Logup*: faster, cheaper logup argument for small-table indexed lookups

Lev Soukhanov [*]

May 2025

Abstract

Logup argument (in it's modern GKR version, as described in eprint:2023/1284 paper) is a logarithmic derivative-based unindexed lookup argument. An indexed lookup argument can be constructed from unindexed one using standard trick.

In this short informal note, we explain a different way of obtaining indexed lookup from logup, which does not commit any additional arrays of the size of the indexing array. That makes it particularly amenable for lookups in small tables (giving, to our knowledge, a first argument with this property).

Additionally, this argument is not subject to numerator overflow issue that requires additional mitigation described in eprint:2024/2067.

Improvements to SPARK / Lasso protocols are also discussed.

1 Introduction

Logup argument, as described in [4] [5], is a following unindexed lookup argument: for two arrays X, T with values in some cryptographically large field $\mathbb{F}$ of characteristic p we wish to prove that $X \subset T$ (as sets). For simplicity of exposition, also assume that Y's elements do not repeat.

Denote $|X| = n, |T| = m$.

Prover commits these arrays and additionally commits an array acc of size m which represents *access counts* - $\mathsf{acc}[i]$ is the number of times that the value $T[i]$ is visited by X. Assume, additionally, that $|X| < p$, so these access counts never overflow.

[*][[alloc]init]. email:0xdeadfae@gmail.com

Then, it is enough to check, for a random challenge $c \in \mathbb{F}$ that

$$\sum_{0 \leq i < n} \frac{1}{c - X[i]} = \sum_{0 \leq j < m} \frac{\mathsf{acc[j]}}{c - T[j]}$$

Soundness error of this argument is $\frac{n+m}{|\mathbb{F}|}$, and the check itself is nowadays usually done using GKR protocol as described in [5].

Logup is sometimes used in an indexed lookup form; say as a drop-in replacement for permutation argument lookups in Lasso [8] and [6]. In these cases, a standard reduction of indexed lookup to non-indexed lookup is typically used.

Let table T be as before and let I be a set of indices $|I|$. We denote (I^*T) the indexed lookup of table T with indices I (also called *pullback* further), defined as

$$(I^*T)[i] = T[I[i]]$$

Standard reduction does the following:

1. (Assuming I, T already committed) commit to I^*T, acc.

2. Get challenges c, γ.

3. Validate

$$\sum_{0 \leq i < n} \frac{1}{c - (I^*T[i] + \gamma I)} = \sum_{0 \leq jm} \frac{\mathsf{acc[j]}}{c - (T[j] + \gamma j)}$$

Interpretation here is that we are actually doing an unindexed lookup argument of the set of pairs $(I[i], I^*T[i])$ in a table $(j, T[j])$, with random challenge γ used to fold a tuple into a single representative value.

We are mostly interested in the following setting:

1. $m << n$.

2. Indices live in some "small" subset of a field that can be efficiently committed to (maybe $\mathbb{F}$ is an extension field of some smaller base field, or maybe our commitment scheme is just more efficient for small elements).

3. Table T consists of large, essentially random elements of $\mathbb{F}$ (this frequently happens, for example, in SPARK the table represents the evaluations of polynomial eq_r).

In this regime, the logup argument has a significant overhead, commitment-wise. In addition to the indexing array I, which is n *small* elements, the prover needs to commit the array of looked up values I^*T, which is n *large* elements.

Certain ad-hoc mitigations are available for elliptic curve commitments; but for hash-based schemes, this is a significant performance killer.

We suggest an alternative protocol that doesn't have these issues.

Acknowledgement. Author thanks Yar Rebenko, Misha Komarov, Ron Rothblum, Srinath Setty and Liam Eagen for useful discussions that led to this work.

2 Notation

2.1 Multilinear polynomials

We assume some familiarity of the reader with the multilinear setting. We refer to excellent book by Justin Thaler [10], and to foundational papers on sumcheck [3] and gkr protocol [2][9].

