Natively Compatible Super-Efficient Lookup Arguments and How to Apply Them

Matteo Campanelli, Dario Fiore, and Rosario Gennaro

[1] Matter Labs
matteo@matterlabs.dev
[2] IMDEA Software Institute, Madrid
dario.fiore@imdea.org
[3] The City College of New York
rosario@ccny.cuny.edu

Abstract. Lookup arguments allow an untrusted prover to commit to a vector $\mathbf{f} \in \mathbb{F}^n$ and show that its entries reside in a predetermined table $\mathbf{t} \in \mathbb{F}^N$. One of their key applications is to augment general-purpose SNARKs making them more efficient on subcomputations that are hard to arithmetize. In order for this "augmentation" to work out, a SNARK and a lookup argument should have some basic level of compatibility with respect to the commitment on $\mathbf{f}$. However, not all existing efficient lookup arguments are fully compatible with other efficient general-purpose SNARKs. This incompatibility can for example occur whenever SNARKs use multilinear extensions under the hood (e.g. Spartan) but the lookup argument is univariate in flavor (e.g. Caulk or `cq`).

In this paper we discuss how to widen the spectrum of "super-efficient" lookup arguments (where the proving time is independent of the size of the lookup table): we present a new construction inspired by `cq` and based on multilinear polynomial encodings (MLE). Our construction is the first lookup argument for any table that is also natively compatible with MLE-based SNARKs at comparable costs with other state-of-the-art lookup arguments, particularly when the large table is unstructured. This case arises in various applications, such as using lookups to prove that the program in a virtual machine is fetching the right instruction and when proving the correct computation of floating point arithmetic (e.g., in verifiable machine learning).

We also introduce a second more general construction: a compiler that, given any super-efficient lookup argument compatible with univariate SNARKs, converts it into a lookup argument compatible with MLE-based SNARKs with a very small overhead. Finally, we discuss SNARKs that we can compose with our constructions as well as approaches for this composition to work effectively.

1 Introduction

Lookup arguments are an important building block in modern SNARK constructions. They can be used as a tool to strengthen the efficiency of general-purpose proof systems [CBBZ23], but also as a tool of their own to construct

one from scratch [AST24, Whi]. In this work we are mainly motivated by the former aspect—lookup arguments used in combination with other SNARKs. At the same time, we provide techniques that may be useful for constructions where lookups are used on their own.

Quick background on lookup arguments Given a table $T \in \mathbb{F}^N$ and a vector $f \in \mathbb{F}^n$, we would like to check through a proof whether all the items in f appear in T. The verifier, in addition to the proof, takes as input succinct commitments c_T and c_f respectively to T and f. Verification should run in time significantly lower than N or n. Typically, these commitments exploit a polynomial encoding of T and f and are based on polynomial commitments such as [KZG10, PST13]. Very recently there has been a flourishing of lookup constructions in literature [CFF+24, ZBK+22a, ZBK+22b, PK22, GK22, Hab22, ZGK+22, EFG22, GW20, STW24].

Integrating SNARKs and lookup arguments Lookup arguments are useful because they allow one to efficiently prove specific computations. For example they allow more efficient operations that are non-native to field arithmetic such as XOR of bit strings or range checks. Range checks are useful to enforce that a variable w is (say) actually a byte and this can be enforced as a lookup by enforcing that w is in the table $T_{<256} := 0, 1, \ldots, 255$. This is in contrast to having to show 256 bit constraints of the type $(w_i - 0)(w_i - 1) = 0$. For the case of XOR, a lookup consists in checking that a triple (a, b, c) is in the table of all possible XOR values $(x, y, x \oplus y)$ (for bytes this table is of size $2^{16} = (2^8)^2$).

Lookup arguments are thus used as aids to SNARKs for general-purpose computations. The latter outsources part of the work to the lookup arguments as in the examples above. In the context of a larger proof we then show knowledge of a witness w such that some property $P(w)$ holds (proved with a general-purpose SNARKs) and in addition some subset at indices I of w satisfy a lookup relation (proved with a lookup argument) which we compactly denote with $w_I \prec T$.

In order for lookup arguments to be able to work together with SNARKs, they need to be able to refer to the *same witness* as the one for which the general property P is being shown. This is typically done through commitments (a succinct collision-resistant digest) and by three (idealized) facts: *(i)* The larger SNARK will commit to the witness w through a commitment scheme producing c_w; *(ii)* The lookup argument is able to express statements of the type c_f commits to an f that is contained in the table T; *(iii)* Another argument proves that the content of c_f actually refers to the parts of interest—the ones at indices I—in the witness committed in c_w.

For the lookup approach to be cost effective when integrated with SNARKs, it is crucial that steps *(ii)* and *(iii)* are both as cheap as possible. We refer to this as the *challenge of native compatibility*, to which we will get back later in this introduction.

Super-efficient lookup arguments An efficiency measure that is often desirable in lookup arguments is what we call *super efficiency*. We say a lookup argument is super-efficient if its proving time is independent of the size of the table. In particular, after a one-time preprocessing of the table, the prover can run only in time dependent on the size of the vector being looked up. This property makes a difference whenever the table size dominates the size of the witness proven by the general SNARK, and thus of the looked up vector. Use cases where this occurs are for example the lookup table sizes used for SHA256[4] or tables for wide range checks.

Due to the importance of this property, several lookup arguments in the state of the art are super-efficient [STW24, CFF+24, ZBK+22a, ZBK+22b, PK22, GK22, ZGK+22, EFG22]. With the only exception of [STW24], all these lookup arguments rely on the KZG polynomial commitment. This means that the c_f of steps *(ii)–(iii)* is a KZG commitment to a univariate polynomial that interpolates the vector f. KZG-based lookup arguments have two main advantages: (a) they exploit special properties of univariate polynomials and pairing-based verification to achieve super efficiency, and (b) they can be efficiently integrated with KZG-based SNARKs (e.g., [CHM+20, GWC19]) that are among the most popular and efficient ones.

If one wishes to integrate a super efficient lookup argument with SNARKs based on multivariate polynomial commitments (e.g., [Set20, CBBZ23, CGG+23]) the only choice left is the work in [STW24]. The latter though achieves super efficiency only for tables that present a specific, regular structure.

Therefore, the state of the art presents the following open question:

Can we construct a super-efficient lookup argument for arbitrary tables that is natively compatible with SNARKs based on multivariate polynomials?

Addressing this question is the main focus of this work.

The challenge of compatibility In the question above we emphasize the requirement that the solution should be "natively compatible". This is due to the wish of implementing the step *(iii)* above in the cheapest possible way. Otherwise, a straw-man solution could be to pick one of the existing super efficient lookup arguments where c_f is a KZG commitment, and then use the multilinear-based SNARK to additionally prove the link of step *(iii)* w.r.t. f in the KZG commitment c_f (e.g., by encoding the commitment computation as a circuit).

One obvious example of the problem of native compatibility in literature appears in the HyperPLONK paper [CBBZ23]. Here the authors, in order to achieve compatibility of lookups with multivariate polynomial commitments, need to redesign the Plookup lookup argument [GW20] (a lookup argument

[4] The table for SHA256 is of size approximately 2^{16} and requires approximately the order of the few thousands lookups *per input block* [hbc]. The number of lookups stays low in applications where not many hashes are performed. This is not uncommon: see for example proofs of hashing-to-primes in the setting of proving set membership[BBF19, BCF+21].

based on univariate encodings) for the multivariate setting. Through the solutions in this work, we can enable super-efficient lookups to be used in MLE-based SNARKS such as HyperPlonk and without the efficiency penalty due to the very general transformation above.

1.1 Our Contributions

We contribute two constructions: the first one is a "direct" construction, while the second is a general compiler which effectively yields a family of constructions.

Our first construction, μ-seek: we propose a direct construction of super-efficient lookup arguments natively compatible with multilinear encodings. This construction is inspired by cq and we call it μ-seek[5]. The construction is *modular* and can be instantiated with an appropriate multilinear polynomial commitment scheme (see Section 1.3.1 for a discussion).

The starting point of μ-seek is the technique of *cached quotients* in cq [EFG22]. The latter is an approach to efficiently compute at proving time a set of "committed quotients"[6] One lens through which to describe our approach is this: by looking at cq as a "*Polynomial IOP*"[7], it is possible to isolate the polynomial f encoding the vector being looked up: this is the vector whose polynomial encoding will be provided through a commitment for multilinear polynomials. This change is not enough by itself. In fact, some of the checks on this polynomial in cq work through a polynomial equation that leverages the representation of f as a univariate polynomial commitment. We then need to change how some of these checks are performed representing them as appropriate sumcheck equations and porting other polynomial commitments to the multilinear world.

The resulting construction can be seen as a modular version of cq instantiatable with a generic commitment scheme for multilinear polynomials with minimal requirements. The proving time is mostly dominated by one execution of the multivariate sumcheck protocol [GKR08] and computing two evaluation proofs of the multilinear polynomial commitment scheme. In Fig. 1 we describe a concrete breakdown of the performance of μ-seek when instantiated with the polynomial commitment obtained by applying the Gemini compiler [BCHO22] on top of KZG [KZG10].

Our generic compiler: our second contribution consists of a compiler which on input:

[5] The name of this construction (pronounced "*mee-seek*") is a pun in a style similar to cq [EFG22]. See also [RA].

[6] These quotients are the ones required in a crucial lemma from [Hab22].

[7] An abstract, information-theoretic blueprint underlying several efficient SNARKs where the prover is limited to providing oracle access to polynomials at each round of interaction with the verifier. Polynomial IOPs [BFS20] are also known—with minor differences—as Algebraic Holographic Proofs [CHM+20] or—in their most general version—as Polynomial Holographic IOPs [CFF+21]

– a super-efficient lookup argument over *univariate* polynomial commitments;
– a multivariate polynomial commitment;

produces a super-efficient lookup argument over *multivariate* polynomial commitments. From a technical standpoint the compiler mostly consists of a subargument "linking" the two representations (respectively univariate and multivariate) of the looked up vector through a multivariate sumcheck. We refer the reader to Section 4 for a technical overview.

Like μ-seek, our compiler-based lookup argument is modular: it can be instantiated with an arbitrary multilinear polynomial commitment scheme and with an arbitrary lookup argument—the only compatibility requirement, for efficiency reasons, is that both work over the same finite field. From the efficiency standpoint, our compiler has a profile similar to that of μ-seek: both require one sumcheck and a few proofs of evaluations for committed polynomials. Nonetheless, our compiler is slightly less efficient. Its (relatively) worse performance profile comes from its generality: it requires additional checks on top of a *black-box* lookup argument, whereas μ-seek is designed to integrate cq's framework within a multilinear framework as seamlessly as possible.

Applications to general-purpose SNARKs: One of the applications of lookup arguments is to support general-purpose SNARKs into efficiently proving arbitrary computations that are hard to represent as a purely arithmetic set of constraints. As an additional contribution, we show how to integrate our flavor of lookup arguments with SNARKs for R1CS (Rank-One Constraint Systems), a popular form of *intermediate representations* for circuits. For this purpose, we introduce a simple intermediate abstraction—projective lookup arguments—and we show how to instantiate it in two different ways, through existing work and through simple adaptations of our techniques in specific settings.

