

SAK: Sparse Arguments of Knowledge, from Sparse Lookup Arguments

Abhiram Kothapalli¹, Sriram Sridhar², and Arantxa Zapico³

¹ Microsoft Research

² University of California, Berkeley

³ Field Extension Institute

Abstract. We formalize and construct *sparse* arguments of knowledge, where given a length N witness that contains n non-zero values the prover time is $O(n) \circ o(N)$. In particular, we achieve a sparse argument of knowledge for the customizable constraint system relation, which generalizes circuit-satisfiability. We achieve this by first utilizing *sublinear* lookup arguments to provably select only the subset of the constraint system that touches the non-zero entries of the variable assignment, and then running a standard argument of knowledge with mild qualifications. Our resulting sparse argument of knowledge is publicly verifiable and supports a universal and updatable setup. It features an $O(N \log N)$ preprocessing phase, an $O(n \log n)$ prover time, and an $O(1)$ verifier time, improving on the prior $O(n \log^2 n)$ and $O(n \log N)$ prover time.

1 Introduction

Arguments of knowledge [16] are a powerful cryptographic primitive that enables a prover to interactively and succinctly demonstrate to a verifier that a computation (e.g., the execution of a blockchain [4]) is executed correctly without revealing any secret inputs. In this work, we formalize and construct *sparse* arguments of knowledge, where the prover work only scales with the number of *non-zero* gates of a circuit execution representing the computation. The core technical idea is to use a *lookup argument* [6, 11, 14, 22, 31, 32] to provably select a subset of the circuit that is non-zero (in sublinear time in the size of the circuit [11]) and then prove that subset using a standard argument of knowledge with mild qualifications. In essence, we prove the following theorem.

Theorem 1 (Sparse Argument of Knowledge). *There exists an argument of knowledge for circuit satisfiability in the algebraic group model [12], such that for a circuit of size N and a satisfying wire assignment with n non-zero values, the preprocessing time is $O(N \log N)$, the prover time is $O(n \log n)$, and the verifier time is $O(1)$.*

1.1 Why Sparsity Matters

In the past two decades there has been substantial progress in reducing the enormous proving costs for arguments of knowledge for general computations [8,

17, 24]. Today, proving costs range from 1 [27, 30] to $4\times$ [8, 24] orders of magnitude over the cost of native circuit execution, depending on the underlying assumptions requirements on the verifier run-times. A major limitation with most modern proof systems is that the prover scales linearly in the size of the *entire* circuit representing the computation, regardless of the size of the active path of the circuit, leaving significant optimizations on the table.

This lead to a diverse set of works focused on capturing savings from sparsity for various specific applications, either from removing any sparsity in the computational statement itself, or by designing bespoke proof machinery that does not perform significant work over sparse sections of specialized computation. Sparse arguments can be viewed as a *general* solution to this problem: Any application that can be cast as a circuit satisfiability instance would have a guaranteed sparse solution. We discuss several example applications below.

Verifiable Machine Execution Recently verifiable virtual machine execution [2, 20] and underlying techniques such as incrementally verifiable computation (that is, iteratively proving repeated applications of a computation [26]) has become an actively studied area. The predominant approach is to prove repeated applications of a circuit representing a single cycle of the virtual machine [20]. Naively, such a circuit encodes all possible instructions, meaning that the prover would have to prove all of the unexecuted instructions alongside the one executed instruction.

To circumvent this, SuperNova [20], SublonK [9], MuxProofs [10], and Jolt designed proof systems around selecting subsets of the original computation. A second wave of efforts including Protostar [5] and Nebula [1] achieve sparse incremental proofs by observing that the algebraic proof machinery itself performs no meaningful computation over the zero terms.

Proving computations of this form can be interpreted as proving a single large circuit consisting of many copies of a uniform circuits (each of which contains all the possible instructions). As such, sparse arguments give a general solution for ensuring that the prover pays only for the instruction run at each step.

