiplications in the resultant protocol against algebraic adversaries.

4 Protocol for subtable extraction

The protocol in the following section is similar to one implicit in Caulk+ [PK22]. Based on the innovation of Caulk, Caulk+ uses fractional decomposition to efficiently "extract"

a vanishing polynomial Z_I of a subset $I \subset T$ from Z_T. In [PK22], the large set T is always a multiplicative subgroup. This is fine for their protocol, as there T represents *indices* of table values, rather than *the table values themselves*.

Our main innovation in this section is an algorithm that computes all subtable commitments of size $|T|-1$ efficiently - this insures that also when T is an arbitrary set, our preprocessing remains quasilinear rather than quadratic. This allows us later to work with vanishing polynomials representing the actual table values.

Lemma 4.1. *Given $T \subset \mathbb{F}$ of size N and $\left\{\left[x^i\right]_2\right\}_{i\in\{0,\ldots,N-1\}}$ there is an algorithm using $O(N\log^2 N)$ $\mathbb{G}_2$-scalar multiplications and $\mathbb{F}$-operations for computing the set of elements*

$$\mathcal{T} = \left\{\left[Z_{T\setminus\{i\}}(x)\right]_2\right\}_{i\in T}$$

Proof. Denote by $\overline{Z_{T\setminus\{i\}}}$ the vector $[a_0\ a_1\ \cdots a_{N-1}]$ (N columns, 1 row) of coefficients of the polynomial $Z_{T\setminus\{i\}}$. Then consider a matrix whose rows are coefficients of $Z_{T\setminus\{i\}}(X)$:

$$Z_{T\setminus *} = \begin{bmatrix} \overline{Z_{T\setminus\{0\}}} \\ \overline{Z_{T\setminus\{1\}}} \\ \cdots \\ \overline{Z_{T\setminus\{N-1\}}} \end{bmatrix}$$

Thus we have to compute $\mathcal{T} = Z_{T\setminus *} \times SRS$ where

$$SRS = \left[[1]_2\,,\ [x]_2\,,\ \cdots,\ \left[x^{N-2}\right]_2\,,\ \left[x^{N-1}\right]_2\right]^T$$

Before we describe an algorithm to compute $\mathcal{T}$, we first introduce the ideas behind it:

1. For polynomials $a(X), b(X)$ of degree $N/2$ and $c(X)$ such that $c(X) = a(X)b(X)$ with coefficient vectors $\overline{c}, \overline{a}, \overline{b}$ it holds that

$$\begin{aligned} &\begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_{N-2} & c_{N-1} & c_N \end{bmatrix} = \\ &= \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{N/2-1} & a_{N/2} \end{bmatrix} \cdot \begin{bmatrix} b_0 & b_1 & b_2 & b_3 & \cdots & b_{N/2} & \cdots & 0 \\ 0 & b_0 & b_1 & b_2 & \cdots & b_{N/2-1} & \cdots & 0 \\ 0 & 0 & b_0 & b_1 & \cdots & b_{N/2-2} & \cdots & 0 \\ & & \ddots & & & & & \\ 0 & 0 & 0 & 0 & \cdots & b_2 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & b_1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & b_0 & \cdots & b_{N/2} \end{bmatrix} \end{aligned}$$

   or in matrix form

$$\overline{c} = \overline{a} \times A_b,$$

   where A_b is a matrix with $N/2+1$ rows and $N+1$ columns.

2. Let us split T into $T_1 = \{v_1, v_2, \ldots, v_{N/2}\}$ and $T_2 = \{v_{N/2+1}, \ldots, v_N\}$. Then we have

$$Z_{T\backslash *} = \begin{bmatrix} \overline{Z_{T_1\backslash\{0\}} \cdot Z_{T_2}} \\ \overline{Z_{T_1\backslash\{1\}} \cdot Z_{T_2}} \\ \cdots \\ \overline{Z_{T_2\backslash\{n-1\}} \cdot Z_{T_1}} \end{bmatrix} = \begin{bmatrix} \overline{Z_{T_1\backslash\{0\}}} \times A_{Z_{T_2}} \\ \overline{Z_{T_1\backslash\{1\}}} \times A_{Z_{T_2}} \\ \cdots \\ \overline{Z_{T_2\backslash\{N-1\}}} \times A_{Z_{T_1}} \end{bmatrix} = \begin{bmatrix} Z_{T_1\backslash *} \times A_{Z_{T_2}} \\ Z_{T_2\backslash *} \times A_{Z_{T_1}} \end{bmatrix}$$