We will use the following notation. For an array P of size 2^k we will denote with the same letter the unique multilinear polynomial with values $P(i_0, ..., i_{k-1}) = P[i_0 + 2i_1 + ... + 2^{k-1}i_{k-1}]$ on a boolean hypercube $(i_0, ..., i_{k-1}) \in \mathbb{B}^k$.

We will frequently use the inner product. For two arrays P, Q of the same size 2^k the value

$$\langle P, Q \rangle = \sum_{0 \leq i < 2^k} P[i]Q[i] = \sum_{x \in \mathbb{B}^k} P(x)Q(x)$$

Evaluation of a polynomial P in a point $r = (r_0, ..., r_{k-1})$ can be computed as an inner product with the multilinear Lagrange kernel:

$$P(r) = \langle P, \mathsf{eq}_r \rangle$$

where

$$\mathsf{eq}_r(x) = \prod_{0 \leq i < k} (r_i x_i + (1 - r_i)(1 - x_i))$$

We will use the sumcheck protocol. Its internal workings are not very important here, what is important is that it is a claim reduction protocol that, for a multivariate polynomial U reduces the claim about it's sum

$$\sum_{x \in \mathbb{B}^k} U(x) = s$$

to a claim about its value in a challenge point

$$U(r) = s'$$

We will use the sumcheck to reduce the claim of the form $\langle P, Q\rangle$ to evaluation claims of individual multilinear polynomials P and Q.

Our protocol will also have the form of claim reduction: we will avoid committing to I^*T directly and instead we will commit to data that allows us to evaluate it in a point r.

2.2 Pullback and pushforward

We will use the language of pullbacks and pushforwards. Let A, B be two field-valued functions on the sets $\mathbf{n} = \{0...n-1\}$ and $\mathbf{m} = \{0...m-1\}$, respectively, and let I be a mapping $I : \mathbf{n} \longrightarrow \mathbf{m}$.

Then, as already said before, a pullback of B along I is defined as:

$$I^*B[i] = B[I[i]]$$

We define a dual notion, called pushforward:

$$I_*A[j] = \sum_{i | I[i]=j} A[i]$$

Lemma 1. ***Duality.***

$$\langle I_*A, B\rangle = \langle A, I^*B\rangle$$

Proof. Expanding the definitions,

$$\langle I_*A, B\rangle = \sum_{0\le j<m} I_*A[j]B[j] = \sum_{0\le j<m} \sum_{i|I[i]=j} A[i]B[j] =$$

$$= \sum_{0\le i<n} \sum_{j=I[i]} A[i]B[j] = \sum_{0\le i<n} A[i]B[I[i]] = \langle A, I^*B\rangle$$

□

Another view of this duality is that pullback I^*B is the application of matrix of pairs $(i, I[i])$ to B, and pushforward is an application of transposed matrix to A.

3 Trading pullback on pushforward

Duality of pullback and pushforward sometimes allows replacing claims about pullbacks with claims about pushforwards, or vice versa. Notable example of this strategy is SPARK protocol, which trades pushforward on pullback (though authors didn't call it this way).

We want to go in the opposite direction. Assume that the prover had already committed the indexing array I, and the table T, and we are now in the situation where $I^*T(r)$ needs to be computed. Then, we will run the following protocol.

1. Claim evaluation value $I^*T(r) = e$.
2. Commit $I_*\mathsf{eq}_r$.
3. Notice that $I^*T(r) = \langle I^*T, \mathsf{eq}_r\rangle = \langle T, I_*\mathsf{eq}_r\rangle$.
4. Run the sumcheck for the claim $\langle T, I_*\mathsf{eq}_r\rangle = e$ and obtain claims $T(r') = c$, $I_*\mathsf{eq}_r(r') = c'$.
5. Open T and $I_*\mathsf{eq}_r$.