1.2 Related Work

Lookup arguments were introduced by Bootle, Cerulli, Groth, Jakobsen and Maller [BCG+18]. The work of Gabizon and Williamson [GW20], called Plookup, sparked an interest in the development of lookup arguments for SNARKs based on the Kate et al. (commonly known as KZG) polynomial commitment scheme [KZG10]. In this research line, Caulk [ZBK+22a] by Zapico et al. was the first to achieve a protocol where the prover complexity depends only logarithmically on the size of the bigger table, after preprocessing. A rapid sequence of works improved the prover's efficiency of lookup arguments for KZG commitments: Caulk+ by Posen and Kattis [PK22], flookup by Gabizon and Khovratovich [GK22], Baloo [ZGK+22] by Zapico et al., and cq by Eagen, Fiore, Gabizon [EFG22]. cq is among the most efficient and extremely succinct lookup arguments for arbitrary tables. Relying on the technique of logarithmic derivates of Haböck [Hab22], it achieves proofs of constant size and a prover complexity proportional only to the size of the smaller vector. Recently Campanelli et al. [CFF+24]

proposed cq+, a version of cq with even smaller proofs and zero-knowledge capabilities (among other extensions). All these schemes require preprocessing and support the lookup of vectors committed using the KZG polynomial commitment, and thus are efficiently applicable to SNARKs based on KZG.

Another important class of SNARKs rely on techniques based on the sum-check protocol [LFKN90] and multilinear polynomial commitments, e.g., [PST13]. To benefit of lookup capabilities, these schemes would need a lookup argument where the small vector is committed using a multilinear polynomial commitment. We mention a few works that address this problem. Hyperplonk [CBBZ23] gives a protocol for lookups inspired to Plookup [GW20]. Haböck [Hab22] introduces the logarithmic derivatives technique and uses it to construct a lookup argument that, while asymptotically similar to that of Hyperplonk, is concretely more efficient, especially for multi-column lookups. In a subsequent work, Papini and Haböck [PH23] shows a variant of Haböck's protocol [Hab22] using the GKR protocol and then extends it work with vectors committed using a univariate polynomial commitment scheme. All these works, however, have prover complexity linear in the size of the table, and thus fail to achieve the super-efficiency property which is the goal of this work. Recently, Setty, Thaler and Wahby [STW24] introduced a new lookup argument, called Lasso, that achieves prover complexity sub-linear in the size of the table, albeit this holds only for a class of tables that they call "structured". In a nutshell, structured tables are tables that can be succinctly described when encoded with a multilinear polynomial. Lasso is extremely efficient, and in particular more efficient than cq, for this class of tables. On the other hand, when it comes to handling arbitrary tables outside the structured class, Lasso still fails to achieve super efficiency and is therefore asymptotically slower than our solutions.

The concurrent work in [GNS24] has proposed *dual polynomial commitment schemes* which allows to link the opening in a univariate polynomial commitment with that in a multivariate one. Their techniques are radically different from the ones we apply in our compiler. We leave as an interesting future work direction to explore how to combine their approach with ours.

Scheme	**Table-specific Preprocessing**	**Proof size**	**Prover work** group, field	**Verifier work**	**MLE-compatible?**
Plookup [GW20]	–	$5\mathbb{G}1, 9\mathbb{F}$	$O(N), O(N \log N)$	$2P$	✗
Halo2 [BGH19]	–	$6\mathbb{G}_1, 5\mathbb{F}$	$O(N), O(N \log N)$	$2P$	✗
Caulk [ZBK+22a]	$O(N \log N)$ exps	$14\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$15m, O\left(m^2 + m\log(N)\right)$	$4P$	✗
Caulk+ [PK22]	$O(N \log N)$ exps	$7\mathbb{G}_1, 1\mathbb{G}_2, 2\mathbb{F}$	$8m, O\left(m^2\right)$	$3P$	✗
flookup [GK22]	$O\left(N \log^2 N\right)$ exps	$7\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$O(m), O\left(m \log^2 m\right)$	$3P$	✗
Baloo [ZGK+22]	$O(N \log N)$ exps	$12\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$14m, O\left(m \log^2 m\right)$	$5P$	✗
cq+ [CFF+24]	$O(N \log N)$ exps	$7\mathbb{G}_1, 1\mathbb{F}$	$8m, O(m \log m)$	$5P$	✗
Lasso w/ KZG + Gemini (unstructured table) [STW24]	–	$O(\log m)\mathbb{G}_1$ $\tilde{O}(\log m)\,\mathbb{F}$	$(c+1)m + cN^{\frac{1}{c}}, O(m+N)$	$2P$ $\tilde{O}(\log m)\,\mathbb{F}$	✓
this work μ-seek w/ KZG + Gemini	$O(N \log N)$ exps	$2(\log m + 3)\mathbb{G}_1$ $6(\log m + 1)\mathbb{F}$	$O(m), O(m)$	$2P$ $O(\log m)\mathbb{G}_1$	✓

Fig. 1. Comparison among different lookup arguments. Our proof sizes include optimizations through batching techniques. A comparison with Lasso for structured tables is in Section 1.3.3.

1.3 Discussion

1.3.1 On the generality of our constructions Our two constructions offer some flexibility on the choice of the multivariate polynomial commitment scheme through which to commit the vector to be looked up. For μ-seek the only constraint this multivariate polynomial commitment mPC needs to satisfy has to do with the field $\mathbb{F}$ of the underlying univariate polynomial commitment uPC in the construction (KZG). In particular, mPC will have to support the same field $\mathbb{F}$. A similar constraint holds for our general compiler: mPC needs to be over the same field as the underlying univariate polynomial commitment in the lookup argument in input.

There are several instantiation candidates that satisfy these conditions. Possible candidates include: the multivariate version of KZG, i.e., the PST commitment scheme [PST13]; the commitment scheme obtained by applying the Gemini compiler to KZG [BCHO22]; the Testudo polynomial commitment [CGG$^+$23]; any field-agnostic multivariate polynomial commitment scheme like Orion [XZS22] or Brakedown [GLS$^+$23].

1.3.2 SNARKs compatible with our constructions The promise of natively compatible lookup arguments is to easily augment general purpose SNARKs with lookups through easy "plug-and-play" composition[8]. Our work proposes constructions that are compatible with SNARKs where the proof includes a multivariate polynomial commitment to the witness, i.e., the proof can be parsed as a $(\mathsf{cm}, \pi_{\mathrm{rst}})$ where cm is a commitment to the witness through a multivariate polynomial commitment mPC.

We cite two such examples of constructions with this property. One is HyperPLONK [CBBZ23], a multivariate version of PLONK. Here we can apply the μ-seek construction instantiated with whichever mPC we use as underlying polynomial commitment scheme for HyperPLONK (e.g., KZG+Gemini). Another example is Testudo [CGG$^+$23], a hybrid of Groth16 [Gro16], Spartan [Set20] and Inner Product Arguments [BMM$^+$21] with short CRSs and optimized for data-parallel computations. Here we can apply μ-seek instantiated with the Testudo polynomial commitment scheme as mPC.

1.3.3 Detailed comparison with Lasso For the general case of "unstructured" tables, we have pointed out that Lasso has a cost profile that grows linearly with the table size N in terms of field operations, and sublinearly in terms of group operations ($O(cN^{1/c})$ for an arbitrary constant c). These group operations, it should be noted, are mostly exponentiations with small exponents. In the case of "structured" tables[9] Lasso will avoid this linear cost, but will still

[8] This approach has been explicitly studied and successfully applied in several works [CHA22, CFF$^+$21, ABC$^+$22, CHAK23, FT22, CFQ19, AGM18, BCF$^+$21, CFH$^+$22].

[9] Loosely defined as tables that have a succinct representation allowing the verifier to evaluate the multilinear extension of T on its own.

pay $O(cN^{1/c})$ group operations (again, with the caveat above about the size of the exponents). On the other hand, our construction μ-seek, thanks to preprocessing, has proving time which is *independent* of N. Because our protocol is heavier in group operations than Lasso, it is hard to estimate where this asymptotic advantage translates in a real advantage in practical applications, especially in the case of unstructured tables where Lasso is already sublinear.

We remark, however, that there are several interesting applications that could benefits from efficient lookups over unstructured tables. One is floating point arithmetic, which is particularly relevant in the case of verifiable machine learning (a table of size 2^{32} for 16-bit floating point arithmetic)[10]. Another is using lookups to implement fetching the next instruction in a zkVM program: in this case the table is the sequence of instructions and could be arbitrary.

Outline of this paper

In Section 2 we present preliminaries on some of the abstractions we will adopt (the language of commit-and-prove SNARKs and polynomial commitments formalized under that language) as well as background on some of our basic techniques, including multivariate sumchecks. In Section 3 we present our main construction, μ-seek. Our generic compiler is presented in Section 4. In Section 5 we describe methods to augment SNARKs for R1CS with some of the lookup arguments presented in this paper.

2 Preliminaries

2.1 Notation

We denote the security parameter by λ and a negligible function by $\mathsf{negl}(\lambda)$. If $m \in \mathbb{N}$ we denote by $[m]$ the set $\{1, 2, \ldots, m\}$. If $\mathbf{X} = (X_1, \ldots, X_\mu)$ is a vector of μ formal variables and $I \subseteq [\mu]$ then we denote by $\mathbf{X}^I$ the monomial $\prod_{i \in I} X_i$.

2.2 Commit-and-Prove SNARKs [CFQ19]

A commitment scheme is a tuple of algorithm $\mathsf{CS} = (\mathsf{KGen}, \mathsf{Com})$ where the first algorithm samples a commitment key ck and the second algorithm, upon input of the commitment key, a message p and opening material o, outputs a commitment c. The basic notions of security for the commitment scheme are (perfect) *hiding* and (computational) *binding*. The former property states that no (unbounded) adversary can distinguish commitments of two different messages when the opening materials are sampled at random from their domain, the latter property states that no (polynomial time) adversary, upon input of the commitment key, can find two different messages and two opening materials that commit to the same commitment. We will not require hiding in this work.

[10] While it is conceivable that this table could be expressed in a more succinct way, we are not aware of ways of doing it.

Following Groth et al.[GKM+18], we define a relation $\mathcal{R}$ verifying triple $(\mathsf{pp}; x; w)$. We say that w is a witness to the instance x being in the relation defined by the parameters pp when $(\mathsf{pp}; x; w) \in \mathcal{R}$ (equivalently, we sometimes write $\mathcal{R}(\mathsf{pp}; x; w) = 1$). For example, the parameters pp could be the description of a bilinear group, or additionally contain a commitment key for a commitment scheme or a common reference string. Whenever it is clear of the context, we will write $\mathcal{R}(x; w)$ as a shortcut for $\mathcal{R}(\mathsf{pp}; x; w)$.

Briefly speaking, Commit-and-Prove SNARKs (CP-SNARKs) are zkSNARKs whose relations verify predicates over commitments [CFQ19]. Given a commitment scheme CS, we consider relations $\mathcal{R}$ whose instances are of the form $x = ((\mathsf{c}_j)_{j\in[\ell]}, \hat{x})$, where we can unambiguously parse the witness $w = ((p_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]})$ for some $\ell \in \mathbb{N}$ with $\forall j : p_j$ is in the domain of a commitment scheme CS, and such that there exists a polynomial time relation $\hat{\mathcal{R}}$ such that for $\hat{w} = (p_j)_{j\in[\ell]}$ and pp a tuple that includes the commitment key ck of CS:

$$\mathcal{R}(\mathsf{pp}; x; w) = 1 \iff \hat{\mathcal{R}}(\mathsf{pp}; \hat{x}; \hat{w}) = 1 \wedge \forall j \in [\ell] : \mathsf{c}_j = \mathsf{Com}(\mathsf{ck}, p_j, o_j).$$

We refer to a relation $\hat{\mathcal{R}}$ as derived above as a *Commit-and-Prove* (CP) relation. Given a CP-relation $\hat{\mathcal{R}}$ and a commitment scheme CS, we can easily derive the *associated* NP-relation $\mathcal{R}$. Instances of NP-relations may contain only commitments. Therefore, using the notation above, the instances of the associated CP-relation are empty strings ε, namely, $\hat{\mathcal{R}}$ is a predicate over the committed witness. To avoid cluttering the notation, in these cases, we may omit the (empty) instance and simply write $\hat{\mathcal{R}}(\mathsf{pp}, \hat{w})$.

