Efficient KZG-based Univariate Sum-check and Lookup Argument

Yuncong Zhang[0000−0001−8826−5844]1, Shi-Feng Sun[0000−0003−3742−5275]1⋆, and Dawu Gu[0000−0002−0504−9538]1,2⋆

1. Shanghai Jiao Tong University, China
{shjdzhangyuncong,shifeng.sun,dwgu}@sjtu.edu.cn
2. Shanghai Jiao Tong University (Wuxi) Blockchain Advanced Research Center

Abstract. We propose a novel KZG-based sum-check scheme, dubbed Losum, with *optimal* efficiency. Particularly, its proving cost is *one* multi-scalar-multiplication of size k—the number of non-zero entries in the vector, its verification cost is *one* pairing plus one group scalar multiplication, and the proof consists of only *one* group element.
Using Losum as a component, we then construct a new lookup argument, named Locq, which enjoys a smaller proof size and a lower verification cost compared to the state of the arts cq, cq+ and cq++. Specifically, the proving cost of Locq is comparable to cq, keeping the advantage that the proving cost is independent of the table size after preprocessing. For verification, Locq costs four pairings, while cq, cq+ and cq++ require five, five and six pairings, respectively. For proof size, a Locq proof consists of four $\mathbb{G}_1$ elements and one $\mathbb{G}_2$ element; when instantiated with the BLS12-381 curve, the proof size of Locq is 2304 bits, while cq, cq+ and cq++ have 3840, 3328 and 2944 bits, respectively. Moreover, Locq is zero-knowledge as cq+ and cq++, whereas cq is not. Locq is more efficient even compared to the non-zero-knowledge (and more efficient) versions of cq+ and cq++.

Keywords: Sum-check Scheme, Lookup Argument, zkSNARK

1 Introduction

Lookup arguments [GW20,EFG22] are protocols that allow a prover to convince a verifier that all the elements in a *lookup vector* $\boldsymbol{v}$ appear in another vector $\boldsymbol{t}$ called the *table*, where the verifier holds only the commitments to these vectors. Lookup arguments are usually *succinct*, i.e., the running time of the verifier is sublinear to the vector sizes.

Lookup argument has been widely used to improve SNARKs [PFM+22, CBBZ22] and is one of the key reasons for the recent rapid development of Zero-Knowledge Virtual Machines (ZKVMs) [Ris22, VM22, Mid22]. Before the introduction of lookup arguments, it is very expensive to prove 32- to 256-bit boolean

⋆ Corresponding author.

operations using SNARKs [Gro16, CHM+20, COS, Set20] as it is costly to simulate them in arithmetic circuits; thus they are considered *SNARK-unfriendly*. Lookup arguments mitigate this issue by transforming the SNARK-unfriendly operations into lookups in tables. For example, the expensive-to-verify statement "a op $b = c$" is replaced by "(a, b, c) is a row of T_{op}" that is efficiently handled by lookup arguments, where T_{op} is a table whose rows range over of all possible values of (a, b, c) that are valid input-output combinations, and "op" can be arbitrary operation.

However, the table sizes for the lookup arguments can be huge for operations with even small input sizes. For example, the 16-bit XOR operation has a lookup table whose number of rows is as large as 2^{32}. Therefore, recent works have focused on constructing lookup arguments whose proving cost is sublinear or even independent of the table size [ZBK+22, PK22, GK22, ZGK+22, EFG22]. All these works use KZG [KZG10] as the underlying polynomial commitment scheme, thus enjoying constant proof size (constant number of group and field elements) and verification costs (constant number of pairing checks). As the state of the art in this line of works, Eagen et al. [EFG22] proposed cq, which, for the first time, reduces the proving cost to $O(m)$ group operations plus $O(m \log m)$ field operations where m is the size of the lookup vector. Meanwhile, compared to the previous works, the verification cost of cq is comparable (five pairings), and the proof size is the smallest. As its core technique, cq uses the *logarithmic derivative* [Hab22] method to reduce the lookup argument to the *univariate sum-check* [BCR+19]. To eliminate the dependence of the proving cost on the table size, cq proposes the *cached quotient* method that shifts the majority of the prover work to the preprocessing phase. Despite the performance improvements, cq does not consider zero-knowledge.

Very recently[1], following the framework of cq, Companelli et al. [CFF+23] propose cq+ and cq++ that improve the proof size of cq. More specifically, these schemes use the univariate sum-check [BCR+19] and the cached quotient technique as cq does, and come with the zero-knowledge property with small overheads—the zero-knowledge versions have one more group element in proof sizes compared to the non-zero-knowledge version. In addition, they propose zkcq+ that achieves *full zero-knowledge*, which further conceals the table content from the verifier[2], at the cost of two more group elements in the proof.

Because of the wide applications of lookup arguments in building SNARKs deployed in blockchain-based cryptocurrencies [zkS22, GPR21], any small efficiency improvements to lookup arguments may bring significant financial benefits in practice. In this work, we make further progress in this line of work on table-size independent lookup arguments. Our contributions are summarized in Sect 1.1.

---

[1] Concurrent to this work.

[2] This requires the table has been randomized by a mask when computing its commitment, before putting it into the lookup argument.

Table 1. Locq has a comparable proving cost, a smaller proof size, and a lower verification cost, compared to the state of the arts in the line of KZG-based lookup arguments with sublinear-or-zero independence on table size. Here m is the lookup vector size, N is the table size, P stands for "pairing check". **Prep.** stands for "Preprocessing Cost". **Vrf.** stands for "Verification Cost". The $\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2$ in "Proof Size" refer to field/group elements, and in "Preprocessing" and "Proving Cost" they refer to field/group operations. "#Bits" is the bit size of the proof when instantiated with BLS12-381, for which a $\mathbb{G}_1$ element takes 384 bits, a $\mathbb{G}_2$ element takes 768 bits, and a $\mathbb{F}$ element takes 256 bits. **Hom.** means this lookup argument supports multi-column table lookups by homomorphically combining the columns using random linear combination. **ZK.** stands for zero-knowledge, where ✓ means it is zero-knowledge only if the table polynomial $t(X)$ is not considered secret, while ✓✓ means the protocol is *full zero-knowledge*, i.e., it also works when $t(X)$ is masked by a random $\rho \cdot Z_{\mathbb{H}}(X)$, in which case the verifier learns no information of $t(X)$, either. The "*" stands for the zero-knowledge versions of cq+ or cq++.

Scheme	Prep.	Proof Size				Proving Cost	Vrf.	Hom.	ZK.
	$\mathbb{F}+\mathbb{G}_1$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{F}$	#Bits				
Caulk [ZBK+22]	$O(N\log N)$	14	1	4	7168	$O(m^2+m\log(N))(\mathbb{F}+\mathbb{G}_1)$	$4P$	✓	✓✓
Caulk+ [PK22]	$O(N\log N)$	7	1	2	3968	$O(m^2)(\mathbb{F}+\mathbb{G}_1)$	$3P$	✓	✓✓
Flookup [GK22]	$O(N\log^2 N)$	6	1	4	4096	$6m\mathbb{G}_1+m\mathbb{G}_2+O(m\log^2 m)\mathbb{F}$	$3P$	×	×
Baloo [ZGK+22]	$O(N\log N)$	12	1	4	6400	$13m\mathbb{G}_1+m\mathbb{G}_2+O(m\log^2 m)\mathbb{F}$	$5P$	✓	×
cq [EFG22]	$O(N\log N)$	8	-	3	3840	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$5P$	✓	×
cq+ [CFF+23]	$O(N\log N)$	7	-	1	2944	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$5P$	✓	×
cq++ [CFF+23]	$O(N\log N)$	6	-	1	2560	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$6P$	✓	×
cq+* [CFF+23]	$O(N\log N)$	8	-	1	3328	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$5P$	✓	✓
cq++* [CFF+23]	$O(N\log N)$	7	-	1	2944	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$6P$	✓	✓
zkcq+ [CFF+23]	$O(N\log N)$	9	-	1	3712	$8m\mathbb{G}_1+O(m\log m)\mathbb{F}$	$6P$	✓	✓✓
Locq (This work)	$O(N\log N)$	4	1	-	2304	$6m\mathbb{G}_1+m\mathbb{G}_2+O(m\log m)\mathbb{F}$	$4P$	✓	✓✓

1.1 Contributions

We put forward a novel KZG-based zero-knowledge lookup argument, Locq, with the proving cost independent of the table size and with smaller proof size and verification cost, compared to the state of the arts cq and even the non-zero-knowledge versions of cq+ and cq++. As a core component of Locq, we introduce a new KZG-based univariate sum-check, Losum, with optimal proving cost, verification cost and proof size. Our main contributions are summarized as follows.

- We propose a more efficient univariate sum-check scheme called Losum, which improves the existing univariate sum-check protocol [BCR+19] in the KZG setting. The cost of Losum is (arguably) optimal for KZG-based sum-checks: (1) the proving cost is a single multi-scalar-multiplication (MSM) of size k in $\mathbb{G}_1$—the first group of the pairing scheme, where k is the number of non-zero entries in the input vector; (2) the verification cost is one pairing and one scalar multiplication[3] in $\mathbb{G}_1$; and (3) the proof size is a single $\mathbb{G}_1$ element.
- We then use Losum as a building block to construct a new lookup argument, named Locq. Our new lookup argument keeps the property that the proving

[3] The cost of one scalar multiplication can be ignored compared to the pairing.

cost is independent of the table size. Moreover, it has a smaller proof size ($4\mathbb{G}_1 + 1\mathbb{G}_2$, compared to $8\mathbb{G}_1 + 3\mathbb{F}$ for cq and $6\mathbb{G}_1 + \mathbb{F}$ for the non-zero-knowledge version of cq++) and a smaller verification cost (four pairings checks compared to five in cq and cq+ and six in cq++). Moreover, our scheme enjoys full zero-knowledge as zkcq+. The zero-knowledge property in our scheme is achieved with almost no additional cost, because Locq (a) does not contain any field element in the proof, so adding zero-knowledge is as simple as adding random masks to the committed polynomials; and (b) does not involve any degree check as in cq, cq+ or cq++, thanks to using Losum instead of the traditional univariate sum-check [BCR+19]. More detailed comparisons of this work with existing works on KZG-based table-size-independent lookup arguments are shown in Table 1.

In addition, both Losum and Locq enjoy the property that they can work for *arbitrary* field $\mathbb{F}$.[4] In practice, this allows Losum and Locq to choose from much wider candidates for $\mathbb{F}$ that may enjoy better optimization techniques. Particularly, Losum and Locq work even when $|\mathbb{F}| - 1$ is not *smooth*—has a large power-of-two factor, which is required by all prior schemes because they only work for $\mathbb{F}$ with large smooth multiplicative subgroups. More precisely, the smooth multiplicative subgroups still benefit Losum and Locq, but only in (prover-side) efficiency. Specifically, without such subgroups, the complexity of the preprocessing cost of both Losum and Locq would increase from $O(N \log N)$ to $O(N \log^2 N)$; the number of field operations in the proving cost of Locq would increase from $O(m \log m)$ to $O(m \log^2 m)$; and everything else is not affected. In comparison, all prior schemes completely stop working *unless* $\mathbb{F}$ has large smooth multiplicative subgroups.