Therefore

$$Z_{T\backslash *} \times SRS = \begin{bmatrix} Z_{T_1\backslash *} \times A_{Z_{T_2}} \times SRS \\ Z_{T_2\backslash *} \times A_{Z_{T_1}} \times SRS \end{bmatrix}$$

Algorithm to compute $\mathcal{T} = Z_{T\backslash *} \times SRS$

1. Compute coefficients of $Z_T(X)$ and its tree of subproducts in $O(N \log^2 N)$ time (see below).
2. Split T into halves T_1 and T_2. Retrieve $Z_{T_2}(X)$ and $Z_{T_1}(X)$.
3. Compute vectors $a_2 = A_{Z_{T_2}} \times SRS$ and $a_1 = A_{Z_{T_1}} \times SRS$ as Toeplitz matrix-vector multiplication in $O(N \log N)$ time [GVL13].
4. Apply algorithm recursively (go to step 2) to compute $b_1 = Z_{T_1\backslash *} \times a_2$ and $b_2 = Z_{T_2\backslash *} \times a_1$.
5. Output concatenation of b_1 and b_2.

We have an equation for the complexity $C_1(N)$ of the algorithm:

$$C_1(N) = 2C_1(N/2) + O(N \log N)$$

which gives $C_1(N) = O(N \log^2 N)$.

Algorithm to compute Z_T and subproducts: This algorithm is a direct adaptation of [vzGG] (Alg. 10.3).

1. Split T to T_1 and T_2.
2. Compute Z_{T_1} and Z_{T_2} by two recursive calls to this algorithm with $T = T_1$ and $T = T_2$.
3. Multiply Z_{T_1} by Z_{T_2} in $O(N \log N)$ time using FFT.

We have

$$C_2(N) = 2C_2(N/2) + O(N \log N)$$

which gives $C_2(N) = O(N \log^2 N)$. This ends the proof. □

We proceed to describe the subtable extraction protocol.

IsVanishingSubtable$_T(g(X))$

Preprocessed polynomials: Let $P_1 = \{Z_T\}$. For each $i \in T$ insert into P_2 the polynomial $Z_{T\setminus\{i\}}$.

Inputs: $g(X) \in \mathbb{F}_{<d}[X]$.

Protocol:

- $\mathrm{P}_{\mathsf{poly}}$ sends $(Z_{T\setminus S}, 2)$ to $\mathcal{I}$.
- $\mathrm{V}_{\mathsf{poly}}$ makes the bi-linear query $g \cdot Z_{T\setminus S} \stackrel{?}{\equiv} Z_T$ and outputs acc iff it returns true.

Lemma 4.2. $\mathsf{IsVanishingSubtable}_{\mathsf{T}}$ *is a d-BLIOP for the language* $\mathcal{L} := \{g(X) \in \mathbb{F}_{<d}[X] | g(X) = Z_S(X) \text{ for some } S \subseteq T\}$. *On input* $g = Z_S$, *the prover complexity is* $O(m \log^2 m)$ $\mathbb{F}$*-operations and* $O(m)$ $\mathbb{G}_1$ *and* $\mathbb{G}_2$*-scalar multiplications, where* $m = |S|$. *Denoting* $|T| = N$, *preprocessing takes* $O(N \log^2 N)$ $\mathbb{G}_2$*-scalar multiplications and* $\mathbb{F}$*-operations.*

Proof. Correctness and soundness are obvious: The check in Step 4 passes if and only if g divides Z_T which happens if and only if $g = Z_S$ for some $S \subseteq T$.