A CP-SNARK for $\hat{\mathcal{R}}$ over a commitment scheme CS is a zkSNARK for the associated relation $\mathcal{R}$ as described above. More in detail, we consider a tuple of algorithms $\mathsf{CP} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ where:

- $\mathsf{Setup}(\mathsf{ck}) \to \mathsf{srs}$ is a probabilistic algorithm that takes as input a commitment key ck for CS and it outputs $\mathsf{srs} := (\mathsf{ek}, \mathsf{vk}, \mathsf{pp})$, where ek is the evaluation key, vk is the verification key, and pp are the parameters for the relation $\mathcal{R}$ (which include the commitment key ck).
- $\mathsf{Prove}(\mathsf{ek}, x, w) \to \pi$ takes an evaluation key ek, a statement x, and a witness w such that $\mathcal{R}(\mathsf{pp}, x, w)$ holds, and returns a proof π.
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to b$ takes a verification key, a statement x, and either accepts $(b = 1)$ or rejects $(b = 0)$ the proof π. The length of vk_{ind} is $\mathsf{poly}(\lambda, \log |ind|)$.

In some cases, the Setup algorithm would simply (and deterministically) re-parse the commitment key ck information. In these cases, we might omit Setup and refer to the CP-SNARK as a tuple of two algorithms.

We require a CP-SNARK to be correct and knowledge sound.

Definition 1 (Correctness). *A CP-SNARK is correct if for any* pp, x, w *such that* $\mathcal{R}(\mathsf{pp}, x, w) = 1$ *then*

$$\Pr\begin{bmatrix}\mathsf{ck} \leftarrow \mathsf{CS.KGen}(1^\lambda, d); \mathsf{srs} \leftarrow \mathsf{CP.Setup}(\mathsf{ck});\\ \pi \leftarrow \ \mathsf{CP.Prove}(\mathsf{srs}, \mathsf{pp}, x, w);\\ \mathsf{CP.Verify}(\mathsf{srs}, \mathsf{pp}, x, \pi) = 1\end{bmatrix} = 1$$

Definition 2 (Knowledge Soundness). *A CP-SNARK is knowledge extractableif for any PPT adversary $\mathcal{A}$, there exists a polynomial-time (not necessarily uniform) extractor $\mathcal{E}$ such that:*

$$\Pr\left[\begin{array}{l}\mathsf{ck} \leftarrow \mathsf{CS.KGen}(1^\lambda, d); \mathsf{srs} \leftarrow \mathsf{CP.Setup}(\mathsf{ck}); \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{srs}); w \leftarrow \mathcal{E}(\mathsf{srs}) \\ \mathsf{CP.Verify}(\mathsf{srs}, x, \pi) = 1 \wedge \neg\mathcal{R}(\mathsf{pp}, x, w)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Indexed Relations and Universal CP-SNARKs We extend the notion of relations to indexed relations [CHM+20]. We define a polynomial-time indexed relation $\mathcal{R}$ verifying tuple (pp, ind, x, w). We say that w is a witness to the instance x being in the relation defined by the pp and index ind when $(\mathsf{pp}, ind, x, w) \in \mathcal{R}$ (equivalently, we sometimes write $\mathcal{R}(\mathsf{pp}, ind, x, w) = 1$).

Briefly, we say that a CP-SNARK is *universal* if there exists a deterministic algorithm Derive that takes as input an srs and an index ind, and outputs a specialized reference string $\mathsf{srs}_{ind} = (\mathsf{vk}_{ind}, \mathsf{ek}_{ind})$ where vk_{ind} is a succinct verification key and ek_{ind} is a proving key for such an index. Moreover, we require that the verifier Verify (resp. the P) of a Universal CP-SNARK takes as additional input the specialized verification key vk_{ind} (resp. the specialized ek_{ind}).

Consider the relation $\mathcal{R}'$ such that $\mathcal{R}'(\mathsf{pp}, (ind, x), w) \iff \mathcal{R}(\mathsf{pp}, ind, x, w)$, the tuple of algorithms $(\mathsf{KGen}, \mathsf{Prove}, \mathsf{Verify}')$ is an argument system for the relation $\mathcal{R}'$ and Verify' is the algorithm that upon input srs, instance (ind, x) and a proof π, first runs Derive on srs and index ind, then runs $\mathsf{Verify}(\mathsf{vk}_{ind}, x, \pi)$.

2.3 Lookup arguments

We formalize lookup arguments as CP-SNARKs for the indexed relation

$$\mathcal{R}_{\mathsf{lookup}} := \{(\mathbf{t}; \hat{\mathsf{c}}_{\mathbf{f}}; (\mathbf{f}, o_{\mathbf{f}})) : \mathbf{f} \prec \mathbf{t} \wedge \mathsf{c}_{\mathbf{f}} = \mathsf{Com}(\mathsf{ck}, \mathbf{f}, o_{\mathbf{f}})\}$$

where we write $\mathbf{f} \prec \mathbf{t}$ to compactly denote that there exists a (multi) set $K = \{k_1, \ldots, k_n\} \subseteq [N]$ such that for every $j \in [n]$ $\mathbf{f}_j = \mathbf{t}_{k_j}$.

2.4 Polynomial Commitment Schemes

We define polynomial commitments using the CP-SNARKs framework. We will not require hiding properties from the commitments schemes. Below $\mathbb{F}$ denotes a finite field which will be defined by the context in which we use them. We assume that parameters of the polynomials being committed (such as maximum degree or maximum number of variables) are determined by an appropriate function of the security parameters λ.

Definition 3 (Univariate polynomial commitment). *A univariate polynomial commitment scheme is a pair* $(\mathsf{uPC}, \mathsf{CP}_u)$ *where:*

– $\mathsf{uPC} = (\mathsf{KGen}, \mathsf{Com})$ *is a binding commitment scheme whose message space is* $\mathbb{F}^{<d}[X]$ *where* d *is a function of the security parameter* λ.
– CP_u *is a CP-SNARK over* uPC *for the relation* $\mathcal{R}_{eval}(x, y; p) :=$ "$p(x) = y$".

Definition 4 (Multilinear polynomial commitment). *A multilinear polynomial commitment scheme is a pair* $(\mathsf{mPC}, \mathsf{CP}_m)$ *where:*

– $\mathsf{mPC} = (\mathsf{KGen}, \mathsf{Com})$ *is a binding commitment scheme whose message space is* $\mathbb{F}[X_1, \ldots, X_\mu]$ *of multivariate polynomials of individual degree 1 where* μ *is a function of the security parameter* λ.
– CP_m *is a CP-SNARK over* mPC *for the evaluation relation* $\mathcal{R}_{eval}((x_i)_{i\in[\mu]}, y; p) :=$ "$p(x_1, \ldots, x_\mu) = y$".

For mPC we overload notation and use it to commit to vectors by writing $\mathsf{mPC.Com}(\mathsf{ck}, \mathbf{f})$ as a shortcut notation for $\mathsf{mPC.Com}(\mathsf{ck}, \tilde{f})$ where $\tilde{f} \in \mathbb{F}[X_1, \ldots, X_\mu]$ is the *unique multilinear extension* of vector $\mathbf{f} \in \mathbb{F}^n$, i.e., the unique polynomial such that for all $\mathbf{b} \in \{0,1\}^\mu$ we have $\tilde{f}(\mathbf{b}) = \mathbf{f}_{\sum_j b_j 2^j}$, with $\mu = \log n$.

2.5 (Vanilla) Sumcheck Protocol

Let $p(X_1 \ldots, X_\mu)$ be a multilinear[11] polynomial in μ variables defined over a field $\mathbb{F}$. Consider the value $a = \sum_{\mathbf{b}\in\{0,1\}^\mu} p(\mathbf{b})$, i.e., the sum of the value of p on all the vertices of the Boolean hypercube. This computation takes $n = 2^\mu$ time and the sumcheck protocol [LFKN90] described in Figure 2, is a way for a Prover to convince a Verifier that a is correct in $O(\mu)$ time, plus a *single* query to the polynomial p on a random point in $\mathbb{F}^\mu$.

2.6 Generalized Sumcheck Protocol

In the previous section we described a sumcheck protocol for the case where the verifier has oracle access to the polynomial in the sum. In some cases, this polynomial will not be available "directly" to the verifier. This is the case, for example, if the verifier is interesting in checking the sum

$$\sum_{\mathbf{b}} (p(\mathbf{b}) + \gamma) \cdot q(\mathbf{b}) \cdot f(\mathbf{b}) + f'(\mathbf{b}) \cdot t(\mathbf{b}) \tag{1}$$

where p, q, t are polynomials for which the verifier has a *commitment*, γ is a public value and f, f' are public polynomials that the verifier can compute on their own.

Here we define a sumcheck protocol for the generalized sumcheck relation for expressions like the above and that allows for committed polynomials. The

[11] We only care about multilinear polynomials for the context of this work but the sumcheck protocol can be run on any multivariate polynomial (with a different efficiency profile).

1. P sends the univariate polynomial $p_1(X) = \sum_{\mathbf{b}\in\{0,1\}^{\mu-1}} p(X, \mathbf{b})$.
2. V checks that $p_1(0) + p_1(1) = a$ and sends back $r_1 \in_R \mathbb{F}$.
3. P sends the polynomial $p_2(x) = \sum_{\mathbf{b}\in\{0,1\}^{\mu-2}} p(r_1, X, \mathbf{b})$.
4. V checks that $p_2(0) + p_2(1) = p_1(r_1)$ and sends back $r_2 \in_R \mathbb{F}$.

...

5. At round j P sends the polynomial $p_j(X) = \sum_{\mathbf{b}\in\{0,1\}^{\mu-j}} p(r_1, \ldots, r_{j-1}, X, \mathbf{b})$.
6. V checks that $p_j(0) + p_j(1) = p_{j-1}(r_{j-1})$ and sends back $r_j \in_R \mathbb{F}$.

...

7. At the last round P sends the polynomial $p_{\mu-1}(x) = p(r_1, \ldots, r_{\mu-1}, X)$.
8. V checks that $p_{\mu-1}(0) + p_{\mu-1}(1) = p_{\mu-2}(r_{\mu-2})$, selects $r_\mu \in_R \mathbb{F}$ and checks that $p_{\mu-1}(r_\mu) = p(r_1, \ldots, r_\mu)$ via a single query to p.

Fig. 2. The Sumcheck Protocol

relation is parametrized by a (multilinear) polynomial commitment PC and is defined as follows:

$$\mathcal{R}_{\mathrm{SC}}\left(\mathsf{pp}, y, f_1, \ldots, f_d, \mathcal{J}_1, \ldots, \mathcal{J}_d, c_1, \ldots, c_\ell; g_1, \ldots, g_\ell\right) :=$$
$$\sum_{\mathbf{b}\in\{0,1\}^\mu} \left(\sum_{k=1}^{d} f_k\left(\mathbf{b}\right) \cdot \prod_{j\in\mathcal{J}_k} g_j\left(\mathbf{b}\right) \right) \wedge$$
$$c_1 = \mathsf{PC.Commit}(\mathsf{pp}, g_1) \wedge \cdots \wedge c_\ell = \mathsf{PC.Commit}(\mathsf{pp}, g_\ell)$$

Above, d is a natural number, the f_k-s are public multilinear polynomials and the $\mathcal{J}_k$ are subsets of $[\ell]$. All these elements are used to describe the public expression[12] (such as the one in Eq. (1)) on which we are performing the sumcheck.

A protocol for $\mathcal{R}_{\mathrm{SC}}$ is described in Fig. 3. This protocol is folklore. For its security analysis we refer the reader to [CFQ19, Section 5.2].

