SoK: Lookup Table Arguments

Hossein Hafezi[1*], Gaspard Anthoine[2,4], Matteo Campanelli[3], and Dario Fiore[2]

[1] New York University, USA
[2] IMDEA Software Institute, Spain
[3] Offchain Labs, USA
[4] Universidad Politécnica de Madrid, Madrid, Spain

Abstract. Lookup arguments have become a central tool in proof systems, powering a range of practical applications. They enable the efficient enforcement of non-native operations, such as bit decomposition, range checks, comparisons, and floating-point arithmetic. They underpin zk-VMs by modelling instruction tables, provide set membership proofs in stateful computations, and strengthen extractors by ensuring witnesses belong to small domains. Despite these broad uses, existing lookup constructions vary widely in assumptions, efficiency, and composability. In this work, we systematize the design of lookup arguments and the cryptographic primitives they rely on. We introduce a unified and modular framework that covers standard, projective, indexed, vector, and decomposable lookups. We classify existing protocols by proof technique—multiset equality, Logup-based, accumulators, and subvector extraction (matrix–vector)—and by composition style. We survey and evaluate existing protocols along dimensions such as prover cost, dependence on table size, and compatibility with recursive proofs. From this analysis, we distill lessons and guidelines for choosing lookup constructions in practice and highlight the benefits and limitations of emerging directions in literature, such as preprocessing and decomposability.

1 Introduction

Succinct non-interactive arguments of knowledge (aka SNARKs) allow a prover to convince a verifier that a statement is valid with a short and efficiently verifiable proof. A striking property of modern SNARKs is that they are general-purpose, namely, they can prove the correctness of arbitrary computation while maintaining succinct proofs and fast verification. This combination has driven widespread adoption, both in theory and practice.

A recent development in this space is the growing interest in *lookup arguments* [Boo+18; GW20; Zap+22a; EFG22; STW24]. Informally, a lookup argument allows proving that each element of a witness vector belongs to a predefined table.[5]

* Work primarily conducted at IMDEA Software.

[5] Even though the term *arguments* technically refers to computationally sound proof systems, much of the literature uses *lookup arguments* more broadly to denote the proof techniques employed in constructing argument systems that incorporate lookup checks.

Despite their apparent simplicity, this functionality has proven to be a potent tool. Notably, it allows SNARK designers to reduce non-native operations — such as bit decomposition, range checks, comparisons, and floating-point arithmetic — to table checks, thereby significantly reducing the number of circuit constraints and prover time. Lookup arguments have already found diverse applications within SNARKs, including efficient proofs for hash functions [Sze+23; Gra+22], modular arithmetic, membership proofs, and general virtual machine executions [AST24].

Research on lookup arguments has progressed at an extraordinary pace. In just a few years, numerous constructions have been proposed, each with distinct design methodologies, efficiency trade-offs, and compatibility requirements. In this rapid progress, the focus was primarily on introducing new techniques that improve efficiency. However, this has often been achieved in specialized contexts, using definitions of the lookup functionality tailored to the application or proof system at hand, rather than guided by a general framework. For example, several works [GW20; Zap+22a; EFG22; Zap+22b; Cam+24] develop lookup arguments for witnesses and tables committed using the KZG polynomial commitment [KZG10], implicitly adopting a notion of lookup specialized to this (univariate) polynomial-based setting. The context in which the lookup argument operates is a crucial aspect of its use. Seen as standalone proof systems, lookup arguments may indeed be rather uninteresting, as they would only allow proving membership of elements in a table. Their power instead arises when they are composed with other proofs about the same witness. Understanding when, how, and under what assumptions this composition is possible is therefore a key question we address.

This work. We provide a systematization of knowledge (SoK) on lookup arguments. We aim to clarify definitions, highlight key methodologies for composing lookup proofs with other proof systems, identify useful variants, analyze the current state of the art, and applications. In doing so, we shed light on the unifying principles underlying existing proposals and guide researchers and practitioners seeking to design or deploy lookup arguments. Our main contributions include:

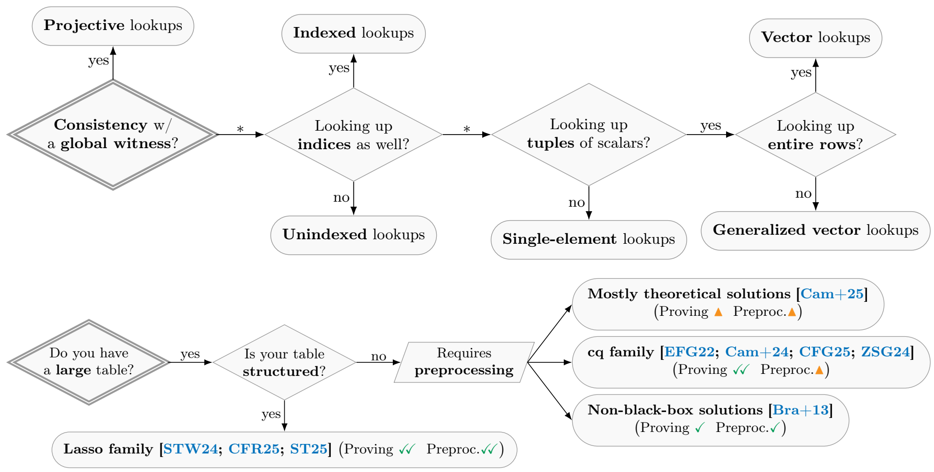
Definitions. We introduce a modular framework of definitions for lookup arguments. We first define the lookup relation, i.e., checking that each element w_j of a vector $\mathbf{w}$ appears in another vector $\mathbf{t}$. Then we formalize proof systems for this relation in two flavours: commit-and-prove SNARKs [CFQ19] and polynomial interactive oracle proofs (PIOP) [Chi+20; BFS20]. The former formalizes the case of a (succinct) proof about *committed* vectors $\mathbf{t}$ and $\mathbf{w}$ in which the commitment to $\mathbf{t}$ is usually publicly computed by the verifier in an offline preprocessing phase. This commit-and-prove flavour essentially corresponds to the notion of lookup arguments. The latter formalizes proof systems for the lookup relation in the information-theoretic PIOP framework, in which the vectors $\mathbf{t}$ and $\mathbf{w}$ are encoded as polynomials and available to the verifier as *oracles*. The oracle model is an idealization that can later be removed via a popular compilation strategy that uses polynomial commitments. Formalizing lookup proof systems using this perspective highlights two distinct composition methodologies for using lookups when constructing zkSNARKs: (i) argument-level composition via the commit-and-prove paradigm, and (ii) PIOP-level composition.

Beyond simple lookups. We present several variants of the lookup problem, which include: *indexed lookups*, where the membership relation is enforced with respect to a given vector of indices $\boldsymbol{i}$, i.e., for every j, $w_j = t_{i_j}$; *vector lookups*, where the entries of the witness and table are tuples, and *projective lookups*, where only a subset of the witness entries are checked to be in the table, i.e., for a given set of indices $\boldsymbol{i}$, it holds $w_j \in \mathbf{t}$ for every $j \in \boldsymbol{i}$. We also define the notion of *online lookup tables*, in which the table depends on the witness and hence cannot be preprocessed. We find that the projective setting is an overlooked aspect of lookup tables in prior work. As noted earlier, lookup tables become most useful when composed with other proof systems, where they are typically used to prove that *part of the witness* lies in a table. However, the syntax of lookup relations and arguments in most existing definitions does not capture projectiveness; instead, they enforce that the entire witness must appear in the table. We generalize our definitions to the projective setting and demonstrate that existing techniques—such as the logup lemma (and works building on it, including cq [EFG22] and Protostar [BC23]), as well as the matrix–vector technique (e.g., Lasso [STW24])—naturally extend to the projective setting with minimal overhead.

State of the art and applications. We survey the state of the art in lookup arguments, focusing on three main aspects. First, we distill and present the main algebraic techniques underlying existing lookup arguments, and we map them to the existing schemes (see Section 4). The main categories include multiset equality, accumulators, Logup-based, and subvector extraction (matrix-vector). Second, in Section 5, we examine two practical aspects of lookup arguments: (i) their compatibility, meaning which SNARKs they can be efficiently integrated with, and (ii) their efficiency, in particular how performance scales with the size of the lookup table. When it comes to supporting large tables (i.e., $|\mathbf{t}| \gg \mathbf{w}$), our analysis points out limitations in the state of the art, which either assumes the tables to have some structure (e.g., they can be decomposed in the product of smaller tables or be described by a short function) or relies on preprocessing methods that are concretely too expensive to scale (see Figure 1, we leave a full discussion to the full version). Third, we provide an overview of the most important applications of lookup arguments, identifying which features matter most in different applications. We group these applications in three main categories: replacement for non-native or non-arithmetic operations, set membership, and verifiable memory.

Through this systematization, our work aims to provide a unified understanding of lookup arguments, distilling lessons and guidelines for choosing lookup constructions in practice and highlighting the benefits and limitations of emerging directions in literature, such as preprocessing and decomposability.

Full version. A full version of this paper is available in [Haf+25]. We briefly summarize the main differences here. In the conference version, we omit the



$*$ ≈ "regardless of whether yes/no" *Performance:* ✓/✓✓ ≈ "good/best" ▲ ≈ "often undesirable"

Fig. 1: *Above*: mapping requirements on lookup relations to properties in respective schemes. *Below*: an informal performance taxonomy of schemes for large tables.

formal definitions of cryptographic primitives, as well as some extended discussions. Finally, the full version includes a comprehensive survey of existing schemes based on various techniques, which we omit in this version due to space constraints.

2 Preliminaries

Notation. Let $\mathbb{F}$ be a finite field and $\mathbb{G}$ a group with scalars in $\mathbb{F}$, with additive notation. For $a \in \mathbb{F}$ and $g \in \mathbb{G}$, the scalar multiplication of g by a is denoted as $a \times g$. For an asymmetric pairing group, we define it as a tuple $(p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where p is the order of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and e is an efficiently computable, non-degenerate bilinear map. For a natural number n, $[n]$ denotes the set of integers $\{0, 1, \ldots, n-1\}$.

