TASSLE: LASSO FOR THE COMMITMENT-PHOBIC

TESSERACT DORE

Abstract. We present TaSSLE, a new lookup argument for decomposable tables with minimal commitment costs. The construction generalizes techniques introduced in Lasso [9] which take advantage of the internal structure present in such tables to avoid the need for any party to need to commit to, or even construct, the entire table. This allows the use of lookups against very large tables, with applications including new design strategies for "zero-knowledge virtual machines". We show that these techniques may be combined in a generic way with any existing lookup argument to achieve similar results. We then give a construction of TaSSLE by applying this observation to a recent lookup argument, introduced in [7], which combines logarithmic derivatives with the GKR protocol to achieve a lookup argument with minimal commitment costs.

1. Introduction

Lasso[9] is a recent lookup argument with the attractive feature that, for *structured* lookup tables, avoids the need to commit to the entire lookup table at any stage (pre-processed or otherwise). However, it has the drawback of requiring the prover to commit to several vectors of metadata, whose length is the same as the vector of looked-up values. While these metadata vectors have entries of small bit-length, lowering commitment costs when using commitments requiring operations in a prime-order group whose discrete logarithm is hard, this benefit matters less when using commitments based on hash functions or error-correcting codes.

In this note, we present a variant of Lasso, called TaSSLE (Tensors and Sumcheck for Structured Lookup Efficiency), that avoids the need to commit to these metadata vectors at all: indeed, the set of committed vectors is a strict subset of those committed to in Lasso. We achieve this by replacing the product argument in Lasso with a more flexible logarithmic derivative argument. As in Lasso, we use a version of the GKR protocol[7] to avoid needing to commit to any intermediate values used in the computation of this logarithmic derivative. The costs for running this protocol are much the same as in Lasso's product argument, so we expect the resulting lookup protocol to be strictly faster than Lasso with similarly sized proofs.

Lita Foundation
E-mail address: dorebell@gmail.com.
Date: July 2, 2024.

The lookup tables with the structure needed to apply TaSSLE, as in Lasso, are called *decomposable*. Loosely, these are tables T of size N which can be "built up from" smaller tables $T_1, \ldots, T_\alpha$ each of size $O(N^{1/c})$. Jolt[1] shows how a number of lookup tables arising in the context of a zero-knowledge virtual machine (zk-VM) can be realized as decomposable tables.

The rest of this note is structured as follows: in Section 3, we present an abstracted version of the "tensor-product argument" used in Lasso. This argument demonstrates how to reduce lookup into a "decomposable" table T into α separate lookup instances into its sub-tables T_j: we show that this works with *any* lookup argument for the sub-tables and *any* IOP for polynomial constraints.

Then, in Section 5, we show how to instantiate this general argument using sumcheck, logarithmic derivatives, and GKR to obtain TaSSLE. As another application of the general result in 3, we show how to obtain a lookup argument in the *univariate* context which still has the property that the prover does not need to construct or commit to the table T or any vectors of length N.

1.1. **Acknowledgements.** This work would not be possible without the support of the whole team at the Lita Foundation, especially Hadas Zeilberger, Ventali Tan, and Morgan Thomas. I'd also like to thank Psi Vesely for their helpful comments and ongoing mentorship. Finally, this work, as with much of my life, is dedicated to Extra for diligently keeping me company through the long hours of writing and coding.

2. Preliminaries

2.1. Lookups and decomposable tables.

Problem 2.1 (The lookup relation)**.** Let $v = (v_1, \ldots, v_k)$ be k vectors of length L and $T = (t_1, \ldots, t_k)$ k vectors (the "lookup table") of length $N = 2^n$. The (vector-valued) *lookup relation* $\mathsf{Lookup}_{k,N,L}$ consists of all such pairs (T, v) such that for every $j \in [L]$, there is some $n \in [N]$ such that $(v_1[j], v_2[j], \ldots, v_k[j]) = (t_1[n], t_2[n], \ldots, t_k[n])$. Write $\mathsf{Lookup}_{N,L}$ for the case $k = 1$, i.e. the case of scalar-valued lookups.

Our argument provides a solution to this problem in the special case that T is *decomposable*:

Definition 2.2. Let $T = (t_1, \ldots, t_k)$ be a lookup table with k columns t_i consisting of $N = 2^n$ scalar entries. Fix some $c, \alpha > 0$, and set $m = n/c, M = 2^m = N^{1/c}$. Let $T_1, \ldots, T_\alpha$ be tables consisting of M entries each. Let $g = (g_1, \ldots, g_k)$: $\mathbf{F}^\alpha \to \mathbf{F}^k$ be a polynomial function, and let d be the maximum total degree of the polynomials g_i.

We say that T is *decomposable* with *roots* $T_1, \ldots, T_\alpha$ if the entries of $T, T_1, \ldots, T_\alpha$ may be ordered in such a way that, for some $1 \leq k_1 \leq k_2 \leq \cdots \leq k_c = \alpha$:

$$T[r] = g(T_1[r_1], \ldots, T_{k_1}[r_1], T_{k_1+1}[r_2], \ldots, T_{k_2}[r_2], \ldots, T_{k_c}[r_c]) \tag{1}$$

for all $r \in [N]$. Here, $r = r_1 + 2^m r_2 + \ldots + 2^{(c-1)m} r_c$ for $r_i \in \{0, \ldots, 2^m - 1\}$, i.e. the r_i's are the result of splitting the binary decomposition of r into c groups of m digits. Note that both sides of (1) are vectors of length k.

Note 2.3. This definition of a decomposable table is a slight generalization of the definition given in [9]: there, the definition requires the number of roots T_i depending on a given r_j in (1) to be the same for all j. Equivalently, this requirement states that there is some k such that $k_i = i * k$ for all i. This restriction appears to be for convenience only, and the construction in [9] readily extends to this case.

In addition, we allow the table T to be vector-valued, where [9] only discusses the case of scalar-valued lookup tables. We include this straightforward generalization only because the extension to vector-valued tables is *not* the usual reduction from vector-valued to scalar-valued lookups.

An example of a decomposable table, justifying the 'T' in TaSSLE, is a *tensor product* table $T = \otimes_{i=1}^{c} T_i$: here, the polynomial g is just the product $X_1 \cdots X_c$. Another example is a *range check table*, with $T[r] = r$ and $T_i[r] = r_i$. Then $g(T_1, \ldots, T_c) = T_1 + 2^m T_2 + \ldots + 2^{(c-1)m} T_c$.

2.2. **Polynomials.** We recall some algebra facts and constructions which we will use throughout this note. A core fact from algebra, upon which many cryptographic constructions rest, is the following:

Lemma 2.4 (Schwartz-Zippel Lemma)**.** *Let $f \in \mathbf{F}[X_1, \ldots, X_k]$ be a* non-zero *polynomial of total degree d in k variables over a finite field $\mathbf{F}$. Then for any set $S \subseteq \mathbf{F}$, the probability that $f(s) = 0$ for s drawn uniformly at random from S is*

$$\epsilon_{\text{SZ}} \leq \frac{dk}{|S|}.$$

This follows by a straightforward induction from the well-known univariate case, i.e. the statement that a univariate polynomial of degree d has at most d zeroes.

Definition 2.5. A polynomial $f(X_1, \ldots, X_k) \in \mathbf{F}[X_1, \ldots, X_k]$ is *multilinear* if it is of the form

$$f(X_1, \ldots, X_k) = \sum_{\alpha \in \{0,1\}^k} c_\alpha \prod_{i=1}^{k} X_i^{\alpha_i} \tag{2}$$

for some coefficients $c_\alpha \in \mathbf{F}$. In other words, it must have degree at most one in each variable separately. We write $\mathbf{M}_k$ for the set of multilinear polynomials in k variables.

As is immediate from (2), the set $\mathbf{M}_k$ of multilinear polynomials in k variables is a vector space over $\mathbf{F}$ of dimension 2^k. They are determined by their values on the *boolean hypercube* $\mathbf{B}_k = \{0, 1\}^k$, whose size is 2^k.

Throughout, we identify $\mathbf{B}_k$ with $[2^k]$, the set of integers from 0 to $2^k - 1$, in the natural way: $j \in [2^k]$ corresponds to $(j_1, \ldots, j_k) \in \mathbf{B}_k$ if and only if

$$j = j_1 + 2j_2 + \ldots + 2^{k-1} j_k. \tag{3}$$

Reflecting this, an alternative basis for $\mathbf{M}_k$ is:

Definition 2.6 (The Multilinear Lagrange Kernel)**.** The *multilinear Lagrange kernel* consists of the basis elements

$$L_b(X_1, \ldots, X_k) = \prod_{j=1}^{k} \left(b_i X_i + (1 - b_i)(1 - X_i)\right), \quad b = (b_1, \ldots, b_k) \in \mathbf{B}_k \tag{4}$$

The crucial property of the multilinear Lagrange kernel is that if $b, b' \in \mathbf{B}_k$:

$$L_b(b') = \begin{cases} 1 & \text{if } b = b' \\ 0 & \text{otherwise} \end{cases}$$

Thus, the multilinear Lagrange kernel defines an isomorphism $L_{\mathbf{F}} \colon \mathbf{F}^{\mathbf{B}_k} \xrightarrow{\sim} \mathbf{M}_k$, where $\mathbf{F}^{\mathbf{B}_k}$ is the set of $\mathbf{F}$-valued functions on $\mathbf{B}_k$.