We turn to analyzing efficiency. Denote $m = |S|$. The coefficients of polynomial Z_S can be computed in time $O(m \log^2 m)$ (Lemma 4.1). Then, as noted in [TAB+20, vzGG], it holds that

$$Z_{T\setminus S}(X) = \sum_{i \in S} c_i Z_{T\setminus\{i\}}(X)$$

where coefficients c_i are computed via the derivative $Z'_S(X)$ as $c_i = \frac{1}{Z'_S(i)}$ in $O(m \log^2 m)$ time. Thus we can compute $\left[Z_{T\setminus S}(x)\right]_2$ with m $\mathbb{G}_2$ scalar multiplications from the elements of P_2.

We move to analyze the cost of preprocessing. We must compute given $\left\{\left[x^j\right]_2\right\}_{j\in\{0,\ldots,N-1\}}$ the set

$$\mathcal{T} = \left\{\left[Z_{T\setminus\{i\}}(x)\right]_2\right\}_{i\in T}.$$

The complexity of this is $O(N \log^2 N)$ $\mathbb{G}_2$-scalar multiplications and $\mathbb{F}$-operations according to Lemma 4.1. □

5 A PIOP for lookups when given the table in vanishing form

The previous section gives us a way to extract the vanishing polynomial of the subtable we are interested in. We could use a grand product argument to convert the subtable into evaluation/Lagrange form, and then use a lookup protocol like plookup that expects to have the table in this form. Instead, we give a protocol that works directly with the vanishing form of the table. In fact, it is considerably more efficient than plookup in group operations: It requires roughly $3m$ $\mathbb{G}_1$-scalar multiplications, when both witness and table are of size m, as opposed to plookup requiring roughly $5m$ (Lemma 3.2 in [GW20]).

Unnormalized rational Lagrange functions Central to our analysis is the idea of defining "rational Lagrange functions" also for points outside of the relevant set. Roughly speaking, this allows us to check inclusion in the set by checking if we ended up with a polynomial or rational function. Details follow.

Fix a set $T \subset \mathbb{F}$. For $v \in \mathbb{F}$, we denote by Γ_v^T the rational function

$$\Gamma_v^T(X) := \frac{Z_T(X)}{X - v}$$

Note that Γ_v^T is a polynomial exactly when $v \in T$.

The following lemma allows us to reduce lookups to distinguishing between polynomials and rational functions.

Lemma 5.1. *Fix any vectors $v, a \in \mathbb{F}^m$, and any subset $T \subset \mathbb{F}$. Define the rational function $R(X) := \sum_{j\in[m]} a_j \Gamma_{v_j}^T(X)$.*

1. *If for all $j \in [m]$, $v_j \in T$; then $R(X) \in \mathbb{F}[X]$.*
2. *Let $S \subset [m]$ be the set of $j \in [m]$ such that $v_j \notin T$. Assume that $S \neq \emptyset$. Then if $\sum_{j\in S} a_j \neq 0$, $R(X) \notin \mathbb{F}[X]$. In particular, assuming $|\mathrm{char}(\mathbb{F})| > m$, $R(X) \notin \mathbb{F}[X]$ when taking $a_j = 1$ for all $j \in [m]$.*

Proof. The first item in the lemma is obvious - a sum of polynomials is a polynomial. We prove the second. Let S be as in the lemma statement and assume it is non-empty. Our task is essentially to show the rational functions do not "cancel out" and create a polynomial. Let $a \in \mathbb{F}^m$ be such that $\sum_{j\in S} a_j \neq 0$. We can write

$$R(X) = R_1(X) + R_2(X)$$

where $R_1(X) = \sum_{j\in[m]\setminus S} \Gamma_{v_j}^T(X)$ and $R_2(X) := \sum_{j\in S} a_j \Gamma_{v_j}^T(X)$. Since R_1 is a polynomial, R is a polynomial if and only if R_2 is. If $R_2(X) \in \mathbb{F}[X]$, then we have the polynomial identity

$$Z_T(X) \sum_{j\in S} \frac{a_j}{X - v_j} = R_2(X).$$

Multiplying denominators, we get

$$Z_T(X)Q'(X) = R_2(X)Q(X)$$

where $Q'(X) := \sum_{j\in S} a_j \prod_{i\in S\setminus\{j\}}(X - v_i)$ and $Q(X) := \prod_{j\in S}(X - v_j)$. We first rule out the possibility $R_2(X) \equiv 0$. If this was the case, we would have $Q'(X) \equiv 0$. However, the coefficient of $X^{|S|-1}$ in Q' is $\sum_{j\in S} a_j$ which we are assuming is non-zero.