Sparse State Updates Various proof systems prove that a large data structure was correctly updated on a small subset of points. For instance Volatile and Persistent Memory for zkSNARKs [21] designs verifiable read-write memory accesses while ensuring that the prover runtime only scales with the number of accessed cells. Verdict [25], through carefully designed proof machinery, provides proofs that a large Merkle tree was updated over small number of points while ensuring that the proof only scales with the number of points updated. Alternatively, sublinear and table-independent lookup arguments [6, 11, 22, 31, 32] prove a small number of data accesses in a large table in time sublinear in the size of the table.

Proving computations of this form can be interpreted as proving a large circuit that takes as input a “one-hot” encoding of which entries are updated and masks the data structure with respect to this one-hot vector. Although this

representation is enormous, a sparse argument over such a circuit generically ensure that the proving cost still scales with only the entries that are updated.

Verifiable Conditionals Computations As another example various proof systems such as Speed-Stacking [15], consider proofs of a large number of disjunctions, where only one of the computations is true. These computations can be interpreted as a single circuit selecting between both disjunctions based on an additional selector bit. Sparse arguments ensure that the prover only pays for the selected computation.

1.2 Our Solution

As intuited above, our solution for sparse arguments for general circuits involves utilizing a lookup argument to first verifiably select the non-zero subset of the circuit execution, and then running a standard argument of knowledge on the resulting circuit.

Recall that a lookup argument enables provably selecting a (committed) subtable from a (committed) table according to a (committed) list of indices. As such our approach involves (1) carefully encoding the circuit structure as a set of tables such that appropriate subtables (representing a subcircuit) can be selected, and (2) ensuring that this process only scales with the size of the subcircuit.

We achieve the former by requiring that the circuit encoding is *column-indexable*, which roughly states that each gate constraint in the circuit is encoded as only a single entry in each table (thus ensuring that subtables coherently remove gates). We ensure the latter by utilizing a *sublinear* lookup table, cq [11], which guarantees that (after preprocessing) the lookup argument prover time only scales with the size of the subtable. As such our techniques can be viewed as lifting the sublinearity of the lookup argument to achieve sublinearity for general circuits. We formalize our machinery in a modular fashion, using the reductions of knowledge framework [19], ensuring that our technique for selecting a non-zero subcircuit can be used independently.

1.3 Related Works

There are broadly two lines of works which are related to our results. The first builds *dynamic arguments of knowledge* [28, 29, 33], which are proofs that can be updated as the underlying witness (e.g., circuit wire assignment) is updated in time that only scales with the number of updates. Dynamic arguments naturally induce sparse arguments as an all-zero circuit can be included in a preprocessing phase and then dynamically updated to prove any subcircuit execution in time that scales with that subcircuit. Below we compare our result to sparse arguments that are induced by recent dynamic argument systems.

The second line of work [2, 9, 10] focuses on RAM computation and utilizes sublinear lookup tables to ensure that only the instruction run at each step is selected to be proven in the aggregate computation. While these approaches are

Scheme	Preprocessing	Proving	Public verif.	Univ. setup
This work	$(3\nu + 1)\text{MSM}_N^{\mathbb{G}_1}$ $+ 2(3\nu + 1)\text{MSM}_N^{\mathbb{G}_2};$ $O((3\nu + 1)N \log N) \mathbb{G}_1$ $+ O(1) \mathbb{G}_2;$ $O(\nu N \log N) \mathbb{F}$	$(3\nu + 13)\text{MSM}_n^{\mathbb{G}_1}$ $+ 3\text{MSM}_{2n}^{\mathbb{G}_1};$ $(3\nu + 1)n \mathbb{G}_1;$ $O(n \log n + \nu n) \mathbb{F}$	✓	✓
Aero [29]	$O(N \log N)$	$O(n \log N + n \log^2 n)$	✓	✓
Dynark [33]	$O(N \log N) \mathbb{G}_1$	$O(n) \mathbb{G}_1 + O(n) \mathbb{G}_2;$ $O(n \log^2 n) \mathbb{F}$	✓	
Delphus [28]	$(2N \log N + 4N) \mathbb{G}_1;$ $(2N \log N + N) \mathbb{G}_2;$ $O(N \log N) \mathbb{F}$	$O(n \log N) \mathbb{G}_1;$ $O(n \log N) \mathbb{G}_2;$ $O(n \log N) \mathbb{P}$		
Dynasaurs + Plonk [3]	$O(N \log N)$	$O(n) \mathbb{G}_1;$ $O(n \log^2 n) \mathbb{F}$	✓	✓