Explicitly, this isomorphism is:

$$L_{\mathbf{F}} : \mathbf{F}^{\mathbf{B}_k} \xrightarrow{\sim} \mathbf{M}_k : \qquad \mathbf{F}^{\mathbf{B}_k} \ni f \mapsto \widetilde{f} \coloneqq \sum_{b \in \mathbf{B}_k} f(b) L_b. \tag{5}$$

We call $\widetilde{f} \in \mathbf{M}_k$ the *multilinear extension* of the function f.

2.3. **Sumcheck.** The *sumcheck protocol* is an IOP for the following relation:

Definition 2.7. Let $v = (v_1, \ldots, v_k)$ be k vectors of length $L = 2^\ell$, $\sigma \in \mathbf{F}$, and g a polynomial in k variables of maximum total degree d. We say that $(\sigma, g; v)$ is in the *sum relation* $\mathsf{Sum}_{d,\ell,k}$ if

$$\sum_{j \in [L]} g(v_1[j], \ldots, v_k[j]) = \sigma. \tag{6}$$

Now, we give the sumcheck protocol, introduced in [6]:

Protocol 2.8 (Sumcheck)**.** *There is an IOP to prove the relation*

$$(\sigma, g; v_1, \ldots, v_k) \in \mathsf{Sum}_{d,\ell,k}$$

whose communication costs are:

- *ℓ prover messages of length d,*
- *one prover message of length k,*
- *ℓ verifier challenges $r_i \in \mathbf{F}$,*
- *k polynomial oracles for $v_1, \ldots, v_k$, and*
- *one multilinear evaluation query to each of these k oracles.*

The complexity for the prover and verifier is:

- *The prover performs $O(d(k + b)L)$ field operations, where b is the number of monomials in g, and*

• *the verifier performs $O(d\ell)$ field operations plus one evaluation of g, and the soundness error of the protocol is*

$$\epsilon_{\text{sumcheck}} \leq \frac{d\ell}{|\mathbf{F}|}$$

Proof. Identifying $[L]$ with $\mathbf{B}_\ell$ using (3), we define multilinear polynomial oracles $\widetilde{v}_i$ by applying (5) to the vectors v_i.

- First, the prover sends multilinear polynomial oracles for $\widetilde{v}_1, \ldots, \widetilde{v}_k$ to the verifier.

Define a polynomial function $G \in \mathbf{F}[X_1, \ldots, X_\ell]$ by:

$$G(X_1, \ldots, X_\ell) = g(\widetilde{v}_1(X_1, \ldots, X_\ell), \ldots, \widetilde{v}_k(X_1, \ldots, X_\ell)) \tag{7}$$

Note that the degree of G is at most d in each variable separately. Rewriting the sum in (6) as a sum over $\mathbf{B}_\ell$, our claim is that:

$$\sum_{b \in \mathbf{B}_\ell} G(b) = \sigma. \tag{8}$$

Define a *univariate* polynomial $g^{(1)} \in \mathbf{F}[X_1]$ by

$$g^{(1)}(X_1) = \sum_{b \in \mathbf{B}_{\ell-1}} G((X_1, b)). \tag{9}$$

Supposing that (9) is true, the claim (8) is equivalent to the requirement

$$g^{(1)}(0) + g^{(1)}(1) = \sigma^{(0)} \coloneqq \sigma \tag{10}$$

- The prover sends the verifier the d scalars

$$m^{(1)} = \{g^{(1)}(0), g^{(1)}(2), \ldots, g^{(1)}(d)\}. \tag{11}$$

- The verifier responds with a challenge $r_1 \in \mathbf{F}$, drawn uniformly at random.

Note that the d scalars in (11) along with the requirement (10) uniquely determine the degree-d polynomial $g^{(1)}$. The verifier need not check (10), since it is true *a fortiori*.

Now, the goal is to prove that $g^{(1)}$ satisfies (9), where both sides are degree-d polynomials in X_1. By the univariate Schwartz-Zippel Lemma 2.4, except with probability

$$\epsilon^{(1)} \leq \frac{d}{|\mathbf{F}|},$$

if $g^{(1)}$ satisfies (9) after evaluating both sides at the random $r^{(1)} \in \mathbf{F}$, then $g^{(1)}$ satisfies (9) in $\mathbf{F}[X_1]$. This is

$$g^{(1)}(r^{(1)}) = \sum_{b \in \mathbf{B}_{\ell-1}} G((r^{(1)}, b)). \tag{12}$$

- The verifier computes $\sigma^{(1)} = g^{(1)}(r^{(1)})$ by interpolating the values (11), applying (10) to obtain $g^{(1)}(1)$.

- The prover computes functions $v_1^{(1)}, \ldots, v_k^{(1)} \in \mathbf{F}^{\mathbf{B}_{\ell-1}}$ by

$$v_i^{(1)}(b_2, \ldots, b_\ell) = \widetilde{v}_i(r^{(1)}, b_2, \ldots, b_\ell).$$

We have now reduced the original claim $(\sigma, g; v_1, \ldots, v_k) \in \mathsf{Sum}_{d,\ell,k}$ to the claim $(\sigma^{(1)}, g; v_1^{(1)}, \ldots, v_k^{(1)}) \in \mathsf{Sum}_{d,\ell-1,1}$. We may now iterate this process ℓ times to obtain the desired result. At the end, we are left with:

- a random point $r = (r^{(1)}, \ldots, r^{(\ell)}) \in \mathbf{F}^\ell$,
- ℓ prover messages $m^{(i)}$ of length d,
- a total soundness error of

$$\epsilon_{\text{sumcheck}} \leq \epsilon^{(1)} + \cdots + \epsilon^{(\ell)} \leq \frac{d\ell}{|\mathbf{F}|},$$

- the final claim

$$\sigma^{(\ell)} = G(r) = g(\widetilde{v}_1(r), \ldots, \widetilde{v}_k(r)) \tag{13}$$

Finally, we prove (13) with one multilinear evaluation query to each of $\widetilde{v}_1, \ldots, \widetilde{v}_k$.

In each round, the verifier needs to perform $O(d)$ field operations to compute the interpolation $g^{(i)}(r^{(i)})$ from $m^{(i)}$ and $\sigma^{(i-1)}$, and at the end the verifier needs to evaluate the degree-d polynomial $g(X_1, \ldots, X_k)$ given $X_1, \ldots, X_k$. Thus, the verifier sends ℓ messages in total and the verifier's work consists of $O(d\ell)$ field operations plus the cost of evaluating g.

In each round, the prover must compute the functions

$$v_j^{(i)}(b) = v_j^{(i-1)}((r^{(i)}, b)) \tag{14}$$

for $b \in \mathbf{B}_{\ell-i}$ as well as the sums

$$g^{(i)}(j) = \sum_{b \in \mathbf{B}_{\ell-i}} G((r_1, \ldots, r_{i-1}, j, b)) \tag{15}$$

$$= \sum_{b \in \mathbf{B}_{\ell-i}} g\left(\widetilde{v}_1^{(i-1)}((j, b)), \ldots, \widetilde{v}_k^{(i-1)}((j, b))\right) \tag{16}$$

for $j \in \{0, \ldots, d\}$. Since the $v_j^{(i-1)}(X, b)$ are linear in X, the prover may compute the $d + 2$ values

$$v_j^{(i-1)}((x, b)), \qquad x \in \{r^{(i)}, 0, 1, \ldots, d\}$$

by linear interpolation from $v_j^{(i-1)}(0, b), v_j^{(i-1)}(1, b)$, which have been computed in the prior step. Thus, the prover may compute all of these values in $O(2^{\ell-i} * (d + 2) * k) = O(d * k * 2^{\ell-i})$ field operations. Additionally, the prover must compute (16) from these values, which amounts to evaluating the degree-d polynomial g at $2^{\ell-i}$ points. This amounts to $O(d * b * 2^{\ell-i})$ field operations. Thus, the total prover work in the ith round is $O(d(b + k) * 2^{\ell-i})$. Summing these up, the total prover work is:

$$O(d(b + k) * (1 + 2 + \ldots + 2^{\ell-1})) = O(d(b + k)L).$$

□

2.4. **Interactive Oracle Proofs.** We describe TaSSLE as an interactive oracle proof (IOP), built from smaller IOPs as sub-protocols.

Definition 2.9 (IOP)**.** An *interactive oracle proof* (IOP) is an interactive argument between a prover and a verifier, in which the prover is allowed to send the verifier *polynomial oracles*.

- Such an oracle uniquely represents a polynomial f over some finite field $\mathbf{F}$, and the verifier may later query the oracle at any point x in the domain of f to obtain $f(x)$.
- We say that the IOP is *univariate* if the oracles f represent polynomials in a single variable.
- Likewise, the IOP is *multilinear* if the oracles f represent *multilinear polynomials*, polynomials in multiple variables with degree at most one in each variable.
- At the end of the protocol, the verifier *accepts* or *rejects*.
- We say that the IOP *proves* a relation $\mathcal{R} \subseteq \{0,1\}^*$ if it constitutes an *argument of knowledge* for $\mathcal{R}$.

2.5. **Polynomial evaluation IOPs.** In addition to the Lookup relation, we will need the following relation:

Definition 2.10. Let $w_1, \ldots, w_k$ be vectors of length L and $g = (g_1, \ldots, g_\kappa)$ a collection of polynomials in k variables of maximum total degree d. We say that $(g; w_1, \ldots, w_k)$ is in the *polynomial constraint relation* $\mathsf{PolyCon}_{k,\kappa,d,L}$ if for each $i \in [L], j \in [\kappa]$, $g_j(w_1[i], \ldots, w_k[i]) = 0$.