Given a vector $\mathbf{w}$, we denote its associated oracle by $[[\mathbf{w}]]$. The double-bracket notation indicates an *oracle handle* to the underlying data. While this notation can represent various interpretations (depending on context), we do not explicitly distinguish between them for the sake of clarity. For example, $\mathbf{w}$ might represent the evaluations of a multilinear polynomial over the Boolean hypercube. In this case, the oracle $[[\mathbf{w}]]$ allows the client to query the corresponding multilinear polynomial at any point in the domain. Let $\mathbf{t} \in \mathbb{S}^N$ denote the lookup table, represented as a vector of length N, and let $\mathbf{w} \in \mathbb{S}^n$ denote the witness, represented as a vector of length n. We use the notation $M_{n\times m}$ to refer to a matrix with n rows and m columns; $M_{i,j}$ denotes the entry on row i and column j of M.

Polynomial commitment scheme (PCS) [KZG10]. PCS is a cryptographic primitive that enables a prover to *commit* to a polynomial succinctly and later prove the correctness of its evaluation at some chosen points from the domain, without revealing the full polynomial. When a verifier requests an opening at a point x, the prover returns both the claimed evaluation $y = p(x)$ and a proof π attesting to its correctness. The verifier can efficiently check that y is the correct value of the committed polynomial at x using the proof π. We defer a formal definition to the full version of the paper. Here, the polynomial can be of different types, such as univariate, multilinear, or multivariate.

Sumcheck protocol [Lun+92]. Let g be some ℓ-variate polynomial defined over a finite field $\mathbb{F}$. The purpose of the sumcheck protocol is for the prover to provide the verifier with the following sum: $H := \sum_{\mathbf{b} \in \{0,1\}^\ell} g(\mathbf{b})$. To compute H unaided, the verifier would have to evaluate g at all 2^ℓ points in $\{0,1\}^\ell$ and sum the results. The sumcheck protocol is an interactive protocol that allows the prover to reduce a claim on summation above to a claim on an evaluation of g at a random point. The verifier's runtime is $O(\ell)$, plus the time required to evaluate g at a single point $\mathbf{r} \in \mathbb{F}^\ell$, this evaluation can be provided by the prover directly by committing to g and opening it at point $\mathbf{r}$. This is exponentially faster than the 2^ℓ time that would generally be required for the verifier to compute H.

Argument of knowledge. An argument of knowledge (called an SNARK if succinct) lets a prover prove to a verifier that it knows a witness to an NP statement. It can support properties such as *succinctness*, i.e., the verifier's computation being sublinear in the witness size or *zero-knowledge*, i.e., the verifier learns nothing beyond the satisfiability of the statement. We leave formal definitions to the full version of the paper.

Accumulation scheme [BC23]. An accumulation scheme is a fundamental building block for recursive proof systems. At a high level, it enables a prover and a verifier to reduce the satisfiability of two NP statements into a single new statement through a lightweight interaction between the two parties. We leave a formal definition of an accumulation scheme and how it relates to recursive proofs to the full version of the paper.

Commit-and-prove SNARKs [CFQ19]. The commit-and-prove paradigm enables one to prove *commit-and-prove relations*. In such relations, the instance contains a commitment to part of the witness, even potentially the entire witness. This feature is handy for composing proof systems. For example, when composing a lookup argument with another proof system, the instance of the proof system contains a commitment to the relevant portion of the witness (which may even be the entire witness) that will be checked against the lookup table. Since lookup arguments are naturally commit-and-prove (their instances also include commitments to the looked-up values), this shared commitment structure allows for seamless composition between the two arguments.

Multilinear extensions. An n-variate polynomial $p : \mathbb{F}^n \to \mathbb{F}$ is said to be *multilinear* if p has degree at most one in each variable. Let $f : \{0,1\}^n \to \mathbb{F}$ be any function mapping the n-dimensional Boolean hypercube to a field $\mathbb{F}$. Then there exists a unique multilinear polynomial $\tilde{f} : \mathbb{F}^n \to \mathbb{F}$ that extends f to $\mathbb{F}^n$. This is referred to as the *multilinear extension (MLE)* of f. We will denote by $\widetilde{\mathsf{eq}}$ the

MLE of the function $\mathsf{eq} : \{0,1\}^s \times \{0,1\}^s \to \{0,1\}$ such that $\mathsf{eq}(x,e) = 1$ if and only if $x = e$. An explicit expression for $\widetilde{\mathsf{eq}}$ is:

$$\widetilde{\mathsf{eq}}(x,e) = \prod_{i=1}^{s} \big(x_i e_i + (1-x_i)(1-e_i)\big).$$

Multilinear extensions of vectors. Given a vector $u \in \mathbb{F}^m$, we denote its multilinear polynomial by $\tilde{u}$, obtained by viewing u as a function mapping $\{0,1\}^{\log m} \to \mathbb{F}$ in the natural way. A matrix $M_{n\times m}$ can also be seen as a function $M : \{0,1\}^{\log n} \times \{0,1\}^{\log m} \to \mathbb{F}$; its multilinear extension is defined as a natural extension of the vector case.

3 A Unified Definitional Framework for Lookups

In this section, we present a modular framework for defining *lookup arguments*. Unlike most of the prior work—which typically defines the lookup argument directly or encodes the lookup witness and table as evaluations of a univariate polynomial (e.g., using KZG [KZG10])—our approach is modular. This modularity facilitates seamless composition of the lookup argument with existing proof systems. At its core, in Section 3.1, we define a *lookup relation* (Definition 1)—a polynomial-time relation that asserts whether a given vector appears within a fixed table. We begin by formalizing a basic (deterministic) relation for lookup: given a table $\mathbf{t}$ parametrizing the relation, a vector $\mathbf{w}$, passed as public input, is contained in $\mathbf{t}$. The deterministic relation above is not very useful on its own. In fact, the real power of lookup stems from its ability to be composed with other relations. In Section 3.2, we then extend it to its more useful, non-deterministic version where the verifier only has a "handle" to the witness. We introduce this non-deterministic relation in two forms to support different modes of composition with proof systems: the *committed lookup relation* (Definition 2) and the *oracle lookup relation* (Definition 3). In each, the "handle" is instantiated with a commitment and an oracle, respectively. We elaborate on both approaches later in this section.

In Section 3.3, we define *projective lookup*, a generalization of the standard lookup relation. Unlike the standard lookup, where the prover demonstrates that *all* the elements in the witness belong to a table, projective lookups allow the prover to assert that *only specific indices of the witness* appear in the table. This variant aligns better with practical use cases, in which a proof system often requires checking membership only for specific indices of the witness (e.g., verifying that certain elements in the witness are small). We next define two further generalizations of the lookup relation: *indexed lookup* and *vector lookup*, presented in Sections 3.4 and 3.5, respectively. In an indexed lookup, both the witness and the table are treated as ordered lists (i.e., vectors), and the order of elements becomes significant. In this setting, the verifier receives a commitment to the relevant subtable along with a commitment to the indices of its elements in the original table. This formulation enables new applications, such as proving custom functions without encoding the function logic directly into the arithmetic circuit. We then define *vector lookup*, a natural extension where each entry in the table is a tuple of elements rather than a single element. Next, in Section 3.6, we define the notation of *an online lookup table*, such that the table itself is part of the witness, instead of being in the index. In the full version of the paper, we also define the notion of *decomposability*, a property of lookup tables that enables the prover to reduce lookups in a large table to lookups in a number of smaller tables, improving the prover's efficiency. This is particularly useful for constructing lookups over extensive tables (e.g., size 2^{128}), which cannot be materialized explicitly.

3.1 Lookup relation

Definition 1 (Lookup Relation). *Given a finite set* $\mathbb{S}$*, a lookup index is defined as* $\mathrm{I} := (\mathbb{S}, n, \mathbf{t})$ *where* n *is the number of lookups and* $\mathbf{t} \in \mathbb{S}^N$ *is the table, the relation* $\mathrm{LK_I}$ *is the set of tuples* $\mathbf{w} \in \mathbb{S}^n$ *such that* $w_i \in \mathbf{t}$ *for all* $i \in [n]$*. In other words:*

$$\mathrm{LK_I} = \{\mathbf{w} \in \mathbb{S}^n \, : \, \mathbf{w} \subseteq \mathbf{t}\}$$

Note that an index is valid if $0 < n$*,* N *and* $\mathbf{t} \in \mathbb{S}^N$*. We slightly abuse notation by extending the subset relation* $A \subseteq B$ *to apply to multisets as well.*

What is the finite set $\mathbb{S}$? The lookup relations introduced above are defined generically over a finite set $\mathbb{S}$. In practice, different lookup constructions instantiate $\mathbb{S}$ with specific finite structures—most commonly a finite field—to exploit certain algebraic properties. Early constructions, particularly those based on the KZG commitment scheme, typically operated over large prime fields of 256 bits [GW20; GK22; Zap+22a; PK22; Zap+22b; EFG22]. Other approaches, such as Logup [Hab22] and Lasso [STW24], introduced techniques that extend the applicability of lookup relations to smaller prime fields [Pol22] (e.g., 64-bit fields), binary fields [DP25], and even to rings in recent work [Cry; Gar+25].

Treating table and witness as vectors instead of sets. In the discussion above, we defined the lookup relation using the convention that the table $\mathbf{t}$ is represented as an ordered tuple in $\mathbb{S}^N$, i.e., as a vector. Alternatively, one could define lookup relations over sets rather than vectors. Some constructions [Cam+22] support only this *set-based* lookup formulation. However, this perspective is generally less compelling for two main reasons. First, the commitment schemes, such as *vector* [CF13] or *polynomial commitment schemes*, employed in SNARKs naturally treat the witness as an ordered list rather than sets. Second, the compiler from general lookup relations to indexed lookup relations is not applicable in the set-based setting, yet indexed lookups are a crucial form used in many SNARK constructions. We also regard the witness $\mathbf{w}$, which is originally a multiset, as an ordered tuple in $\mathbb{S}^n$. As we will see in the next section, the witness must be committed by the prover, while the verifier only receives a commitment to it. Since there is no standard notion of a *multiset commitment scheme*, we instead model the lookup instance as an ordered collection, such as a vector. Consequently, we assume that it is committed using, for instance, a vector commitment scheme or a polynomial commitment scheme.