Table 1: Dominant cost of (implied) sparse proof systems for general circuits. Here N is the ambient circuit size, n is the number of nonzero wire values, and ν bounds the number of nonzero matrix entries per column. For simplicity, we assume no public-input. The Dynark row reports its sparse specialization, not its fully dynamic update cost. $\mathbb{F}$ denotes field operations, $\mathbb{G}_1$ and $\mathbb{G}_2$ denote group scalar multiplications, $\mathbb{P}$ denotes pairings between these two groups, and MSM_r denotes a length- r multi-scalar multiplication.

comparable in their use of lookup arguments, they focus on highly structured RAM computations whereas we focus on fully general circuits.

For our construction, we instantiate the SNARK with Basilisk [23], that assumes bounded fan-out ν . The total proving cost is $O(n \log n + \log N)$. The additive logarithm accounts for evaluating the vanishing polynomial of the unchanged ambient row domain.

Dynamic Arguments of Knowledge

Aero. Wang et al. [29] combine a sparse argument for wiring permutations, with dynamic proofs of multiplication gates via polynomial divisibility checks restricted to the affected gates. Aero specializes the resulting dynamic prover to an all-zero base proof to achieve sparse arguments of knowledge. For the universal-setup variant considered here, the sparse proving cost is $O(n \log N + n \log^2 n)$: universal consistency checks contribute $O(n \log N)$, while dynamic polynomial divisibility checks contribute $O(n \log^2 n)$. Our construction avoids dynamic polynomial divisibility checks, reducing the latter term to $O(n \log n)$.

Dynark (sparse specialization). Dynark [33] introduces a sparse convolution technique as an alternative method to achieve sparse polynomial divisibility checks for Groth16 [17]. We compare only with its sparse specialization, rather than its fully dynamic update bound. This specialization uses $O(n)$ scalar multiplications in each of $\mathbb{G}_1$ and $\mathbb{G}_2$, and $O(n \log^2 n)$ field operations. Our construction avoids sparse convolution, reducing this n -dependent field-operation cost to $O(n \log n)$. Moreover, unlike Dynark’s circuit-specific setup, we achieve a universal and updatable setup.

Delphus. Delphus [28] decomposes Groth16’s quotient polynomial into a linearly updatable part and cross terms, maintaining the latter’s commitment in the pairing target group using a binary tree. This gives $O(\log N)$ work per gate change but requires secret verifier state. For sparse proving from an all-zero base proof, our prover costs $O(n \log n + \log N)$ rather than $O(n \log N)$. Our prover also avoids Delphus’s $O(n \log N)$ prover-side pairings, and our proofs are publicly verifiable.

Lookup-Based Sparse Arguments for RAM Computations

SublonK. SublonK [9] uses preprocessed segment lookups to assemble the constraints and commitments for the executed branches of a repeating circuit (each subcircuit representing a cycle of the CPU). For k executed layers chosen from b branches of size s , its prover uses $O(ks(\log(ks) + \log b))$ work. If the same computation admits a sparse circuit encoding with $n = \Theta(ks)$, our prover costs $O(ks \log(ks) + \log N)$. It removes the $n \log b$ term while retaining an additive ambient logarithm.