Remark 1 (On Notation for proving/verifying $\mathcal{R}_{SC}$). The formal description of the statements for $\mathcal{R}_{\mathrm{SC}}$ is quite involved when expressed through the subsets $\mathcal{J}_k$ as above. Nonetheless the expressions proved are often simple enough and their formal representation should be immediate to the reader. For this reason, in our protocols, we adopt a more readable solution whenever proving/verifying $\mathcal{R}_{\mathrm{SC}}$: we simply provide a description of the expression we are proving in the pseudocode and make explicit the public/witness inputs to be passed to the SNARK algorithms.

[12] In our pseudocode, whenever using the generalized sumcheck, we will use a simpler notation that should make the expression obvious from the context. We present the formal version here for sake of completeness.

1. P and V run the protocol in Fig. 2 with polynomial $p(\mathbf{X}) := \sum_{k=1}^{d} f_k(\mathbf{X}) \cdot \prod_{j \in \mathcal{J}_k} g_j(\mathbf{X})$. On the last round, since the verifier has no direct access to p it proceeds to compute $p(r_1, \ldots, r_\mu)$ through the following steps.
2. P sends V the tuples $(y_i, \pi_i)_{i \in [\ell]}$ where $y_i = g_i(r_1, \ldots, r_\mu)$ and π_i is a polynomial evaluation proof that $y_i = g_i(r_1, \ldots, r_\mu)$.
3. V checks each pair (y_i, π_i) with respect to commitment c_i
4. V computes $y^* = \sum_{k=1}^{d} f_k(r_1, \ldots, r_\mu) \cdot \prod_{j \in \mathcal{J}_k} y_j$ and checks that $y^* = p_{\mu-1}(r_\mu)$ (where $p_{\mu-1}$ is defined as in Fig. 2)

Fig. 3. The Generalized Sumcheck Protocol for $\mathcal{R}_{\text{SC}}$.

Proof size The proof size of the protocol in Fig. 3 is

$$\ell \cdot |\pi_{\text{mPC}}| + \mu(\ell + 2)|\mathbb{F}| + \ell|\mathbb{F}|$$

where $|\pi_{\text{mPC}}|$ is the size of one opening proof for the multilinear polynomial commitment.

3 Our Construction μ-seek

In this section we describe our construction μ-seek of a lookup argument for multilinear polynomial commitments. Namely, given a commitment scheme mPC for multilinear polynomials, μ-seek is a CP-SNARK for the following indexed relation

$$\mathcal{R}_{\mathsf{lookup}} := \{(\mathbf{t}; \hat{\mathsf{c}}_{\mathbf{f}}; \mathbf{f}) : \mathbf{f} \prec \mathbf{t}, \hat{\mathsf{c}}_{\mathbf{f}} = \mathsf{mPC.Com}(\mathsf{ck}, \mathbf{f})\} \tag{2}$$

Remark 2. In the relation above we do not explicitly include N and n, the table and lookup vector size in order to simplify notation. These are, however, assumed to be parameters of the relation and can be embedded in the evaluation and verification keys.

The main ingredients of our construction are:

- The commitment scheme mPC for multilinear polynomials with $\mu = \log n$ variables. This is the scheme we use for committing to the vectors of which we want to test lookup relations (and thus also the vectors to be used in larger SNARK protocols). To commit to a vector $\mathbf{f}$ we use its unique multilinear extension $\tilde{f}$ (see below).
- A CP-SNARK CP_{SC} for the generalized sumcheck relation over polynomials committed with mPC. As mentioned in Section 2.6, CP_{SC} can be in turn realized from a CP-SNARK CP_m for the basic polynomial evaluation functionality.

– The KZG polynomial commitment for univariate polynomials of degree $< N$. We use KZG to commit to the table $\mathbf{t}$ and store this commitment in the verification key $\mathsf{vk}_{\mathbf{t}}$ for the indexed relation defined by $\mathbf{t}$. We commit to $\mathbf{t}$ using Lagrange interpolation over a multiplicative subgroup $\mathbb{K} \subset \mathbb{F}$ of order N. We emphasize that KZG is not the main commitment scheme of μ-seek; it rather acts as an auxiliary building block for our protocol.

For these building blocks to work together we only require that mPC supports polynomials over the same finite field $\mathbb{F}$ used in KZG.

Notation and preliminaries For a vector $\mathbf{v} \in \mathbb{F}^n$, its *multilinear extension*, denoted $\tilde{v}$, is the unique polynomial that evaluates to $\mathbf{v}$ on the boolean hypercube, i.e., for $\mu = \log n$, $\tilde{v} \in \mathbb{F}[X_1, \ldots, X_\mu]$ such that for all $\mathbf{b} \in \{0,1\}^\mu$ $\tilde{v}(\mathbf{b}) = \mathbf{f}_{\sum_j b_j 2^j}$.

To commit a vector with mPC we use multilinear extensions; nevertheless we abuse notation and write $\mathsf{mPC.Com}(\mathsf{ck}, \mathbf{v})$ as a shortcut for $\mathsf{mPC.Com}(\mathsf{ck}, \tilde{v})$ where $\tilde{v}$ is $\mathbf{v}$'s multilinear extension.

Let $\mathbb{K} \subset \mathbb{F}$ be a multiplicative subgroup of order N, and let ω be a fixed generator of $\mathbb{K}$. Then we denote by $\nu_{\mathbb{K}}(X) = (X^N - 1)$ the vanishing polynomial of $\mathbb{K}$, and by $\lambda_j^{\mathbb{K}}(X) = \nu_{\mathbb{K}}(X)\omega_N^{j-1}/(N(X - \omega_N^{j-1}))$ the Lagrange polynomials of $\mathbb{K}$. To commit to a vector $\mathbf{t} \in \mathbb{F}^N$ with KZG, we use the polynomial $\mathsf{T}(X) = \sum_j \mathbf{t}_j \cdot \lambda_j^{\mathbb{K}}(X)$.

We use the univariate sumcheck lemma from Aurora [BCR+19] and the following lemma from [Hab22].

Lemma 1 (Univariate Sumcheck, [BCR+19]). *Let $\mathbb{H}$ be multiplicative subgroup of $\mathbb{F}$ of order n. For any $P(X)$ of degree $< n$ we have*

$$\sum_i P(\omega_n^i) = n \cdot P(0)$$

Lemma 2 (Set inclusion, [Hab22]). *Let $\mathbb{F}$ be a field of characteristic $p > N$, and suppose that $(a_i)_{i=1}^N$, $(b_i)_{i=1}^N$ are arbitrary sequences of field elements. Then $\{a_i\} \subseteq \{b_i\}$ as sets (with multiples of values removed), if and only if there exists a sequence $(m_i)_{i=1}$ of field elements from $\mathbb{F}_p \subseteq \mathbb{F}$ such that*

$$\sum_{i=1}^N \frac{1}{X - a_i} = \sum_{i=1}^N \frac{m_i}{X - b_i} \tag{3}$$

in the function field $\mathbb{F}(X)$. Moreover, we have equality of the sets $\{a_i\} = \{b_i\}$, if and only if $m_i \neq 0$, for every $i = 1, \ldots, N$.

Bilinear groups Finally, our construction relies on type-3 bilinear groups that are defined by a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q, and we set $\mathbb{F} = \mathbb{Z}_q$. P_1, P_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$. $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently-computable non-degenerate bilinear map, and there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We

use the implicit notation $[a]_i \coloneqq aP_i$, for elements in $\mathbb{G}_i, i \in \{1, 2, T\}$ and set $P_T \coloneqq e(P_1, P_2)$.

For security we rely that in these groups the Power Discrete Logarithm assumption, defined below, holds.

Definition 5 (Power Discrete Logarithm [Lip12]). *Let $d_1(\lambda), d_2(\lambda) \in \mathsf{poly}$. The (d_1, d_2)-PDL (Power Discrete Logarithm) assumption holds in the group described by $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ if for any non-uniform PPT $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{pdl}}_{d_1,d_2,\mathcal{A}}(\lambda) := \Pr\left[s \leftarrow\!\$\, \mathbb{F}^* : \mathcal{A}\left(\mathsf{pp}, \left[(s^i)_{i=0}^{d_1}\right]_1, \left[(s^i)_{i=0}^{d_2}\right]_2\right) = s \right] = \mathsf{negl}(\lambda) \ .$$

Finally, we prove the security of our construction in the algebraic group model [FKL18]. An algorithm $\mathcal{A}$ is called *algebraic* if for all group elements that $\mathcal{A}$ outputs, it additionally provides the representation relative to all previously received group elements. That is, if $\mathbf{h} \in \mathbb{G}_i^N$ is the list of group elements received by $\mathcal{A}$, then for any group element $\mathsf{z} \in \mathbb{G}_i$ returned in output, the adversary must also provide a vector $\mathbf{c} \in \mathbb{F}^N$ such that $\mathsf{z} = \langle \mathbf{c}, \mathbf{h} \rangle$.

Our protocol The protocol is described in Figure 4. Below we provide a textual description of its main steps along with some intuitive explanations.

Setup We assume a universal SRS that consists of an SRS for KZG, $\mathsf{srs}_{\mathsf{KZG}} = (\left[(s^j)_{j=0}^{N-1}\right]_1, \left[(s^j)_{j=0}^{N}\right]_2)$, and an SRS $\mathsf{srs}_{\mathsf{mPC}}$ for the mPC scheme.

Derivation We encode the table $\mathbf{t}$ in the univariate polynomial

$$\mathsf{T}(X) := \sum_{j=1}^{N} = \mathbf{t}_j \lambda_j^{\mathbb{K}}(X)$$

we store its KZG commitment $[\mathsf{T}(s)]_2$ (in group $\mathbb{G}_2$), and then we compute the cached quotients of $\mathsf{T}(X)$ (using the $O(N \log N)$ algorithm of [EFG22]):

$$[Q_j(s)]_1 \ \text{ where } \ Q_j(X) := \frac{(\mathsf{T}(X) - \mathbf{t}_j)\lambda_j^{\mathbb{K}}(X)}{\nu_{\mathbb{K}}(X)} \ .$$

Additionally we precompute the commitments $\left[r_j^{\mathbb{K}}(s)\right]_1$ to the polynomials $r_j^{\mathbb{K}}(X) = (\lambda_j^{\mathbb{K}}(X) - \lambda_j^{\mathbb{K}}(0))/X$, which help computing a KZG evaluation proof on 0 for sparse polynomials in Lagrange basis.

Prover The first steps of the prover algorithm closely follow the cq protocol:

- the prover commits to the *sparse* vector $\mathbf{m}$ of multiplicities, such that $\sum_{j=1}^{N} \frac{m_i}{\mathbf{t}_i + X} = \sum_{j=1}^{n} \frac{1}{\mathbf{f}_i + X}$;

– the verifier sends a random challenge β. Then, the prover's goal is showing

$$\sum_{j=1}^{N} \frac{m_i}{\mathbf{t}_j+\beta} = \sum_{j=1}^{n} \frac{1}{\mathbf{f}_i+\beta} \tag{4}$$

This is done by committing to vectors $\mathbf{a}$, $\mathbf{b}$ such that

$$a_j = \frac{m_j}{\mathbf{t}_j+\beta} \quad \text{and} \quad b_i = \frac{1}{\mathbf{f}_i+\beta} \ . \tag{5}$$

The commitment to $\mathbf{a}$ is done using KZG, i.e., by committing to the interpolation $A(X)$ of $\mathbf{a}$ over $\mathbb{K}$, that is $[A(s)]_1$. This is like in [EFG22]. The commitment to $\mathbf{b}$ is done using the multilinear commitment mPC, which differs from cq.