The PolyCon relation is the "core" of the relation consisting of witnesses for the AIR constraint system (as well as a major component of the relation for the Plonkish constraint system). As such, the IOPs used to create SNARKs for those constraint systems readily specialize to IOPs for PolyCon. One major example of such is the following, used as a component in e.g. [2] and [8]:

Protocol 2.11 (Sumcheck IOP for PolyCon)**.** *We obtain an IOP to prove* $(g; w_1, \ldots, w_k) \in \mathsf{PolyCon}_{k,\kappa,d,L}$ *for* $L = 2^\ell$ *whose costs are:*

- k *multilinear polynomial oracles for* $w_1, \ldots, w_k$*, each in* ℓ *variables,*
- *verifier challenges* $\alpha \in \mathbf{F}, \tau \in \mathbf{F}^\ell$*,*
- *one evaluation query to each of these* k *oracles at the same point* $\tau \in \mathbf{F}^\ell$*,*
- *the costs of running the sumcheck protocol on an instance of* $\mathsf{Sum}_{d+1,\ell,k}$*:*
  - $O(d\ell + k)$ *total prover communication,*
  - $O(d(\ell + b))$ *total verifier work, where* b *is the total number of monomials among* $g_1, \ldots, g_\kappa$*,*
  - ℓ *verifier challenges* $r_i \in \mathbf{F}$*,*
  - $O(d(k + b)L)$ *prover work,*

– *a soundness error* $\epsilon_{\text{sumcheck}} \leq \frac{(d+1)\ell}{|\mathbf{F}|} = O\left(\frac{(d+\kappa)\ell}{|\mathbf{F}|}\right)$.

The total soundness error is

$$\epsilon_{\text{polycon}} \leq \frac{(2d+\kappa+1)\ell}{|\mathbf{F}|}.$$

Proof. Identify $[L]$ with $\mathbf{B}_\ell$ as in (3), and define multilinear polynomial oracles $\widetilde{w}_i$ by applying (5) to the vectors w_i. Define a polynomial G_i in ℓ variables with degree at most d in each variable by:

$$G_i(X_1, \ldots, X_\ell) = g_i(\widetilde{w}_1(X_1, \ldots, X_\ell), \ldots, \widetilde{w}_k(X_1, \ldots, X_\ell)), \tag{17}$$

and define $\widetilde{g_i} \in \mathbf{M}_\ell$ as the multilinear extension of G_i, i.e.

$$\widetilde{g_i}(X_1, \ldots, X_\ell) = \sum_{b \in \mathbf{B}_\ell} G_i(b) L_b(X_1, \ldots, X_\ell). \tag{18}$$

We may rephrase the claim to be proven, $(g; w_1, \ldots, w_k) \in \mathsf{PolyCon}_{k,d,L}$, as:

$$G_i(b) = 0 \text{ for all } b \in \mathbf{B}_\ell, \tag{19}$$

for all $i \in [\kappa]$. By (18), this is equivalent to the requirement that $\widetilde{g_i} = 0$ in $\mathbf{M}_\ell$.

First, we batch the κ constraints $g_1, \ldots, g_\kappa$:

- The verifier sends random challenges $\alpha \in \mathbf{F}, \tau \in \mathbf{F}^\ell$.

Define a combined constraint polynomial G by:

$$G(X_0, X_1, \ldots, X_\ell) = \sum_{i=1}^{\kappa} X_0^i * \widetilde{g_i}(X_1, \ldots, X_\ell), \tag{20}$$

and a combined multilinear extension $\widetilde{g} \in \mathbf{M}_\ell$ by:

$$\widetilde{g}(X_1, \ldots, X_\ell) = \sum_{i=1}^{\kappa} \alpha^i * \widetilde{g_i}(X_1, \ldots, X_\ell). \tag{21}$$

By the Schwartz-Zippel Lemma 2.4, if each $\widetilde{g_i}$ is not uniformly 0, then the probability that $G(\alpha, \tau) = \widetilde{g}(\tau) = 0$ is at most $\frac{(\kappa+d)\ell}{|\mathbf{F}|}$. (Note that by the definition (17) of the G_i in terms of multilinear extensions, the condition that G_i is uniformly 0 on $\mathbf{B}_\ell$ is the same as the condition that $G_i = 0$ in $\mathbf{F}[X_1, \ldots, X_\ell]$.)

Now, we will use Protocol 2.8 (sumcheck), to prove $\widetilde{g}(\tau) = 0$. Since $\widetilde{g}$ is multilinear, applying (5) tells us

$$\begin{aligned}\widetilde{g}(\tau) &= \sum_{b \in \mathbf{B}_\ell} \widetilde{g}(b) L_b(\tau) \\ &= \sum_{b \in \mathbf{B}_\ell} \sum_{j=0}^{\kappa-1} \alpha^j G_j(b) L_b(\tau) \\ &= \sum_{b \in \mathbf{B}_\ell} \sum_{j=0}^{\kappa-1} \alpha^j G_j(v_1[b], \ldots, v_k[b]) L_b(\tau). \qquad (22)\end{aligned}$$

We define a polynomial $h \in \mathbf{F}[X_1, \ldots, X_k]$ of total degree at most $d+1$ in each variable by:

$$h(X_0, X_1, \ldots, X_k) = \sum_{j=0}^{\kappa-1} \alpha^j g_j(X_1, \ldots, X_k) * X_0. \qquad (23)$$

Letting $v_0 \in \mathbf{F}^{\mathbf{B}_\ell}$ be the function $b \mapsto L_b(\tau)$, we may rewrite (22) as:

$$0 = \sum_{b \in \mathbf{B}_\ell} h(v_0[b], v_1[b], \ldots, v_k[b]), \qquad (24)$$

which is an instance of $\mathsf{Sum}_{d+1,\ell,k}$ with $\sigma = 0$.

Now, we run the sumcheck protocol to reduce this claim to a single multilinear evaluation query to each of the k oracles at the random point $r \in \mathbf{F}^\ell$ sampled in the course of the sumcheck protocol.

The total soundness error follows from a union bound among the soundness error of the sumcheck protocol and the soundness error of the application of the Schwartz-Zippel Lemma above:

$$\epsilon_{\text{polycon}} \leq \epsilon_{\text{sumcheck}} + \frac{(\kappa + d)\ell}{|\mathbf{F}|} = \frac{(2d + \kappa + 1)\ell}{|\mathbf{F}|}.$$

□

3. Abstracting Lasso's tensor-product argument

Protocol 3.1. *Let T be a decomposable table of length $N = 2^n$ and width κ with roots $T_1, \ldots, T_\alpha$ of length $M = 2^m = N^{1/c}$. Let d be the maximum degree of the polynomials g_i as in Definition 2.2. Suppose we are given:*

- *an IOP for the relation $\mathsf{PolyCon}_{\alpha+1,d,L}$, and*
- *an IOP for the relations $\mathsf{Lookup}_{k_i - k_{i-1},M,L}$ for $i = 1, \ldots, c$.*

From these, we obtain an IOP for the relation $\mathsf{Lookup}_{\kappa,N,L}$. The costs are the same as:

- *those of applying the lookup IOP to c instances, $\mathsf{Lookup}_{k_i - k_{i-1},M,L}$ for $i = 1, \ldots, c$,*
- *and one instance of the IOP for $\mathsf{PolyCon}_{\alpha+1,d,L}$,*
- *with the addition of α polynomial oracles of size L.*

The soundness error is at most the sum of the soundness errors of the subprotocols.

Proof. Define vectors v_i, $i = 1, \ldots, \alpha$ of length L as follows: if $v[j] = T[r]$, then

$$v_i[j] \coloneqq T_i[r_\iota], \tag{25}$$

where ι is such that $k_{\iota-1} \leq i \leq k_\iota$. Here, $r = r_1 + 2^m r_2 + \ldots + 2^{(c-1)m} r_c$ and $1 = k_0 \leq k_1 \leq \cdots \leq k_c = \alpha$ are as in Definition 2.2. Then, (T, v) is in the lookup relation $\mathsf{Lookup}_{N,L}$ if and only if:

(i) $(h; v, v_1, \ldots, v_\alpha)$ is in the polynomial constraint relation $\mathsf{PolyCon}_{\alpha+1,d,L}$. Here
$$h(X_0, X_1, \ldots, X_\alpha) = g(X_1, \ldots, X_\alpha) - X_0$$
(ii) $((T_{k_{i-1}+1}, \ldots, T_{k_i}), (v_{k_{i-1}+1}, \ldots, v_{k_\iota}))$ is in the vector-valued lookup relation $\mathsf{Lookup}_{k_i - k_{i-1}, M, L}$ for each $i \in \{0, \ldots, c-1\}$, and

For the forward direction, assume $(T, v) \in \mathsf{Lookup}_{N,L}$. Then for each $j \in [L]$, there is some $r \in [N]$ such that $v[j] = T[r]$. By Definition 2.2, if $r_1, \ldots, r_c$ are such that $r = r_1 + 2^m r_2 + \ldots + 2^{(c-1)m} r_c$, then

$$v[j] = T[r] = g(T_1[r_1], \ldots, T_{\kappa_1}[r_1], T_{\kappa_1+1}[r_2], \ldots, T_{\kappa_c}[r_c]).$$

Since $v_i[j] = T_i[r_\iota]$ by construction, we have $v[j] = g(v_1[j], \ldots, v_\alpha[j])$, so $(h; v, v_1, \ldots, v_\alpha) \in \mathsf{PolyCon}_{\alpha+1,d,L}$, so we have (i). Moreover, for each $i \in \{1, \ldots, c\}$,

$$(v_{k_{i-1}+1}[j], \ldots, v_{k_i}[j]) = (T_{k_{i-1}-1}[r_i], \ldots, T_{k_i}[r_i])$$

by construction, giving (ii).