So assume now that $R_2(X) \not\equiv 0$. Since none of the factors of Q divide Z_T, we must have $Q|Q'$. However, we have $\deg(Q) = |S|$ and $\deg(Q') < |S|$; so Q doesn't divide Q' . Therefore, $R_2 \notin \mathbb{F}[X]$.

In summary, $R \notin \mathbb{F}[X]$ in this case, and the second item in the lemma holds.

□

The above lemma suggests the following protocol. Let $\mathbb{H} = \{\mathbf{g}, \mathbf{g}^2, \ldots, \mathbf{g}^m = 1\} \subset \mathbb{F}$ be a multiplicative subgroup of size m with generator $\mathbf{g}$. Given a polynomial $\phi(X) \in \mathbb{F}_{<d}[X]$, a set $T \subset \mathbb{F}$ and $\mathbb{H}$, define $R_{T,\phi}(X) := \sum_{v \in \mathbb{H}} \Gamma^T_{\phi(v)}(X)$ ($\mathbb{H}$ is an implicit parameter in this definition). We commit to $R_{T,\phi}(X)$ and prove the commitment is correct. This will show $R_{T,\phi}$ is a polynomial and therefore $\phi|_{\mathbb{H}} \subset T$ according to the lemma. To show the commitment is indeed to $R_{T,\phi}(X)$, we open it at random $\beta \in \mathbb{F}$, and compare the value to an independent evaluation of $R_{T,\phi}(\beta)$. To compute this independent evaluation of $R_{T,\phi}(\beta)$ we use a "grand sum argument" suggested by Justin Drake [Dra] similar to [GWC19]'s grand product argument.

Below we denote by $\{L_i(X)\}_{i \in [m]}$ the Lagrange basis of $\mathbb{H}$. That is, $L_i(X) \in \mathbb{F}_{<m}[X]$, $L_i(\mathbf{g}^i) = 1$ and $L_i(\mathbf{g}^j)$ for $i \neq j \in [m]$. Given subsets $\mathbb{H}, T \subset \mathbb{F}$ we define the following protocol.

$\underline{\mathsf{IsInVanishing}_{\mathbb{H},\mathsf{T}}(\phi)}$

Preprocessed polynomials: Let $P_1 = \{Z_T\}$

Inputs: A polynomial $\phi \in \mathbb{F}_{<d}[X]$

Protocol:

1. $\mathrm{P}_{\mathsf{poly}}$ sends the polynomial $g(X) := R_{T,\phi}(X)$
2. $\mathrm{V}_{\mathsf{poly}}$ chooses and sends random $\beta \in \mathbb{F}$ and queries the value $z := g(\beta)$.
3. $\mathrm{P}_{\mathsf{poly}}$ computes and sends a polynomial $Z(X) \in \mathbb{F}_{<m}[X]$ defined as follows
   - For each $i \in [m]$, $Z(\omega^i) = \sum_{j=1}^{i} \Gamma^T_{\phi(\mathbf{g}^i)}(\beta)$
4. $\mathrm{V}_{\mathsf{poly}}$ queries the value $Z_T(\beta)$.
5. $\mathrm{V}_{\mathsf{poly}}$ checks on $\mathbb{H}$ the identities
   (a) $L_1(X)(Z(X)(\beta - \phi(X)) - Z_T(\beta)) = 0$.
   (b) $(X - \mathbf{g})\left(Z(X) - Z(X/\mathbf{g})\frac{Z_T(\beta)}{\beta - \phi(X)}\right) = 0$.
   (c) $L_m(X)(Z(X) - z) = 0$.