To prove (4), we perform the following steps:

1. Prove that $[A(s)]_1$ is wellformed, that is $\forall j = 1...N$, $A(\omega^{j-1})(\mathsf{T}(\omega^{j-1})+\beta) = M(\omega^{j-1})$, or equivalently $A(X)(\mathsf{T}(X)+\beta) = M(X) \mod \nu_{\mathbb{K}}(X)$. Following cq, we commit to the quotient $Q_A(X)$ in $O(n)$ time, using the cached quotient technique, i.e., compute $[Q_A(s)]_1$ relying on the precomputed $[Q_j(s)]_1$.
2. Compute $z \leftarrow \sum_{j=1}^{N} a_j = \sum_{i=1}^{n} b_i$
3. Prove that $z = \sum_{j=1}^{N} A(\omega^{j-1})$. By relying on the univariate sumcheck lemma and the fact that the SRS imposes $\deg(A) < N$, we have $\sum_{j=1}^{N} A(\omega^{j-1}) = N \cdot A(0)$.
4. Prove that $\hat{\mathsf{c}}_{\mathbf{b}}$ is wellformed w.r.t. $\hat{\mathsf{c}}_{\mathbf{f}}$. The idea is to reduce it to a sumcheck statement on the multilinear polynomials $\tilde{f}(\mathbf{X})$ and $\tilde{b}(\mathbf{X})$ that encode $\mathbf{f}$ and $\mathbf{b}$ respectively. Namely, we want to prove

$$\forall \mathbf{h} \in \{0,1\}^{\mu} : \tilde{b}(\mathbf{h})(\tilde{f}(\mathbf{h}) + \beta) = 1$$

which we can test, using a random challenge $\boldsymbol{\rho} \leftarrow\!\!\$\ \mathbb{F}^{\mu}$, as the following sumcheck

$$0 = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} (\tilde{b}(\mathbf{h})(\tilde{f}(\mathbf{h}) + \beta) - 1)\chi(\boldsymbol{\rho}, \mathbf{h})$$

where $\chi(\mathbf{X}, \mathbf{h}) := \prod_{j=1}^{\mu}((1-X_j)(1-h_j)+X_j h_j)$ is the multilinear extension of the identity function that returns 1 on $\mathbf{X} = \mathbf{h}$ and 0 on a boolean $\mathbf{h}' \neq \mathbf{h}$.
5. Prove that $z = \sum_{i=1}^{n} b_i$. This is a standard sumcheck to show that

$$z = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{b}(\mathbf{h})$$

As an optimization, we batch the last two steps above, with a random challenge $\tau \leftarrow\!\!\$\ \mathbb{F}$, into a single sumcheck:

$$z = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{b}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{b}(\mathbf{h}) \cdot (\tilde{f}(\mathbf{h}) + \beta) - 1)$$

$\underline{\mathsf{Setup}}(1^\lambda, N, n)$**:**

$s \leftarrow\$ \mathbb{F}$; $\mathsf{srs}_{\mathsf{KZG}} = (\left[(s^j)_{j=0}^{N-1}\right]_1, \left[(s^j)_{j=0}^{N}\right]_2)$;

$\mathsf{srs}_{\mathsf{mPC}} \leftarrow\$ \mathsf{mPC.Setup}(1^\lambda, \log n)$

$\underline{\mathsf{Derive}}(\mathsf{srs}, \mathbf{t})$**:**

Define $\mathsf{T}(X) := \sum_{j=1}^{N} = \mathbf{t}_j \lambda_j^{\mathbb{K}}(X)$;

Let $\left\{ r_j^{\mathbb{K}}(X) = \frac{\lambda_j^{\mathbb{K}}(X) - \lambda_j^{\mathbb{K}}(0)}{X},\ Q_j(X) = \frac{(\mathsf{T}(X) - \mathbf{t}_j)\lambda_j^{\mathbb{K}}(X)}{\nu_{\mathbb{K}}(X)} \right\}_{j \in [N]}$;

Compute $\mathsf{ek}_{\mathbf{t}} := \left[(\lambda_j^{\mathbb{K}}(s), r_j^{\mathbb{K}}(s), Q_j(s))_{j=1}^{N}\right]_1$; $\mathsf{vk}_{\mathbf{t}} := ([1, s, \nu_{\mathbb{K}}(s), \mathsf{T}(s)]_2, \mathbf{t})$

Return $(\mathsf{ek}_{\mathbf{t}}, \mathsf{vk}_{\mathbf{t}})$.

$\underline{\mathsf{Prove}}(\mathsf{ek}_{\mathbf{t}}, \hat{\mathsf{c}}_{\mathbf{f}}, \mathbf{f})$**:**

$\mathbf{m} \leftarrow (m_1, \ldots, m_N)$ s.t. $\forall j : \mathbf{t}_j$ appears m_j times in $\mathbf{f}$;

$[M(s)]_1 \leftarrow \sum_{j=1}^{N} m_j \cdot \left[\lambda_j^{\mathbb{K}}(s)\right]_1$; // KZG-commit to $\mathbf{m}$

$\underline{\text{Send}}$ $[M(s)]_1$;

$\underline{\text{Get}}$ $\beta \leftarrow\$ \mathbb{F}$;

Let $a_j \leftarrow m_j/(\mathbf{t}_j + \beta)$ $\forall j = 1, \ldots, N$ and $b_i \leftarrow 1/(\mathbf{f}_i + \beta)$ $\forall i = 1, \ldots, n$;

$[A(s)]_1 \leftarrow \sum_{j=1}^{N} a_j \left[\lambda_j^{\mathbb{K}}(s)\right]_1$; // KZG-commit to $\mathbf{a} = (a_1, \ldots, a_N)$

$\hat{\mathsf{c}}_{\mathbf{b}} \leftarrow \mathsf{mPC.Com}(\mathsf{srs}_{\mathsf{mPC}}, \mathbf{b})$; // mPC-commit to $\mathbf{b} = (b_1, \ldots, b_n)$

Prove wellformedness of $A(X)$

$[Q_A(s)]_1 \leftarrow \sum_{m_j \neq 0} a_j \cdot [Q_j(s)]_1$; // $Q_A(X) = (A(X)(\mathsf{T}(X) + \beta) - M(X))/\nu_{\mathbb{K}}(X)$

$z \leftarrow \sum_{m_j \neq 0} a_j$ // $z = \sum_{j=1}^{N} A(\omega^{j-1}) = N \cdot A(0)$

$[A'(s)]_1 \leftarrow \sum_{m_j \neq 0} a_j \cdot \left[r_j^{\mathbb{K}}(s)\right]_1$ // $A'(X) = (A(X) - A(0))/X = (A(X) - z/N)/X$

$\underline{\text{Send}}$ $([A(s), Q_A(s), A'(s)]_1, z, \hat{\mathsf{c}}_{\mathbf{b}})$;

$\underline{\text{Get}}$ $\boldsymbol{\rho} \leftarrow\$ \mathbb{F}^\mu, \tau \leftarrow\$ \mathbb{F}$;

Prove wellformedness of $\mathbf{b}$ and that $z = \sum_{i=1}^{n} b_i$ via the following sumcheck

$z = \sum_{\mathbf{h} \in \{0,1\}^\mu} \tilde{b}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{b}(\mathbf{h}) \cdot (\tilde{f}(\mathbf{h}) + \beta) - 1)$

$\pi_{SC} \leftarrow \mathsf{CP}_{SC}.\mathsf{Prove}(\mathsf{srs}_{\mathsf{mPC}}, \mathcal{R}_{\mathrm{SC}}, (z, \chi(\boldsymbol{\rho}, \mathbf{X}), \beta, \tau, \hat{\mathsf{c}}_{\mathbf{b}}, \hat{\mathsf{c}}_{\mathbf{f}}), (\mathbf{f}, \mathbf{b}))$

Return $\pi = ([M(s), A(s), Q_A(s), A'(s)]_1, z, \hat{\mathsf{c}}_{\mathbf{b}}, \pi_{SC})$.

$\underline{\mathsf{Verify}}(\mathsf{vk}_{\mathbf{t}}, \mathsf{c}_{\mathbf{f}}, \pi)$**:** Return 1 if and only if the following holds:

(i) $e([A(s)]_1, [\mathsf{T}(s)]_2 + [\beta]_2) = e([M(s)]_1, [1]_2) \cdot e([Q_A(s)]_1, [\nu_{\mathbb{K}}(s)]_2)$,

(ii) $e([A(s) - z/N]_1, [1]_1) = e([A'(s)]_1, [s]_2)$,

(iii) $\mathsf{Verify}(\mathsf{srs}_{\mathsf{mPC}}, (z, \chi(\boldsymbol{\rho}, \mathbf{X}), \beta, \tau, \hat{\mathsf{c}}_{\mathbf{b}}, \hat{\mathsf{c}}_{\mathbf{f}}), \pi_{SC}) = 1$

Fig. 4. Our lookup argument μ-seek.

Proof size. The proof size of our protocol is

$$4|\mathbb{G}_1| + 3|\mathbb{F}|(\log n + 2) + 1|\mathcal{C}_{\mathrm{mPC}}| + 2|\pi_{\mathrm{mPC}}|$$

where $|\mathcal{C}_{\mathrm{mPC}}|$ and $|\pi_{\mathrm{mPC}}|$ are respectively the size of the commitment and opening proof in the multilinear polynomial commitment.

Theorem 1. *If* mPC *is an extractable commitment scheme,* CP_{SC} *is a CP-SNARK for* mPC *and the generalized sumcheck relation* $\mathcal{R}_{SC}$*, and the power-*

discrete logarithm (PDL) assumption holds, then the protocol μ-seek in Figure 4 is knowledge-sound in the AGM and ROM.

Proof. Consider an algebraic adversary $\mathcal{A}$ that, after interacting with the honest verifier, returns: a commitment $\hat{\mathsf{c}}_{\mathbf{f}}$, a valid proof

$$\pi = \left([M(s), A(s), Q_A(s), A'(s)]_1, z, \hat{\mathsf{c}}_{\mathbf{b}}, \pi_{SC}\right)$$

and algebraic explanations for each group element consisting of the polynomials

$$M(X), A(X), Q_A(X), A'(X) \in \mathbb{F}[X] \text{ of degree } < N.$$

For every $\mathcal{A}$ we can also define the adversary $\mathcal{A}_{SC}$ that runs $\mathcal{A}$ and then outputs the commitments $\hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\mathbf{b}}$, the description $(z, \chi(\boldsymbol{\rho}, \mathbf{X}), \beta, \tau)$ of the generalized sumcheck statement, and the proof π_{SC} returned by $\mathcal{A}$. By the extractability of mPC and the knowledge-soundness of CP_{SC}, for every $\mathcal{A}_{SC}$ there exists a corresponding extractor Ext_{SC} that returns openings $\mathbf{f}, \mathbf{b}$ for the commitments $\hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\mathbf{b}}$.

We define the extractor $\mathsf{Ext}_{\mathcal{A}}$ as the algorithm that runs Ext_{SC} and then outputs the opening $\mathbf{f}$ for $\hat{\mathsf{c}}_{\mathbf{f}}$.

To prove knowledge soundness for $(\mathcal{A}, \mathsf{Ext}_{\mathcal{A}})$, notice that $M(X), \mathbf{f}$ do not depend on $\beta, \boldsymbol{\rho}, \tau$, while $A(X), Q_A(X), A'(X), \mathbf{b}$ do not depend on $\boldsymbol{\rho}, \tau$. The independence follows by the fact that they can be extracted prior to receiving those random challenges.

Consider the polynomials

$$\begin{aligned} V_1(X) &:= A(X)(\mathsf{T}(X) + \beta) - M(X) - Q_A(X)\nu_{\mathbb{K}}(X), \\ V_2(X) &:= A(X) - z/N - XA'(X) \end{aligned}$$

Since $\mathcal{A}$ returns a valid proof, by the verification equation ((i)) (resp. ((ii))) we have $V_1(s) = 0$ (resp. $V_2(s) = 0$). If either $V_1(X) \neq 0$ or $V_2(X) \neq 0$, we can make a reduction that computes s and breaks the PDL problem.