We remark that our Losum and Locq require additional trusted setups besides that of KZG. This slight disadvantage is acceptable in real-world scenarios, as the setup is only executed once for each different *vector size*. This reliance on the trusted party is much weaker than the application-specific trusted setup of Groth16, which is still widely used in practice. Moreover, the setup for Locq is only executed *once* for each different *table size* and can be reused for all sizes of lookup vectors. Therefore, in practice, the setups of Losum and Locq can be accomplished together with that of KZG to avoid additional invocation of the trusted third party.

1.2 Technical Overview

For a quick understanding of our work, we give a high-level explanation about how we achieve the smaller proof size and verification cost in both Losum and Locq.

Univariate sum-check. Let $\mathbb{F}$ be a finite field whose size is a large prime. Given a commitment to a polynomial $f(X)$ and a domain $\mathbb{H} \subset \mathbb{F}$, the goal of the

[4] As long as $\mathbb{F}$ is sufficiently large, as required by all succinct arguments.

univariate sum-check protocol is to prove to a succinct verifier that the sum of the evaluations of $f(X)$ over $\mathbb{H}$ is $s \in \mathbb{F}$, i.e., $\sum_{h\in\mathbb{H}} f(h) = s$. Our new sum-check method is based on the following observation: for the special case where $s = 0$, which we refer to as the *zero sum-check*, the set of all the polynomials that satisfy the requirement forms a linear space, and thus can be represented as the linear combination of a set of basis polynomials, denoted by $c_1(X), \cdots, c_d(X)$, where d is the dimension of this linear space and is determined by the degree bound on $f(X)$. Therefore, the zero sum-check on $f(X)$ is equivalent to proving that there exist coefficients $b_1, \cdots, b_d \in \mathbb{F}$ such that $f(X) = \sum_{i=1}^{d} b_i c_i(X)$. For a concrete choice of basis polynomials, please refer to Sect. 3.

When $f(X)$ is committed using the KZG scheme, this relation of linear combination is easy to prove using the following technique, which is heavily used in Groth16 [Gro16]. In the setup phase, we sample a secret random α, and precompute the commitments to $\alpha \cdot c_i(X)$ for every i. In the online phase, the prover computes the commitment to $f'(X) := \alpha \cdot f(X)$ by linearly combining the precomputed commitments to $\alpha \cdot c_i(X)$, which is only possible when $f(X)$ is a linear combination of $c_i(X)$ since the prover does not know α. The proof thus contains a single group element, which is the commitment to $f'(X)$. The verification costs a single pairing that checks the relation $f(X) \cdot \alpha = f'(X) \cdot 1$, and no additional group scalar multiplication is needed. By properly choosing the basis polynomials $c_i(X)$, computing the coefficients b_i brings no cost at all, so the total prover cost is a single multi-scalar multiplication (MSM) for computing the commitment to $f'(X)$.

For the case where $s \neq 0$, we pick a polynomial $\ell(X)$ that trivially satisfies $\sum_{h\in\mathbb{H}} \ell(h) = 1$, and apply the above technique to $f(X) - s \cdot \ell(X)$ instead. The prover cost is still a single MSM, the proof size is still one group element, and the only additional verification cost besides the pairing is one group scalar multiplication for computing the commitment to $f(X) - s \cdot \ell(X)$.

Table-size independent lookup argument. We then use this sum-check scheme to construct a lookup argument. Given the commitments to two polynomials $f(X)$ and $t(X)$ and two domains $\mathbb{D}, \mathbb{H} \subset \mathbb{F}$, where $\mathbb{D}$ is a subset of $\mathbb{H}$ and $|\mathbb{D}|$ is far smaller than $|\mathbb{H}|$, the goal of the lookup argument is to prove that for every $u \in \mathbb{D}$, there exists $h \in \mathbb{H}$ such that $f(u) = t(h)$.

Our work follows the framework of cq by exploiting the following statement equivalent to the one to prove by the lookup argument: there exists a polynomial $m(X)$, such that $\sum_{u\in\mathbb{D}} \frac{1}{X-f(u)} = \sum_{h\in\mathbb{H}} \frac{m(h)}{X-t(h)}$. Intuitively, this equality implies that every hole (position where the equation evaluates to infinity) in the left function is also a hole in the right function, so $f(u)$ must equal some $t(h)$.

Using the Schwartz-Zippel Lemma, this equation is checked at a random β sampled by the verifier, and the equality between two fractional functions is transformed to a sum-check for the polynomials $g(X)-w(X)$, where $g(X), w(X)$ are prover-committed polynomials that satisfy $g(u) = \frac{1}{\beta-f(u)}$ for $h \in \mathbb{D}$ and $g(h) = 0$ for $h \in \mathbb{H}\backslash\mathbb{D}$, and $w(h) = \frac{m(h)}{\beta-t(h)}$ for every $h \in \mathbb{H}$. Note that here Locq differs from cq, as cq does not require $g(h) = 0$ for $h \in \mathbb{H}\backslash\mathbb{D}$, so cq executes two sum-checks for $g(X)$ and $w(X)$ respectively over $\mathbb{D}$ and $\mathbb{H}$, whereas Locq only uses one sum-check.

We use Losum for the zero sum-check on $g(X)-w(X)$. It remains to prove the correctness of $g(X)$ and $w(X)$. We accomplish this by exploiting their definitions—it suffices for the prover show that both $g(X)\cdot(\beta-f(X))-U(X)$ and $w(X)\cdot(\beta-t(X))-m(X)$ are divided by $Z_{\mathbb{H}}(X)$, where $U(X)$ is the polynomial that evaluates to 1 over $\mathbb{D}$ and 0 over $\mathbb{H}\backslash\mathbb{D}$, and $Z_{\mathbb{H}}(X) := \prod_{h\in\mathbb{H}}(X-h)$ is the *vanishing polynomial* over $\mathbb{H}$.

To prove this divisibility, the prover computes $q_1(X)$ and $q_2(X)$ by dividing the two polynomials with $Z_{\mathbb{H}}(X)$, respectively. By random linear combination, the two divisibility checks can be merged into one, and the prover only needs to commit to a single quotient polynomial $q(X)$.

Note that all the polynomials involved in computing $q(X)$ are of degree $O(|\mathbb{H}|)$. To reduce the prover cost from $O(|\mathbb{H}|)$ to $O(|\mathbb{D}|)$, making the lookup argument table-size independent, we note that $g(X)$, $U(X)$ and $Z_{\mathbb{H}}(X)$ are all divided by $Z_{\mathbb{H}}(X)/Z_{\mathbb{D}}(X)$, so $q_1(X)$ can be computed by dividing two polynomials of degree $O(|\mathbb{D}|)$ instead. Then we apply the *cached quotient* technique that is the same as in cq to compute the commitment to $q_2(X)$. The details will be explained in Sect 4.

Applying all the techniques above, our lookup argument proof consists of only five group elements, i.e., the polynomial commitments for $m(X), g(X), w(X), q(X)$, and the proof of Losum.

1.3 Related Works

The concept of lookup argument is initially introduced by Gabizon et al. in Plookup [GW20], though the related ideas have been demonstrated in some earlier works [BEG+91, BCG+18]. Since lookup arguments are particularly useful for proving SNARK-unfriendly relations, i.e., relations that are expensive to express as arithmetic computations, they have been extensively used to boost the performance of SNARKs [PFM+22, CBBZ22]. Moreover, they work as an indispensable component in the recent popular ZKVM projects [Mid22, VM22, Ris22, zkS22, Scr22]. Lookup argument has been extensively studied since its introduction and researchers have focused on improving its efficiency.

Starting from Caulk [ZBK+22], whose proving cost relies on the table size logarithmically, there is a line of follow-up works that assume the table is much larger than the lookup vector, and focus on achieving table-size-independent proving cost, including Caulk+ [PK22], Flookup [GK22], Baloo [ZGK+22], cq [EFG22], and cq+, cq++, zkcq+ by Campanelli et al. [CFF+23]. These schemes can be used to prove lookups for large tables such as the 32-bit range check and 16-bit boolean operations, where the tables are fixed and can be preprocessed offline. Among these schemes, Flookup, Baloo and cq are not zero-knowledge. The latest cq, cq+ and cq++ are the state of the arts in this line as they have the smallest asymptotic proving cost. Meanwhile, their verifications cost 2 or 3 more pairings compared to Caulk+ and Flookup. Our work follows this line of research to further reduce the proof size and verification cost, and achieves zero-knowledge.

The recently proposed Lasso [STW23] works with huge tables with exponential sizes, but requires that the table is structured, i.e., is a generalized tensor product of smaller (size close to the lookup vector) tables.

Other lookup arguments assume the table size is close to that of the lookup vector, just as Plookup, so these works can be used in cases where the table is dynamically generated and committed by the prover online. These schemes include mvlookup [Hab22], Tip5 [SLST23], and the lookup argument inside the HyperPlonk SNARK [CBBZ22]. The mvlookup scheme proposes a powerful technique called *logarithmic derivative*, which reduces the lookup argument into a sum-check [LFKN90] statement. This technique is then adapted to the univariate case by cq and Tip5, whereas cq uses the *univariate sum-check* [BCR+19] and Tip5 uses the running sum vector for doing the sum-check. Both the multivariate and univariate sum-checks are widely used in constructing succinct argument systems [GKR08, BCR+19, CHM+20, COS, ZXZS20, XZZ+19].

2 Preliminaries

Let λ denote the security parameter. Let p be a prime of λ bits. Let $\mathbb{F} = \mathbb{F}_p$ be the prime field of size p. Let $\boldsymbol{v} \in \mathbb{F}^N$ be a vector of size N. Let $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ be the inner product between the two vectors. We say a probabilistic algorithm is *PPT* if it runs in polynomial time.

2.1 Bilinear Pairing

A bilinear pairing is a tuple $\mathsf{bp} = (p, e, g, h, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ where p is a prime, $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$, g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$. The function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map that satisfies:

- $e(g, h)$ is a generator of $\mathbb{G}_T$.
- $e(g^a, h^b) = e(g, h)^{ab}$ for every $a, b \in \mathbb{Z}_p$.

For any $x \in \mathbb{F}_p$, let $[x]_1$ denote g^x, $[x]_2$ denote h^x and $[x]_T$ denote $e(g, h)^x$. Based on this notation, we use the addition notation for group operations, i.e., $[x]_1 + [y]_1 = [x + y]_1$ and $c \cdot [x]_1 = [c \cdot x]_1$.

This work assumes that q-DLOG problem is hard for the bilinear pairing groups, i.e., no PPT adversaries can solve this problem with more than negligible probability.