Theorem 5.2. $\mathsf{IsInVanishing}_{\mathbb{H},\mathsf{T}}$ *is a d-PIOP for the language* $\mathcal{L} := \{\phi(X) \in \mathbb{F}_{<d}[X] | \phi|_{\mathbb{H}} \subset T\}$. *When* $\deg(\phi), |T| = O(m)$, *the prover runs in time* $O(m \log^2 m)$.

Additionally, after a preprocessing phase depending on T *consisting of* $O(m \log^2 m)$ $\mathbb{G}_1$*-scalar multiplications; the prover only requires* $O(m \log m)$ $\mathbb{F}$*-operations and* $O(m)$ $\mathbb{G}_1$*-scalar multiplications.*

Proof. Assume that $\phi \notin \mathcal{L}$. We show that $\mathrm{V}_{\mathsf{poly}}$ accepts with $\mathsf{negl}(\lambda)$ probability. Let E be the event that $g(\beta) = R_{T,\phi}(\beta)$. When $\phi(X) \notin \mathcal{L}$, Lemma 5.1 implies that $R_{T,\phi(X)} \notin \mathbb{F}[X]$. As $g \in \mathbb{F}_{<d}[X]$, E has probability $\mathsf{negl}(\lambda)$.

Note that the checks in step 5 passing imply that $R_{T,\phi}(\beta) = g(\beta)$:

- The first check implies $Z(\mathbf{g}) = \Gamma^T_{\phi(\mathbf{g})}(\beta)$.
- The second check implies for $i \in \{2, \ldots, m\}$, $Z(\mathbf{g}^i) = Z(\mathbf{g}^{i-1}) + \Gamma^T_{\phi(\mathbf{g}^i)}(\beta)$. Thus, the two first checks together imply $Z(\mathbf{g}^m) = \sum_{i\in[m]} \Gamma^T_{\phi(\mathbf{g}^i)}(\beta) = R_{T,\phi}(\beta)$.
- The third check implies $Z(\mathbf{g}^m) = z = g(\beta)$. Hence together with the first two checks, we have $R_{T,\phi}(\beta) = g(\beta)$.

Thus, $\mathrm{V}_{\mathsf{poly}}$ outputs acc only during a $\mathsf{negl}(\lambda)$ probability event.

Turning to analyze the $\mathrm{P}_{\mathsf{poly}}$'s runtime, the heaviest component is computing the coefficients of $R_{T,\phi}$. We show that we can derive $R_{T,\phi}$'s values on T in time $O(m \log^2 m)$. From there, we can interpolate the coefficients in time $O(m \log^2 m)$. To do so, $\mathrm{P}_{\mathsf{poly}}$ computes in $O(m \log m)$ time[1] the values $\{a_v\}_{v\in T}$ where a_v is defined to be the number of $x \in \mathbb{H}$ with $\phi(x) = v$. We have that

$$R_{T,\phi}(X) = \sum_{v\in T} a_v \Gamma^T_v(X).$$

Let $\{\tau_v(X)\}_{v\in T}$ be the Lagrange base of T. We have for $v \in T$, that $\Gamma^T_v(X) = c_v\tau_v(X)$ for the constant $c_v = \prod_{v\neq i\in T}(v - i)$. Thus, we have that

$$R_{T,\phi}(X) = \sum_{v\in T} a_v c_v \tau_v(X).$$

Equivalently, $R_{T,\phi}(v) = a_v c_v$ for each $v \in T$. Thus, once we obtain the values $\{a_v c_v\}_{v\in T}$, we can interpolate the coefficients of $R_{T,\phi}(X)$ in $O(m \log^2 m)$ time.

For this purpose, similarly to the proof of Lemma 4.2, we note that the constants $\{c_v\}_{v\in T}$ are precisely the evaluations of the derivative $Z'_T(X)$ of $Z_T(X)$ at T. Thus they can be computed in the required time bound.