3.2 Lookup relation interfaces for composition

The relation LK_{I} belongs to the class P because it does not involve a witness; it simply consists of an index and an instance $\mathbf{w}$. We now introduce the notion of a *committed lookup relation*, where the instance is a commitment to $\mathbf{w} \in \mathrm{LK}_{\mathrm{I}}$, and the witness is $\mathbf{w}$ in plaintext.

Definition 2 (Committed Lookup Relation). *Given lookup index* $\mathrm{I} := (\mathbb{S}, n, \mathbf{t})$ *and its corresponding relation* LK_{I} *and a commitment scheme* $\mathcal{C} = (\mathrm{Com}, \mathrm{Open})$ *on* $\mathbb{S}^n$*,* CLK *is defined as follows:*

$$\mathrm{CLK}_{\mathrm{I},\mathcal{C}} = \{(c; \mathbf{w}) \,:\, \mathbf{w} \in \mathrm{LK}_{\mathrm{I}},\, c = \mathrm{Com}(\mathbf{w})\}$$

Even though a committed lookup relation is a natural notation, it isn't easy to compile it into a PIOP directly. That's why, similar to HyperPlonk [Che+23], we define the following notion of lookup relation where the verifier has oracle access to the witness. In practice, the oracle is typically instantiated using a polynomial commitment scheme. However, treating this commitment as an oracle enables the composition of the oracle lookup relation with the proof system at the PIOP level. Concretely, one can construct a PIOP for each relation, compose them sequentially, and finally compile the protocol into an argument of knowledge by replacing the oracles with polynomial commitments.

Definition 3 (Oracle Lookup Relation). *Given lookup index* $\mathrm{I} := (\mathbb{S}, n, \mathbf{t})$ *and its corresponding relation* LK_{I}*,* OLK *is defined as follows:*

$$\mathrm{OLK}_{\mathrm{I}} = \{([[\mathbf{w}]]; \mathbf{w}) \,:\, \mathbf{w} \in \mathrm{LK}_{\mathrm{I}}\}$$

Composition of lookup arguments with proof systems. We define both the committed lookup relation and the oracle lookup relation to support different modes of composability. There are two main ways to compose a lookup with an existing proof system[6]. The first approach is composition at the argument level, leveraging the commit-and-prove paradigm [CFQ19], where both the lookup argument and the underlying proof system share commitments to the same witness. This shared commitment facilitates seamless composition. The second approach operates at the PIOP level, by sequentially composing PIOPs [BSCS16]. More concretely, assuming the existence of a PIOP for the lookup relation and one for the main proof system, one can compose them directly. The composed PIOP can then be compiled into an argument of knowledge by replacing the interactive oracles with polynomial commitments. Each approach suits different contexts. For instance, in Groth16 [Gro16], the argument is constructed directly without

[6] This discussion applies more generally to proof schemes beyond lookups.

relying on a PIOP. To compose it with a lookup argument, one needs a commit-and-prove variant such as Mirage [Kos+20], together with a commitment scheme compatible with the committed lookup relation. In contrast, for systems such as Plonk [GWC19], which first defines a PIOP and then compiles it into an argument using a PCS, we can directly compose the lookup PIOP with the proof system's PIOP and then compile the composed PIOP into an argument of knowledge. Generally, composition via the commit-and-prove paradigm is more general, since it allows compiling the PIOPs into knowledge arguments first and then composing them via the commit-and-prove paradigm.

One key difference is that the commit-and-prove paradigm offers better modularity and a more plug-and-play architecture. In this paradigm, the underlying components, namely the lookup argument and the main proof system, remain separate and need only share a single commitment. This separation simplifies the developer experience, making it easier to implement and maintain. In contrast, composing components at the PIOP level requires tightly coupling the lookup argument with the proof system. Although this makes development more complex, it can lead to greater efficiency and smaller proof sizes. For example, it enables batching of the lookup polynomial openings with those of the main proof system, reducing overall proof size.

Preprocessing PIOP [Chi+20; BFS20]. A preprocessing PIOP is a type of PIOP that frequently appears in the context of lookup PIOPs. At a high level, this protocol consists of two phases: an *offline phase* and an *online phase*. In the offline phase, an honest indexer, given the index of the relation, generates a set of preprocessed polynomials p_i. These polynomials are provided to the prover and can be queried by the verifier during the online phase. The online phase proceeds similarly to a standard PIOP. The prover, given the instance, the witness, and access to the preprocessed polynomials, interacts with the verifier. The verifier, in turn, has access to the instance and oracle access to the preprocessed polynomials. The goal of this interaction is for the prover to convince the verifier that it possesses a valid witness for the given instance. A formal definition of a preprocessing PIOP is deferred to the full version of the paper. In the context of preprocessing PIOPs for lookups, intuitively, the offline phase can be seen as a preprocessing stage where, given the table $\mathbf{t}$, an honest preprocessor computes some auxiliary polynomials. These polynomials are designed to enable the prover to perform its computation more efficiently during the online phase.

Lookup argument. A lookup argument is an argument of knowledge for the committed lookup relation. Such an argument can be constructed either directly, for the committed lookup relation, or by first designing a (possibly preprocessing) PIOP for the oracle lookup relation and then compiling it into an argument of knowledge by replacing the oracles with a polynomial commitment scheme.

Efficiency of lookup arguments. Following the terminology of prior work [CFG25], we define a lookup argument as *sublinear* if the prover's runtime is *sublinear* in the size of the lookup table $\mathbf{t}$. If the prover's runtime is *independent* of the size of $\mathbf{t}$, we refer to the argument as *super-sublinear*. Similarly, consider a compound PIOP composed of a lookup PIOP and the main proof system PIOP. When this compound PIOP is compiled into an argument of knowledge, we say that the resulting lookup argument is *sublinear* if the prover runtime is sublinear in the size of $\mathbf{t}$ and *super-sublinear* if the runtime is completely independent of the size of $\mathbf{t}$. We do not define a lookup PIOP itself to be efficient, since the efficiency of the resulting argument also depends on the polynomial commitment scheme used to compile it into an argument. For example, consider cq, which is a generic PIOP that can be compiled with different polynomial commitment schemes beyond KZG. In such cases, the prover's running time is not necessarily independent of the table size. The efficiency of cq specifically arises from its compilation with KZG and the underlying properties of KZG, such as homomorphism.

Zero-knowledge lookup. Zero-knowledge for lookup arguments is defined analogously to standard arguments of knowledge. Specifically, given a lookup instance and an index, there exists a polynomial-time simulator that generates a proof computationally indistinguishable from a real proof. Since the table is considered part of the public index rather than the witness, it is not hidden in the proof; however, there are examples such as Duplex [Han+25] and cq++ [Cam+24], which consider the table as the witness, so it is hidden too. Most lookup schemes are typically presented as a lookup PIOP, which can be compiled into a zero-knowledge argument when the underlying PIOP is zero-knowledge and the polynomial commitment is hiding and supports zero-knowledge opening too. Nevertheless, most existing lookup PIOP schemes are introduced without zero-knowledge, leaving it for extension. For example, PIOPs described in cq [EFG22] and Lasso [STW24] are not zero-knowledge, whereas schemes such as Caulk [Zap+22a] and Caulk+ [PK22] do provide zero-knowledge guarantees.

3.3 Projective lookup

Lookup tables are primarily used as subroutines in proof systems to verify that specific components of a witness vector are contained within a predefined table, rather than checking the entire witness. This selective verification is not compatible with the standard lookup definitions previously introduced, such as Definition 1, which require the full witness vector to match a table entry. To bridge this gap, inspired by [Ang+24; CFG25], we introduce the notion of a *projective lookup*. In a projective lookup, given a witness $\mathbf{w} \in \mathbb{S}^m$, we fix a set of indices $\mathbf{i} = (i_1, i_2, \ldots, i_n)$ and check whether the projection $(w_{i_1}, \ldots, w_{i_n})$ belongs to the table $\mathbf{t}$. This concept, referred to as the *projective lookup relation*, allows proof systems to verify properties of partial witnesses—for example, ensuring that some components consist of small field elements—thereby supporting more flexible and practical applications.

Definition 4 (Projective Lookup Relation). *Let $(\mathbb{S}, n, \mathbf{t})$ be a lookup index. We define a projective lookup index as $\mathrm{I} = \big((\mathbb{S}, n, \mathbf{t}),\, m,\, \mathbf{i} = \{i_1, i_2, \ldots, i_n\}\big)$, and the corresponding projective lookup relation is defined as follows:*

$$\mathrm{PLK}_{\mathrm{I}} = \{\mathbf{w} \in \mathbb{S}^m \,:\, \mathbf{w}_{\mathbf{i}} \subseteq \mathbf{t}\}$$

$\mathbf{w_i}$ *denotes the subset of* $\mathbf{w}$ *indexed by* $\mathbf{i}$. *The projective index* I *is valid if* $(\mathbb{S}, n, \mathbf{t})$ *is valid lookup index and* $\mathbf{i} = \{i_1, i_2, \ldots, i_n\}$ *satisfies* $0 \leq i_1 < \cdots < i_n < m$.