For the reverse direction, let $j \in [L]$ be arbitrary: we want to show that there is some $r \in [N]$ such that $v[j] = T[r]$. By (i), we first see that

$$0 = h(v[j], v_1[j], \ldots, v_\alpha[j]) = g(v_1[j], \ldots, v_\alpha[j]) - v[j]$$

or $v[j] = g(v_1[j], \ldots, v_\alpha[j])$. Now, (ii) tells us that for each $i \in \{1, \ldots, c\}$, there is some $r_i \in [M]$ such that

$$(v_{k_{i-1}+1}[j], \ldots, v_{k_i}[j]) = (T_{k_{i-1}+1}[r_i], \ldots, T_{k_i}[r_i]).$$

Now, setting $r = r_1 + 2^m r_2 + \ldots + 2^{(c-1)m} r_c$, we have by the definition of a decomposable table:

$$v[j] = g(v_1[j], \ldots, v_\alpha[j]) = g(T_1[r_1], \ldots, T_{k_1}[r_1], T_{k_1+1}[r_2], \ldots, T_{k_c}[r_c]) = T[r],$$

as desired.

Therefore, given IOPs for $\mathsf{PolyCon}_{\alpha+1,d,L}$ and $\mathsf{Lookup}_{k_i-k_{i-1},M,L}$, we obtain an IOP for $\mathsf{Lookup}_{N,L}$ as follows:

- The prover sends the verifier polynomial oracles for $v, v_1, \ldots, v_\alpha$.
- The prover and verifier engage in the IOP for $\mathsf{PolyCon}_{\alpha+1,d,L}$, applied to the $\alpha+1$ oracles $v, v_1, \ldots, v_\alpha$ and the polynomial $h(X_0, X_1, \ldots, X_\alpha) = g(X_1, \ldots, X_\alpha) - X_0$. Note that the degree of h is equal to d, the degree of g.

- The prover and verifier engage in c independent instances of the IOP for $\mathsf{Lookup}_{k_i-k_{i-1},M,L}$, $i = 1, \ldots, c$, applied to the pairs
$$((T_{k_{i-1}+1}, \ldots, T_{k_i}), (v_{k_{i-1}+1}, \ldots, v_{k_i}))$$
for $i \in \{1, \ldots, c\}$.
  - If the IOP for $\mathsf{Lookup}_{k,M,L}$ used allows *batching*, the cost of applying it to c instances independently may be less than c times the cost of applying it to a single instance.

□

4. Avoiding commitments with sumcheck

Now, we describe an IOP for the relation $\mathsf{Lookup}_{M,L}$ which uses the *GKR protocol* to avoid the need to provide oracles for any auxiliary polynomials of length L.

4.1. **Logarithmic derivative argument.**

It is based on a *logarithmic derivative argument*, introduced in [3] and [5], which uses the following:

Lemma 4.1. *Let $f : \mathbf{F} \to \mathbf{F}$ be any function. Then*
$$L_f(X) \coloneqq \sum_{\alpha \in \mathbf{F}} \frac{f(\alpha)}{X + \alpha} = 0$$
in the field $\mathbf{F}(X)$ of rational functions over $\mathbf{F}$ if and only if $f(\alpha) = 0$ for all $\alpha \in \mathbf{F}$.

The reason for the terminology "logarithmic derivative" is that, *if f takes values in $\{0, \ldots, p-1\}$* where p is the *characteristic* of $\mathbf{F}$, then the sum $L_f(X)$ above is the *logarithmic derivative* $\frac{\pi'(X)}{\pi(X)}$ of the *monic* polynomial
$$\pi(X) = \prod_{\alpha \in \mathbf{F}} (X + \alpha)^{f(\alpha)}.$$

Proof. Since $\mathbf{F}(X)$ is a field, the sum $L_f(X)$ is zero in $\mathbf{F}(X)$ if and only if the sum
$$\sum_{\alpha \in \mathbf{F}} f(\alpha) \cdot \prod_{\beta \neq \alpha} (X + \beta) = 0 \tag{26}$$
in the ring $\mathbf{F}[X]$ of polynomials.

If $f(\alpha) = 0$ for all $\alpha \in \mathbf{F}$, (26) is clearly equal to 0. In the other direction, supposing (26) is equal to 0, in particular the result of evaluating it at any $-\alpha \in \mathbf{F}$ is 0. On the other hand, this evaluation is equal to
$$f(\alpha) \cdot \prod_{\beta \neq \alpha} (\beta - \alpha).$$
This is 0 if and only if $f(\alpha) = 0$, as desired. □

Using this, we show how to construct an IOP to interactively reduce the relation $\mathsf{Lookup}_{M,L}$ to the following:

Definition 4.2 (Rational sumcheck relation)**.** Let $p, q \in \mathbf{F}^L$ be two vectors and $\sigma \in \mathbf{F}$ a scalar. We say that $(\sigma; p, q)$ is in the *rational sumcheck relation* RatSum_L if and only if

$$\sum_{[L]} \frac{p[j]}{q[j]} = \sigma$$

Protocol 4.3 (Generic LogUp IOP)**.** *Let L be smaller than the* characteristic *of the finite field $\mathbf{F}$. Given an IOP for* RatSum*, we obtain an IOP for* $\mathsf{Lookup}_{k,M,L}$ *whose costs are:*

- *Those of an instance of* RatSum_L omitting *the cost, if present, of sending oracles for p, q,*
- *those of an instance of* RatSum_M*,* omitting *the cost, if present, of sending oracles for p, q,*
- *k polynomial oracles of size L for the vectors $v_1, \ldots, v_k$ of lookups,*
- *k polynomial oracles of size M for the lookup table $t = (t_1, \ldots, t_k)$,*
- *1 polynomial oracle of size M of "multiplicities", with all entries in $\{0, \ldots, L\}$,*
- *one query to each of these oracles,*
- *one additional scalar prover message, and*
- *two additional verifier challenges.*

The soundness error is at most the sum of the soundness errors of the RatSum *protocols plus*

$$\epsilon_{\text{logup}} \leq \frac{k + L + M}{|\mathbf{F}|}.$$

Proof. First, we show how to reduce $\mathsf{Lookup}_{k,M,L}$ to $\mathsf{Lookup}_{M,L}$ (this reduction is general and works for any lookup argument, not just LogUp): we want to show that for every $j \in [L]$ there is some $r \in [M]$ such that $v_i[j] = t_i[r]$ for each $i \in [k]$. We simply batch these k claims into a single claim:

- The verifier sends a "batching" challenge $\alpha \in \mathbf{F}$.

The prover sets $t = \sum_{i=1}^{k} \alpha^{i-1} t_i$ and $v = \sum_{i=1}^{k} \alpha^{i-1} v_i$. Now, we must prove the claim that $(t, v) \in \mathsf{Lookup}_{M,L}$, i.e. that for all $j \in [L]$, there is some $r \in [M]$ such that $v[j] = t[r]$. For such j, r, if it is not the case that $v_i[j] = t_i[r]$ for all $i \in [k]$, then by the Schwartz-Zippel Lemma 2.4, the probability that $v[j] = t[r]$ is at most

$$\epsilon_{\text{batch}} \leq \frac{k}{|\mathbf{F}|}.$$

Thus, it suffices to prove $(t, v) \in \mathsf{Lookup}_{M,L}$, i.e. that for all $j \in [L]$, there is some $r \in [M]$ such that $v[j] = t[r]$. Observe that this is equivalent to the requirement that the prover knows some vector m of size M such that for all $\alpha \in \mathbf{F}$, the following equation holds *in* $\mathbf{F}$:

$$\sum_{\substack{j \in [L] \\ v[j] = \alpha}} 1 = \sum_{\substack{r \in [M] \\ t[r] = \alpha}} m[r]. \tag{27}$$

Indeed, if $v[j] = \varphi$ for some $j \in [L]$, then *by the assumption that L is less than the characteristic of* $\mathbf{F}$, the left-hand side of (27) is nonzero. Thus, if the prover knows some m making (27) true for all α, in particular the right-hand side is nonzero for $\alpha = \varphi$, so there is some $r \in [M]$ such that $t[r] = \varphi = v[j]$.

Conversely, the prover computes m as follows: if $r \in [M]$ is such that $t[r] \neq t[r']$ for any $r' < r$, set

$$m[r] = \sum_{\substack{j \in [L] \\ v[j] = t[r]}} 1,$$

and otherwise set $m[r] = 0$. Then, for all $\alpha \in \mathbf{F}$, the right-hand side of (27) is equal to $m[r]$ where r is the first index in $[M]$ such that $t[r] = \alpha$, and this is equal to the left-hand side of (27) by construction.