Therefore, let us continue assuming that both $V_1(X)$ and $V_2(X)$ are zero.

From $V_1(X) = 0$ we get

$$A(X)(\mathsf{T}(X)+\beta) = M(X) \bmod \nu_{\mathbb{K}}(X) \iff \forall j \in [N] : A(\omega^{j-1}) = \frac{M(\omega^{j-1})}{(\mathsf{T}(\omega^{j-1}) + \beta)}$$

From $V_2(X) = 0$ we get that $A(X) - z/N$ is divisible by X which, in combination with the fact that $A(X)$ is of degree $< N$ and the univariate sumcheck lemma, yields that $z = N \cdot A(0) = \sum_j A(\omega^{j-1}) = \sum_j \frac{M(\omega^{j-1})}{\mathsf{T}(\omega^{j-1})+\beta}$.

On the other hand, from the knowledge soundness of the sumcheck proof we have that $z = \sum_{\mathbf{h}\in\{0,1\}^\mu} \tilde{b}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{b}(\mathbf{h}) \cdot (\tilde{f}(\mathbf{h}) + \beta) - 1)$.

By the independence of $\tilde{f}, \tilde{b}, z$ on $\boldsymbol{\rho}, \tau$ the above sumcheck implies (except with negligible probability $(\mu + 1)/|\mathbb{F}|$)

$$z = \sum_{\mathbf{h}\in\{0,1\}^\mu} \tilde{b}(\mathbf{h}) \quad \wedge \quad \forall \mathbf{h} \in \{0,1\}^\mu : \tilde{b}(\mathbf{h}) = (\tilde{f}(\mathbf{h}) + \beta)^{-1}$$

namely $z = \sum_{\mathbf{h} \in \{0,1\}^\mu} \frac{1}{(\tilde{f}(\mathbf{h}) + \beta)}$.

Putting things together, we have reached

$$\sum_{j \in [N]} \frac{M(\omega^{j-1})}{\mathbf{t}_j + \beta} = \sum_{i \in [n]} \frac{1}{\mathbf{f}_i + \beta}$$

and since $M(X), \mathbf{f}, \mathbf{t}$ do not depend on the random challenge β, the following equality holds

$$\sum_{j} \frac{M(\omega^{j-1})}{\mathbf{t}_j + X} = \sum_{i} \frac{1}{\mathbf{f}_i + X}$$

which by the inclusion lemma implies $\mathbf{f} \prec \mathbf{t}$. □

4 Our Generic Construction

In this section we present our generic construction of a lookup argument for multilinear polynomial commitments, based on one for univariate polynomial commitments.

The building blocks of our construction are:

- The commitment scheme mPC for multilinear polynomials with $\mu = \log n$ variables and coefficients in $\mathbb{F}$. We assume that $\mathsf{srs}_{\mathsf{mPC}}$ is a universal SRS for mPC and for all the CP-SNARKs for this commitment scheme.
- A CP-SNARK CP_{SC} for the generalized sumcheck relation over polynomials committed with mPC.
- A commitment scheme uPC for univariate polynomials of degree $< n$ and coefficients in $\mathbb{F}$. We assume that $\mathsf{srs}_{\mathsf{uPC}}$ is a universal SRS for uPC and for all the CP-SNARKs for this commitment scheme.
- A CP-SNARK CP_u for the evaluation of univariate polynomials committed with uPC.
- A super-efficient lookup argument for uPC, that is a CP-SNARK $\mathsf{CP}_{\mathsf{lookup}}$ for the relation

  $$\mathcal{R}_{\mathsf{lookup}} := \{(\mathbf{t}; \mathsf{c}_{\mathbf{f}}; (\mathbf{f}, o_{\mathbf{f}})) : \mathbf{f} \prec \mathbf{t} \wedge \mathsf{c}_{\mathbf{f}} = \mathsf{uPC.Com}(\mathsf{srs}_{\mathsf{uPC}}, \mathbf{f})\}$$

  where $\mathsf{uPC.Com}(\mathsf{ck}, \mathbf{f})$ denotes committing to the Lagrange interpolation of $\mathbf{f}$ over a domain $\mathbb{H}$ of size n.

Our protocol The construction is described in full detail in Fig. 5. Here we describe its main ideas.

In this protocol, the prover holds a commitment $\hat{\mathsf{c}}_{\mathbf{f}}$ to the vector $\mathbf{f}$ in multilinear encoding (i.e., a commitment to the multilinear extension of $\mathbf{f}$). Then the main idea is that the prover: (a) generates a commitment $\mathsf{c}_{\mathbf{f}}$ to the univariate polynomial that interpolates $\mathbf{f}$ over a domain $\mathbb{H}$, (b) uses the (super-efficient) lookup argument to prove the lookup relation w.r.t. $\mathsf{c}_{\mathbf{f}}$, (c) proves that $\mathsf{c}_{\mathbf{f}}$ and $\hat{\mathsf{c}}_{\mathbf{f}}$ commit to the same vector.

To address (c), we use a random point σ to test that

$$f(\sigma) = \sum_{\mathbf{h}\in\{0,1\}^\mu} \tilde{f}(\mathbf{h}) \cdot \lambda^{\mathbb{H}}_{\sum_j h_j 2^j}(\sigma)$$

The left-hand side of the equation can be proven via a standard polynomial evaluation that $y = f(\sigma)$.

For the right-hand side, we proceed as follows. First, the prover commits in $\hat{\mathsf{c}}_{\mathbf{r}}$ to the multilinear extension of the vector $\mathbf{r} = (\lambda^{\mathbb{H}}_1(\sigma), \ldots, \lambda^{\mathbb{H}}_n(\sigma))$. Second, it proves that $y = \sum_{i=1}^n \mathbf{f}_i \cdot \mathbf{r}_i$ with standard sumcheck for inner products

$$y = \sum_{\mathbf{h}\in\{0,1\}^\mu} \tilde{f}(\mathbf{h})\tilde{r}(\mathbf{h})$$

Third, it proves that $\hat{\mathsf{c}}_{\mathbf{r}}$ is wellformed as follows. For every $i \in [n]$ it holds $\mathbf{r}_i = \lambda^{\mathbb{H}}_i(\sigma) = \frac{\omega^{i-1}(\sigma^n-1)}{n(\sigma-\omega^{i-1})}$, which can be reduced to the following statement

$$\forall \mathbf{h} \in \{0,1\}^\mu : \tilde{r}(\mathbf{h})n(\sigma - \tilde{\omega}(\mathbf{h})) = \tilde{\omega}(\mathbf{h})(\sigma^n - 1)$$

w.r.t. the multilinear polynomials $\tilde{r}(\mathbf{X})$ and $\tilde{\omega}(\mathbf{X})$, which are the multilinear extensions of $\mathbf{r}$ and $\boldsymbol{\omega} = (\omega^{i-1})_{i=1}^n$ respectively[13].

By using a random challenge $\boldsymbol{\rho} \leftarrow\!\$\ \mathbb{F}^\mu$ we reduce the statement above to the following sumcheck

$$0 = \sum_{\mathbf{h}\in\{0,1\}^\mu} (\tilde{r}(\mathbf{h})(\sigma - \tilde{\omega}(\mathbf{h})) - \tilde{\omega}(\mathbf{h})n(\sigma^n - 1))\chi(\boldsymbol{\rho}, \mathbf{h})$$

Finally, we optimize and batch the two sumchecks, with a random challenge $\tau \leftarrow\!\$\ \mathbb{F}$, into a single sumcheck:

$$y = \sum_{\mathbf{h}\in\{0,1\}^\mu} \tilde{f}(\mathbf{h})\tilde{r}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{r}(\mathbf{h})n(\sigma - \tilde{\omega}(\mathbf{h})) - \tilde{\omega}(\mathbf{h})(\sigma^n - 1))$$

Proof size. The proof size of our generic protocol is

$$1|\mathcal{C}_{\mathrm{uPC}}| + 3|\mathbb{F}|(\log n + 2) + 1|\mathcal{C}_{\mathrm{mPC}}| + 4|\pi_{\mathrm{mPC}}| + 1|\pi_{\mathsf{uPC\text{-}lu}}|$$

where $|\mathcal{C}_{\mathrm{uPC}}|$ is the size of the commitment in the univariate polynomial commitment, $|\mathcal{C}_{\mathrm{mPC}}|$ and $|\pi_{\mathrm{mPC}}|$ are respectively the size of the commitment and opening proof in the multilinear polynomial commitment, $|\pi_{\mathsf{uPC\text{-}lu}}|$ is the size of the lookup argument for univariate polynomial commitments.

Remark 3 (Batching optimizations in our constructions). We stress that in construction in Fig. 4 (resp. Fig. 5) one is able to avoid the full size and verification cost of the two (resp. four) multilinear evaluation proofs if the underlying polynomial commitment supports batching.

Let $\mathsf{srs} := (\mathsf{srs}_{\mathsf{uPC}}, \mathsf{srs}_{\mathsf{mPC}})$

$\underline{\mathsf{Derive}}(\mathsf{srs}, N, n, \mathbf{t})$**:**

$(\mathsf{ek}_{\mathbf{t}}, \mathsf{vk}_{\mathbf{t}}) \leftarrow \mathsf{CP}_{\mathsf{lookup}}.\mathsf{Derive}(\mathsf{srs}_{\mathsf{uPC}}, N, n, \mathbf{t})$

$\hat{\mathsf{c}}_{\boldsymbol{\omega}} \leftarrow \mathsf{mPC}.\mathsf{Com}((1, \omega, \ldots, \omega^{n-1}))$

Return $(\hat{\mathsf{ek}}_{\mathbf{t}}, \hat{\mathsf{vk}}_{\mathbf{t}}) := (\mathsf{ek}_{\mathbf{t}}, (\mathsf{vk}_{\mathbf{t}}, \hat{\mathsf{c}}_{\boldsymbol{\omega}}))$.