Similar to the standard lookup relations, we can define *projective versions* of the committed lookup relation and the oracle lookup relation, e.g., *the projective oracle lookup relation* would be pairs $([[\mathbf{w}]], \mathbf{w})$ such that $\mathbf{w} \in \mathrm{PLK}_{\mathrm{I}}$. In the way projective relations are defined above, the indices $i_1, i_2, \ldots, i_n$ are fixed within the witness by embedding them directly into the index. This reflects a common practice in SNARKs, where the statement is typically preprocessed, and we know in advance which witness wires need to be checked for inclusion in a lookup table. Essentially, to prove such a statement, we must compose a projective oracle lookup PIOP with the underlying proof system's PIOP. The central question then arises: since most prior work only addresses the non-projective setting, is it possible to efficiently construct a PIOP for the projective case given an existing PIOP designed for the non-projective setting?

How to Design a PIOP for Projective Lookup Relations? As discussed earlier, lookup tables serve as a useful subprotocol component. The key challenge lies in partially selecting the witness and proving that it appears in a predefined table. A natural approach is to commit to a subvector and demonstrate that it matches the entries at specific indices. However, doing so efficiently is non-trivial; for example, without constructing a commitment within the arithmetic circuit, concretely because the size of the underlying vector which is committed to changes. In what follows, we explore several strategies for adapting or extending existing techniques, such as the Logup lemma and approaches based on matrix-vector products, to efficiently support projective lookup without requiring the explicit construction of the subvector. We extend the Logup lemma to the projective setting (deferred to the full version) and, based on this, generalize `cq` and Protostar to support projective lookups. This approach can also be applied to other lookup arguments that rely on Logup. We also introduce a way to modify Lasso to support projective lookups, with almost no overhead. We defer this extension of Lasso to the full version of the paper, too. Another approach, briefly noted in Halo2 [Zca25], is the *polynomial selector* technique. At a high level, the idea is that the prover modifies the witness polynomial so that the indices corresponding to the intended lookups remain unchanged, while all other positions are filled with a trivial element from the lookup table. As a result, verifying membership of this modified witness is equivalent to performing a projective lookup on the original witness.

3.4 Indexed lookup

Indexed lookup [STW24] extends the standard definition of lookup tables. In the traditional lookup setting, there is no requirement for the table $\mathbf{t}$ or the witness $\mathbf{w}$ to be ordered lists—these can even be modeled as a set and a multiset, respectively.[7] The indexed lookup setting introduces two key differences: (I) both

[7] As discussed in Section 3.1, our definitions treat both the witness $\mathbf{w}$ and table $\mathbf{t}$ as ordered lists because that is the case of interest in proof systems.

the table and the witness are assumed to be ordered lists, also known as vectors, and (II) the verifier additionally receives a commitment to the indices of the witness elements in the table (see Definition 5).

Definition 5 (Indexed Lookup Relation). *Given a* lookup index $\mathrm{I} := (\mathbb{S}, n, \mathbf{t})$*, an* indexed lookup relation *is the set of tuples* $(\mathbf{w}, \mathbf{i})$ *such that* $\mathbf{w} \subseteq \mathbb{S}$ *and* $\mathbf{i} \subseteq [N]$ *where* $|\mathbf{i}| = |\mathbf{w}| = n$*, and for all* $k \in [n]$*, the entry* w_k *equals the table entry at index* i_k*, i.e.,* $w_k = t_{i_k}$*. Formally:*

$$\left\{(\mathbf{w}, \mathbf{i}) \,:\, \forall j \in [n],\ w_j = t_{i_j}\right\}.$$

Indexed lookups offer greater expressiveness than standard lookups and enable a wider range of applications. While standard lookups are sufficient for simple tasks such as range checks—ensuring that certain witness values lie within a predefined set—they fall short when implementing arbitrary functions. Consider a function $f : [N] \to \mathbb{S}$, and let $\mathbf{t}_f$ be a table such that the entry at index i is $f(i)$. With indexed lookup tables, the function f can be externally represented via table $\mathbf{t}_f$, avoiding the need to encode f directly in the circuit. This is the core idea behind Jolt [AST24], which constructs tables for all CPU operations (e.g., AND, OR, etc.), resulting in significantly more efficient circuits for general-purpose programs.

Generic compiler from standard to indexed Lookup [STW24]. Lasso introduces a general compiler to transform a standard lookup argument into an indexed one when the underlying set is a field with a large enough characteristic, i.e., $\mathbb{S} = \mathbb{F}$. Let $\mathbf{t} \in \mathbb{F}^N$ be a lookup table, and suppose that all table entries lie in the range $[r]$. Assume that $m \cdot N$ is less than the characteristic of the field $\mathbb{F}$. We define a modified table $\mathbf{t}^* \in \mathbb{F}^N$ where each entry is encoded as $t_i^* = i \cdot r + t_i$. For each standard lookup pair (b_j, a_j), define the combined field element $a_j^* = b_j \cdot r + a_j$. A range check must be applied to ensure that $a_j \in [r]$ for each lookup. Under this constraint, the equality $t_{b_j} = a_j$ holds if and only if $t_{b_j}^* = a_j^*$. Conversely, if $t_{b_j} \neq a_j$, then $a_j^* \notin \mathbf{t}^*$. This compiler introduces an additional range check, which increases its cost. As we will see in the next section, some constructions, such as Lasso—natively support indexed lookups. In contrast, others, such as constructions based on the Logup lemma, can be extended to handle indexed lookups more efficiently through vector lookups.

Indexed lookup via vector lookups. A natural method to enable indexed lookups is to use vector lookups. Suppose we have a table $\mathbf{t}$ indexed by i with entries t_i. If the underlying lookup relation supports vector lookups, we can treat the table as a vector (or tuple) of pairs (i, t_i). This approach can be remarkably efficient—for instance, when using the Logup lemma—since it natively supports vector lookups and avoids the need for additional checks, such as range constraints.

3.5 Vector lookup

Vector lookup is a natural generalisation of standard lookup, where each entry in the table is a tuple of elements from $\mathbb{S}$, rather than a single element from $\mathbb{S}$.

This concept has appeared in different works [GW20; Hab22; Cam+24; Cho+24; Di+24][8]. We formally define the vector lookup relation in Definition 6.

Definition 6 (Vector Lookup Relation). *Given a finite set* $\mathbb{S}$*, a* vector lookup index *is defined as* $\mathrm{I} := (\mathbb{S}, n, k, \mathbf{t})$*, where* n *is the number of lookups, and* $\mathbf{t} \in \left(\mathbb{S}^{(k)}\right)^N$ *is a table consisting of* N *tuples of size* k *over* $\mathbb{S}$*. The corresponding* vector lookup relation *is the set of* k*-tuples over* $\mathbb{S}$ *of size* n *such that each* k*-tuple appears in the table* $\mathbf{t}$*. Formally,*

$$\left\{ \mathbf{w} \in \left(\mathbb{S}^{(k)}\right)^n \,\middle|\, \forall i \in [n],\, \mathbf{w}_i \in \mathbf{t} \right\}.$$

Certain techniques, such as the Logup lemma, naturally extend to vectors. The description of the extended lemma is deferred to the full version of the paper. For instance, MuxProofs [Di+24] and Subplonk [Cho+24] construct their vector lookup arguments based on this extension. Another line of work, introduced in Plonkup and later refined by Campanelli et al. [Cam+24], leverages *homomorphic tables* [EFG22] to realize vector lookups when the underlying lookup table supports homomorphic proofs.

Vector lookup from homomorphic proofs. A straightforward way to support vector lookups over tuples of size k using a lookup argument supporting *aggregatable* proofs (e.g., homomorphic proofs) is to rely on k separate lookup tables and aggregate the lookup proofs. However, even with proof aggregation, the verifier's running time remains linear in k using existing homomorphic table schemes, since the verifier must compute a random linear combination of the proofs. This overhead becomes inefficient for large tuple sizes. An alternative solution for vector lookups, in which the verifier's time does not depend on the tuple size, was suggested by Campanelli et al. [Cam+24]. At a high level, the technique linearizes the lookup table (and subtable) of k-tuples into a table of 3-tuples, where each entry contains the element, its position within the tuple, and its index in the table. For example, given a tuple $\mathbf{r}$ of length k, the transformation produces $\{(x_i, y_j, r_i)\}_{i\in[k],\, j\in[N]}$, where x_i denotes the position of element r_i within the tuple, and y_j denotes the index of the tuple in the lookup table. This transformation requires additional consistency checks to ensure well-formedness, namely that the resulting vector indeed corresponds to a concatenation of k-tuples. For this transformation to be valid, all x_i values must be equal across the tuple. After this transformation and the corresponding correctness checks, the lookup reduces to a lookup over 3-tuples. This can be interpreted as three separate lookup tables, with the proofs then aggregated. Concretely, Campanelli et al. [Cam+24] employs cq, which supports homomorphic proof aggregation.

Generalized vector lookups. In Definition 6, the table can be seen as composed of *tuples*, and our witness corresponds to a vector of tuples in the table. That is, what we are looking up are tuples *in their entirety*. In some settings, we may want to lookup just part of the tuple (and we may not be interested in paying

[8] These works use different terminology of segment lookup, matrix lookup and vector lookup. We prefer to go with the more natural choice of vector lookup here.

the additional overhead of simply applying a vector lookup); for example, the witness could consist of a pairs $(a_i, b_i)_i$, such that there exists some tuple in the table such that the first element is equal to a_i and the tenth element is equal to b_i. We do not further discuss this more general notion here, but we refer the reader to [Cam+24, Appendix D.1] where the authors define and construct this different notion.

3.6 Online lookup table

In standard lookup table use, the table is fixed and can often be preprocessed. However, in several cases, the table is not fixed during setup but instead depends on the witness. A well-known example is Spark [Set20], which constructs an online table[9] for $\widetilde{\mathsf{eq}}(\mathbf{x}, \mathbf{r})$ over a set of $\mathbf{x} \in \{0,1\}^n$, where $\mathbf{r}$ is the verifier's challenge. Similar ideas arise in distributed proving systems such as Hekaton [Ros+24] and Soloist [Li+25], which generate online lookup tables for mutual witness wires shared across subcircuits. Since these tables depend on the witness, they cannot be preprocessed during the offline phase, making them incompatible with preprocessing schemes such as cq [EFG22]. Exploring additional applications of online lookup tables remains an interesting direction for future work. To formalize this setting, we introduce the following definition of a lookup relation where the table itself appears within the lookup instance rather than in the index. We emphasize that all previous definitions—such as the committed lookup relation, oracle lookup relation, and projective lookup etc.—can be extended to the *online* setting as well. For simplicity, we omit these generalizations here.