Definition 1 (q-DLOG Problem). *Let x be uniformly randomly sampled from $\mathbb{F}$. On input the bilinear pairing parameters* bp *and group elements* $[1]_1$, $[x]_1$, $[x^2]_1$, $\cdots$, $[x^q]_1$, $[1]_2$, $[x]_2$, $\cdots$, $[x^q]_2$*, find x.*

Assuming q-DLOG is hard on the given bilinear pairing groups, it is easy to see that the following problem is also hard: given $[x^{-1}]_1$, $[1]_1$, $[x]_1$, $[x^{-1}]_2$, $[1]_2$, $[x]_2$, to find x. To see why this is hard, assume we have an algorithm that solves this problem efficiently, we show that we can solve q-DLOG for $q = 2$, i.e., given any $[1]_1, [x]_1, [x^2]_1, [1]_2, [x]_2, [x^2]_2$, find x. To accomplish this, we choose new generators $g' = [x]_1$ and $h' = [x]_2$, and let $\mathsf{bp}' := (p, e, g', h', \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. Using this new set of parameters, the inputs are written as $[x^{-1}]_1$, $[1]_1$, $[x]_1$, $[x^{-1}]_2$, $[1]_2$ and $[x]_2$, which can be solved efficiently by our assumption, leading to contradiction.

2.2 The KZG Polynomial Commitment

The KZG-polynomial commitment [KZG10] is constructed based on bilinear pairings. This work only involves the setup and committing algorithms of KZG and does not use the opening or verification algorithms, because all the polynomial equations in our scheme are checked by the *ideal check* introduced later in Sect. 2.4, without evaluating any polynomials. Therefore, we recall the setup and committing algorithms of KZG as follows.

- $\mathsf{Setup}(D_1, D_2) \rightarrow \sigma_{KZG}$: Given the bilinear pairing parameters bp and the degree bounds D_1, D_2, uniformly sample $x \in \mathbb{F}$ and output $\sigma_{KZG} = ([1]_1, [x]_1, \cdots, [x^{D_1}]_1, [1]_2, [x]_2, \cdots, [x^{D_2}]_2)$.
- $\mathsf{Commit}(\sigma_{KZG}, f(X), b) \rightarrow \mathsf{cm}$: Given the setup parameters σ_{KZG}, polynomial $f(X) = f_0 + f_1 X + \cdots + f_d X^d$, and $b \in \{1, 2\}$ indicating which group this polynomial is committed in, check that $d \leq D_b$, and output $\mathsf{cm} = [f(x)]_b = f_0 \cdot [1]_b + \cdots + f_d \cdot [x^d]_b$.

2.3 Polynomials and Lagrange Basis

Let $\mathbb{H}$ be a subset of $\mathbb{F}$ with $|\mathbb{H}| = N$. The Lagrange basis polynomials over $\mathbb{H}$ are the set of polynomials $\{L_h(X)\}_{h\in\mathbb{H}}$ where $L_h(h) = 1$ and $L_h(h') = 0$ for any $h' \in \mathbb{H}\backslash\{h\}$. For any polynomial $f(X)$ of degree less than N, $f(X)$ can be uniquely represented as a linear combination of the Lagrange basis, i.e., $f(X) = \sum_{h\in\mathbb{H}} f(h)L_h(X)$. We call $Z_{\mathbb{H}}(X) := \prod_{h\in\mathbb{H}}(X-h)$ the *vanishing polynomial* over $\mathbb{H}$. Assuming an implicit ordering over the elements in $\mathbb{H}$, we let $f(\mathbb{H})$ denote either the vector $(f(h))_{h\in\mathbb{H}}$ or the set $\{f(h)\}_{h\in\mathbb{H}}$, depending on the context. For any two polynomials $f(X)$ and $g(X)$, the vectors $f(\mathbb{H}) = g(\mathbb{H})$ if and only if $f(X) - g(X)$ can be divided by $Z_{\mathbb{H}}(X)$.

We will use the univariate version of the Schwartz-Zippel Lemma about polynomials.

Lemma 1 (Schwartz-Zippel). *Let $f(X)$ be a univariate polynomial of degree d over $\mathbb{F}$, S be a finite subset of $\mathbb{F}$, and z be selected randomly and uniformly from S. Then*

$$\Pr[f(z) = 0] \leq \frac{d}{|S|}.$$

Schwartz-Zippel Lemma can be extended to rational functions of the form $f(X)/g(X)$, in which case the probability is bounded by $\frac{d}{|S|}$ where $d := \max\{\deg(f(X)), \deg(g(X))\}$.

2.4 Algebraic Group Model

The algebraic group model (AGM) [FKL17], introduced by Fuchsbauer et al., is widely used to prove the security of protocols and schemes that involve elliptic curve groups. In this model, it is required that whenever the adversary A outputs an element a in $\mathbb{G}_i$ for $i \in \{1, 2\}$, A must simultaneously output a vector $\boldsymbol{s} \in \mathbb{F}_p^\ell$ such that $\langle \boldsymbol{s}, \boldsymbol{t} \rangle = a$, where $\boldsymbol{t} \in \mathbb{G}_i^\ell$ is the collection of all $\mathbb{G}_i$ elements that A has received. We say such an adversary is *algebraic*.

Assuming q-DLOG is hard, the bilinear pairing check becomes computationally equivalent to checking quadratic relations on polynomials, explained as follows.

Suppose the adversary is given inputs $[1]_1, [x]_1, \cdots, [x^q]_1$ and $[1]_2, [x]_2, \cdots, [x^q]_2$, where x is secretly uniformly chosen from $\mathbb{F}$, and outputs $a, b \in \mathbb{G}_1$ and $c, d \in \mathbb{G}_2$ such that $e(a, c) = e(b, d)$. Being algebraic, the adversary simultaneously outputs the coefficients $a_0, \cdots, a_q \in \mathbb{F}$, such that $a = \sum_{i=0}^{q} a_i [x^i]_1$ (similar for b, c, d).

Let $f_a(X) = a_0 + a_1 X + \cdots + a_q X^q$, then by definition $a = [f_a(x)]_1$ (similar for $f_b(X), f_c(X)$ and $f_d(X)$). For convenience, we say whenever the adversary outputs $a \in \mathbb{G}_1$, it *simultaneously outputs the polynomial* $f_a(X)$ such that $a = [f_a(x)]_1$.

Since $e(a, c) = e(b, d)$, we have $f_a(x) f_c(x) = f_b(x) f_d(x)$. Now we claim that the polynomial equation $f_a(X) f_c(X) = f_b(X) f_d(X)$ also holds. Otherwise, the polynomial $f_a(X) f_c(X) - f_b(X) f_d(X)$ would be a non-zero polynomial that has a root at x. Then by computing the at most $2q$ roots of this polynomial, we restrict x to $2q$ candidates and then solve the q-DLOG problem by brute force. Therefore, the hardness of q-DLOG problem implies the polynomial equation $f_a(X) f_c(X) = f_b(X) f_d(X)$.

The same argument can be extended to any quadratic relations on the polynomials. This technique is heavily exploited by the line of works from Caulk to cq, where the pairing check on the group elements is referred to as the *real pairing check*, and the implied polynomial equation is referred to as the *ideal check*.

2.5 Argument of Knowledge

An argument of knowledge Π is a protocol involving two parties, a prover and a verifier. In general, it consists of three algorithms, namely Setup, Prove and Verify, and allows the prover to convince the verifier that given a string x it knows a witness w such that (x, w) is in an NP relation.

Definition 2. *An argument of knowledge for an indexed family of NP relations* $\{\mathcal{R}_{ind}\}_{ind \in \mathcal{I}}$ *is a triple of algorithms* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *with the following syntax:*

– $\mathsf{Setup}(ind, 1^\lambda) \to (\sigma, \sigma_V, \tau)$*: in the offline phase,* Setup *is given the index ind and* 1^λ *outputs a common reference string (SRS) denoted by* σ*, a verifier SRS* σ_V*, and a trapdoor* τ *that is optional for zero-knowledge.*

– $\langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma_V, x)\rangle \to b$: *in the online phase,* Prove *receives input* σ, σ_V *and a pair of* $(x, w) \in \mathcal{R}_{ind}$*, and* Verify *receives input* σ_V *and* x*. The parties interact with each other. Finally,* Verify *outputs* 0 *or* 1*.*

The algorithms should satisfy the following security requirements.

– Completeness. *For any* $ind \in \mathcal{I}$ *and* $(x, w) \in \mathcal{R}_{ind}$*,*

$$\Pr\left[b = 1 \,\middle|\, \begin{array}{l}\mathsf{Setup}(ind, 1^\lambda) \to (\sigma, \sigma_V, \tau)\\ \langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma_V, x)\rangle \to b\end{array}\right] = 1.$$

– Argument-of-Knowledge. *For any PPT algorithm* Prove^**, there exists a PPT extractor* E *such that for any* $ind \in \mathcal{I}$ *and auxiliary inputs* $z \in \{0,1\}^*$

$$\Pr\left[b = 1 \wedge (x, w) \notin \mathcal{R}_{ind} \,\middle|\, \begin{array}{l}\mathsf{Setup}(ind, 1^\lambda) \to (\sigma, \sigma_V, \tau)\\ \mathsf{Prove}^*(\sigma, z, \perp; r) \to (x, \mathsf{st})\\ \langle \mathsf{Prove}^*(\sigma, z, \mathsf{st}; r), \mathsf{Verify}(\sigma_V, x)\rangle \to b\\ \mathsf{E}(\sigma, z; r) \to w\end{array}\right] = \mathsf{negl}.$$

An argument of knowledge may optionally be:

– *Non-interactive*: The entire interaction consists of a single message π from the prover to the verifier.
– *Public-coin*: All the messages sent from the verifier are fresh random coins. In this case, the argument of knowledge can be transformed into a non-interactive protocol by the Fiat-Shamir transformation [FS86].
– *Succinct*: The communication cost is sublinear to the witness size, and the verification cost is sublinear to the cost of verifying $(x, w) \in \mathcal{R}_{ind}$ using its standard NP verification.
– *Zero-Knowledge*: Let $tr\langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma_V, x)\rangle$ denote the *transcript* of an execution, i.e., all the messages exchanged during the interaction. There exists a PPT simulator S such that for any $ind \in \mathcal{I}$, $(x, w) \in \mathcal{R}_{ind}$, the following two distributions have negligible statistical distance

$$\left\{\begin{array}{l} tr\langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma_V, x)\rangle : \\ \mathsf{Setup}(ind, 1^\lambda) \to (\sigma, \sigma_V, \tau)\end{array}\right\} \approx_s \left\{\begin{array}{l}\mathsf{S}(\sigma_V, x, \tau) : \\ \mathsf{Setup}(ind, 1^\lambda) \to (\sigma, \sigma_V, \tau)\end{array}\right\}.$$

Univariate sum-checks (in the KZG-setting) are a class of arguments of knowledge for the following relation, indexed by the bilinear pairing parameters, the domain $\mathbb{H} \subset \mathbb{F}$ and the KZG setup parameters $\sigma_{KZG} := (\{[x^i]_1\}_{i=0}^D, \{[1]_2, [x]_2\})$.

$$\mathcal{R}_{\mathsf{Sum}} := \left\{((c, s), f(X)) : c = [f(x)]_1, \sum_{h \in \mathbb{H}} f(h) = s\right\}_{\mathsf{bp}, \mathbb{H}, \sigma_{KZG}}. \quad (1)$$

Lookup arguments are a class of arguments of knowledge for the following relation, indexed by the bilinear pairing parameters, the domains $\mathbb{D}, \mathbb{H} \subset \mathbb{F}$ and the KZG setup parameters $\sigma_{KZG} := (\{[x^i]_1\}_{i=0}^D, \{[x^i]_2\}_{i=0}^D)$:

$$\mathcal{R}_{\mathsf{Lookup}} := \left\{\left(\begin{array}{c}(c_f, c_t),\\ (f(X), t(X))\end{array}\right) \,\middle|\, \begin{array}{l} c_f = [f(x)]_1, c_t = [t(x)]_2,\\ \deg(f(X)) < |\mathbb{D}|, \deg(t(X)) < |\mathbb{H}|,\\ f(\mathbb{D}) \subseteq t(\mathbb{H})\end{array}\right\}_{\mathsf{bp}, \mathbb{D}, \mathbb{H}, \sigma_{KZG}}. \quad (2)$$

3 Losum: Optimal Sum-check for KZG

We introduce Losum, a new univariate non-interactive sum-check scheme whose communication cost is a single group element, proving cost is a single MSM, and verification cost is dominated by one pairing. Moreover, unlike the existing univariate sum-check [BCR+19], Losum does not require the interpolation domain $\mathbb{H}$ to have any special structure, so it works for any field $\mathbb{F}$ as long as the field is sufficiently large.