MuxProofs. MuxProofs [10] similarly uses succinct vector lookups to assemble commitments to executed instruction constraints from preprocessed tables. Mux-PLONK and Mux-Marlin prove t executed instructions from an instruction set of size ℓ in $\tilde{O}((t + \ell)C)$ time, where C is the constraint-size upper bound for one instruction. For a sparse circuit encoding of the same machine relation with $n = O(tC)$, our prover time is $O(tC \log(tC) + \log N)$ and has no additive ℓC term.

Concurrent Work

Dynasaurs + Plonk. Dynasaurs [3] designs sparse arguments for *linear* relations, a compiler to lift them to dynamic arguments for linear relations, and then combines these arguments with the dynamic Hadamard-product of Dynark [33] leading to a much more efficient Plonk-based system than what was initially achieved by Wang et al. [29]. We compare with the sparse specialization of its Plonk-based system, which uses $O(n)$ scalar multiplications in $\mathbb{G}_1$ and $O(\log N + n \log^2 n)$ field operations. The additive $\log N$ term comes from evaluating an ambient-domain vanishing polynomial. Our lookup-based technique reduces the n -dependent field-operation term to $O(n \log n)$.

2 Technical Overview

The goal. We aim to construct sparse SNARKs for NP relations: for a witness with n non-zero entries inside an ambient witness space of dimension N , the prover’s work should be linear in n and sublinear in N . The difficulty is that the witness is not the only object whose size may depend on N . In a standard SNARK, the prover is given a representation of the relation – equivalently, of the circuit – and this representation itself typically has size proportional to N . Consequently, even when the witness is sparse, a prover that must read the entire circuit description cannot run in sparse time. Sparse proving therefore requires a way to restrict the circuit description itself, consistently with the sparse witness: informally, to remove the parts of the circuit that become irrelevant once the zero coordinates of the witness are fixed, prove satisfiability of the resulting smaller circuit, and convince the verifier that this smaller circuit was obtained from the original one by a valid restriction.

From lookup arguments to sparse encodings. We build on the line of lookup arguments developed by Plookup, Caulk, Caulk+, cq, Baloo, and subsequent refinements [6, 11, 14, 22, 31, 32]. The customizable constraint system (CCS) representation makes this restriction concrete for circuits encoded via matrices and matrix-vector multiplication: satisfiability is expressed via terms $T_i \cdot z$ for structure matrices T_i and extended witness z , so restricting z to a sparse support J and restricting the columns of each T_i to the same set J are equivalent operations. This suggests using a lookup argument to enforce the restriction: a lookup argument allows a prover to show that a committed vector of values is drawn entrywise from a committed table, and for lookup arguments admitting table-dependent preprocessing, this holds at an online cost that scales only with the size of the queried vector rather than the table itself. If we can phrase “the restricted columns of T_i ” as a lookup into “the full columns of T_i ,” a lookup argument lets us enforce this restriction directly – though not for free, since the prover must still commit to the restricted matrices and prove their consistency with the original ones. The obstacle is that T_i is a two-dimensional object, and the operation we care about is not selecting individual table entries but consistently restricting entire *columns* across every matrix T_i at once. Since a SNARK verifier only has access to a succinct or committed representation of these matrices, this representation must itself make column restrictions verifiable via a lookup: each element of the committed encoding must be traceable back to a single, fixed column of the underlying matrix, so that removing a column from the input has a well-defined, corresponding effect on the encoded output. We formalize this as a *column-indexable encoding* (Definition 17): an encoding whose j th output column is determined by a single column of the input matrix – not necessarily the j th one, and not necessarily the same column for every j – or, alternatively, depends only on shared key-generation randomness. Under such an encoding, restricting the columns of T_i to J corresponds exactly to taking a subvector of its encoded representation, so the prover can commit to the re-

stricted matrix descriptions and use a subvector (lookup) argument to certify consistency with the originally committed matrices.