Definition 7 (Online Lookup Relation). *Given a finite set $\mathbb{S}$, a lookup index is defined as $\mathrm{I} := (\mathbb{S}, n, N)$, where n is the number of lookups and N is the size of the table. The relation consists of all tuples $(\mathbf{w}, \mathbf{t})$ such that $\mathbf{w} \in \mathbb{S}^n$ and $\mathbf{t} \in \mathbb{S}^N$, with the condition that $w_i \in \mathbf{t}$ for all $i \in [n]$. Equivalently, $\{(\mathbf{w}, \mathbf{t}) : \mathbf{w} \subseteq \mathbf{t}\}$.*

4 An Overview of Existing Techniques

In this section, we survey existing schemes categorised based on their underlying technique. In Table 1, we give a non-exhaustive list of schemes ordered by techniques together with their compatibility and some asymptotic costs. Given the variety of possible instantiations and thus the complexity of comparing these schemes, our primary goal is to provide insight to those with explicit constraints in mind rather than to present an overall winner.

Multiset equality technique. Early approaches to (unindexed) lookup arguments namely Arya [Boo+18], Plookup [GW20] and Halo2 [Zca25] are all based on the notion of *multiset equality*. The central idea is that since verifying subset inclusion is more involved than checking multiset equality, these works reduce the problem to a multiset equality check, i.e., $\mathbf{w} \subseteq \mathbf{t}$ if and only if $\mathbf{w} \cup \mathbf{t}$ contains

[9] In Spartan, this is referred to as a *write-once-read-many* memory scheme.

the same elements as $\mathbf{t}$, up to multiplicities. A shared characteristic among these approaches is that the prover's cost depends on the size of the table, making them most efficient when $|\mathbf{t}| \leq |\mathbf{w}|$. We observe that the underlying multiset equality techniques used in Plookup and Halo2 do not depend on a large field size, and they require no preprocessing (aside from committing to the table and the witness), which makes them suitable for online lookup tables. For example, VerITS [DCB25] applies Plookup over a prime field of size $2^{31}-1$. RISC0 [ris25b] utilizes Plookup as its underlying lookup argument, demonstrating efficiency even with hash-based polynomial commitment schemes.[10] We defer a full review on the existing schemes based on the multiset equality notion to the full version of the paper.

Logup-based. This is a different family of lookup table arguments that reduces a lookup relation into a summation over rational functions. This line of work began with [Eag+24] and was later formalized as a lookup argument in [Hab22]. To prove that a witness vector $\mathbf{w}$, encoded as polynomial $W(x) = \prod_{i\in[n]}(x+w_i)$, is a subset of a lookup table $\mathbf{t}$, encoded as $T(x) = \prod_{i\in[N]}(x+t_i)$, it suffices to show that there exist non-negative integers $\{m_i\}$ such that $W(x) = \prod_{i\in[N]}(x+t_i)^{m_i}$. However, directly proving this identity leads to high costs, similar to those in Arya. The key insight is that this polynomial identity holds if and only if their logarithmic derivatives match. In other words, for two polynomials g_1 and g_2, we have:

$$g_1(x) = g_2(x) \iff \frac{d}{dx}\log g_1(x) = \frac{d}{dx}\log g_2(x).$$

$$\xRightarrow[g_2(x)=\prod_{i\in[N]}(x+t_i)^{m_i}]{g_1(x)=W(x)} W(x) = \prod_{i\in[N]}(x+t_i)^{m_i} \iff \sum_{i\in[n]}\frac{1}{x+w_i} = \sum_{i\in[N]}\frac{m_i}{x+t_i}.$$

The original work introduces what is referred to as the Logup, which reformulates the lookup inclusion condition as a rational function identity. One key idea is that verifying grand summations is generally simpler than verifying grand products, for example, by using the univariate or standard sumcheck. The Logup lemma does not rely on a large field size. For instance, Stwo [Sta25] and SP1 [Lab25], the state-of-the-art STARK-based zkVMs employ Logup for lookups, demonstrating the technique's efficiency even when instantiated with non-homomorphic PCSs, such as FRI-based ones. Existing work such as Logup [Hab22], `cq` [EFG22], and their extensions [PH23; Cam+24; ZSG24; Dor24; BC24; CFG25] build on the Logup lemma; a detailed overview of these schemes is provided in the full version of the paper.

Subvector extraction (matrix-vector) technique. One general technique to verify a lookup relation is *subvector extraction*, commonly implemented as a matrix–vector multiplication, i.e., a sparse constraint system of rank one. The high-level idea is as follows: if the witness $\mathbf{w}$ is included in the table $\mathbf{t}$, then for every w_i that equals t_j, there exists an elementary vector e_j of the same length as $\mathbf{t}$ such that $e_j \cdot \mathbf{t} = w_i$. Here, e_j is zero everywhere except at index j, where

[10] They plan to transition from Plookup to Logup-based techniques in their upcoming version [ris25a].

it is equal to 1. By concatenating these elementary vectors as rows, we obtain an *elementary matrix* $M_{n\times N}$, where each row has exactly one nonzero entry equal to 1. The lookup relation is then equivalent to the matrix equation $M \times \mathbf{t} = \mathbf{w}$. With this new interpretation, the verifier is given commitments to a table $\mathbf{t}$ and a witness $\mathbf{w}$. The prover's goal is to prove that there exists an elementary matrix M such that $M \times \mathbf{t} = \mathbf{w}$. Existing approaches based on this technique include Caulk [Zap+22a] and Caulk+ [PK22], which perform subvector extraction implicitly, as well as Baloo [Zap+22b], Lasso [STW24], and Shout [ST25], which carry out the extraction explicitly using the matrix–vector technique. The essence of the matrix–vector technique does not rely on a large field size; for instance, Lasso has been adapted to binary fields [DP25] as well as to ring elements [Cry]. An overview of these approaches is provided in the full version of the paper.

Accumulator-based. Another approach to building lookup arguments is through set accumulators that support *commit-and-prove batch openings*. In a nutshell, these accumulators allow one to commit to the table $\mathbf{t}$ succinctly and to a subtable $\mathbf{t}'$, and to generate a succinct proof that $\mathbf{t}' \subseteq \mathbf{t}$. Flookup [GK22] follows this approach, with pairing-based accumulators, to extract the subtable of all witness entries and then gives a protocol to show that these entries of the committed witness are in the subtable. Other works [Cam+22; Han+25] instead rely on RSA accumulators, in which the table must be encoded with prime numbers, and then use specialized protocols to efficiently link RSA accumulators with SNARKs.

Index preprocessing. One theoretical technique used to build lookup arguments is polynomial preprocessing [Cam+25]. To provide background, in Lasso [STW24], the authors introduce a variant of their protocol called *generalized Lasso* that constructs a lookup PIOP using a sparse sumcheck protocol. However, this construction relies critically on the fact that the *table is structured* and can therefore its multilinear extension can be evaluated efficiently—meaning it only applies to a restricted class of tables. The key idea in Campanelli et al. [Cam+25] is to preprocess *any* table so that its evaluations can be performed efficiently, in sublinear time with respect to the size of the polynomial (or table). This allows them to employ generalized Lasso to design an efficient lookup PIOP even for unstructured tables. The polynomial preprocessing technique builds on prior works [KU08; KU11], which, at a high level, show that given a polynomial, one can preprocess it so that evaluations can be computed in sublinear time while requiring only polylogarithmic storage. The authors note, however, that their approach is unlikely to be immediately practical due to the large constants involved in the specific preprocessing scheme [KU11].

Lookup accumulation. Accumulation schemes are an efficient way to build recursive proof systems such as Incremental Verifiable Computation (IVC) [Val08]. The use of lookup tables in the context of recursive SNARKs is non-trivial. For example, using a lookup argument within a single IVC step typically requires one of the following strategies: (I) *Encoding the lookup verification directly into the step circuit.* This can substantially increase the circuit size and cost, especially when the underlying lookup verifier involves expensive operations such as pairings—as is the case with KZG-based lookup arguments like cq. (II) *Accumulating the lookup verifier using a running accumulator.* In this case, the verifier logic for the lookup argument is not encoded in the circuit. Instead, only the verification logic for the accumulation lookup is encoded, which can significantly reduce overhead by, for example, avoiding pairing operations. We later review existing techniques for accumulating lookup relations, namely Protostar [BC23], nLookup [KS24] and FLI [GM24], which enable the efficient integration of specific lookup methods into the *recursive proofs setting*. An overview of these schemes is provided in the full version of the paper.

5 Considerations on Lookup Arguments in Practice

In order to use lookup arguments in practice, a first crucial consideration is *compatibility* between the lookup scheme and the underlying proof system. This compatibility manifests itself in two main ways. First, the lookup scheme and the proof system must operate over the same finite structure, e.g., the same finite field. For example, a proof system such as Plonky2 [Pol22], which operates over a small prime field, is not compatible with KZG-based constructions that require a large prime field. Second, their composition must be structurally compatible: the two systems should either share the same commitment when composed at the argument level, or rely on the same polynomial commitment scheme when the PIOPs are compiled into arguments. For example, if the lookup PIOP uses a univariate polynomial commitment scheme like KZG, but the proof system relies on multivariate commitments, then composing these two may require simultaneously committing to the same witness in both univariate and multivariate forms. Proving that these two commitments correspond to the same underlying witness within an arithmetic circuit can be highly inefficient, and general compilers [Zha+20; Boo+22] would add some extra overhead. To address such challenges, recent works propose scheme adaptations. For example, μ-seek [CFG25] enables cq to operate with witnesses committed via multilinear polynomial commitments (more details on μ-seek can be found in the full version of the paper), making cq compatible with proof systems based on multilinear polynomials such as HyperPlonk. Another idea to solve this problem might be to link the a univariate polynomial to a multivariate one at the commitment level; such relevant primitive as been called *dual polynomial commitment* [GNS24] and leverage some properties[11] of specific polynomials to map efficiently from a univariately committed witness to the same one committed using a multivariate basis. We leave the study of this primitive in the context of lookup compatibility to future work.