3.1 Overview

Our method is based on the following observation: for a polynomial $f(X)$ of degree at most D, where $D \geq |\mathbb{H}|$, proving the equality $\sum_{h\in\mathbb{H}} f(h) = 0$ is equivalent to proving that $f(X)$ is a linear combination of a given set of basis polynomials. In more detail:

- The set $\mathcal{Z} := \{f(X) \in \mathbb{F}^{\leq D}[X] : \sum_{h\in\mathbb{H}} f(h) = 0\}$ forms a linear space of dimension D.
- For all $h \in \mathbb{H}\backslash\{h^*\}$, $L_h(X) - L_{h^*}(X) \in \mathcal{Z}$ for a fixed h^* in $\mathbb{H}$.
- For all $i \leq D - |\mathbb{H}|$, $X^i Z_{\mathbb{H}}(X) \in \mathcal{Z}$.
- The set $B := \{L_h(X) - L_{h^*}(X)\}_{h\in\mathbb{H}\backslash\{h^*\}} \cup \{X^i Z_{\mathbb{H}}(X)\}_{i=0}^{D-|\mathbb{H}|}$ is a linearly independent subset of $\mathcal{Z}$. Since $|B| = D$, B is a basis of $\mathcal{Z}$.

Therefore, proving that a committed $f(X)$ satisfies $\sum_{h\in\mathbb{H}} f(h) = 0$ is equivalent to proving that $f(X)$ is a linear combination of B. For a more general sum-check, i.e., $\sum_{h\in\mathbb{H}} f(h) = s$ for any $s \in \mathbb{F}$, we choose a representative polynomial $\ell_s(X)$ whose sum is trivially s, and prove that $f(X) - \ell_s(X) \in \mathcal{Z}$. One obvious choice of $\ell_s(X)$ is $s \cdot L_{h^*}(X)$.

Formally, the sum-check problem is reduced to the following statement: there exist $\{b_h\}_{h\in\mathbb{H}\backslash\{h^*\}}$ and $\{q_i\}_{i=0}^{D-|\mathbb{H}|}$ such that

$$f(X) - s \cdot L_{h^*}(X) = \sum_{h\in\mathbb{H}\backslash\{h^*\}} b_h \cdot (L_h(X) - L_{h^*}(X)) + \sum_{i=0}^{D-|\mathbb{H}|} q_i \cdot X^i Z_{\mathbb{H}}(X).$$

Note that q_i are exactly the coefficients of the quotient polynomial from dividing $f(X)$ by $Z_{\mathbb{H}}(X)$. To compute b_h, we evaluate both sides of this equation at every $h \in \mathbb{H}\backslash\{h^*\}$, and get that b_h is just $f(h)$. Therefore, if the prover starts the protocol from the evaluation representation of $f(X)$, the prover does not need any additional computation to compute the linear combination coefficients. This is the reason why we choose this basis. Depending on the scenario in which the protocol is used, a different basis may be more appropriate. For example, if the prover stores $f(X)$ by its coefficients, then a better basis would be the set of polynomials of the form $a_k X^k - b_k$.

Using the same technique as in Pinocchio [PHGR13], Groth16 [Gro16] and Marlin [CHM+20], we can force the prover to output a linear combination of a given polynomial set $\{c_i(X)\}$ as follows. The idea is to let the trusted third party select a random secret $\alpha \in \mathbb{F}$ and produce a set of $\mathbb{G}_1$ elements $\{[\alpha c_i(x)]_1\}$. These elements are included in the SRS. Moreover, the verifier SRS should include $[\alpha]_2$. The prover simply produces the proof $\pi := [\alpha f(x)]_1$ computed by linearly combining the elements $\{[\alpha c_i(x)]_1\}$, and the verifier checks that $e(\pi, [1]_2) = e([f(x)]_1, [\alpha]_2)$. Intuitively, without learning the secret α, the only way for the prover to generate $[\alpha f(x)]_1$ is linearly combing $\{[\alpha c_i(x)]_1\}$, which would be impossible if $f(X)$ is not a linear combination of $\{c_i(X)\}$.

3.2 Protocol Description

Exploiting the above techniques, we propose Losum for proving that the sum of $f(X)$ over $\mathbb{H}$ is s, presented as follows.

Setup. On input the pairing $\mathsf{bp} = (p, e, g, h, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, the domain $\mathbb{H}$ of size N, and the SRS $\sigma_{KZG} := (\{[x^i]_1\}_{i=0}^D, \{[1]_2, [x]_2\})$ previously generated by the KZG setup where $D \geq N$, the trusted third party samples $\alpha \in \mathbb{F}$, and outputs the SRS that include:

- $\{[\alpha \cdot (L_h(x) - L_{h^*}(x))]_1\}_{h \in \mathbb{H} \setminus \{h^*\}}$, and $\{[\alpha \cdot x^i Z_{\mathbb{H}}(x)]_1\}_{i=0}^{D-N}$ where h^* is picked from $\mathbb{H}$ arbitrarily;
- Verifier SRS: $[1]_2$, $[L_{h^*}(x)]_1$, and $[\alpha]_2$.

Prove. On input $f(X)$ and $\mathbb{H}$:

1. Divide $f(X)$ by $Z_{\mathbb{H}}(X)$ and let the quotient be $q(X) = \sum_{i=0}^{d} q_i X^i$.
2. Output the proof π computed as below

$$\pi := \sum_{h \in \mathbb{H} \setminus \{h^*\}} f(h) \cdot [\alpha \cdot (L_h(x) - L_{h^*}(x))]_1 + \sum_{i=0}^{d} q_i \cdot [\alpha x^i \cdot Z_{\mathbb{H}}(x)]_1.$$

Verify. On input $\pi, s, [f(x)]_1$, check

$$e([f(x)]_1 - s \cdot [L_{h^*}(x)]_1, [\alpha]_2) = e(\pi, [1]_2).$$

3.3 Security and Efficiency Analysis

We prove that Losum is an argument of knowledge.

Theorem 1. *The three algorithms in Sect. 3.2 form an argument of knowledge for the relation $\mathcal{R}_{\mathsf{Sum}}$ defined in Equation (1), in the algebraic group model.*

Proof. Completeness. Let $\tilde{f}(X)$ denote $f(X) \bmod Z_{\mathbb{H}}(X)$, then the right side, after divided by α, has exponent

$$\begin{aligned}
&\sum_{h\in\mathbb{H}\setminus\{h^*\}} f(h)\cdot(L_h(x)-L_{h^*}(x))+q(x)\mathbb{Z}_{\mathbb{H}}(x)\\
=&\sum_{h\in\mathbb{H}\setminus\{h^*\}} f(h)L_h(x)-\sum_{h\in\mathbb{H}\setminus\{h^*\}} f(h)L_{h^*}(x)+q(x)Z_{\mathbb{H}}(x)\\
=&(\tilde{f}(x)-f(h^*)L_{h^*}(x))-(s-f(h^*))L_{h^*}(x)+q(x)Z_{\mathbb{H}}(x)\\
=&\tilde{f}(x)-sL_{h^*}(x)+q(x)Z_{\mathbb{H}}(x)\\
=&f(x)-sL_{h^*}(x),
\end{aligned}$$

which is the same as the exponent of the left side.

Knowledge soundness. Let $\mathcal{A}$ be any PPT adversary. Since $\mathcal{A}$ is algebraic, whenever $\mathcal{A}$ outputs $c,\pi\in\mathbb{G}_1$, $\mathcal{A}$ simultaneously outputs the linear coefficients corresponding to $\sigma_{KZG}, [L_{h^*}(x)]_1, [\alpha\cdot(L_h(x)-L_{h^*}(x))]_1$ and $\alpha x^i Z_{\mathbb{H}}(x)$. We can then build the extractor E that executes whatever $\mathcal{A}$ executes, obtains the linear coefficients, and computes the polynomials $f_1(X), f_2(X), p_1(X)$ and $p_2(X)$ satisfying $c=[f_1(x)]_1+\alpha\cdot[f_2(x)]_1$ and $\pi=[p_1(x)]_1+\alpha\cdot[p_2(x)]_2$ by linearly combining the polynomials corresponding to the $\mathbb{G}_1$ elements in the SRS. Then our extractor outputs $f(X)=f_1(X)$. Now we prove that $[f(x)]_1=c$, i.e. $f_2(X)$ must be the zero polynomial, and $\sum_{h\in\mathbb{H}} f(h)=s$.

If the verification passes, i.e.,

$$e([f_1(x)]_1+\alpha\cdot[f_2(x)]_1-s\cdot[L_{h^*}(x)]_1,[\alpha]_2)=e(\pi,[1]_2),$$

we have

$$\alpha\cdot(f_1(x)-sL_{h^*}(x))+\alpha^2\cdot f_2(x)=p_1(x)+\alpha\cdot p_2(x)$$

which can be rewritten into

$$-p_1(x)+\alpha\cdot(f_1(x)-p_2(x)-sL_{h^*}(x))+\alpha^2\cdot f_2(x)=0.$$

Then by q-DLOG assumption, $p_1(x)=0$, $f_1(x)-p_2(x)-sL_{h^*}(x)=0$ and $f_2(x)=0$, except with negligible probability. Otherwise, we will get a non-trivial equation of α. We can then build an adversary that solves α with non-negligible probability from $[\alpha]_1$ and $[\alpha]_2$, by preparing the SRS using the target $[\alpha]_1$ and $[\alpha]_2$, breaking the q-DLOG assumption.