$\underline{\mathsf{Prove}}(\hat{\mathsf{ek}}_{\mathbf{t}}, \hat{\mathsf{c}}_{\mathbf{f}}, \mathbf{f})$**:**

$f(X) := \sum_{i=1}^{n} = \mathbf{f}_i \lambda_i^{\mathbb{H}}(X)$

$\mathsf{c}_{\mathbf{f}} \leftarrow \mathsf{uPC}.\mathsf{Com}(\mathsf{srs}_{\mathsf{uPC}}, f(X))$

$\pi_{\mathsf{uPC\text{-}lu}} \leftarrow \mathsf{CP}_{\mathsf{lookup}}.\mathsf{Prove}(\mathsf{ek}_{\mathbf{t}}, \mathsf{c}_{\mathbf{f}}, \mathbf{f})$ Prove lookup for the univariate polynomial encoding

Prove that $\mathsf{c}_{\mathbf{f}}$ and $\hat{\mathsf{c}}_{\mathbf{f}}$ commit to the same vector $\mathbf{f}$

$\underline{\mathsf{Send}}$ $(\mathsf{c}_{\mathbf{f}}, \pi_{\mathsf{uPC\text{-}lu}})$;

$\underline{\mathsf{Get}}$ $\sigma \leftarrow\!\$\ \mathbb{F}$;

Compute $y \leftarrow \sum_{i=1}^{n} = \mathbf{f}_i \cdot \lambda_i^{\mathbb{H}}(\sigma) = f(\sigma)$

$\pi_{\mathbf{f}} \leftarrow \mathsf{CP}_u.\mathsf{Prove}(\mathsf{srs}_{\mathsf{uPC}}, \mathsf{c}_{\mathbf{f}}, \sigma, y, f(X))$ Prove that $y = f(\sigma)$

$\hat{\mathsf{c}}_{\mathbf{r}} \leftarrow \mathsf{mPC}.\mathsf{Com}(\mathbf{r})$ where $\mathbf{r} := (\lambda_1^{\mathbb{H}}(\sigma), \ldots, \lambda_n^{\mathbb{H}}(\sigma))$

$\underline{\mathsf{Send}}$ $(y, \pi_{\mathbf{f}}, \hat{\mathsf{c}}_{\mathbf{r}})$;

$\underline{\mathsf{Get}}$ $\boldsymbol{\rho} \leftarrow\!\$\ \mathbb{F}^{\mu}, \tau \leftarrow\!\$\ \mathbb{F}$;

Prove linking of $\hat{\mathsf{c}}_{\mathbf{f}}$ to $\mathsf{c}_{\mathbf{f}}$ via the following sumcheck

$y = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{f}(\mathbf{h})\tilde{r}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{r}(\mathbf{h}) n(\sigma - \tilde{\omega}(\mathbf{h})) - \tilde{\omega}(\mathbf{h})(\sigma^n - 1))$

$\pi_{SC} \leftarrow \mathsf{CP}_{SC}.\mathsf{Prove}(\mathsf{srs}_{\mathsf{mPC}}, \mathcal{R}_{\mathrm{SC}}, (y, \chi(\boldsymbol{\rho}, \mathbf{X}), \sigma, \tau, \hat{\mathsf{c}}_{\mathbf{r}}, \hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\boldsymbol{\omega}}), (\mathbf{f}, \mathbf{r}, \boldsymbol{\omega}))$

Return $\pi = (\mathsf{c}_{\mathbf{f}}, \pi_{\mathsf{uPC\text{-}lu}}, y, \hat{\mathsf{c}}_{\mathbf{r}}, \pi_{\mathbf{f}}, \pi_{SC})$.

$\underline{\mathsf{Verify}}(\mathsf{vk}_{\mathbf{t}}, \mathsf{c}_{\mathbf{f}}, \pi)$**:** Return 1 if and only if the following holds:

(i) $\mathsf{CP}_{\mathsf{lookup}}.\mathsf{Verify}(\hat{\mathsf{vk}}_{\mathbf{t}}, \mathsf{c}_{\mathbf{f}}, \pi_{\mathsf{uPC\text{-}lu}}) = 1$

(ii) $\mathsf{CP}_u.\mathsf{Verify}(\mathsf{srs}_{\mathsf{uPC}}, (y, \sigma), \mathsf{c}_{\mathbf{f}}, \pi_{\mathbf{f}}) = 1$

(iii) $\mathsf{CP}_{SC}.\mathsf{Verify}(\mathsf{srs}_{\mathsf{mPC}}, \mathcal{R}_{\mathrm{SC}}, (y, \chi(\boldsymbol{\rho}, \mathbf{X}), \sigma, \tau, \hat{\mathsf{c}}_{\mathbf{r}}, \hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\boldsymbol{\omega}}), \pi_{SC}) = 1$

Fig. 5. Our generic lookup argument for multilinear polynomials.

Theorem 2. *If* mPC *is an extractable commitment scheme,* CP_{SC} *is a CP-SNARK for* mPC *and the generalized sumcheck relation* $\mathcal{R}_{SC}$*,* CP_u *is a CP-SNARK for* uPC *and polynomial evaluations, and* $\mathsf{CP}_{\mathsf{lookup}}$ *is a CP-SNARK for* uPC *for* $\mathcal{R}_{\mathsf{lookup}}$*, then the protocol in Figure 5 is knowledge-sound in the ROM.*

Proof. Consider an adversary $\mathcal{A}$ that, after interacting with the honest verifier, returns: a commitment $\hat{\mathsf{c}}_{\mathbf{f}}$ and a valid proof

$$\pi = (\mathsf{c}_{\mathbf{f}}, \pi_{\mathsf{uPC\text{-}lu}}, y, \hat{\mathsf{c}}_{\mathbf{r}}, \pi_{\mathbf{f}}, \pi_{SC})$$

For every $\mathcal{A}$ we can define the adversary $\mathcal{A}_{\mathsf{mPC}}$ that runs $\mathcal{A}$ until the first round and then outputs $\hat{\mathsf{c}}_{\mathbf{f}}$. By the extractability of mPC, for $\mathcal{A}_{\mathsf{mPC}}$ there is an extractor $\mathsf{Ext}_{\mathsf{mPC}}$ that outputs a valid opening $\tilde{f}$ of $\hat{\mathsf{c}}_{\mathbf{f}}$ with overwhelming probability.

[13] A commitment to $\boldsymbol{\omega}$, which only depends on the size n of the vector, can be created during the preprocessing phase.

We define the extractor $\mathsf{Ext}_{\mathcal{A}}$ as the algorithm that runs $\mathsf{Ext}_{\mathsf{lookup}}$ and then outputs the opening $\tilde{f}$ for $\hat{\mathsf{c}}_{\mathbf{f}}$.

To prove that the pair $(\mathcal{A}, \mathsf{Ext}_{\mathcal{A}})$ satisfies knowledge soundness, we also define the following additional extractors.

First, for every $\mathcal{A}$ we define the adversary $\mathcal{A}_{\mathsf{lookup}}$ that runs $\mathcal{A}$ until the first round and then outputs $\mathsf{c}_{\mathbf{f}}, \pi_{\mathsf{uPC\text{-}lu}}$. By the knowledge-soundness of $\mathsf{CP}_{\mathsf{lookup}}$, for every $\mathcal{A}_{\mathsf{lookup}}$ there is a corresponding extractor $\mathsf{Ext}_{\mathsf{lookup}}$ that returns a polynomial $f(X)$ that, with overwhelming probability, is a valid opening for $\mathsf{c}_{\mathbf{f}}$ and whose encoded vector $\mathbf{f}$ (i.e., such that $f(X) = \sum_{i=1}^{n} \mathbf{f}_i \cdot \lambda_i^{\mathbb{H}}(X)$) satisfies $\mathbf{f} \prec \mathbf{t}$.

Second, we define $\mathcal{A}_{\mathsf{CP}_u}$ that runs $\mathcal{A}$ until the second round and then outputs $\mathsf{c}_{\mathbf{f}}$ and proof $\pi_{\mathbf{f}}$. By the knowledge-soundness of CP_u there is an extractor $\mathsf{Ext}_{\mathsf{CP}_u}$ that returns a polynomial $f'(X)$ that, with overwhelming probability, is a valid opening for $\mathsf{c}_{\mathbf{f}}$ and such that $y = f'(\sigma)$.

Third, we define the adversary $\mathcal{A}_{SC}$ that runs $\mathcal{A}$ until the end and returns the commitments $\hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\mathbf{r}}$, the description $(y, \chi(\boldsymbol{\rho}, \mathbf{X}), \sigma, \tau)$ of the generalized sumcheck statement, and the proof π_{SC}. By the knowledge-soundness of CP_{SC}, for every $\mathcal{A}_{SC}$ there exists a corresponding extractor Ext_{SC} that returns $\tilde{f}', \tilde{r}$ that are valid openings for the commitments $\hat{\mathsf{c}}_{\mathbf{f}}, \hat{\mathsf{c}}_{\mathbf{r}}$ and are consistent with the generalized sumcheck statement.

By the computational binding of the commitment schemes we can exclude that $f(X) \neq f'(X)$, $\tilde{f} \neq \tilde{f}'$.

Next, we notice that $\tilde{f}, f(X)$ do not depend on $\sigma, \boldsymbol{\rho}, \tau$, while $\tilde{r}$ does not depend on $\boldsymbol{\rho}, \tau$ (the same holds clearly for $\boldsymbol{\omega}$). The independence follows by the fact that they can be extracted prior to receiving those random challenges.

From the knowledge soundness of the sumcheck proof we have that

$$y = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{f}(\mathbf{h})\tilde{r}(\mathbf{h}) + \tau \cdot \chi(\boldsymbol{\rho}, \mathbf{h}) \cdot (\tilde{r}(\mathbf{h}) n(\sigma - \tilde{\omega}(\mathbf{h})) - \tilde{\omega}(\mathbf{h})(\sigma^n - 1)).$$

By the independence of $\tilde{f}, \tilde{r}, y$ (and the rest of the statement) on $\boldsymbol{\rho}, \tau$, the above sumcheck implies

$$y = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{f}(\mathbf{h})\tilde{r}(\mathbf{h}) \quad \wedge \quad \forall \mathbf{h} \in \{0,1\}^{\mu} : \tilde{r}(\mathbf{h}) = \frac{\tilde{\omega}(\mathbf{h})(\sigma^n - 1)}{n(\sigma - \tilde{\omega}(\mathbf{h}))}$$

namely $y = \sum_{\mathbf{h} \in \{0,1\}^{\mu}} \tilde{f}(\mathbf{h}) \cdot \lambda^{\mathbb{H}}_{\sum_j h_j 2^j}(\sigma)$.

Putting things together, we have reached

$$f(\sigma) = \sum_{i=1}^{n} \hat{\mathbf{f}}_i \cdot \lambda_i^{\mathbb{H}}(\sigma)$$

where $\hat{\mathbf{f}}$ is the vector represented by the multilinear polynomial $\tilde{f}$. Since $f(X), \tilde{f}$ do not depend on the random challenge σ, the following identity

$$f(X) = \sum_{i=1}^{n} \hat{\mathbf{f}}_i \cdot \lambda_i^{\mathbb{H}}(X) \iff \mathbf{f} = \hat{\mathbf{f}}$$

holds with probability $1 - n/|\mathbb{F}|$. □

5 Lookup arguments and general-purpose SNARKs

In this section we are interested in augmenting proof systems for rank-1 constraint systems (R1CS) with lookup capabilities.

We briefly recap the definition of R1CS:

Definition 6 (R1CS). *An R1CS relation $\mathcal{R}_{\mathsf{R1CS}}$ is defined by three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{m \times (\ell+n)}$ and accepts $(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^{\ell} \times \mathbb{F}^{n}$ such that, for $\mathbf{z} = (\mathbf{x}, \mathbf{w})$,*

$$(\mathbf{A} \cdot \mathbf{z}) \circ (\mathbf{B} \cdot \mathbf{z}) = (\mathbf{C} \cdot \mathbf{z})$$

The augmented constraint system we want to support is the following:

Definition 7. *Let $\mathbf{A}, \mathbf{B}, \mathbf{C}$ be matrices describing an R1CS with witness of size n, let $I \subseteq [n]$ and let $\mathbf{t}$ be a lookup table. We define the relation $\mathcal{R}_{\mathsf{R1CS}+\mathsf{lookup}}$ such that*

$$(\mathbf{x}, \mathbf{w}) \in \mathcal{R}_{\mathsf{R1CS}+\mathsf{lookup}} \iff (\mathbf{x}, \mathbf{w}) \in \mathcal{R}_{\mathsf{R1CS}} \wedge \mathbf{w}_I \prec \mathbf{t}$$

where $\mathbf{w}_I$ denotes the subvector of $\mathbf{w}$ restricted to the indices in I.

We can build a (CP) SNARK for $\mathcal{R}_{\mathsf{R1CS}+\mathsf{lookup}}$ by using the following building blocks:

- A commitment scheme mPC for multilinear polynomials.
- A CP-SNARK Π_{R1CS} over commitment scheme mPC for $\mathcal{R}_{\mathsf{R1CS}}$.
- A CP-SNARK $\Pi_{\mathsf{prj\text{-}lkup}}$ over commitment scheme mPC for the projective lookup relation $\mathcal{R}_{\mathsf{prj\text{-}lkup}}$ defined as follows: $(\bot, \mathbf{w}) \in \mathcal{R}_{\mathsf{prj\text{-}lkup}} \iff \mathbf{w}_I \prec \mathbf{t}$ where[14] $\mathbf{t}$ and I parametrize the relation and are respectively a lookup table and a subset of $[n]$ where n denotes the size of w (this argument is what we call a *projective lookup* in this section).