When selecting among compatible schemes, important considerations are the table size, its potential for decomposition, efficiency of the scheme, and whether a lookup table offers greater efficiency than a naive implementation.

[11] Such as a specific linear isomorphism between a univariate Lagrange basis over a set of roots of unity and a multivariate Lagrange basis over the boolean hypercube.

Table size. A crucial factor in the efficiency of lookup protocols is table size. Many schemes are designed under the assumption that the table is significantly larger than the number of lookups. In such cases, the solution is to use *table-efficient* lookup arguments in which the running time of the prover grows sublinearly in the table size. Existing table-efficient schemes can be divided into two families (cf. Figure 1). The first family achieves table-efficiency relying on some structure of the large table (e.g., if it can be reconstructed with sublinear computation from smaller tables). In this setting the family of constructions based on Lasso [STW24; CFR25; ST25] offers the best tradeoffs between proving time and other performance metrics. The second family achieves table-efficiency for arbitrary, unstructured tables, by means of a preprocessing stage that produces auxiliary material which the prover can use to generate lookup proofs in time sublinear in the table size. We observe a few patterns in the efficiency profile of existing preprocessing-based lookups. One construction [Cam+25] obtains table efficiency via preprocessing algorithms for polynomial evaluation; this construction is generic (e.g., they can be realized only from hash functions) but mostly of theoretical interest since preprocessing and proving time have very large constants. Several schemes—whose state of the art is represented by the `cq` family [EFG22; Cam+24; CFG25; ZSG24] of constructions—achieve table-efficiency relying on specific properties of pairing-based cryptography, including a preprocessing method for computing in quasilinear time the membership proofs of all the elements of a committed table. These schemes have extremely fast provers but their preprocessing has concrete scalability limitations. For instance, for a table of size 2^{32}, preprocessing requires around 2^{37} group operations—an effort that would take days even on a high-performance server[12]. Finally, a different line of work achieves a better trade-off by committing to the table using a lightweight accumulator (e.g., a Merkle tree), precomputing membership proofs for all elements, and then proving lookups by proving the existence of the relevant subset of membership proofs with a general-purpose SNARK. These constructions are called non-black-box, since the membership check must be encoded in the circuit; an early instance of this method appears in Pantry [Bra+13].

This state of affairs raises several open questions in this line of work, such as whether pairing-based performance can be replicated under alternative assumptions, or if one can design black-box solutions that offer a balanced profile where both proving and preprocessing remain feasible for larger tables.

Applications. Precomputed tables are a well-established technique in computer science to accelerate computationally intensive operations. By storing the results of commonly used or complex computations ahead of time, systems can replace expensive runtime calculations with efficient table lookups. In proof systems, lookup tables serve a similar role: they reduce computation by replacing *repetitive or complex operations* with direct access to precomputed values. Rather than embedding such operations directly into the arithmetic circuit, the prover can read the result from a pre-defined lookup table. This strategy decreases both the size and cost of the circuit, resulting in more efficient proof generation. However,

[12] https://zka.lc/

this efficiency comes at the cost of an additional requirement: the prover must now demonstrate that the lookup accesses are valid. This trade-off is typically favorable for two key reasons. First, thanks to the preprocessing explained in the previous paragraph, most of the work is pushed offline. Second, lookup systems support batching, enabling multiple lookup operations to be verified in a single invocation of the argument. This section provides an overview of how lookup tables are used in practice.

We organize these applications into the following main categories: (I) The biggest application of lookup tables in practice is for reducing constraint count for replacing non-native or not arithmetic-friendly operations with lookup operations [Sze+23; Gra+22; AST24; Woo+25; SLZ24; Xin+25; DCB25; Kan+22; Xio+23; Der+23; Lu+24; GFN24; Sou24]. Such non-native or non-arithmetic-friendly operations may include operations such as bit decomposition, range proofs, value comparison, floating-point operations and other functions that use such operations as their primitives, such as hash functions. In this case, the table is often small or structured, and the Lasso family of techniques shines. (II) Another less-explored application of lookup arguments is set membership, where one proves that all elements in a commitment belong to a given set or vector. This technique is advantageous in scenarios where you want to access a large and unstructured set, e.g., a set of public keys. In this case, it is interesting to use a scheme with the table-efficiency property. (III) Lookups are used in the context of memory correctness [Zha+23; Dut+24] and state machines and finite automata [Ang+24; Di+24] too, e.g., to prove that the execution of the machine adheres to its transition rules. In this case, the table is big and unstructured, leaning towards indexed lookup arguments supporting precomputations. It is interesting to mention that, as shown by Dutta et al. [Dut+24], when the memory also supports write operations, known approaches fail because they require some updatable table. How to build such a scheme with an efficiency that is close to the state-of-the-art for classical preprocessing is still an open question. (IV) A theoretically interesting application of lookup tables is strengthening an extractor by proving the witness belongs to a finite set. Zinc [Gar+25] employs a novel use of lookup PIOP to strengthen the extractor of its PCS. At a high level, Zinc aims to build an argument of knowledge for an integer witness, while the underlying PCS extractor is only capable of extracting bounded rational numbers. Since knowledge soundness requires the extraction of an integer witness, Zinc composes its PIOP with a lookup PIOP, ensuring that the witness indeed consists of integers. When combined with the PCS extractor, this composition ensures the extraction of a satisfying integer witness.

Projective ratio. One last key consideration in the use of lookup tables is the proportion of the witness that benefits from the lookup argument. For instance, to prove a single range proof with a lookup argument might introduce more overhead than the benefit it provides when using a lookup argument. Verifying that a portion of the witness is contained in a table requires *projective lookups*, which may still incur a cost linear in the total size of the witness—e.g., for witness selection. Therefore, if the portion of the witness subject to lookup is

Technique	Scheme/Family	T_{precomp}	$\lvert\pi\rvert$	T_{prove}	T_{vfy}	Compatibility
Multisets	Plookup [GW20]	—	const.	$(N+n)\log(N+n)$	const.	uni/multivar.
	Halo2 [Zca25]	—	const.	$N\log N$	const.	univar.
Logup	Logup [PH23]	—	const.	$N\log N$	const.	multivar.
	Logup+GKR [EFG22]	—	$\log(N+n)$	$N+n$	$\log(N+n)$	univar.
	cq† [EFG22; Cam+24]	$N\log N$	const.	$n\log n$	const.	KZG
Subvector Extractable	Caulk† [Zap+22a]	$N\log N$	const.	$n^2+n\log N$	const.	KZG
	Caulk+† [PK22]	$N\log N$	const.	n^2	const.	KZG
	Baloo† [Zap+22b]	$N\log N$	const.	$n\log^2 n$	const.	KZG
	Lasso† [STW24; CFR25]	—	$\log^2 n$	$N+n, cn^\star$	$\log^2 n$	multivar.
	Shout [ST25]	—	$d\log n$	$cn^\star$	$d\log n$	multivar.
Accumulator	Flookup [GK22]	$N\log N$	const.	$n\log^2 n$	const.	KZG
	Duplex† [Cam+22; Han+25]	$N\log N$	const.	$n\log n$	const.	Pedersen

N: table size n: number of lookups   †: supports zero-knowledge $\star$: structured tables only
On compatibility: Pedersen $\supseteq$ KZG, univariate $\supseteq$ KZG (i.e., KZG is a special case of both)

Table 1: Asymptotic comparison of lookup argument schemes (Big-Oh is implicit). N is the size of the table and n is the number of lookups. For Lasso, we consider costs as it is initiated with Dory [Lee21], the prover time considers two different cases when the table is structured and unstructured.

relatively small compared to the total witness size, employing a lookup argument may not yield a meaningful efficiency gain.

6 Lookup Arguments vs Other Primitives

Set accumulator and vector commitments. Set accumulators [BM94; BP97; CL02] and vector commitments [CF13] are cryptographic primitives that allow committing to sets and vectors, respectively—and later proving the inclusion of elements or values. Possibly, they may support batch openings, enabling a single proof to attest to multiple inclusions. These primitives are closely related to the *lookup relations.* The key distinction between a lookup protocol and a set accumulator or vector commitment is the existence of duplicates. A lookup witness may contain duplicate elements, whereas in set membership proofs or subvector opening, the elements are unique, and duplication does not occur. Another important difference is that in lookup relations, the verifier holds a commitment to the witness, rather than receiving it in plaintext. In this sense, lookup relations generalize these primitives by introducing commit-and-prove functionality on both sides of the relation. Moreover, in lookup relations the table is commonly[13] assumed to be public, whereas in set accumulators and vector commitments, the underlying set or vector may remain private.

Memory checking. A related concept to lookup tables is *memory checking* [Blu+91; Set+18; ST25]. At a high level, memory checking allows a prover to demonstrate that a transcript of memory reads and writes is consistent—for example, that every read returns the value from the most recent write. One key difference from

[13] See exceptions we discuss in Section 3.6.

lookups is that memory checking supports *write* operations, whereas lookup tables can be viewed as read-only memory. In some works, write operations are leveraged to construct online lookup tables [Xio+23; Ros+24]. Memory checking is a broad research area, and we do not cover it in detail here. For an overview of memory-checking techniques, we refer the reader to Appendix B of Jolt [AST24].

Updateable lookup tables. A notable subclass of lookup tables, closely related to memory-checking, is the class of *updateable lookup tables* [Dut+24; Han+25]. In contrast to conventional preprocessing tables—where any modification invalidates the preprocessing phase—updateable lookup tables are specifically designed to permit modifications after their initial construction. It is important to distinguish updateable lookup tables from online lookup tables: while the latter are constructed once during the online phase and remain fixed thereafter, updateable lookup tables may be precomputed offline, yet can be modified multiple times during the online phase.