By q-DLOG assumption again, $p_1(X)=0$, $f_1(X)-p_2(X)-sL_{h^*}(X)=0$ and $f_2(X)=0$, except with negligible probability. Otherwise, we would obtain non-trivial equations for x and construct an adversary that solves x given the KZG parameters.

Therefore, $f(X)=f_1(X)=p_2(X)+sL_{h^*}(X)$, where $p_2(X)$ is the linear combination of $L_h(X)-L_{h^*}(X)$ and $X^iZ_{\mathbb{H}}(X)$, thus sums to 0 over $\mathbb{H}$. We then have $\sum_{h\in\mathbb{H}} f(h)=s$ and $[f(x)]_1=[f_1(x)]_1=c$. □

Efficiency. The setup algorithm is dominated by computing $\{\alpha\cdot(L_h(x)-L_{h^*}(x))\}$ and $\{[\alpha x^i Z_{\mathbb{H}}(x)]_1\}$. So the cost is $O(D\log D)$ scalar multiplications in $\mathbb{G}_1$ if $\mathbb{H}$ is a multiplicative group, in which case we use FFT on $\mathbb{G}_1$ elements. Otherwise, these can be computed with $O(D\log^2 D)$ scalar multiplications using the multi-point evaluation algorithm [Kun73].

The prover is dominated by computing π and $q(X)$. Computing π requires an MSM of size $k+\deg(f(X))-N$ or simply k if $\deg(f(X))<N$, where k is the number of non-zero entries in $f(\mathbb{H})$. Computing $q(X)$ costs only $O(\deg(f(X))-N)$ additions in $\mathbb{F}$ if $\mathbb{H}$ is a multiplicative group, since $Z_{\mathbb{H}}(X)$ would have the simple form X^N-1. For general $\mathbb{H}$, this takes $O(\deg(f(X))-N)\log^2(\deg(f(X))-N)$ field operations. Note that computing $q(X)$ can be ignored if $\deg(f(X))<N$. In practice, $f(X)$ is usually computed by adding a masking polynomial $\rho\cdot Z_{\mathbb{H}}(X)$ to a polynomial of degree less than N in the first place, as in our lookup argument described in the next section. In this case, $q(X)$ is simply ρ.

The verifier is dominated by a pairing, plus a scalar multiplication for computing $s\cdot[L_{h^*}(x)]_1$. Note that this scalar multiplication is omitted if $s=0$, which is a common situation in the use cases of sum checks.

4 Locq: Improved Lookup Argument

We construct a new zero-knowledge lookup argument Locq that has a smaller proof size and a smaller verification cost than cq and its subsequent works cq+, cq++, the state-of-the-art lookup arguments.

4.1 Overview

The design of Locq essentially follows the framework of cq. To explain our idea more clearly, we briefly recall the cq scheme. The goal of lookup argument is to make the prover convince the verifier that, given two committed polynomials $f(X)$ and $t(X)$, the set $f(\mathbb{D}) := \{f(u) : u \in \mathbb{D}\}$ is a subset of $t(\mathbb{H}) := \{t(h) : h \in \mathbb{H}\}$, where $\mathbb{D}$ and $\mathbb{H}$ are two different domains inside $\mathbb{F}$. We will refer to $f(X)$ or $f(\mathbb{D})$ as the lookup vector and $t(X)$ or $t(\mathbb{H})$ as the table. Let $m := |\mathbb{D}|$ and $N := |\mathbb{H}|$. We assume m is much smaller than N and $\mathbb{D}$ is a subset of $\mathbb{H}$. Note that, like in Losum, we do not require any algebraic structure on $\mathbb{D}$ or $\mathbb{H}$. However, we will mention where the running time can be reduced from $O(m\log^2 m)$ to $O(m\log m)$ when $\mathbb{D}$ or $\mathbb{H}$ are multiplicative subgroups.

To prove $f(\mathbb{D})\subseteq t(\mathbb{H})$, cq leverages a technique called *logarithmic derivative*, first proposed by Habock et al. [Hab22], which is based on the observation that $f(\mathbb{D})\subseteq t(\mathbb{H})$ if and only if there exist m_h for every $h\in\mathbb{H}$ such that $\sum_{u\in\mathbb{D}}\frac{1}{X-f(u)}=\sum_{h\in\mathbb{H}}\frac{m_h}{X-t(h)}$. Intuitively, this equality is possible only if both sides have the same set of holes, implying that every $f(u)$ is a hole on the right side, which is possible only if $f(u)\in t(\mathbb{H})$. Note that when $f(\mathbb{D})\subseteq t(\mathbb{H})$, the unique choice of m_h is the number that $t(h)$ appears in $f(\mathbb{D})$.

To prove this equality between two rational functions, the prover commits the polynomial $m(X)$ such that $m(h)=m_h$ for $h\in\mathbb{H}$. Note that committing $m(X)$

only requires $O(m)$ scalar multiplications because $\{m_h\}_{h\in\mathbb{H}}$ contain at most m non-zero elements. Then the verifier samples a random β and asks the prover to show that $\sum_{u\in\mathbb{D}} \frac{1}{\beta-f(u)} = \sum_{h\in\mathbb{H}} \frac{m(h)}{\beta-t(h)}$. Intuitively, by Schwartz-Zippel Lemma (for rational functions), the unpredictability of β guarantees that the original equality holds with overwhelming probability. Now it remains for the prover to convince the verifier of the equality between these two sums.

To show this equality, the prover commits two polynomials $g(X)$ and $w(X)$ of degree less than $|\mathbb{D}|$ and $|\mathbb{H}|$, respectively, such that $g(u) = \frac{1}{\beta-f(u)}$ for $u \in \mathbb{D}$ and $w(h) = \frac{m(h)}{\beta-t(h)}$ for $h \in \mathbb{H}$. To prove that it has committed to the correct $g(X)$ and $w(X)$, the prover shows the committed polynomials satisfy that $g(X)(\beta - f(X))-1$ is divided by $Z_{\mathbb{D}}(X)$ and $w(X)(\beta-t(X))-m(X)$ is divided by $Z_{\mathbb{H}}(X)$. These are achieved by committing the quotient polynomials $q_1(X)$ and $q_2(X)$, where the commitment to $q_2(X)$ is computed by the *cached quotient* technique to make the prover complexity independent of N. Finally, the prover shows that $\sum_{u\in\mathbb{D}} g(u) = \sum_{h\in\mathbb{H}} w(h)$, which is equivalent to the original equality. This step is accomplished using univariate sum-check [BCR+19].

We explain the cached quotient technique in more detail as it is also used in Locq to compute $[q_2(x)]_1$. Note that $q_2(X) = \lfloor \frac{w(X)t(X)}{Z_{\mathbb{H}}(X)} \rfloor$ since $w(X)t(X)$ is the only item that has degree at least N. To compute the commitment $[q_2(x)]_1$ with $O(m)$ group operations online, the prover preprocesses, in the offline phase, $[q_h(x)]_1$ for $h \in \mathbb{H}$ where $q_h(X) := \lfloor \frac{L_h(X)t(X)}{Z_{\mathbb{H}}(X)} \rfloor$. These pre-computed $[q_h(x)]_1$ are called the cached quotients for the table $t(X)$. In the online phase, $[q_2(x)]_1$ is computed by $[q_2(x)]_1 = \sum_{h\in\mathbb{H}} w(h)[q_h(x)]_1$, which involves at most m group scalar multiplications because there are at most m nonzero $w(h)$.

Locq improves over cq exploiting the following techniques.

Interpolate $g(X)$ over $\mathbb{H}$ instead of over $\mathbb{D}$. Instead of defining $g(X)$ by interpolating $\frac{1}{\beta-f(u)}$ over $\mathbb{D}$, we additionally require that $g(h) = 0$ for $h \in \mathbb{H}\backslash\mathbb{D}$. In this way, the sum of $g(X)$ over $\mathbb{H}$ is still the desired sum $\sum_{u\in\mathbb{H}} \frac{1}{\beta-f(u)}$, we thus merge the two sum-checks: the prover only needs to show that the sum of $g(X)-w(X)$ over $\mathbb{H}$ is zero. This optimization is also adopted by cq+, although presented in a different form.

This redefinition of $g(X)$ brings several challenges, which we address as follows.

1. The degree of $g(X)$ increases from m to N, so the prover should avoid computing its coefficients explicitly, and instead compute its commitment using precomputed commitments for $L_h(X)$.
2. To prove the correctness of $g(X)$, in addition to checking that $g(X)(\beta-f(X))$ evaluates to 1 over $\mathbb{D}$, the verifier should also check that it evaluates to zero over $\mathbb{H}\backslash\mathbb{D}$. Therefore, we redefine $q_1(X)$ to be the quotient between $g(X)(\beta - f(X)) - U(X)$ and $Z_{\mathbb{H}}(X)$, where $U(X)$ is the polynomial that evaluates to 1 over $\mathbb{D}$ and 0 over $\mathbb{H}\backslash\mathbb{D}$. In general, $U(X)$ has degree $N - 1$.

However, when $\mathbb{H}$ and $\mathbb{D}$ are multiplicative subgroups, $U(X)$ is $\frac{m}{N} \cdot \frac{Z_{\mathbb{H}}(X)}{Z_{\mathbb{D}}(X)} = \frac{m \cdot (X^N - 1)}{N \cdot (X^m - 1)}$ and has degree $N - m$.

3. Although the degree of $q_1(X)$ is still $O(m)$, computing $q_1(X)$ naïvely would bring $O(N \log^2 N)$ cost to the prover. To address this, note that all of $Z_{\mathbb{H}}(X)$, $g(X)$ and $U(X)$ are divided by the polynomial $\frac{Z_{\mathbb{H}}(X)}{Z_{\mathbb{D}}(X)}$, so $q_1(X)$ can be alternatively computed by dividing $g'(X)(\beta - f(X)) - U'(X)$ with $Z_{\mathbb{D}}(X)$, where $g'(X)$ and $U'(X)$ are defined by dividing $g(X), U(X)$ with $\frac{Z_{\mathbb{H}}(X)}{Z_{\mathbb{D}}(X)}$, respectively. Note that $U'(X)$ can be precomputed offline, and $g'(X)$ can be computed by interpolating $g(u) \cdot c_u^{-1}$ over $\mathbb{D}$, where c_u is the evaluation of $\frac{Z_{\mathbb{H}}(X)}{Z_{\mathbb{D}}(X)}$ at $u \in \mathbb{D}$. Both can be accomplished with $O(m \log^2 m)$ complexity. When $\mathbb{H}$ and $\mathbb{D}$ are multiplicative subgroups, both $U'(X)$ and c_u^{-1} become the constant $\frac{m}{N}$, and the total cost of computing $q_1(X)$ can be reduced to $O(m \log m)$.

Merge the two quotient polynomials. Since the correctness of both $g(X)$ and $w(X)$ are reduced to divisibility by $Z_{\mathbb{H}}(X)$, the two divisibility checks can be merged into one by random linear combination. Specifically, the verifier samples ζ, and the prover shows that the polynomial

$$(\beta - f(X)) \cdot g(X) - U(X) + \zeta \cdot ((\beta - t(X) \cdot w(X)) - m(X))$$

is divided by $Z_{\mathbb{H}}(X)$ by sending one quotient polynomial $q(X)$, instead of $q_1(X)$ and $q_2(X)$ separately.