Now, given a vector $m \in \mathbf{F}^M$, define a function $f_m \colon \mathbf{F} \to \mathbf{F}$ by subtracting the two sides of (27):

$$f_m(\alpha) = \sum_{\substack{j \in [L] \\ v[j] = \alpha}} 1 - \sum_{\substack{r \in [M] \\ t[r] = \alpha}} m[r]. \tag{28}$$

We have seen that the prover knows some vector m such that $f_m(\alpha) = 0$ for all $\alpha \in \mathbf{F}$ if and only if $(t, v) \in \mathsf{Lookup}_{M,L}$. By Lemma 4.1, we see that $f_m(\alpha) = 0$ for all $\alpha \in \mathbf{F}$ if and only if

$$0 = \sum_{\alpha \in \mathbf{F}} \frac{f_m(\alpha)}{X + \alpha} = \sum_{j \in [L]} \frac{1}{X + v[j]} - \sum_{r \in [M]} \frac{m[r]}{X + t[r]} \tag{29}$$

in the field $\mathbf{F}(X)$. With this observation, we construct our IOP as a proof of knowledge of vectors m, v, t satisfying (29):

- Given $(t, v) \in \mathsf{Lookup}_{M,L}$, the prover computes a "multiplicity" vector m of size M, where
$$m[r] = \#\{j \in [L] \mid v[j] = t[r]\}.$$
Note that $m[r] \in \{0, L\}$ for all $r \in [M]$.
- The prover sends the verifier polynomial oracles for m, $v_1, \ldots, v_k$, and $t_1, \ldots, t_k$.
- The verifier sends a challenge $\beta \in \mathbf{F}$.
- The prover sends a claimed sum $\sigma \in \mathbf{F}$.
- The prover and verifier engage in the IOP for RatSum_L, applied to the pair
$$p[j] \equiv 1, q[j] = \beta + v[j]$$
and sum σ, i.e. proving that
$$\sum_{j \in [L]} \frac{1}{\beta + v[j]} = \sigma.$$

  - Note that the verifier can compute evaluation queries to p and q given α, β and the oracles for $v_1, \ldots, v_k$, so the prover does not need to send oracles for p and q separately.
- The prover and verifier engage in the IOP for RatSum_M, applied to the pair
$$p[r] = m[r], q[r] = \beta + t[r]$$
and sum σ, i.e. proving that
$$\sum_{r \in [M]} \frac{m[r]}{\beta + t[r]} = \sigma.$$
  - Note that the verifier can compute evaluation queries to p and q using α, β and the oracles for $m, t_1, \ldots, t_k$, so the prover does not need to send oracles for p and q separately.

The IOPs for RatSum_M and RatSum_L prove that

$$\sum_{j \in [L]} \frac{1}{\beta + v[j]} - \sum_{r \in [M]} \frac{m[r]}{\beta + t[r]} = 0, \tag{30}$$

which is (29) evaluated at the *random* point $X = \beta$. By the Schwartz-Zippel Lemma 2.4 (applied to the polynomial of degree at most $L + M$ obtained by multiplying the left-hand side of (29) by the least common multiple of the denominators), the probability that (30) holds but (29) does not hold in $\mathbf{F}(X)$ is at most
$$\epsilon_{SZ} \leq \frac{L + M}{|\mathbf{F}|}.$$
As we have seen, this suffices to prove $((t_1, \ldots, t_k), (v_1, \ldots, v_k)) \in \mathsf{Lookup}_{k,M,L}$. The total soundness error is at most the sum of the soundness errors of the IOPs for RatSum_M and RatSum_L plus
$$\epsilon_{\text{logup}} \leq \epsilon_{\text{batch}} + \epsilon_{\text{SZ}} \leq \frac{k + M + L}{|\mathbf{F}|}.$$

□

4.1.1. *Batching multiple lookup columns with the same table.* A further attractive feature of the LogUp protocol is the ability to efficiently handle multiple lookup columns with the same table, i.e. the claim that $v_1, \ldots, v_C$ are vectors[1] such that $(t, v_i) \in \mathsf{Lookup}_{M,L}$ for $i = 1, \ldots, C$. Rather than running Protocol 4.3 C times independently, we can batch the lookups together as follows:

- The prover sends oracles for $v_1, \ldots, v_C$ and t,
- the prover sends *one* multiplicity vector m of size M, where $m[r]$ is the number of $(i, j) \in [C] \times [L]$ such that $v_i[j] = t[r]$,

[1]This works just as well for vector-valued lookups: we stick to the scalar case here just for notational convenience.

- instead of evaluating C different rational sums of length L as in Protocol 4.3, the prover and verifier engage in the IOP for RatSum_{CL} to compute the sum

$$\sum_{i=1}^{C} \sum_{j \in [L]} \frac{1}{\beta + v_i[j]}.$$

See [5] for a detailed treatment of this point.

4.2. **The GKR protocol for the rational sumcheck relation.** The GKR protocol [4] is a general IOP for proving satisfiability of *layered circuits*. These are arithmetic circuits consisting of a series of *layers*, with the outputs from each layer fed into the inputs of the next layer. In each layer, a single *inner circuit* is repeated some number of times in parallel.

It is based on the *sumcheck protocol* of Proposition 2.8, and has the attractive feature that the prover only needs to send oracles for the inputs at the first layer and the outputs at the last layer, not for any intermediate values.

Following [7], we describe how to use the GKR protocol to prove the rational sumcheck relation. The layered circuit in question computes the sum

$$\sum_{j \in [L]} \frac{p[j]}{q[j]} \tag{31}$$

using a binary tree of depth $\ell := \log(L)$.

The inner circuit computes the sum of two fractions $\frac{p_1}{q_1} + \frac{p_2}{q_2}$ using "projective coordinates":

$$(p_1, q_1, p_2, q_2) \mapsto (p_1 q_2 + p_2 q_1, q_1 q_2). \tag{32}$$

Protocol 4.4 (GKR for the rational sumcheck relation)**.** *We construct an IOP for the relation $(\sigma; p, q) \in \mathsf{RatSum}_L$ with $L = 2^\ell$ whose costs are:*

- *2 multilinear polynomial oracles of size L for the vectors p, q,*
- *one multilinear evaluation query at some $\rho \in \mathbf{F}^\ell$ to each of these oracles,*
- *$O(\ell^2)$ total communication,*
- *additional prover work consisting of $O(L)$ field operations.*

The soundness error is at most

$$\epsilon_{\mathrm{GKR}} \leq \frac{3\ell(\ell+5)}{2|\mathbf{F}|}$$

Proof. We label the layers of the circuit computing (31) from ℓ to 1, with layer ℓ as the input layer. Each layer i takes as input two multilinear polynomials $p^{(i)}$ and $q^{(i)}$ in i variables, with

$$(p^{(\ell)}[r], q^{(\ell)}[r])_{r \in \{0,1\}^\ell} := (p[r], q[r]),$$

identifying $\{0,1\}^\ell$ with $[L]$ with (3).

Layer i has as outputs two multilinear polynomials $p^{(i-1)}, q^{(i-1)}$ in $i-1$ variables, which are computed from the inputs by applying (32) 2^{i-1} times in parallel:

$$p^{(i-1)}[r] \coloneqq p^{(i)}[(0,r)] * q^{(i)}[(1,r)] + p^{(i)}[(1,r)] * q^{(i)}[(0,r)], \tag{33}$$
$$q^{(i-1)}[r] \coloneqq q^{(i)}[(0,r)] * q^{(i)}[(1,r)],$$

where $r \in \{0,1\}^{i-1}$. These outputs are then identified used as the inputs to layer $i-1$. In other words, each (p,q) pair among the outputs of layer i gives the projective coordinates of the fractional sum of two adjacent (p,q) pairs from the inputs to layer i:

$$\frac{p^{(i-1)}[r]}{q^{(i-1)}[r]} = \frac{p^{(i)}[(0,r)]}{q^{(i)}[(0,r)]} + \frac{p^{(i)}[(1,r)]}{q^{(i)}[(1,r)]}. \tag{34}$$

Layer 1 has 2 outputs labeled $p^{(0)}, q^{(0)}$, and $\frac{p^{(0)}}{q^{(0)}}$ is the sum (31).

Next, we will apply Protocol 2.8 once for each layer, reducing the computation of a multilinear evaluation query to the oracles for $p^{(i-1)}, q^{(i-1)}$ to multilinear evaluation queries to the oracles for $p^{(i)}, q^{(i)}$, as follows:

- The prover starts the protocol by sending multilinear polynomial oracles for $p^{(\ell)} = \widetilde{p}, q^{(\ell)} = \widetilde{q}$.

Now, we recursively define multilinear polynomial oracles $p^{(i)}$ for $i = 1, \ldots, \ell - 1$ by applying (33) to the oracles for $p^{(i+1)}, q^{(i+1)}$, starting with $p^{(\ell)}, q^{(\ell)}$.

The initial claim is that

$$\frac{p^{(0)}}{q^{(0)}} = \sigma.$$

Applying (33) once, we have

$$p^{(0)} = p^{(1)}[0] * q^{(1)}[1] + p^{(1)}[1] * q^{(1)}[0], \qquad q^{(0)} = q^{(1)}[0] * q^{(1)}[1] \tag{35}$$

- The prover sends the four claimed values

$$m^{(1)} \coloneqq (p_0^{(1)}, p_1^{(1)}, q_0^{(1)}, q_1^{(1)}) \in \mathbf{F}^4. \tag{36}$$

  If the prover is honest, then these satisfy:

$$p_b^{(1)} = p^{(1)}[b], \qquad q_b^{(1)} = q^{(1)}[b], \qquad \text{for } b \in \{0,1\}. \tag{37}$$

- The verifier computes $p^{(0)}, q^{(0)}$ using (35) and the claimed values $m^{(1)}$, then checks whether $p^{(0)}/q^{(0)} = \sigma$, aborting otherwise.