The proof of the following theorem follows immediately from the CP-SNARK properties of its building blocks (see [CFQ19, Section 3.2]).

Theorem 3. *The construction in Fig. 6 is a CP-SNARK for $\mathcal{R}_{\mathsf{R1CS}+\mathsf{lookup}}$ (Definition 7).*

Below we discuss two ways to add projective properties to existing lookup arguments.

5.1 Projective lookups from indexed lookups.

The first method to build a CP-SNARK for $\mathcal{R}_{\mathsf{prj\text{-}lkup}}$ is to create a commitment $\mathsf{c}_{\mathbf{f}}$ to the subvector $\mathbf{f} = \mathbf{w}_I$, run a CP-SNARK for lookup with $\mathsf{c}_{\mathbf{f}}$ (to prove that $\mathbf{f} \prec \mathbf{t}$) and then use a CP-SNARK for indexed lookups to prove that $\mathbf{f} = \mathbf{w}_I$ w.r.t. commitments $\mathsf{c}_{\mathbf{w}}$ an $\mathsf{c}_{\mathbf{f}}$. An indexed lookup can be realized via one for proving that $\mathbf{f} = \mathbf{M} \cdot \mathbf{w}$ for an appropriately defined projection matrix $\mathbf{M}$, see e.g., [STW24, RZ21].

[14] Notice that from the definition of the relation the public input of the CP-SNARK verification algorithm is a commitment to $\mathbf{w}$ and not to $\mathbf{w}_I$. This specific feature makes our first construction particularly simple.

$\underline{\mathsf{Prove}(\mathsf{ek} = (\mathsf{ek}_{\mathbf{R1CS}}, \mathsf{ek}_{\mathbf{prj\text{-}lkup}}), x; w)\textbf{:}}$

$\mathsf{c}_{\mathbf{w}} \leftarrow \mathsf{mPC.Com}(\mathsf{srs}_{\mathsf{mPC}}, \tilde{w})$ // commit to the MLE of $\mathbf{w}$

$\pi_{\mathsf{R1CS}} \leftarrow \mathit{\Pi}_{\mathsf{R1CS}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{R1CS}}, \mathbf{x}; \mathbf{w})$

$\pi_{\mathsf{prj\text{-}lkup}} \leftarrow \mathit{\Pi}_{\mathsf{prj\text{-}lkup}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{prj\text{-}lkup}}; \mathbf{w})$

return $(\pi_{\mathsf{R1CS}}, \pi_{\mathsf{prj\text{-}lkup}})$

$\underline{\mathsf{Verify}(\mathsf{vk} = (\mathsf{vk}_{\mathbf{R1CS}}, \mathsf{vk}_{\mathbf{prj\text{-}lkup}}), \mathbf{x}, \mathsf{c}_{\mathbf{w}}, (\pi_{\mathbf{R1CS}}, \pi_{\mathbf{prj\text{-}lkup}}))\textbf{:}}$

return $1 \iff$

$\mathit{\Pi}_{\mathsf{R1CS}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{R1CS}}, \mathbf{x}, \mathsf{c}_{\mathbf{w}}, \pi_{\mathsf{R1CS}}) \wedge \mathit{\Pi}_{\mathsf{prj\text{-}lkup}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{prj\text{-}lkup}}, \mathsf{c}_{\mathbf{w}}, \pi_{\mathsf{prj\text{-}lkup}})$

Fig. 6. Our construction for augmented R1CS relations through projective lookups. Key generation and derivation algorithm are not described in figure: they simply consist in running the key generation of $\mathit{\Pi}_{\mathsf{R1CS}}$ and $\mathit{\Pi}_{\mathsf{prj\text{-}lkup}}$ and returning the concatenation of their outputs.

5.2 How to convert our constructions into projective lookups.

In alternative to the method described above, we show how to turn a lookup argument into a projective one in the case of subsets $I \subseteq [n]$ that have a simple representation through a wildcard pattern.[15] In such a case, we support projective lookups without introducing more commitments, at virtually no overhead. For example the subset $I := \{i : 1 \leq i \leq 2^{\frac{\mu}{2}}\} \subseteq [2^\mu]$ can be represented through the wildcard pattern $\underbrace{00\ldots00}_{\mu/2}\underbrace{\star\star\cdots\star\star}_{\mu/2}$ since each bit replacement of the $\star$-s corresponds to the binary representation of an integer j such that $j+1 \in I$. On the other hand there is no such wildcard pattern for a subset like $I' := \{\underbrace{00\ldots00}_{\mu}, \underbrace{11\ldots11}_{\mu}\}$.

We now capture these intuitions formally:

Definition 8 (Wildcard pattern). *Let $\mu \in \mathbb{N}$. A wildcard pattern $\mathbb{P}$ of size μ is a vector in $\{0,1,\star\}^\mu$ (e.g., the string $00010\star1\star$). We denote by $\kappa(\mathbb{P})$ the number of star symbols ($\star$) in the pattern $\mathbb{P}$. Given a binary string $\mathbf{b} \in \{0,1\}^{\kappa(\mathbb{P})}$, we denote by $\mathbf{b} \sqcup \mathbb{P}$ the binary string obtained by replacing the $\star$-s in $\mathbb{P}$ with the bits in $\mathbf{b}$ in order (e.g., $001 \sqcup 110\star1\star1\star \to 110\underline{0}1\underline{0}1\underline{1}$, where the replaced positions are underlined). We will also extend this notation $\mathbf{u} \sqcup \mathbb{P}$ in the natural way to the case where the vector $\mathbf{u}$ is of arbitrary field elements (rather than bits only).*

Definition 9 (Wildcard-expressible subset). *Let $n \in \mathbb{N}$ where $n = 2^\mu$. Let $I \subseteq [n]$. We say that I is wildcard-expressible if there exists a wildcard pattern $\mathbb{P}_I$*

[15] In the application of lookups to R1CS, such wildcard representation of the set I may be achieved by appropriately rearranging the indices of the lookup gates.

of length μ such that we can represent each element of I as a string "matched" by $\mathbb{P}_I$ (most-significant bit on the left), i.e., if

$$\forall i \in I \ \exists \mathbf{b} \in \{0,1\}^{\kappa(\mathbb{P})} : i = 1 + \sum_{j=1}^{\mu} 2^{\mu-j} \cdot z_j, \ \textit{where } \mathbf{z} := \mathbf{b} \sqcup \mathbb{P}_I$$

The key observation we will use to make our lookup arguments "projective" is that for a wildcard-expressible subset I we can describe $\tilde{w}_I$—the MLE of the subvector $\mathbf{w}_I$ of $\mathbf{w}$ (its restrictions to indices $i \in I$)—through a partial evaluation of $\tilde{w}$ on the non-wildcard characters. More formally, let $I \subseteq [n]$ be a wildcard-expressible subset with associated pattern $\mathbb{P}_I$, let $\mathbf{w} \in \mathbb{F}^n$ be a vector, $\mathbf{w}_I \in \mathbb{F}^{2^{\kappa(\mathbb{P}_I)}}$ the I-subvector of $\mathbf{w}$, and let $\tilde{w}$ and $\tilde{w}_I$ be their respective MLE. Then for any $\mathbf{x} \in \mathbb{F}^{\kappa(\mathbb{P}_I)}$ it holds $\tilde{w}_I(\mathbf{x}) = \tilde{w}(\mathbf{x} \sqcup \mathbb{P}_I)$.

Given the observation above, we can turn both our constructions for $\mathcal{R}_{\mathsf{lookup}}$ of Sections 3 and 4 into constructions for $\mathcal{R}_{\mathsf{R1CS+lookup}}$. For $\mathcal{R}_{\mathsf{R1CS+lookup}}$, let us assume that the prover holds a commitment $\mathsf{c}_{\mathbf{w}}$ to the vector $\mathbf{w}$ and wishes to prove that $\mathbf{w}_I \prec \mathbf{t}$. Then we let the prover algorithm work in the same way by setting $\mathbf{f} = \mathbf{w}_I$ and by "simulating" $\tilde{f}(\mathbf{x}) = \tilde{w}_I(\mathbf{x})$ as $\tilde{w}(\mathbf{x} \sqcup \mathbb{P}_I)$ following the observation above. From the point of view of the verifier, the only change is that, whenever it needs to verify an evaluation of $\tilde{f}$ on point $\mathbf{x}$ w.r.t. the commitment $\mathsf{c}_{\mathbf{w}}$, the verifier runs CP_m's verification on the augmented point $\mathbf{x} \sqcup \mathbb{P}_I$.

Acknowledgements

We thank Justin Thaler for fruitful discussions about the cost profile of Lasso and applications of unstructured tables. We also thank Dario Catalano and Emanuele Giunta who contributed to discussions during the early stages of this work. Finally, we thank Helger Lipmaa for useful comments on the presentation of our results.

References

ABC+22. Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: Enhanced compiling method for pedersen-committed zkSNARK engines. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 584–614. Springer, Heidelberg, March 2022.

AGM18. Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 643–673. Springer, Heidelberg, August 2018.

AST24. Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2024.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

BCF+21. Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 393–414. Springer, Heidelberg, March 2021.

BCG+18. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Heidelberg, December 2018.

BCHO22. Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 427–457. Springer, Heidelberg, May / June 2022.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

BMM+21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Heidelberg, December 2021.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Heidelberg, April 2023.

CFF+21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021.

CFF+24. Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 337–369. Springer, Heidelberg, April 2024.

CFH+22. Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accu-

mulators. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 455–469. ACM Press, November 2022.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

CGG+23. Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover snarks with constant size proofs and square root size universal setup. In *International Conference on Cryptology and Information Security in Latin America*, pages 331–351. Springer, 2023.

CHA22. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 652–666. ACM Press, May / June 2022.

CHAK23. Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. Curve trees: Practical and transparent {Zero-Knowledge} accumulators. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4391–4408, 2023.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

EFG22. Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022. `https://eprint.iacr.org/2022/1763`.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

FT22. Dario Fiore and Ida Tucker. Efficient zero-knowledge proofs on signed data with applications to verifiable computation on data streams. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1067–1080. ACM Press, November 2022.

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

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs

for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 193–226. Springer, Heidelberg, August 2023.

GNS24. Chaya Ganesh, Vineet Nair, and Ashish Sharma. Dual polynomial commitment schemes and applications to commit-and-prove SNARKs. Cryptology ePrint Archive, Paper 2024/943, 2024. https://eprint.iacr.org/2024/943.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

GW20. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://eprint.iacr.org/2020/315.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022. https://eprint.iacr.org/2022/1530.

hbc. halo2 book contributors. The halo2 book. 16-bit table chip for sha-256. https://zcash.github.io/halo2/design/gadgets/sha256/table16.html#16-bit-table-chip-for-sha-256.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

LFKN90. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg, March 2012.

PH23. Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Paper 2023/1284, 2023. https://eprint.iacr.org/2023/1284.

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

PST13. Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242. Springer, Heidelberg, March 2013.

RA. Rick and Morty Wiki Authors. Mr. meeseeks. rick and morty wiki. https://rickandmorty.fandom.com/wiki/Mr._Meeseeks.

RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020.

STW24. Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 180–209. Springer, Heidelberg, May 2024.

Whi. Barry Whitehat. Lookup singularity. `https://zkresear.ch/t/lookup-singularity/65/7`.

XZS22. Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 299–328. Springer, Heidelberg, August 2022.

ZBK+22a. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022.

ZBK+22b. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Report 2022/621, 2022. `https://eprint.iacr.org/2022/621`.

ZGK+22. Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565, 2022. `https://eprint.iacr.org/2022/1565`.