Use Losum *instead of univariate sum-check.* We apply a slightly modified Losum to prove that the sum of $g(X) - w(X)$ over $\mathbb{H}$ is zero, rather than the univariate sum-check [BCR+19] used by cq. The modification to Losum is because $g(X)$ and $w(X)$ are committed in different groups, so instead of multiplying α to $g(X) - w(X)$, the verifier multiplies α^{-1} to the proof π in the pairing check. Moreover, because the degree of $g(X) - w(X)$ is smaller than N, there is no need to include $\alpha X^i Z_{\mathbb{H}}(X)$ in the SRS (unless for $i = 0$, if zero-knowledge is needed, as explained later). Losum reduces the overall cost of Locq because it costs a single $\mathbb{G}_1$ element in the proof and a single pairing in the verification. Moreover, it does not require any degree check as the original univariate sum-check, nor introduce any additional divisibility check.

Add zero-knowledge. Finally, we make the protocol zero-knowledge with almost zero overhead. The idea is to add random multiples of $Z_{\mathbb{H}}(X)$ to $m(X), g(X)$ and $w(X)$, respectively. This will make the degree of $g(X) - w(X)$ achieve N, so we additionally add $\alpha Z_{\mathbb{H}}(X)$ to the SRS of Losum. Because our protocol does not involve any degree check and the proof does not contain any polynomial evaluations, these randomizations suffice to guarantee zero-knowledge without affecting the performance.

4.2 Protocol Description

The complete protocol is presented as follows, where we split the setup algorithm into a universal setup and a preprocessing procedure. The universal setup is executed once for all tables of the specific size, while the preprocessing is executed for each table without a trusted third party.

Setup. On input $\mathbb{D}, \mathbb{H}$ of size m, N, respectively, where $\mathbb{D} \subset \mathbb{H}$, the bilinear pairing parameter $\mathsf{bp} = (p, e, g, h, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and the SRS for KZG $\sigma_{KZG} = (\{[x^i]_1\}_{i=0}^{D}, \{[x^i]_2\}_{i=0}^{D})$, the setup procedure outputs the SRS computed as follows:

1. Let $\{L_h(X)\}_{h\in\mathbb{H}}$ be the Lagrange basis polynomials over $\mathbb{H}$, $U(X) = \sum_{u\in\mathbb{D}} L_u(X)$, $U'(X) = \frac{U(X)\cdot Z_{\mathbb{D}}(X)}{Z_{\mathbb{H}}(X)}$, and c_u be the evaluation of $\frac{Z_{\mathbb{H}}(X)}{Z_{\mathbb{D}}(X)}$ at u for $u \in \mathbb{D}$.
2. Execute the modified setup algorithm of Losum to generate $\sigma_{losum} = ([\alpha^{-1}]_2, [\alpha\cdot Z_{\mathbb{H}}(x)]_1, \{[\alpha \cdot (L_h(x) - L_{h^*}(x))]_1\}_{h\in\mathbb{H}\setminus\{h^*\}})$, where h^* is picked from $\mathbb{D}$ arbitrarily.
3. Output srs that includes
   - σ_{KZG}, σ_{losum}, $\{[L_h(x)]_1, [L_h(x)]_2\}_{h\in\mathbb{H}}$, $[Z_{\mathbb{H}}(x)]_1$, $U'(X)$, and $\{c_u\}_{u\in\mathbb{D}}$.
   - Verifier SRS: $[U(X)]_1$, $[Z_{\mathbb{H}}(x)]_2$, $[\alpha^{-1}]_2$, $[1]_1$, and $[1]_2$.

Preprocess. On input $t(X)$ and srs, the preprocessor outputs $\mathsf{srs}_{t(X)}$ as follows:

1. For $h \in \mathbb{H}$, divide $L_h(X)t(X)$ by $Z_{\mathbb{H}}(X)$ and get the quotient $q_h(X)$.
2. Output $\{[q_h(x)]_1\}_{h\in\mathbb{H}}$ and $[t(x)]_1$.

Prover. On input $f(X), t(X), [f(x)]_1, [t(x)]_2$, srs and $\mathsf{srs}_{t(X)}$, the prover interacts with the verifier as follows:

Round 1. For $h \in \mathbb{H}$, let m_h be the number of times that $t(h)$ appears in $f(\mathbb{D})$. Sample $\delta_1 \xleftarrow{\$} \mathbb{F}$. Send $[m(x)]_1 := \sum_{h\in\mathbb{H}} m_h[L_h(x)]_1 + \delta_1 \cdot [Z_{\mathbb{H}}(x)]_1$ to the verifier.

Round 2. Receive β from the verifier.

Round 3.

1. Sample $\delta_2, \delta_3 \xleftarrow{\$} \mathbb{F}$ and let

$$g(X) := \sum_{u\in\mathbb{D}} \frac{1}{\beta - f(u)} \cdot L_u(X) + \delta_2 \cdot Z_{\mathbb{H}}(X),$$

$$w(X) := \sum_{h\in\mathbb{H}} \frac{m_h}{\beta - t(h)} \cdot L_h(X) + \delta_3 \cdot Z_{\mathbb{H}}(X).$$

2. Compute the commitments $[g(x)]_2$ and $[w(x)]_1$ by

$$[g(x)]_2 := \sum_{u\in\mathbb{D}} \frac{1}{\beta - f(u)} \cdot [L_u(x)]_2 + \delta_2 \cdot [Z_{\mathbb{H}}(x)]_2$$

$$[w(x)]_1 := \sum_{h\in\mathbb{H}} \frac{m_h}{\beta - t(h)} \cdot [L_h(x)]_1 + \delta_3 \cdot [Z_{\mathbb{H}}(x)]_1.$$

3. Invoke the proving algorithm of Losum to compute π_{sum} for $g(X) - w(X)$. In detail,

$$\pi_{sum} = \sum_{u \in \mathbb{D} \setminus \{h^*\}} \frac{1}{\beta - f(u)} \cdot [\alpha \cdot (L_u(x) - L_{h^*}(x))]_1 - \sum_{h \in \mathbb{H} \setminus \{h^*\}} \frac{m_h}{\beta - t(h)} \cdot [\alpha \cdot (L_h(x) - L_{h^*}(x))]_1 + (\delta_2 - \delta_3) \cdot [\alpha \cdot Z_{\mathbb{H}}(x)]_1.$$

4. Send $[g(x)]_2, [w(x)]_1, \pi_{sum}$ to the verifier.

Round 4. Receive ζ from the verifier.

Round 5.

1. Interpolate $\frac{c_u^{-1}}{\beta - f(u)}$ over $\mathbb{D}$ to get $g'(X) := \sum_{u \in \mathbb{D}} \frac{c_u^{-1}}{\beta - f(u)} \cdot K_u(X)$ where $K_u(X)$ is the Lagrange basis polynomial over $\mathbb{D}$. Divide $g'(X) \cdot f(X)$ by $Z_{\mathbb{D}}(X)$, take the quotient, then add $\delta_2 \cdot (\beta - f(X))$ to get $q_1(X)$.
2. Compute $[q_1(x)]_1$ using its coefficients and σ_{KZG}, and compute $[q_2(x)]_1$ by

$$[q_2(x)]_1 := \sum_{h \in \mathbb{H}} \frac{m_h}{\beta - t(h)} \cdot [q_h(x)]_1 + \delta_3 \cdot (\beta \cdot [1]_1 - [t(x)]_1).$$

3. If $t(X)$ has been masked by a random $\rho \cdot Z_{\mathbb{H}}(X)$, then $[q_2(x)]_1$ should additionally add $\rho \cdot [w(x)]_1$.
4. Send $[q(x)]_1 := [q_1(x)]_1 + \zeta \cdot [q_2(x)]_1$ to the verifier.

Verifier. On input $[f(x)]_1, [t(x)]_2$ and the verifier SRS, the verifier interacts with the prover as follows.

Round 1. Receive $[m(x)]_1$ from the prover.

Round 2. Sample a uniformly random $\beta \in \mathbb{F}$ and send β to the prover.

Round 3. Receive $[g(x)]_2, [w(x)]_1, \pi_{sum}$ from the prover.

Round 4. Sample a uniformly random $\zeta \in \mathbb{F} \setminus \{0\}$ and send ζ to the prover.

Round 5. Receive $[q(x)]_1$ from the prover, then

1. Invoke the verification algorithm of Losum to check π_{sum}. In detail

$$e(\pi_{sum}, [\alpha^{-1}]_2) = e(-[w(x)]_1, [1]_2) \cdot e([1]_1, [g(x)]_2).$$

2. Check the correctness of $[g(x)]_2$ and $[w(x)]_1$ in batch

$$e(\beta \cdot [1]_1 - [f(x)]_1, [g(x)]_2) \cdot e(\zeta \cdot [w(x)]_1, \beta \cdot [1]_2 - [t(x)]_2) = e([U(x)]_1 + \zeta \cdot [m(x)]_1, [1]_2) \cdot e([q(x)]_1, [Z_{\mathbb{H}}(x)]_2).$$

Note that in the two pairing checks, $[w(x)]_1$ and $[g(x)]_2$ are repeatedly multiplied to different polynomials, so the two checks can be merged into one using random linear combination to save one pairing. Specifically, they are statistically equivalent to: sample $\delta \in \mathbb{F}$, then check

$$e((\beta+\delta)\cdot[1]_1 - [f(x)]_1, [g(x)]_2)\cdot e([w(x)]_1, (\zeta\beta-\delta)\cdot[1]_2 - \zeta\cdot[t(x)]_2) = \\ e(\delta\cdot\pi_{sum}, [\alpha^{-1}]_2)\cdot e([U(x)]_1 + \zeta\cdot[m(x)]_1, [1]_2)\cdot e([q(x)]_1, [Z_{\mathbb{H}}(x)]_2).$$

4.3 Security and Efficiency Analysis

We prove that Locq is a zero-knowledge argument of knowledge.

Theorem 2. *Viewing the preprocessing algorithm as part of the setup algorithm, the four algorithms in Sect 4.2 form a zero-knowledge argument of knowledge for the relation $\mathcal{R}_{\mathsf{Lookup}}$ defined in Equation (2), in the algebraic group model.*

Proof. For simplicity, we prove the completeness and soundness of the protocol without the final merge of two pairing checks. The equivalence between the original two pairing checks and the merged check holds for any pairing checks of this form.