Acknowledgments

Hossein Hafezi conducted this work while hosted by Dario Fiore at IMDEA Software. We would like to thank, in alphabetical order, Arasu Arun, Joseph Bonneau, Benedikt Bünz, Albert Garreta, Abhiram Kothapalli, Shahar Papini, Alireza Shirzad, and Aranxta Zapico for their comments and opinions.

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects PRODIGY (TED2021-132464B-I00), ESPADA (PID2022-142290OB-I00) and CEX2024-001471-M funded by MCIN/AEI/10.13039 /501100011033/.

References

[Ang+24] Sebastian Angel, Eleftherios Ioannidis, Elizabeth Margolin, Srinath Setty, and Jess Woods. "Reef: fast succinct non-interactive zero-knowledge regex proofs". In: *Proceedings of the 33rd USENIX Conference on Security Symposium*. SEC '24. Philadelphia, PA, USA: USENIX Association, 2024. ISBN: 978-1-939133-44-1.

[AST24] Arasu Arun, Srinath Setty, and Justin Thaler. "Jolt: SNARKs for Virtual Machines via Lookups". In: *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part VI*. Zurich, Switzerland: Springer-Verlag, 2024, 3–33. ISBN: 978-3-031-58750-4. DOI: 10.1007/978-3-031-58751-1_1. URL: https://doi.org/10.1007/978-3-031-58751-1_1.

[BP97] Niko Barić and Birgit Pfitzmann. "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees". In: *Advances in Cryptology — EUROCRYPT '97*. Ed. by Walter Fumy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 480–494. ISBN: 978-3-540-69053-5.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*. Berlin, Heidelberg: Springer-Verlag, 2016, 31–60. ISBN: 9783662536438. DOI: 10.1007/978-3-662-53644-5_2. URL: https://doi.org/10.1007/978-3-662-53644-5_2.

[BM94] Josh Benaloh and Michael de Mare. "One-Way Accumulators: A Decentralized Alternative to Digital Signatures". In: *Advances in Cryptology — EUROCRYPT '93*. Ed. by Tor Helleseth. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 274–285. ISBN: 978-3-540-48285-7.

[Blu+91] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. "Checking the correctness of memories". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. SFCS '91. San Juan, Puerto Rico: IEEE Computer Society, 1991, 90–99. ISBN: 0818624450. DOI: 10.1109/SFCS.1991.185352. URL: https://doi.org/10.1109/SFCS.1991.185352.

[Boo+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: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I*. Brisbane, QLD, Australia: Springer-Verlag, 2018, 595–626. ISBN: 978-3-030-03325-5. DOI: 10.1007/978-3-030-03326-2_20. URL: https://doi.org/10.1007/978-3-030-03326-2_20.

[Boo+22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrú. "Gemini: Elastic SNARKs for Diverse Environments". In: *Advances in Cryptology – EUROCRYPT 2022*. Ed. by Orr Dunkelman and Stefan Dziembowski. Cham: Springer International Publishing, 2022, pp. 427–457. ISBN: 978-3-031-07085-3.

[Bra+13] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. "Verifying computations with state". In: *Proceedings of the twenty-fourth ACM Symposium on Operating Systems Principles*. 2013, pp. 341–357.

[BC23] Benedikt Bünz and Binyi Chen. "Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols". In: 2023, pp. 77–110. DOI: 10.1007/978-981-99-8724-5_3.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: 2020, pp. 677–706. DOI: 10.1007/978-3-030-45721-1_24.

[BC24] Benedikt Bünz and Jessica Chen. "Proofs for Deep Thought: Accumulation for Large Memories and Deterministic Computations". In: Dec. 2024, pp. 269–301. ISBN: 978-981-96-0934-5. DOI: 10.1007/978-981-96-0935-2_9.

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials". In: *Advances in Cryptology — CRYPTO 2002*. Ed. by Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 61–76. ISBN: 978-3-540-45708-4.

[Cam+25] Matteo Campanelli, Mario Carrillo, Ignacio Cascudo, Dario Fiore, Danilo Francati, and Rosario Gennaro. *On the Power of Polynomial Preprocessing: Proving Computations in Sublinear Time, and More*. Cryptology ePrint Archive, Paper 2025/238. 2025. URL: https://eprint.iacr.org/2025/238.

[Cam+24] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. "Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees". In: 2024, pp. 337–369. DOI: 10.1007/978-3-031-57722-2_11.

[CFR25] Matteo Campanelli, Antonio Faonio, and Luigi Russo. "SNARKs for Virtual Machines Are Non-malleable". In: *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*. Ed. by Serge Fehr and Pierre-Alain Fouque. Vol. 15604. Lecture Notes in Computer Science. Springer, 2025, pp. 153–183. DOI: 10.1007/978-3-031-91134-7_6. URL: https://doi.org/10.1007/978-3-031-91134-7_6.

[CFG25] Matteo Campanelli, Dario Fiore, and Rosario Gennaro. "Natively Compatible Super-Efficient Lookup Arguments and How to Apply Them: Natively Compatible Super-Efficient Lookup Arguments". In: *J. Cryptol.* 38.1 (Jan. 2025). ISSN: 0933-2790. DOI: 10.1007/s00145-024-09535-0. URL: https://doi.org/10.1007/s00145-024-09535-0.

[Cam+22] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. "Succinct Zero-Knowledge Batch Proofs for Set Accumulators". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, 455–469. ISBN: 9781450394505. DOI: 10.1145/3548606.3560677. URL: https://doi.org/10.1145/3548606.3560677.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs". In: 2019, pp. 2075–2092. DOI: 10.1145/3319535.3339820.

[CF13] Dario Catalano and Dario Fiore. "Vector Commitments and Their Applications". In: *Public-Key Cryptography – PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 55–72. ISBN: 978-3-642-36362-7.

[Che+23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17.

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: 2020, pp. 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[Cho+24] Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. "SublonK: Sublinear Prover PlonK." In: *Proc. Priv. Enhancing Technol.* 2024.3 (2024), pp. 314–335. URL: http://dblp.uni-trier.de/db/journals/popets/popets2024.html#ChoudhuriGGSS24.

[DCB25] Trisha Datta, Binyi Chen, and Dan Boneh. "VerITAS: Verifying Image Transformations at Scale". In: May 2025, pp. 4606–4623. DOI: 10.1109/SP61157.2025.00097.

[Der+23] Tal Derei et al. "Scaling Zero-Knowledge to Verifiable Databases". In: *Proceedings of the 1st Workshop on Verifiable Database Systems*. VDBS '23. Seattle, WA, USA: Association for Computing Machinery, 2023, 1–9.

ISBN: 9798400707759. DOI: 10.1145/3595647.3595648. URL: https://doi.org/10.1145/3595647.3595648.

[Di+24] Zijing Di, Lucas Xia, Wilson Nguyen, and Nirvan Tyagi. "MuxProofs: Succinct Arguments for Machine Computation from Vector Lookups". In: *Advances in Cryptology – ASIACRYPT 2024: 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9–13, 2024, Proceedings. Part V*. Kolkata, India: Springer-Verlag, 2024, 236–265. ISBN: 978-981-96-0934-5. DOI: 10.1007/978-981-96-0935-2_8. URL: https://doi.org/10.1007/978-981-96-0935-2_8.

[DP25] Benjamin E. Diamond and Jim Posen. "Succinct Arguments over Towers of Binary Fields". In: *Advances in Cryptology – EUROCRYPT 2025*. Ed. by Serge Fehr and Pierre-Alain Fouque. Cham: Springer Nature Switzerland, 2025, pp. 93–122. ISBN: 978-3-031-91134-7.

[Dor24] Tesseract Dore. *TaSSLE: Lasso for the commitment-phobic*. Cryptology ePrint Archive, Paper 2024/1075. 2024. URL: https://eprint.iacr.org/2024/1075.

[Dut+24] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. "Batching-Efficient RAM using Updatable Lookup Arguments". In: Dec. 2024, pp. 4077–4091. DOI: 10.1145/3658644.3670356.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. *cq: Cached quotients for fast lookups*. Cryptology ePrint Archive, Paper 2022/1763. 2022. URL: https://eprint.iacr.org/2022/1763.

[Eag+24] Liam Eagen, Sanket Kanjalkar, Tim Ruffing, and Jonas Nick. "Bulletproofs++: Next Generation Confidential Transactions via Reciprocal Set Membership Arguments". In: *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part V*. Zurich, Switzerland: Springer-Verlag, 2024, 249–279. ISBN: 978-3-031-58739-9. DOI: 10.1007/978-3-031-58740-5_9. URL: https://doi.org/10.1007/978-3-031-58740-5_9.

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

[GW20] Ariel Gabizon and Zachary J. Williamson. *plookup: A simplified polynomial protocol for lookup tables*. Cryptology ePrint Archive, Paper 2020/315. 2020. URL: 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, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[GNS24] Chaya Ganesh, Vineet Nair, and Ashish Sharma. "Dual Polynomial Commitment Schemes and Applications to Commit-and-Prove SNARKs". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, 884–898. ISBN: 9798400706363. DOI: 10.1145/3658644.3690219. URL: https://doi.org/10.1145/3658644.3690219.

[GM24] Albert Garreta and Ignacio Manzur. "FLI: Folding Lookup Instances". In: *Advances in Cryptology – ASIACRYPT 2024: 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9–13, 2024, Proceedings. Part V*. Kolkata, India: Springer-Verlag, 2024, 402–435. ISBN: 978-981-96-0934-5. DOI: 10.1007/978-981-96-0935-2_13. URL: https://doi.org/10.1007/978-981-96-0935-2_13.