This protocol proves the evaluation claim of $I^*T(r)$, as long as $I_*\mathsf{eq}_r$ was committed correctly.

4 Pushforward proof with logup*

Assume that I, T, X, Y are committed and Y is claimed to be pushforward $Y = I_*X$.

Consider the following equality for a random challenge c:

$$\sum_{0\leq i<n} \frac{X[i]}{c - I[i]} = \sum_{0\leq j<m} \frac{Y[j]}{c - j}$$

Lemma 2. *We claim that unless Y actually is a pushforward of X, this equality holds with negligible probability (soundness error $\leq \frac{n+m}{|\mathbb{F}|}$) for a random challenge c.*

Proof. The proof is essentially the same as for normal logup (moreover, normal logup is related as $\mathsf{acc} = I_*1$). Consider the value

$$\sum_{0\leq i<n} \frac{X[i]}{c - I[i]} - \sum_{0\leq j<m} \frac{Y[j]}{c - j}$$

as a rational function in variable c. It is easy to see by direct inspection that this function is nonzero unless $Y = I_* X$ (say, by counting coefficient of the pole in each j on lhs and rhs).

It also has a degree $n + m$, which means that unless random challenge c hits the root of this function, the argument will correctly catch wrong Y. $\square$

In our case we will use $X = \mathsf{eq}_r$. Prover commits to I, T and $I_* T$ (but not eq_r, as verifier is capable of evaluating it by itself), and validates

$$\sum_{0 \leq i < n} \frac{\mathsf{eq}_r[i]}{c - I[i]} = \sum_{0 \leq j < m} \frac{I_* \mathsf{eq}_r[j]}{c - j}$$

This (if the GKR protocol from [5] is used) leads to evaluation claims of $I, T, \mathsf{eq}_r, I_* \mathsf{eq}_r$.

Of these, evaluation claim of eq_r can be computed by verifier itself, and evaluation claims about $I_* \mathsf{eq}_r$ and T can be merged with claims obtained in the sumcheck from the previous section $\langle T, I_* \mathsf{eq}_r \rangle - e$.

5 Performance considerations and applications

In a small-table regime, $m << n$, the added cost of logup* is three sumchecks of degree 2 and size m (one for $\langle T, I_* \mathsf{eq}_r \rangle$, and two for claim reductions about T and eq_r respectively).

The advantage is the absence of a commitment of size n, moreover, *large-field* commitment. For example, if $\mathbb{F}$ is the 5-th extension of 31-bit field (enough to achieve 128-bit level security of logup), this commitment increases the cost of logup $6\times$.

So, this transition is very justified.

Note: one might wonder, in which cases the table T actually is large-valued. This occurs not only in SPARK protocol, but in Lasso too - while individual tables are, typically, small-valued, one tends to do a vectorized lookup, which also has a similar if not worse cost profile.

The main application, as we see, is the extension of sparsity-related arguments, such as Twist and Shout [7] to hash-based commitment schemes: while elliptic curve based commitments have both more efficient ways of committing to I^*T and other ways of dealing with sparsity, hash-based commitment schemes previously had much worse performance for sparse polynomials. Our method is field and commitment scheme agnostic, so this tension is finally solved.

Additional application is that we automatically solve (for an indexed lookup) the overflow problem that normal logup argument had for accumulators, which is complementary to [1].

References

[1] Liam Eagen and Ulrich Haböck. Bypassing the characteristic bound in logUp. Cryptology ePrint Archive, Paper 2024/2067, 2024.

[2] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[3] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.

[4] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023.

[5] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023.

[6] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550, 2019.

[7] Srinath Setty and Justin Thaler. Twist and shout: Faster memory checking arguments via one-hot addressing and increments. Cryptology ePrint Archive, Paper 2025/105, 2025.

[8] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. Cryptology ePrint Archive, Paper 2023/1216, 2023.

[9] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. Cryptology ePrint Archive, Paper 2013/351, 2013. https://eprint.iacr.org/2013/351.

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