Why a lookup is enough, except for public I/O. A lookup argument is deliberately weak: it only certifies that every entry of the queried vector occurs *somewhere* in the table, not that the correspondence between query positions and table positions is a fixed, order-preserving embedding. Concretely, when the prover looks up the restricted columns of T_i against the full columns of T_i , the extractor only obtains an index map $\sigma : [n] \rightarrow [N]$ with $T'_i[:, j] = T_i[:, \sigma(j)]$, and nothing forces σ to be injective: two different restricted-column slots j, j' may both pull the *same* original column of T_i . This is harmless precisely because $T_i \cdot z$ is linear in z and a repeated slot is a repeated (identical) column, not a different one. This slack is not harmless for the public-input and constant coordinates of z , however: their values are supplied directly by the verifier at fixed positions, so there is no freedom to *redefine* z_i after the fact — the reduction must instead guarantee that (say) the third restricted column really is the third public input, not some merge of two distinct public inputs that happens to satisfy the linear equation. A generic lookup does not enforce this. Our fix folds this positional guarantee into the same lookup: we additionally look up a public-I/O *mask* vector, which tags every public-input and constant column with a value unique to that column and tags every witness column with the same value 0, using the common-index batching property so that this mask lookup is forced to share its extracted index map σ with every other table/query pair. Since the tags on public and constant columns are unique, σ cannot merge or reorder them there; since every witness column shares the tag 0, σ is left free to repeat or reorder among them, exactly where the argument above shows this is safe.

A two-step reduction. This observation suggests a natural two-step strategy. First, we use a lookup argument to reduce the committed, full-size instance to a committed instance over only the n relevant coordinates: the prover commits to the restricted encoding of the constraint matrices, and the lookup argument certifies that this restriction is consistent with the original committed matrices under the same column-restriction set J . Second, we run a standard argument of knowledge for the underlying constraint system directly on this smaller, n -sized instance. For this composition to actually deliver an n -dependent (rather than N -dependent) prover, we need three properties to hold simultaneously: the lookup argument’s online prover time must depend only on the size of the queried subvector and not on the ambient table size N , the SNARK prover must be determined by the size of the witness rather than the amount of constraints (assuming a bounded fan-out), and the circuit encoding used by the underlying SNARK must be column-indexable, so that the object being looked up is exactly the object the SNARK needs to run on the smaller instance.

Additional considerations: tracking key size across the reduction. A subtlety arises once we try to formalize this two-step strategy as a single reduction of knowledge (Definition 18). The standard notion of a reduction of knowledge only

tracks how instances and witnesses evolve across the reduction; it says nothing about how the *prover and verifier keys* for the target relation are derived. In our setting this omission is not benign: unless the reduction guarantees that the derived keys for the n -sized instance can themselves be computed (or updated) in time depending only on n , the second step's prover cannot actually run in time independent of N , even though the instance and witness it receives are already appropriately small. To close this gap, we introduce *key-transforming reductions of knowledge* (Definition 8), which explicitly model key generation via algorithms $\mathcal{K}_1, \mathcal{K}_2$ sharing common randomness, and which require the reduction to produce a target key that is consistent with directly invoking $\mathcal{K}_2$ on the reduced structure. We prove that this notion composes sequentially (Lemma 2), and we cast our sparse argument of knowledge (Theorem 3) as an instance of this framework, so that composing it with any other key-transforming reduction preserves the same efficiency guarantees.

3 Preliminaries

3.1 Notation

For vectors x, y , we denote their concatenation by (x, y) . We let $[n]$ denote the set $\{1, \dots, n\}$.