It now suffices to prove the claim that (37) holds.

This is the form of the claim that will be convenient for induction. Namely, for each i, the prover and verifier will compute four values

$$m^{(i)} = (m_{p,0}^{(i)}, m_{p,1}^{(i)}, m_{q,0}^{(i)}, m_{q,1}^{(i)}) \in \mathbf{F}^4.$$

If $i > 1$, they will also compute a challenge value $\overline{r}^{(i)} \in \mathbf{F}^{i-1}$. The inductive claims will be:

$$
\begin{aligned}
m_{p,0}^{(i)} &= p^{(i)}[(0, \overline{r}^{(i)})], \qquad & m_{p,1}^{(i)} &= p^{(i)}[(1, \overline{r}^{(i)})], \\
m_{q,0}^{(i)} &= q^{(i)}[(0, \overline{r}^{(i)})], \qquad & m_{q,1}^{(i)} &= q^{(i)}[(1, \overline{r}^{(i)})].
\end{aligned} \tag{38}
$$

First, we batch these into two claims using a random challenge:

- The verifier sends a challenge $\rho^{(i)} \in \mathbf{F}$.

Now, note that because $p^{(i)}$ and $q^{(i)}$ are multilinear, they are in particular degree one polynomial (or 'affine') functions of their first variable when the other variables are fixed. This implies:

$$
\begin{aligned}
p^{(i)}(\rho^{(i)}, \overline{r}^{(i)}) &= (1 - \rho^{(i)}) * p^{(i)}[(0, \overline{r}^{(i)})] + \rho^{(i)} * p^{(i)}[(1, \overline{r}^{(i)})], \\
q^{(i)}(\rho^{(i)}, \overline{r}^{(i)}) &= (1 - \rho^{(i)}) * q^{(i)}[(0, \overline{r}^{(i)})] + \rho^{(i)} * q^{(i)}[(1, \overline{r}^{(i)})].
\end{aligned} \tag{39}
$$

We define $r^{(i)} = (\rho^{(i)}, \overline{r}^{(i)}) \in \mathbf{F}^i$. Using (39), we batch the claims in (38) as:

$$
\begin{aligned}
p^{(i)}(r^{(i)}) = e_p^{(i)} &\coloneqq (1 - \rho^{(i)}) * m_{p,0}^{(i)} + \rho^{(i)} * m_{p,1}^{(i)}, \\
q^{(i)}(r^{(i)}) = e_q^{(i)} &\coloneqq (1 - \rho^{(i)}) * m_{q,0}^{(i)} + \rho^{(i)} * m_{q,1}^{(i)}.
\end{aligned} \tag{40}
$$

Note that the verifier can compute the claimed evaluations $e_p^{(i)}$ and $e_q^{(i)}$ from $m^{(i)}$ using a constant number of field operations (four multiplications and three additions).

Now, we will use the sumcheck protocol to reduce (40) to (38) for $i + 1$. First, we expand (40) using (5):

$$
e_p^{(i)} = \sum_{b \in \mathbf{B}_i} p^{(i)}[b] * L_b(r^{(i)}), \qquad e_q^{(i)} = \sum_{b \in \mathbf{B}_i} q^{(i)}[b] * L_b(r^{(i)}). \tag{41}
$$

Then, we expand this further using (33):

$$
\begin{aligned}
e_p^{(i)} &= \sum_{b \in \mathbf{B}_i} \Big(p^{(i+1)}[(0, b)] * q^{(i+1)}[(1, b)] + p^{(i+1)}[(1, b)] * q^{(i+1)}[(0, b)] \Big) * L_b(r^{(i)}), \\
e_q^{(i)} &= \sum_{b \in \mathbf{B}_i} q^{(i+1)}[(0, b)] * q^{(i+1)}[(1, b)] * L_b(r^{(i)}).
\end{aligned} \tag{42}
$$

These claims are instances of the sumcheck relation $\mathsf{Sum}_{3,i,5}$ and $\mathsf{Sum}_{3,i,3}$ respectively, each with degree 3 and in i variables. More precisely, we define oracles $p_0^{(i+1)}$, $p_1^{(i+1)}$, $q_0^{(i+1)}$, $q_1^{(i+1)}$ for multilinear polynomials in i variables by:

$$
\begin{aligned}
p_0^{(i+1)}[r] &= p^{(i+1)}[(0, r)], \qquad & p_1^{(i+1)}[r] &= p^{(i+1)}[(1, r)], \\
q_0^{(i+1)}[r] &= q^{(i+1)}[(0, r)], \qquad & q_1^{(i+1)}[r] &= q^{(i+1)}[(1, r)].
\end{aligned} \tag{43}
$$

We also have an oracle for the polynomial $\ell^{(i)}$ which is the multilinear extension of

$$b \mapsto L_b(r^{(i)}). \tag{44}$$

The verifier may compute queries to this oracle explicitly with i multiplications using the following formula:

$$\ell^{(i)}(X_1, \ldots, X_i) = \prod_{j=1}^{i} \left(X_j * r_j - (1 - X_j) * (1 - r_j)\right). \tag{45}$$

To see that this formula provides the multilinear extension of (44), note that (45) defines a multilinear polynomial in $\mathbf{M}_i$, and that if $b = (b_1, \ldots, b_i) \in \mathbf{B}_i$, then evaluating (45) at b gives exactly the definition (4) of $L_b(r^{(i)})$.

Defining

$$\begin{aligned} g_p(X_1, X_2, X_3, X_4, X_5) &= (X_1 * X_4 + X_2 * X_3) * X_5 \\ g_q(X_1, X_2, X_3) &= X_1 * X_2 * X_3, \end{aligned} \tag{46}$$

we rephrase (42) as

$$\begin{aligned} \left(e_p^{(i)}, g_p; p_0^{(i+1)}, q_0^{(i+1)}, p_1^{(i+1)}, q_1^{(i+1)}, \ell^{(i)}\right) &\in \mathsf{Sum}_{3,i,5}, \\ \left(e_q^{(i)}, g_q; q_0^{(i+1)}, q_1^{(i+1)}, \ell^{(i)}\right) &\in \mathsf{Sum}_{3,i,3}. \end{aligned} \tag{47}$$

We batch the two claims in (47) into a single claim, and then use the sumcheck protocol:

- The verifier sends a challenge $\lambda^{(i)} \in \mathbf{F}$.
- The prover and verifier engage in the sumcheck protocol applied to the claim

$$(e_p^{(i)} + \lambda^{(i)} e_q^{(i)}, g_p + \lambda^{(i)} g_q; p_0^{(i+1)}, q_0^{(i+1)}, p_1^{(i+1)}, q_1^{(i+1)}, \ell^{(i)}) \in \mathsf{Sum}_{3,i,5} \tag{48}$$

Applying Protocol 2.8 to (48), we obtain the following:

- A random point $\overline{r}^{(i+1)} \in \mathbf{F}^i$,
- A 4-tuple of claims

$$m^{(i+1)} = (m_{p,0}^{(i+1)}, m_{p,1}^{(i+1)}, m_{q,0}^{(i+1)}, m_{q,1}^{(i+1)}) \in \mathbf{F}^4$$

  for each of $p_0^{(i+1)}, q_1^{(i+1)}, p_1^{(i+1)}, q_0^{(i+1)}$, evaluated at $\overline{r}^{(i+1)}$.

By the soundness of Protocol 2.8, except with probability $\epsilon_{\text{sumcheck}}^{(i)} \leq \frac{3i}{|\mathbf{F}|}$ over the choice of the random value $\overline{r}^{(i+1)}$, (47) is true if and only if:

$$\begin{aligned} m_{p,0}^{(i+1)} &= p^{(i+1)}[(0, \overline{r}^{(i+1)})], & m_{p,1}^{(i+1)} &= p^{(i+1)}[(1, \overline{r}^{(i+1)}), \\ m_{q,0}^{(i+1)} &= q^{(i+1)}[(0, \overline{r}^{(i+1)})], & m_{q,1}^{(i+1)} &= q^{(i+1)}[(1, \overline{r}^{(i+1)})], \end{aligned}$$

which is the inductive claim (38).

At the end of the induction, we are left with claims:

$$m_{p,0}^{(\ell)} = p^{(\ell)}[(0, \overline{r}^{(\ell)})], \qquad m_{p,1}^{(\ell)} = p^{(\ell)}[(1, \overline{r}^{(\ell)}),$$
$$m_{q,0}^{(\ell)} = q^{(\ell)}[(0, \overline{r}^{(\ell)})], \qquad m_{q,1}^{(\ell)} = q^{(\ell)}[(1, \overline{r}^{(\ell)})],$$

We use one final verifier challenge $\rho^{(\ell)} \in \mathbf{F}$ to batch these into two claims as in (40), setting $r^{(\ell)} = (\rho, \overline{r}^{(\ell)})$:

$$p^{(\ell)}(r^{(\ell)}) = e_p^{(\ell)} \coloneqq \rho * m_{p,0}^{(\ell)} + (1-\rho) * m_{p,1}^{(\ell)},$$
$$q^{(\ell)}(r^{(\ell)}) = e_q^{(\ell)} \coloneqq \rho * m_{q,0}^{(\ell)} + (1-\rho) * m_{q,1}^{(\ell)}.$$

- Finally, the verifier makes evaluation queries to the oracles $p^{(\ell)} = \widetilde{p}$, $q^{(\ell)} = \widetilde{q}$ at the point $r^{(\ell)}$.