Completeness. The second pairing equation is satisfied by definition if $g(X)$, $w(X)$, $q(X)$ are computed as expected. Then, by definition, the sums $\sum_{h\in\mathbb{H}} g(u)$ and $\sum_{h\in\mathbb{H}} w(h)$ are equal. Then the first pairing equation follows from the fact that, after divided by α, π_{sum} is a commitment to the polynomial

$$\sum_{u\in\mathbb{D}\setminus\{h^*\}} \frac{1}{\beta-f(u)}(L_u(X)-L_{h^*}(X)) + \sum_{h\in\mathbb{H}\setminus\{h^*\}} \frac{1}{\beta-t(h)}(L_h(X)-L_{h^*}(X)) \\ +(\delta_2-\delta_3)\cdot Z_{\mathbb{H}}(X) = g(X)-w(X).$$

This equality of polynomials holds because, by definition of $g(X)$ and $w(X)$, the two sides (a) have the same leading coefficient for X^N, i.e., $\delta_2-\delta_3$; (b) evaluate to the same value at every $h\in\mathbb{H}\setminus\{h^*\}$; and (c) both sides sum to zero over $\mathbb{H}$, so their evaluations must also match at 1.

Knowledge soundness. Note that all the group elements in srs allow the adversary to compute $\mathbb{G}_1$ elements as commitments to polynomials of the form $a_0(X)+\alpha\cdot a_\alpha(X)$ where the degree of $a_0(X)$ is at most D and the degree of $a_\alpha(X)$ is at most N, and the sum of $a_\alpha(X)$ over $\mathbb{H}$ is zero. For $\mathbb{G}_2$ elements, the adversary can only output commitments to polynomials of the form $a_0(X)+a_{\alpha^{-1}}\cdot\alpha^{-1}$. Being algebraic, whenever the adversary outputs a $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) group element, it simultaneously outputs the linear coefficients that allow us to recover $a_0(X)$ and $a_\alpha(X)$ (resp. $a_{\alpha^{-1}}$).

Specifically, when the adversary outputs c_q, c_m, c_p, c_w, c_f that are supposed to be $[q(x)]_1, [m(x)]_1, \pi_{sum}$, $[w(x)]_1$ in the proof and $[f(x)]_1$ in the instance, respectively, the adversary also outputs $q_0(X), q_\alpha(X)$ where $q_\alpha(X)$ sums to 0 over $\mathbb{H}$, such that $c_q = [q_0(x)]_1 + \alpha\cdot[q_\alpha(X)]_1$, and similarly outputs $m_0(X), m_\alpha(X)$, $p_0(X), p_\alpha(X)$, $f_0(X), f_\alpha(X)$, $w_0(X), w_\alpha(X)$.

Likewise, when the adversary outputs c_g and c_t that are supposed to be $[g(x)]_2$ in the proof and $[t(x)]_2$ in the instance, it simultaneously outputs $g_0(X), g_{\alpha^{-1}}$, $t_0(X), t_{\alpha^{-1}}$, such that $c_g = [g_0(x)]_2 + g_{\alpha^{-1}} \cdot [\alpha^{-1}]_2$ and $c_t = [t_0(x)]_2 + t_{\alpha^{-1}} \cdot [\alpha^{-1}]_2$.

By the first pairing check, we have

$$(g_0(x) + \alpha^{-1} \cdot g_{\alpha^{-1}}) - (w_0(x) + \alpha \cdot w_\alpha(x)) \\ = \alpha^{-1} \cdot (p_0(x) + \alpha \cdot p_\alpha(x)) \quad (3)$$

which can be reformulated into

$$g_{\alpha^{-1}} - p_0(x) + \alpha \cdot (g_0(x) - w_0(x) - p_\alpha(x)) - \alpha^2 \cdot w_\alpha(x) = 0 \quad (4)$$

We then have that

1. $p_0(x) = g_{\alpha^{-1}}$,
2. $g_0(x) - w_0(x) = p_\alpha(x)$,
3. $w_\alpha(x) = 0$.

Otherwise, the adversary would get a non-zero equation of α that allows the adversary to solve for α. We can then build an adversary that computes α from $[\alpha^{-1}]_2$ and $[\alpha]_1$ by breaking the q-DLOG assumption. Then we claim that the corresponding polynomials output from the adversary also satisfy

1. $p_0(X) = g_{\alpha^{-1}}$,
2. $g_0(X) - w_0(X) = p_\alpha(X)$,
3. $w_\alpha(X) = 0$.

Otherwise, we can build an adversary that uses these non-zero polynomials to solve for x, breaking the q-DLOG assumption. Therefore, we have $w(X) = w_0(X)$ and that $g_0(X) - w(X)$ is a polynomial that sums to zero over $\mathbb{H}$ since $p_\alpha(X)$ is guaranteed to have this property.

By the second pairing check (and applying q-DLOG assumption again),

$$(\beta - (f_0(X) + \alpha \cdot f_\alpha(X))) \cdot (g_0(X) + \alpha^{-1} \cdot g_{\alpha^{-1}}) + \zeta \cdot w(X) \cdot (\beta - (t_0(X) + t_{\alpha^{-1}} \cdot \alpha^{-1})) = \\ U(X) + \zeta \cdot (m_0(X) + \alpha \cdot m_\alpha(X)) + (q_0(X) + \alpha \cdot q_\alpha(X)) \cdot Z_{\mathbb{H}}(X), \quad (5)$$

where we have already applied the conclusion that $w_0(X) = w(X)$. Therefore, $Z_{\mathbb{H}}(X)$ divides the polynomial

$$(\beta - (f_0(X) + \alpha \cdot f_\alpha(X))) \cdot (g_0(X) + \alpha^{-1} \cdot g_{\alpha^{-1}}) - U(X) + \\ \zeta \cdot (w(X) \cdot (\beta - (t_0(X) + t_{\alpha^{-1}} \cdot \alpha^{-1})) + (m_0(X) + \alpha \cdot m_\alpha(X))).$$

Because ζ is sampled after all these polynomials are output from the adversary, and this division with non-negligible probability over ζ, we conclude that both

$$(\beta - (f_0(X) + \alpha \cdot f_\alpha(X))) \cdot (g_0(X) + \alpha^{-1} \cdot g_{\alpha^{-1}}) - U(X) \quad (6)$$

and

$$w(X) \cdot (\beta - (t_0(X) + t_{\alpha^{-1}} \cdot \alpha^{-1})) + (m_0(X) + \alpha \cdot m_\alpha(X)) \quad (7)$$

are divided by $Z_{\mathbb{H}}(X)$ with overwhelming probability, i.e., evaluates to zero over $\mathbb{H}$.

For polynomial (6), because β is sampled after $f(X)$ is output from the adversary, so the part $(\beta - (f_0(X) + \alpha \cdot f_\alpha(X)))$ evaluates to nonzero values over $\mathbb{H}$ except with negligible probability. Therefore, $g_0(X) + \alpha^{-1} \cdot g_{\alpha^{-1}}$ must be zero over $\mathbb{H}\backslash\mathbb{D}$. However, if $g_\alpha^{-1} \neq 0$, the adversary would be able to solve for α, breaking the q-DLOG assumption. So $g_\alpha^{-1} = 0$ and $g_0(X) = g(X)$. Then if $f_\alpha(u) \neq 0$ for any $u \in \mathbb{D}$, the adversary can solve α by substituting h in this equation. So $f_\alpha(u) = 0$ for every $u \in \mathbb{D}$. By definition of $U(X)$, we have $g(u) = \frac{1}{\beta - f_0(u)}$ for $u \in \mathbb{D}$.

From polynomial (7), we conclude that for every h, $m_\alpha(h) = 0$, otherwise, we would get a non-trivial equation of α and the adversary solves for α. So $m_\alpha(X)$ is a constant multiple of $Z_{\mathbb{H}}(X)$. We also have $t_{\alpha^{-1}} = 0$, because otherwise, $w(h)$ must be zero for all $h \in \mathbb{H}$ to prevent the adversary from getting a non-trivial equation of α. However, since we have already proved that $g(X) - w(X)$ sums to zero over $\mathbb{H}$, so $w(h) = 0$ over $\mathbb{H}$ would imply that the sum of $g(X)$ over $\mathbb{H}$ is zero. By the conclusion we obtained from polynomial (6), this means $\sum_{u \in \mathbb{D}} \frac{1}{\beta - f_0(u)} = 0$. However, since β is sampled after $f_0(u)$ is fixed, this equality holds with negligible probability. This justifies the claim that $t_{\alpha^{-1}} = 0$, so $t(X) = t_0(X)$. Then we have $w(h) = \frac{m_0(h)}{\beta - t(h)}$ for every $h \in \mathbb{H}$.

Finally, we show that $f(X) = f_0(X)$, i.e., $f_\alpha(X) = 0$. To achieve this, we revisit Equation (5), which can now be simplified to

$$\begin{aligned}(\beta - (f_0(X) + \alpha \cdot f_\alpha(X))) \cdot g(X) + \zeta \cdot (w(X) \cdot (\beta - t(X)) \\ - (m_0(X) + \alpha \cdot m_\alpha(X))) = U(X) + (q_0(X) + \alpha \cdot q_\alpha(X)) \cdot Z_{\mathbb{H}}(X).\end{aligned}$$

We have already proved that $w(X) \cdot (\beta - t(X)) - m_0(X)$ is divided by $Z_{\mathbb{H}}(X)$. Define $m'_\alpha = m_\alpha(X)/Z_{\mathbb{H}}(X)$, which is a constant as we have already proved. So we subtract it from both sides by appropriately redefining $q_0(X)$. Then we have

$$(\beta - (f_0(X) + \alpha \cdot f_\alpha(X))) \cdot g(X) - U(X) = (q_0(X) + \alpha \cdot (q_\alpha(X) + \zeta m'_\alpha)) \cdot Z_{\mathbb{H}}(X),$$

which can be reformulated into

$$(\beta - f_0(X)) \cdot g(X) - U(X) - q_0(X) \cdot Z_{\mathbb{H}}(X) = \alpha \cdot (f_\alpha(X) g(X) + (q_\alpha(X) + \zeta m'_\alpha) \cdot Z_{\mathbb{H}}(X)).$$

Then both sides must be zero. Otherwise, the adversary would solve for α. The right side being zero implies $m'_\alpha = 0$, because $f_\alpha(X)$ and $g(X)$ are both selected before ζ is sampled. If m'_α is nonzero then the prover must select $q_\alpha(X) = -\frac{f_\alpha(X) g(X)}{Z_{\mathbb{H}}(X)} - \zeta m'_\alpha$. However, this is unlikely to be a polynomial whose sum over $\mathbb{H}$ is zero. So m'_α must be zero, and $q_\alpha(X) = -\frac{f_\alpha(X) g(X)}{Z_{\mathbb{H}}(X)}$ sums to zero over $\mathbb{H}$. Since $f_\alpha(X)$ is selected before β is sampled, and we have proved that $g(u) = \frac{1}{\beta - f_0(u)}$, we must have $\frac{f_\alpha(X) L_u(X)}{Z_{\mathbb{H}}(X)}$ sums to zero over $\mathbb{H}$ for every $u \in \mathbb{H}$. Moreover, $f_\alpha(X)$ itself also has a zero sum. Together we have $|\mathbb{H}| + 1$ linearly independent constraints over the coefficient vector of $f_\alpha(X)$, whose size is also at most $|\mathbb{H}| + 1$. We thus conclude that $f_\alpha(X)$ must be zero, so is $q_\alpha(X)$.