We define a bilinear group gk as a tuple $gk = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$, where $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are all groups sharing the same prime order q . The elements $\mathcal{P}_1$ and $\mathcal{P}_2$ serve as generators for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing that is non-degenerate and efficiently computable, and we further assume that no efficient isomorphism exists mapping $\mathbb{G}_1$ to $\mathbb{G}_2$. For each $\gamma \in \{1, 2, T\}$, elements of $\mathbb{G}_\gamma$ are written implicitly using the notation $[a]_\gamma = a\mathcal{P}_\gamma$, with $[1]_T$ defined as $e(\mathcal{P}_1, \mathcal{P}_2)$. This notation lets us express the bilinearity property compactly as $e([a]_1, [b]_2) = [ab]_T$.

For functions $a, b : \mathbb{N} \rightarrow [0, 1]$, we write $a \approx b$ if $|a(\lambda) - b(\lambda)|$ is negligible in the security parameter λ .

3.2 Algebraic Group Model

We prove security in the Algebraic Group Model (AGM) introduced by Fuchs-bauer et al. [12], relying on the *bilinear* variants of the **q-dlog** and **q-sfrac** assumptions [12]. This model limits adversaries to *algebraic* strategies: for any group element $[y]$ that $\mathcal{A}$ returns in a cyclic group $\mathbb{G}$ of order p , the adversary must additionally supply the coefficients expressing $[y]$ as a linear combination of the group elements it has previously seen. Concretely, whenever $\mathcal{A}$ produces $[y] \leftarrow \mathcal{A}([x_1], \dots, [x_m])$, it is also required to output a vector $\mathbf{z}$ satisfying $[y] = \sum_{j=1}^m z_j [x_j]$.

3.3 Commitment Schemes

Definition 1 (Commitment Scheme). A (*binding*) commitment scheme consists of the following algorithms:

- $\text{gen}(1^\lambda) \rightarrow \text{pp}$: A PPT algorithm that generates public parameters pp .
- $\text{commit}(\text{pp}, m, r) \rightarrow \bar{c}$: A deterministic algorithm that takes public parameters pp , message m , and potentially a randomness r and outputs a commitment $\bar{c}$.

A commitment scheme must satisfy the following property:

- **Binding**: For all PPT adversaries $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{gen}(1^\lambda), \\ (m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(\text{pp}), \\ m_1 \neq m_2, \\ \text{commit}(\text{pp}, m_1, r_1) = \text{commit}(\text{pp}, m_2, r_2) \end{array} \right] \leq \text{negl}(\lambda)$$

Definition 2 (Vector Commitment Scheme [7]). A Vector Commitment Scheme is a tuple of algorithms $(\text{gen}, \text{commit}, \text{open}, \text{verify})$ such that:

- $\text{pp} \leftarrow \text{gen}(\text{pp}, N)$: On input the system parameters and a degree bound d , it outputs a structured reference string.
- $\bar{v} \leftarrow \text{commit}(\text{pp}, \mathbf{v})$: It outputs $\bar{v}$.
- $(s, \pi) \leftarrow \text{open}(\text{pp}, \mathbf{v}, i)$: On input vector $\mathbf{v}$ and a position $i \in [N]$, the prover computes and outputs a proof π that $v_i = s$.
- $1/0 \leftarrow \text{verify}(\text{pp}, \bar{v}, i, s, \pi)$: On input the commitment, position i , s , and proof π , the verifier accepts or rejects.

A Vector Commitment Scheme must satisfy Binding (as above) and Extractability:

- **Extractability**: For every PPT adversary $\mathcal{A}$ that outputs a commitment $\bar{v}$, there exists an expected PPT extractor $\mathcal{E}^{\mathcal{A}}$ that outputs a vector $\mathbf{v} \in \mathbb{F}^N$ such that any accepting opening of $\bar{v}$ at position i is to v_i , except with negligible probability.

Construction 1 (Structured Commitment Scheme [18]). Consider group $\mathbb{G}$ over scalar field $\mathbb{F}$. We consider the structured commitment scheme KZG characterized by group $\mathbb{G}$ over message space $\mathbb{F}^n$ and commitment space $\mathbb{G}$ as follows.