[Gar+25] Albert Garreta, Hendrik Waldner, Ilia Vlasov, Katerina Hristova, Luca Dall'Ava, Marko Čupić, and Matthew Klein. "Zinc: Succinct Arguments with Small Arithmetization Overheads from IOPs of Proximity to the Integers". In: *Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part VII*. Santa Barbara, CA, USA: Springer-Verlag, 2025, 259–291. ISBN: 978-3-032-01906-6. DOI: 10.1007/978-3-032-01907-3_9. URL: https://doi.org/10.1007/978-3-032-01907-3_9.

[Gra+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, 1323–1335. ISBN: 9781450394505. DOI: 10.1145/3548606.3560686. URL: https://doi.org/10.1145/3548606.3560686.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[GFN24] Binbin Gu, Juncheng Fang, and Faisal Nawab. *PoneglyphDB: Efficient Non-interactive Zero-Knowledge Proofs for Arbitrary SQL-Query Verification*. 2024. arXiv: 2411.15031 [cs.DB]. URL: https://arxiv.org/abs/2411.15031.

[Hab22] Ulrich Haböck. *Multivariate lookups based on logarithmic derivatives*. Cryptology ePrint Archive, Paper 2022/1530. 2022. URL: https://eprint.iacr.org/2022/1530.

[Haf+25] Hossein Hafezi, Gaspard Anthoine, Matteo Campanelli, and Dario Fiore. *SoK: Lookup Table Arguments*. Cryptology ePrint Archive, Paper 2025/1876. 2025. URL: https://eprint.iacr.org/2025/1876.

[Han+25] Semin Han, Geonho Yoon, Hyunok Oh, and Jihye Kim. "DUPLEX: Scalable Zero-Knowledge Lookup Arguments over RSA Group". In: *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*. ASIA CCS '25. New York, NY, USA: Association for Computing Machinery, 2025, 72–86. ISBN: 9798400714108. DOI: 10.1145/3708821.3733863. URL: https://doi.org/10.1145/3708821.3733863.

[Kan+22] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. *Scaling up Trustless DNN Inference with Zero-Knowledge Proofs*. 2022. arXiv: 2210.08674 [cs.CR]. URL: https://arxiv.org/abs/2210.08674.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[KU08] Kiran S. Kedlaya and Christopher Umans. "Fast Modular Composition in any Characteristic". In: *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. 2008, pp. 146–155. DOI: 10.1109/FOCS.2008.13.

[KU11] Kiran S. Kedlaya and Christopher Umans. "Fast Polynomial Factorization and Modular Composition". In: *SIAM Journal on Computing* 40.6 (2011), pp. 1767–1802. DOI: 10.1137/08073408X. eprint: https://doi.org/10.1137/08073408X. URL: https://doi.org/10.1137/08073408X.

[Kos+20] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. "MIRAGE: succinct arguments for randomized algorithms with applications to universal zk-SNARKs". In: *Proceedings of the 29th USENIX Conference on Security Symposium*. SEC'20. USA: USENIX Association, 2020. ISBN: 978-1-939133-17-5.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. "HyperNova: Recursive Arguments for Customizable Constraint Systems". In: 2024, pp. 345–379. DOI: 10.1007/978-3-031-68403-6_11.

[Lab25] Succinct Labs. *SP1: a zero-knowledge virtual machine for RISC-V*. https://github.com/succinctlabs/sp1. Accessed: 2025-10-05. 2025.

[Lee21] Jonathan Lee. "Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments". In: *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*. Raleigh, NC, USA: Springer-Verlag, 2021, 1–34. ISBN: 978-3-030-90452-4. DOI: 10.1007/978-3-030-90453-1_1. URL: https://doi.org/10.1007/978-3-030-90453-1_1.

[Li+25] Weihan Li, Zongyang Zhang, Yun Li, Pengfei Zhu, Cheng Hong, and Jianwei Liu. *Soloist: Distributed SNARKs for Rank-One Constraint System*. Cryptology ePrint Archive, Paper 2025/557. 2025. URL: https://eprint.iacr.org/2025/557.

[Lu+24] Tao Lu et al. *An Efficient and Extensible Zero-knowledge Proof Framework for Neural Networks*. Cryptology ePrint Archive, Paper 2024/703. 2024. URL: https://eprint.iacr.org/2024/703.

[Lun+92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic methods for interactive proof systems". In: *J. ACM* 39.4 (Oct. 1992), 859–868. ISSN: 0004-5411. DOI: 10.1145/146585.146605. URL: https://doi.org/10.1145/146585.146605.

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

[Pol22] Polygon Labs. *Introducing Plonky2*. Accessed: 2025-08-07. Jan. 2022. URL: https://polygon.technology/blog/introducing-plonky2.

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

[ris25a] risc0 project contributors. *Proof System Sequence Diagram*. https://github.com/risc0/risc0/blob/912c2e198f3abc1094fa55e45840febaee203c22/website/docs/proof-system/proof-system-sequence-diagram.md. Accessed: 2025-10-05. 2025.

[ris25b] risc0 project contributors. *risc0/risc0: RISC Zero — zero-knowledge verifiable computing platform*. https://github.com/risc0/risc0. Accessed: 2025-10-05. 2025.

[Ros+24] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. "Hekaton: Horizontally-Scalable zkSNARKs Via Proof Aggregation". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Salt Lake City, UT, USA:

Association for Computing Machinery, 2024, 929–940. ISBN: 9798400706363. DOI: 10.1145/3658644.3690282. URL: https://doi.org/10.1145/3658644.3690282.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: 2020, pp. 704–737. DOI: 10.1007/978-3-030-56877-1_25.

[Set+18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. "Proving the correct execution of concurrent services in zero-knowledge". In: *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*. OSDI'18. Carlsbad, CA, USA: USENIX Association, 2018, 339–356. ISBN: 9781931971478.

[ST25] Srinath Setty and Justin Thaler. *Twist and Shout: Faster memory checking arguments via one-hot addressing and increments*. Cryptology ePrint Archive, Paper 2025/105. 2025. URL: https://eprint.iacr.org/2025/105.

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. "Unlocking the Lookup Singularity with Lasso". In: 2024, pp. 180–209. DOI: 10.1007/978-3-031-58751-1_7.

[Sou24] Lev Soukhanov. *WARPfold : Wrongfield ARithmetic for Protostar folding*. Cryptology ePrint Archive, Paper 2024/354. 2024. URL: https://eprint.iacr.org/2024/354.

[Sta25] StarkWare Labs. *Stwo: a next-generation Rust implementation of a CSTARK prover and verifier*. https://github.com/starkware-libs/stwo. Version v1.0.0 released July 18, 2025. July 2025.

[SLZ24] Haochen Sun, Jason Li, and Hongyang Zhang. "zkLLM: Zero Knowledge Proofs for Large Language Models". In: Dec. 2024, pp. 4405–4419. DOI: 10.1145/3658644.3670334.

[Sze+23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, Bobbin Threadbare, and Al-Kindi. *The Tip5 Hash Function for Recursive STARKs*. Cryptology ePrint Archive, Paper 2023/107. 2023. URL: https://eprint.iacr.org/2023/107.

[Val08] Paul Valiant. "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency". In: *Proceedings of the 5th Conference on Theory of Cryptography*. TCC'08. New York, USA: Springer-Verlag, 2008, 1–18. ISBN: 354078523X.

[Cry] "Verifiable Computation for Approximate Homomorphic Encryption Schemes". In: Aug. 2025, pp. 643–677. ISBN: 978-3-032-01906-6. DOI: 10.1007/978-3-032-01907-3_21.

[Woo+25] Anna Woo, Alex Ozdemir, Chad Sharp, Thomas Pornin, and Paul Grubbs. "Efficient Proofs of Possession for Legacy Signatures". In: May 2025, pp. 3291–3308. DOI: 10.1109/SP61157.2025.00080.

[Xin+25] Zhibo Xing et al. "Zero-Knowledge Proof-Based Verifiable Decentralized Machine Learning in Communication Network: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* (2025), 1–1. ISSN: 2373-745X. DOI: 10.1109/comst.2025.3561657. URL: http://dx.doi.org/10.1109/COMST.2025.3561657.

[Xio+23] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. "VERIZEXE: decentralized private computation with universal setup". In: *Proceedings of the 32nd*

USENIX Conference on Security Symposium. SEC '23. Anaheim, CA, USA: USENIX Association, 2023. ISBN: 978-1-939133-37-3.

[Zap+22a] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. "Caulk: Lookup Arguments in Sublinear Time". In: 2022, pp. 3121–3134. DOI: 10.1145/3548606.3560646.

[Zap+22b] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive, Report 2022/1565. 2022. URL: https://eprint.iacr.org/2022/1565.

[Zca25] Zcash Developers. *Lookup Argument*. https://zcash.github.io/halo2/design/proving-system/lookup.html. Accessed: 2025-06-23. Zcash, 2025.

[Zha+20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Xiaodong Song. "Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof". In: *2020 IEEE Symposium on Security and Privacy (SP)* (2020), pp. 859–876. URL: https://api.semanticscholar.org/CorpusID:209467198.

[ZSG24] Yuncong Zhang, Shi-Feng Sun, and Dawu Gu. "Efficient KZG-Based Univariate Sum-Check and Lookup Argument". In: *Public-Key Cryptography – PKC 2024: 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15–17, 2024, Proceedings, Part II*. Sydney, NSW, Australia: Springer-Verlag, 2024, 400–425. ISBN: 978-3-031-57721-5. DOI: 10.1007/978-3-031-57722-2_13. URL: https://doi.org/10.1007/978-3-031-57722-2_13.

[Zha+23] Yuncong Zhang, Shi-Feng Sun, Ren Zhang, and Dawu Gu. "Polynomial IOPs for Memory Consistency Checks in Zero-Knowledge Virtual Machines". In: *Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part II*. Guangzhou, China: Springer-Verlag, 2023, 111–141. ISBN: 978-981-99-8723-8. DOI: 10.1007/978-981-99-8724-5_4. URL: https://doi.org/10.1007/978-981-99-8724-5_4.