Combining all the conclusions we have so far, we finally obtain the equality $\sum_{u\in\mathbb{D}}\frac{1}{\beta-f(u)}=\sum_{h\in\mathbb{H}}\frac{m(h)}{\beta-t(h)}$ that holds with overwhelming probability for a non-negligible fraction of β. We then conclude that the rational function $\sum_{u\in\mathbb{D}}\frac{1}{X-f(u)}-\sum_{h\in\mathbb{H}}\frac{m(h)}{X-t(h)}$ must be zero, otherwise this function does not evaluate to 0 except for negligible fraction of β. Therefore, this rational function does not contain any poles, which means that for every $f(u)$, there must exist some $t(h)=f(u)$. We then define the extractor by invoking the adversary and outputting $t(X), f(X)$.

Zero-knowledge. Given σ_{KZG}, c_f, c_t and the trapdoor α, the simulator outputs the transcript $c_m, \beta, c_g, c_w, \pi_{sum}, \zeta, c_q$ as follows. First, uniformly sample $\beta, w, g, q\in\mathbb{F}$ and $\zeta\in\mathbb{F}\backslash\{0\}$ and let $c_w=[w]_1$, $c_g=[g]_2$, $c_q=[q]_1$. Then compute $c_m=\zeta^{-1}\cdot(g\cdot([\beta]_1-c_f)-[U(x)]_1-q\cdot[Z_{\mathbb{H}}(x)]_1)+w\cdot([\beta]_2-c_t)$. Finally, compute $\pi_{sum}=[\alpha\cdot(g-w)]_1$. This transcript passes the verification by design. We then argue that its distribution is statistically close to an honest transcript.

We accomplish this by introducing an intermediate distribution of c_m, β, c_g, c_w, π_{sum}, ζ, c_q and argue that this distribution is close to both the aforementioned distribution and the honest distribution. This distribution is defined as follows. Sample m, β, g, w, ζ uniformly independently, then define $\pi_{sum}=[\alpha\cdot(g-w)]_1$ and $q=\frac{g\cdot(\beta-f(x))+\zeta\cdot w\cdot(\beta-t(x))-U(x)-\zeta\cdot m}{Z_{\mathbb{H}}(x)}$. Finally, let $c_w=[w]_1$, $c_g=[g]_2$, $c_q=[q]_1$, $c_m=[m]_1$. Note that the simulator cannot directly sample following the definition of this intermediate distribution because it does not learn $f(x)$ and $t(x)$.

First, we argue that the distribution is statistically close to the simulated distribution. Because in both distributions, β, g, w, ζ are sampled uniformly independently, we only need to show that the conditional distributions of π_{sum}, c_q, c_m given β, g, w, ζ are close. In both distributions, π_{sum} is deterministically decided by g, w, so its distributions in both cases are the same. Then we note that in both distributions, q and m satisfy the same equation of the form $A\cdot q+B\cdot m=C$, where A, B, C are decided by $\beta, g, w, \zeta, f(x), t(x), U(x), Z_{\mathbb{H}}(x)$. For the simulator, the distribution of q, m is first sampling q uniformly and then solving for m, whereas the intermediate distribution is first sampling m and then solving for q. Both strategies are uniformly sampling the solution space of this equation, thus producing the same distribution.

Next, we show that the intermediate distribution is close to the distribution of an honest transcript. In the honest execution, $\delta_1, \delta_2, \delta_3$ and β, ζ are uniformly and independently sampled, so the distributions of $c_m, \beta, c_g, c_w, \zeta$ are the same as in the intermediate distribution. Then in both distributions, π_{sum} and c_q are the unique elements that satisfy the pairing checks and are decided deterministically by $c_m, \beta, c_g, c_w, \zeta$. We thus conclude that the simulated transcript is statistically indistinguishable from an honest transcript. □

Efficiency. The verifier cost, after the optimization by the final random linear combination, is dominated by four pairings (one pairing check plus three additional pairings). The proof consists of 4 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ elements. The prover cost consists of:

– Computing $[m(x)]_1$ and $[w(x)]_1$ each costs one MSM of size m in $\mathbb{G}_1$.
– Computing $[g(x)]_2$ costs one MSM of size m in $\mathbb{G}_2$.
– Computing $q_1(X)$ can be accomplished as follows:
  1. Interpolate $\frac{c_u^{-1}}{\beta - f(u)}$ over $\mathbb{D}$ into $g'(X)$.
  2. Evaluate $f(X)$ over $\mathbb{D}' \subset \mathbb{F}$ of size m that is disjoint from $\mathbb{D}$ by one interpolation followed by one multi-point evaluation (assuming we only have the evaluation representation of $f(X)$).
  3. Evaluate $g'(X)$ over $\mathbb{D}'$ by one multi-point evaluation of size m.
  4. Evaluate $r(X) := f(X) \cdot g'(X) \bmod Z_{\mathbb{D}}(X)$ over $\mathbb{D}'$, by interpolating $\frac{c_u^{-1} \cdot f(u)}{\beta - f(u)}$ over $\mathbb{D}$ followed by multi-point evaluation over $\mathbb{D}'$.
  5. Divide the evaluation of $f(X) \cdot g'(X) - r(X)$ over $\mathbb{D}'$ by the evaluations of $Z_{\mathbb{D}}(X)$.
  6. Get the coefficients of $q_1(X)$ by one interpolation.

  This costs seven interpolation/multi-point evaluations of size m. Then computing $[q(x)]_1$ costs one MSM of size $2m$ in $\mathbb{G}_1$.
– Computing π_{sum} costs one MSM of size $2m$ in $\mathbb{G}_1$.

In conclusion, the prover cost is dominated by two $\mathbb{G}_1$-MSM of size m, two $\mathbb{G}_1$-MSM of size $2m$, one $\mathbb{G}_2$-MSM of size m, and seven size-m interpolation or multi-point evaluations that have cost $O(m \log^2 m)$. When $\mathbb{H}$ and $\mathbb{D}$ are multiplicative subgroups, the interpolation or multi-point evaluations are replaced with IFFT and FFTs that have cost $O(m \log m)$.

Compared to cq, the proving cost of Locq is almost the same: we exchange $2m$ scalar multiplications in $\mathbb{G}_1$ with m scalar multiplications in $\mathbb{G}_2$. According to the data provided in `zkalc`[5], multiplications in $\mathbb{G}_2$ is roughly three times slower than multiplications in $\mathbb{G}_1$ for the BLS12-381 curve implemented by `ark-works`, so the overall proving efficiency of Locq is slightly worse than cq. However, the concrete impact varies significantly by the concrete implementations.

5 Conclusion

We proposed new schemes respectively for the univariate sum-check and the lookup argument that are essential tools in SNARK and ZKVM constructions. Both schemes advance the state of the art by reducing the proving costs, verification costs, and/or proof sizes. Our schemes can be directly deployed as drop-in replacements for the existing schemes. Meanwhile, our prover complexity is still $O(m \log m)$, and it is still an open question to reduce the number of field operations in the prover to $O(m)$.

Acknowledgement

This work is partially supported by Shanghai Science and Technology Innovation Action Plan (Grant No. 23511101100), the National Key Research and Development Project (Grant No. 2020YFA0712300) and the National Natural Science

[5] `https://zka.lc`

Foundation of China (Grant No. 62272294). We thank Ren Zhang and Alan Szepieniec for their valuable comments and feedback. We thank the anonymous reviewers for their careful examination of our work and their insightful comments and constructive suggestions.

References

BCG+18. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In *Advances in Cryptology – ASIACRYPT 2018*, Lecture Notes in Computer Science, pages 595–626. Springer International Publishing, 2018.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent Succinct Arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11476, pages 103–128. Springer International Publishing, 2019. http://link.springer.com/10.1007/978-3-030-17653-2_4.

BEG+91. Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the Correctness of Memories. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 90–99. IEEE Computer Society, 1991.

CBBZ22. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates, 2022. https://eprint.iacr.org/2022/1355.

CFF+23. Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees, 2023.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 738–768. Springer International Publishing, 2020.

COS. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 769–793. Springer International Publishing.

EFG22. Liam Eagen, Dario Fiore, and Ariel Gabizon. Cq: Cached quotients for fast lookups, 2022. https://eprint.iacr.org/2022/1763.

FKL17. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The Algebraic Group Model and its Applications. Technical Report 620, 2017. http://eprint.iacr.org/2017/620.

FS86. Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology – EUROCRYPT 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. https://doi.org/10.1007/3-540-47721-7_12.

GK22. Ariel Gabizon and Dmitry Khovratovich. Flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size, 2022. https://eprint.iacr.org/2022/1447.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the Fourtieth Annual ACM Symposium on Theory of Computing - STOC 08*, page 113. ACM Press, 2008. http://dl.acm.org/citation.cfm?doid=1374376.1374396.

GPR21. Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo – a Turing-complete STARK-friendly CPU architecture. Technical Report 1063, 2021. http://eprint.iacr.org/2021/1063.

Gro16. Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9666, pages 305–326. Springer Berlin Heidelberg, 2016. http://link.springer.com/10.1007/978-3-662-49896-5_11.

GW20. Ariel Gabizon and Zachary J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. Technical Report 315, 2020. http://eprint.iacr.org/2020/315.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives, 2022.

Kun73. Hsiang-Tsung Kung. *Fast evaluation and interpolation.* Carnegie-Mellon University. Department of Computer Science, 1973.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477, pages 177–194. Springer Berlin Heidelberg, 2010. http://link.springer.com/10.1007/978-3-642-17373-8_11.

LFKN90. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic Methods for Interactive Proof Systems. In *31st Annual Symposium on Foundations of Computer Science, 1990, Volume I*, pages 2–10. IEEE Computer Society, 1990.

Mid22. Team Miden. Miden VM Documentation, 2022. https://maticnetwork.github.io/miden/.

PFM+22. Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Munoz, and Jose Luis Munoz-Tapia. PlonKup: Reconciling PlonK with plookup. Technical Report 086, 2022. https://eprint.iacr.org/2022/086.

PHGR13. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, May 2013. http://ieeexplore.ieee.org/document/6547113/.

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

Ris22. Team RiscZero. RISC Zero : General-Purpose Verifiable Computing, 2022. https://risczero.com/.

Scr22. Team Scroll. Scroll, 2022. https://scroll.io/.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

SLST23. Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, and Bobbin Threadbare. The Tip5 Hash Function for Recursive STARKs, 2023.

STW23. Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with Lasso, 2023.

VM22. Triton VM. Triton VM, September 2022. Triton VM.

XZZ+19. Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology – CRYPTO 2019*, Lecture Notes in Computer Science, pages 733–764. Springer International Publishing, 2019.

ZBK+22. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup Arguments in Sublinear Time. Technical Report 621, 2022.

ZGK+22. Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly Optimal Lookup Arguments, 2022. https://eprint.iacr.org/2022/1565.

zkS22. zkSync Team. zkSync, 2022. https://zksync.io/.

ZXZS20. Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 859–876. IEEE, 2020.