To prove the "additionally" part of the lemma, note that given T, we can precompute the elements $S_v := \left[\Gamma^T_v(x)\right]_1$ for all $v \in T$ using the algorithm of Lemma 4.1 in $O(m \log^2 m)$ operations. Given those values we can compute the KZG-commitment $[R_{T,\phi}(x)]_1$ as $\sum_{v\in T} a_v \cdot S_v$, in $O(m)$ $\mathbb{G}_1$-scalar multiplications and $O(m \log m)$ $\mathbb{F}$-operations. Similarly, we can compute the KZG opening proof of $R_{T,\phi}$ at $r \in \mathbb{F} \setminus T$ to value $z := R_{T,\phi}(r)$ as

$$\left[\frac{R_{T,\phi}(x) - z}{x - r}\right]_1 = \sum_{v\in T} \frac{a_v - z}{v - r} \cdot S_v.$$

□

[1] In applications, this will typically be $O(m)$ time as ϕ is often given in evaluation form over $\mathbb{H}$.

6 Putting it all together

$\underline{\mathsf{IsInVanishingTable}_{\mathbb{H},\mathsf{T}}(\phi)}$

Preprocessed polynomials: $P_1 = \{Z_T\}$ where

Input: $\phi \in \mathbb{F}_{<d}[X]$

Protocol:

1. $\mathrm{P}_{\mathsf{poly}}$ computes the set $I \subseteq T$ such that $I = \phi|_{\mathbb{H}}$.
2. $\mathrm{P}_{\mathsf{poly}}$ computes and sends Z_I.
3. $\mathrm{P}_{\mathsf{poly}}$ and $\mathrm{V}_{\mathsf{poly}}$ run $\mathsf{IsVanishingSubtable}_{\mathsf{T}}(Z_I)$
4. $\mathrm{P}_{\mathsf{poly}}$ and $\mathrm{V}_{\mathsf{poly}}$ run $\mathsf{IsInVanishing}_{\mathbb{H},\mathsf{Z_I}}(\phi)$.

Combining Lemma 4.2 and Theorem 5.2 we have

Theorem 6.1. *Let $N = |T|$. $\mathsf{IsInVanishingTable}_{\mathbb{H},\mathsf{T}}$ is a* d-BLIOP *for the language $\{\phi \in \mathbb{F}_{<d}[X] | \phi|_{\mathbb{H}} \subset T\}$ such that*

- *$O(N \log^2 N)$ $\mathbb{G}_2$-scalar multiplications and $\mathbb{F}$-operations are required in preprocessing.*
- *The prover requires $O(m \log^2 m)$ $\mathbb{F}$-operations and $O(m)$ $\mathbb{G}_1$ and $\mathbb{G}_2$-scalar multiplications.*

Proof. The only thing left to address given previous sections is that the computation of Z_I in the second step can be done in time $O(m \log^2 m)$. This, again, follows from algorithm 10.3 in [vzGG]. □

Acknowledgements

The first author thanks Aztec Network for support of this work. We thank Mary Maller and Arantxa Zapico for helpful discussions. The construction in Section 5 is inspired by a construction of Carla Ràfols and Arantxa Zapico for a similar problem.

References

[BCG+18] J. Bootle, A. Cerulli, J. Groth, S. K. Jakobsen, and M. Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018.

[BGM17] S. Bowe, A. Gabizon, and I. Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. `https://eprint.iacr.org/2017/1050`.

[Dra] J. Drake. https://youtu.be/tbnaud5wgxm?t=2251.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 33–62, 2018.

[Gro16] J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[GVL13] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[GW20] A. Gabizon and Z. J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, page 315, 2020.

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

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. pages 177–194, 2010.

[PK22] J. Posen and A. A. Kattis. Caulk+: Table-independent lookup arguments. 2022.

[TAB+20] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2020.

[vzGG] J. von zur Gathen and J. Gerhard. Fast polynomial evaluation and interpolation. *Modern Computer Algebra, chapter 10*, pages 295–310.

[ZBK+22] A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin. Caulk: Lookup arguments in sublinear time. *IACR Cryptol. ePrint Arch.*, page 621, 2022.