To tally up the costs, we note that there are ℓ sumcheck instances, each proving the relation $\mathsf{Sum}_{3,i,5}$ for $i = 1, \ldots, \ell$. The cost of running Protocol 2.8 for $\mathsf{Sum}_{3,i,5}$ is:

- i prover messages, each of length 3,
- a final prover message of length 4 consisting of evaluation claims for the oracles $p_0^{(i)}, p_1^{(i)}, q_0^{(i)}, q_1^{(i)}$,
  - An evaluation claim for the fifth oracle $\ell^{(i)}$ is not needed, as the verifier may evaluate this polynomial directly using (45).
- i verifier challenges, each of length 1,
- verifier work consisting of $\approx 4i$ field scalar multiplications,
- prover work consisting of $O(2^i)$ field operations,
- a soundness error of size

$$\epsilon_{\text{sumcheck}}^{(i)} \leq \frac{3i}{|\mathbf{F}|}$$

Adding these up, the total costs of the ℓ sumcheck instances are:

- $1 + 2 + \cdots + \ell = \frac{\ell(\ell+1)}{2}$ prover messages of length 3 and ℓ prover messages of length 4, for $O(\ell^2)$ total prover communication,
- $1 + 2 + \cdots + \ell = \frac{\ell(\ell+1)}{2} = O(\ell^2)$ verifier challenges,
- $4 + 8 + \cdots + 4\ell = 4\frac{\ell(\ell+1)}{2} = O(\ell^2)$ verifier work,
- $O(2 + 4 + \cdots + 2^\ell) = O(2^{\ell+1}) = O(L)$ prover work,
- a soundness error of size

$$\epsilon_{\text{sumcheck,tot}} \leq \frac{3}{|\mathbf{F}|}(1 + 2 + \cdots + \ell) = \frac{3\ell(\ell+1)}{2|\mathbf{F}|} = O\left(\frac{\ell^2}{|\mathbf{F}|}\right).$$

The additional costs for each round, coming from batching the claims at the end of each sumcheck, are:

- two additional verifier challenges $\rho^{(i)}, \lambda^{(i)}$,
- an additional soundness error of size $\epsilon_{\text{batch}}^{(i)} \leq \frac{2}{|\mathbf{F}|}$,
  - this additional error comes from the choices of $\rho^{(i)}$ and $\lambda^{(i)}$: each is used to batch the claims for the values of two scalars together,

so by the linear univariate case of the Schwartz-Zippel lemma 2.4, the probability that the original claims are false while the batched claims are true is at most $\frac{2}{|\mathbf{F}|}$.

Finally, we note that the only oracles the prover needs to send over the course of the protocol are those for p, q, and the verifier makes a single evaluation query to each of these oracles at the end of the protocol.

Adding together the soundness errors for each round, we obtain the total soundness error of the protocol:

$$\epsilon_{GKR} \leq \sum_{i=1}^{\ell} \epsilon^{(i)}_{\text{batch}} + \epsilon^{(i)}_{\text{sumcheck}} \leq \frac{3\ell(\ell+5)}{2|\mathbf{F}|}.$$

For a more detailed accounting of the costs for this protocol, see [7]. □

4.2.1. *Batched variant.* It is possible to reduce the number of rounds of sumcheck in this protocol by using a tree of arity $\kappa > 2$ and lower depth to compute the sum: i.e. by using an inner circuit which adds up κ fractions at a time. The tradeoff is that the degree of the sumcheck polynomials is $\kappa + 1$ rather than 3, so the prover work and communication costs of each sumcheck instance increase linearly with κ. For more discussion of this point, see [5] and [7].

5. Putting it all together: TaSSLE

Now, we describe the TaSSLE protocol in full: we apply Protocol 3.1, using a combination of Protocols 4.4 and 4.3 for the lookup argument and Protocol 2.11 for the polynomial constraint argument:

Protocol 5.1. *Let T be a decomposable table of length $N = 2^n$ and width κ with roots $T_1, \ldots, T_\alpha$ of length $M = 2^m = N^{1/c}$. Let d be the maximum degree of the polynomials g_i as in Definition 2.2. We construct an IOP for the relation $(g; T, v) \in \mathsf{Lookup}_{\kappa,N,L}$. The total costs are*

- *κ multilinear oracles for vectors of length L: these are the columns of the table v subject to lookup,*
- *one query to each of these oracles at some $\tau \in \mathbf{F}^\ell$,*
- *α multilinear oracles[2] for vectors of length M: these are the "root" vectors T_i from which T is built as in Definition 2.2,*
- *one query to each of these oracles at some $r \in \mathbf{F}^m$,*
- *α multilinear oracles for vectors of length L: the entries of these vectors belong to the root vectors T_i, so they will be small values if the entries of T_i are,*
- *two queries to each of these oracles: one at τ and one at some $r' \in \mathbf{F}^\ell$,*

[2]For some decomposable lookup tables T of interest, the T_i have the additional useful property of being "MLE-structured", meaning that the verifier may compute multilinear evaluation queries to the T_i directly from a public succinct description in time logarithmic in M. If this is the case, then we do not need the prover to send oracles for the T_i.

- *α multilinear oracles for vectors of length M for the "multiplicity vectors" mult_i: the entries of these vectors lie in $[L]$,*
- *one query to each of these oracles at r,*
- *the costs of running the sumcheck protocol for $\mathsf{Sum}_{d+1,k+\alpha,L}$,*
- *the costs of running the sumcheck protocol for $\mathsf{Sum}_{3,4c+1,2^i}$ for $i = 1, \ldots, \ell - 1$,*
- *total oracle cost of $\kappa + \alpha$ length-L oracles and 2α length-M oracles*
- *total communication of $O(\ell^2 + d\ell + \kappa + \alpha)$ scalars,*
- *total prover work of $O(d(\kappa + \alpha + b)L)$ field operations, where b is the total number of monomials among the κ polynomials g_i comprising g, and*
- *total verifier work of $O(\ell^2 + d\ell + b)$ field operations and sampling $O(\ell^2)$ random scalars.*

The total soundness error is at most

$$\epsilon_{\mathsf{TaSSLE}} \leq \frac{3\ell(\ell + 4c + 1)}{2|\mathbf{F}|} + \frac{3m(m + 4c + 1)}{2|\mathbf{F}|} + \frac{(2d + \kappa + 3)\ell}{|\mathbf{F}|} + \frac{\alpha + c(L + M)}{|\mathbf{F}|};$$

in particular,

$$\epsilon_{\mathsf{TaSSLE}} = O\left(\frac{(d + \kappa)\log(L) + c(L + M)}{|\mathbf{F}|}\right)$$

These claims on soundness and costs follow immediately from the claims proven in Protocols 2.11, 3.1, 4.3, and 4.4. For convenience, we describe the protocol in full:

- The prover sends multilinear oracles for the κ columns $v^{(1)}, \ldots, v^{(\kappa)}$ of the matrix $v \in \mathbf{F}^{\kappa \times L}$ subject to the vector-valued lookup relation $\mathsf{Lookup}_{\kappa,N,L}$.
- The prover sends α multilinear oracles for the "root" vectors T_i of length $M = N^{1/c}$.
- The prover sends α multilinear oracles for the vectors $v_1, \ldots, v_\alpha$ of length L defined by

$$(v_{k_{i-1}+1}, \ldots, v_{k_i})[j] = (T_{k_{i-1}+1}, \ldots, T_{k_i})[r_i] \tag{49}$$

  for $i = 1, \ldots, c$, where the k_i are as in Definition 2.2.

The prover and verifier now engage in Protocol 2.11 to show

$$v^{(i)}[j] = g_i(v_1[j], \ldots, v_\alpha[j]) \tag{50}$$

for all $i \in [\kappa], j \in [L]$.

- The verifier sends a random challenge $\lambda \in \mathbf{F}$, used to combine the κ polynomial equations in (50) into a single claim

$$h(v^{(1)}, \ldots, v^{(\kappa)}, v_1, \ldots, v_\alpha)[j] = 0 \tag{51}$$

  for all $j \in [L]$.
- The verifier sends a challenge point $\tau \in \mathbf{F}^\ell$.

- The prover and verifier engage in one round of sumcheck for $\mathsf{Sum}_{d+1,\kappa+\alpha,L}$ to prove that $\widetilde{h}(\tau) = 0$.
- The verifier makes queries to each of the $\kappa + \alpha$ oracles $v^{(i)}$ and v_j at the same point r, sampled in the course of the sumcheck protocol.

Now, it remains to show (49) holds, which amounts to instances of the vector-valued lookup relation $\mathsf{Lookup}_{k_i - k_{i-1},M,L}$ for $i = 1, \ldots, c$.

First, we use Protocol 4.3 to reduce this claim to the computation of a rational sum:

- The prover computes "multiplicity" vectors mult_i of length M for $i = 1, \ldots, \alpha$, counting the number of times each row of T_i appears as $(v_{k_{i-1}+1}[j], \ldots, v_{k_i}[j])$ for some $j \in [L]$.
- The prover sends the verifier oracles for these multiplicity vectors.
- The verifier sends a random challenge $\alpha \in \mathbf{F}$, used to reduce the length-$(k_i - k_{i-1})$ vector lookup claims in (49) to scalar lookup claims.
- The verifier sends a random challenge $\beta \in \mathbf{F}$ at which to evaluate the logarithmic derivatives arising in Protocol 4.3.
- The prover sends claimed sums $\sigma_i \in \mathbf{F}$, $i = 1, \ldots, c$

At this point, it remains to prove the following sums:

$$\sigma_i = \sum_{j \in [L]} \frac{p_i^v[j]}{q_i^v[j]}, \qquad \sigma_i = \sum_{r \in [M]} \frac{p_i^t[r]}{q_i^t[r]} \tag{52}$$

for $i = 1, \ldots, c$, where

$$p_i^v[j] \equiv 1, \quad p_i^t[r] = \mathsf{mult}_i[r] \tag{53}$$

and

$$q_i^v[j] = \beta + \sum_{s=1}^{k_i - k_{i-1}} \alpha^{s-1} v_{k_{i-1}+s}[j], \qquad q_i^t[r] = \beta + \sum_{s=1}^{k_i - k_{i-1}} \alpha^{s-1} T_{k_{i-1}+s}[r]. \tag{54}$$

We do this by applying a batched version of Protocol 4.4 twice, once for the sums of length L and once for the sums of length M.

For ease of notation, we describe just the first of these, writing p_i for p_i^v, q_i for q_i^v.

- First, the prover computes "partial sum" oracles $p_i^{(s)}, q_i^{(s)} \in \mathbf{M}_s$ for $s = \ell, \ldots, 1$ by applying the recursive formula (33) to the oracles p_i, q_i.

Then, for $s = 1, \ldots, \ell - 1$:

- For each $i = 1, \ldots, c$, the prover sends a vector $m_i^{(s)} \in \mathbf{F}^4$ consisting of claimed values for $p_i^{(s)}[(b, \overline{r}^{(s)})]$ and $q_i^{(s)}[(b, \overline{r}^{(s)})]$ for $b = 0, 1$.
  - For $s = 1$, there is no $\overline{r}^{(s)}$. For $s > 1$, it will have been computed in the prior step.
- The verifier sends c challenges $\rho_i^{(s)} \in \mathbf{F}$, $i = 1, \ldots, c$.

- Using the $\rho_i^{(s)}$, the prover and verifier batch these into two claims for each i as in (40), setting $r_i^{(s)} = (\rho_i^{(s)}, \overline{r}_i^{(s)})$:

$$p_i^{(s)}(r^{(s)}) = e_{p,i}^{(s)}, \quad q_i^{(s)}(r^{(s)}) = e_{q,i}^{(s)}. \tag{55}$$

- We rewrite these as instances of the $\mathsf{Sum}_{3,s,5}$ relation in terms of the $p_i^{(s+1)}, q_i^{(s+1)}$ as in (42):

$$e_{p,i}^{(s)} = \sum_{b\in\mathbf{B}_i} \left(p_i^{(s+1)}[(0,b)] * q_i^{(s+1)}[(1,b)] + p_i^{(s+1)}[(1,b)] * q_i^{(s+1)}[(0,b)] \right) * L_b(r^{(s)}),$$

$$e_{q,i}^{(s)} = \sum_{b\in\mathbf{B}_i} q_i^{(s+1)}[(0,b)] * q_i^{(s+1)}[(1,b)] * L_b(r^{(s)}).$$

- The verifier sends a challenge $\lambda^{(s)} \in \mathbf{F}$, used to batch the $2c$ $\mathsf{Sum}_{3,s,5}$ claims in (55) into a single $\mathsf{Sum}_{3,s,4c+1}$ claim

$$e^{(s)} = \sum_{b\in\mathbf{B}_i} \varphi^{(s)} \left((p_{i,b}^{(s+1)}, q_{i,b}^{(s+1)})_{b\in\{0,1\},i\in[c]}, \ell^{(s)} \right) \tag{56}$$

where

$$\varphi^{(s)}((X_{i,b}, Y_{i,b})_{b\in\{0,1\},i\in[c]}, Z) = Z * \sum_{i=1}^{c} (\lambda^{(s)})^{i-1} * \left(X_{i,0}Y_{i,1} + X_{i,1}Y_{i,0} + (\lambda^{(s)})^c Y_{i,0}Y_{i,1} \right),$$

$$e^{(s)} = \sum_{i=1}^{c} (\lambda^{(s)})^{i-1} * \left(e_{p,i}^{(s)} + (\lambda^{(s)})^c e_{q,i}^{(s)} \right),$$

and $\ell^{(s)}$ is the multilinear extension of the function $b \mapsto L_b(r^{(s)})$ as in (44):

$$\ell^{(s)}(X_1, \ldots, X_s) = \prod_{j=1}^{s} \left(X_j * r_j^{(s)} + (1 - X_j) * (1 - r_j^{(s)}) \right)$$

- The verifier computes $e^{(s)}$.
- The prover and verifier engage in Protocol 2.8 to prove (56), yielding the claim for the next round:
  - A random point $\overline{r}^{(s+1)} \in \mathbf{F}^s$,
  - For $i = 1, \ldots, c$, a 4-tuple of claims $m_i^{(s+1)} \in \mathbf{F}^4$ for each of $p_i^{(s+1)}[(b, \overline{r}^{(s+1)})]$, $q_i^{(s+1)}[(b, \overline{r}^{(s+1)})]$, $b = 0, 1$.

At the end of the round for $s = \ell - 1$, we are left with claims for $p_i[(b, \overline{r}^{(\ell)})]$, $q_i[(b, \overline{r}^{(\ell)})]$ for $b = 0, 1$ and $i = 1, \ldots, c$. Finally:

- The verifier sends a last challenge $\rho^{(\ell)} \in \mathbf{F}$ and sets $r^{(\ell)} = (\rho^{(\ell)}, \overline{r}^{(\ell)})$.
- We batch the final claims using $\rho^{(\ell)}$ as in (40) to obtain $2c$ claims for the values of $p_i(r^{(\ell)})$, $q_i(r^{(\ell)})$, $i = 1, \ldots, c$.
- The verifier checks these claims by making evaluation queries to the α oracles $v_1, \ldots, v_\alpha$ sent by the prover earlier and applying (53) and (54).

  - This is the only part of the application of Protocol 4.4 that differs between the length-L and length-M sums: for the latter, the verifier makes evaluation queries to the 2α oracles $T_1, \ldots, T_\alpha, \mathsf{mult}_1, \ldots, \mathsf{mult}_\alpha$ instead.

5.1. **Comparison with Lasso.** Finally, we discuss how the TaSSLE protocol compares with the Lasso protocol of [9]. Like TaSSLE, the Lasso protocol essentially is the result of combining the general argument of Protocol 3.1 with the sumcheck IOP of Protocol 2.11 and a lookup argument. However, the lookup argument used there is different: it is a novel protocol, based on offline memory checking and the sparse polynomial commitment scheme in [8]. Rather than a rational sum, the lookup argument presented there reduces the lookup claim to the evaluation of a product. Then, similarly to Protocol 4.4, this product is computed without any further commitments using the GKR protocol.

A key benefit of working with logarithmic derivatives is the ability to use the multiplicities of the rows of the lookup table in the argument very directly: to include a fraction $\frac{1}{q}$ in the sum m times, we need only multiply the term $\frac{1}{q}$ by m. On the other hand, including a factor π in a product m times requires computing π^m, which cannot be expressed as directly in terms of polynomial constraints in π and m. As such, lookup arguments based on products require more complex constructions.

In Lasso, the committed vectors are:

- the column[3] v of length L subject to lookup,
- the α "root" vectors T_i of length M,
  - These are not included in the statements of the main theorems of [9], as Lasso works throughout with the assumption that the T_i are MLE-structured.
- α vectors of length L, giving the vectors of lookups into the T_i which we denote $v_1, \ldots, v_\alpha$ above,
- α vectors of length L giving the indices of the lookups of the v_i into the T_i, with all entries in $[M]$.
- α vectors of length L giving "read-timestamps" for the lookups of the v_i into the T_i, with all entries in $[L]$.
- α vectors of length M giving the "final timestamps" for the lookups of the v_i into the T_i, with all entries in $[L]$.
  - These are precisely the vectors mult_i used in the TaSSLE protocol.

Thus, we see that the committed vectors in TaSSLE are a strict subset of those in Lasso: instead of $3\alpha + 1$ vectors of length L and 2α vectors of length M, we have $\alpha + 1$ vectors of length L and 2α vectors of length M, omitting the 2α vectors of indices and read-timestamps.

[3] As [9] only treats the case of scalar-valued lookups, we restrict ourselves to that case here as well.

References

[1] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2024.

[2] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 499–530. Springer, 2023.

[3] Liam Eagen, Sanket Kanjalkar, Tim Ruffing, and Jonas Nick. Bulletproofs++: next generation confidential transactions via reciprocal set membership arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 249–279. Springer, 2024.

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

[5] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. *Cryptology ePrint Archive*, 2022.

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

[7] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using gkr. *Cryptology ePrint Archive*, 2023.

[8] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[9] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 180–209. Springer, 2024.