- $\text{gen}_{\text{KZG}}(t, \tau \in \mathbb{F})$: Output $\text{ck} = ([\lambda_1(\tau)], \dots, [\lambda_t(\tau)])$, where $\lambda_1, \dots, \lambda_t$ are the Lagrange polynomials over $[t]$.
- $\text{Com}_{\text{KZG}}(\text{ck}, w)$: Output $\bar{w} \leftarrow \sum_{i \in [t]} w_i \cdot [\lambda_i(\tau)]$.

3.4 Relations

Definition 3 (Relation). A relation $\mathcal{R}$ is a set of tuples $(\text{pp}, \mathbf{s}, u, w)$, where pp denotes public parameters, $\mathbf{s}$ a public structure, u a public instance, and w a witness. We write $(\text{pp}, \mathbf{s}, u, w) \in \mathcal{R}$ when w is a valid witness for $(\text{pp}, \mathbf{s}, u)$.

3.5 Lookup Arguments

Definition 4 (Lookup Argument). A lookup argument for dimensions (N, n) consists of a deterministic, linearly homomorphic commitment scheme Com for vectors over $\mathbb{F}$, whose commitment algorithm commit may use different commitment spaces for tables and for queries, and a table-preprocessed argument $\Pi_{\text{LU}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with the following properties.

Single-table knowledge soundness. Π_{LU} is an argument of knowledge for

$$\text{cLU}_{\text{Com}}(N, n) = \left\{ \begin{array}{l|l} \text{params} & \text{pp}, \\ \text{structure} & \perp, \\ \text{instance} & (\bar{\mathbf{t}}, \bar{\mathbf{f}}), \\ \text{witness} & (\mathbf{t}, \mathbf{f}, I) \end{array} \middle| \begin{array}{l} \mathbf{t} \in \mathbb{F}^N, \quad \mathbf{f} \in \mathbb{F}^n, \\ I = (i_1, \dots, i_n) \in [N]^n, \\ f_j = t_{i_j} \quad \forall j \in [n], \\ \bar{\mathbf{t}} = \text{commit}(\text{pp}, \mathbf{t}), \\ \bar{\mathbf{f}} = \text{commit}(\text{pp}, \mathbf{f}). \end{array} \right\}.$$

Foldable preprocessing. There are table-independent keys $\text{pk}_{\text{fix}}, \text{vk}_{\text{fix}}$ and linear maps $L_{\mathcal{P}}, L_{\mathcal{V}}$ such that

$$\mathcal{K}(\text{pp}, \mathbf{t}) = ((\text{pk}_{\text{fix}}, L_{\mathcal{P}}(\mathbf{t})), (\text{vk}_{\text{fix}}, L_{\mathcal{V}}(\mathbf{t}))).$$

Thus, from preprocessed keys for tables $\mathbf{t}^{(1)}, \dots, \mathbf{t}^{(s)}$ and coefficients $\rho_1, \dots, \rho_s \in \mathbb{F}$, the parties derive keys for $\mathbf{t}^* = \sum_{a=1}^s \rho_a \mathbf{t}^{(a)}$ by retaining the fixed parts once and linearly combining the table-dependent parts.

Common-index batching. For any polynomial number s of table/query pairs, after all query commitments are fixed, a random linear combination of the pairs followed by one invocation of Π_{LU} is an argument of knowledge for

$$\text{cLU}_{\text{Com}}^s(N, n) = \left\{ \begin{array}{l|l} \text{params} & \text{pp}, \\ \text{structure} & \perp, \\ \text{instance} & (\{\bar{\mathbf{t}}^{(a)}, \bar{\mathbf{f}}^{(a)}\}_{a \in [s]}), \\ \text{witness} & (\{\mathbf{t}^{(a)}, \mathbf{f}^{(a)}\}_{a \in [s]}, I) \end{array} \middle| \begin{array}{l} I = (i_1, \dots, i_n) \in [N]^n, \\ f_j^{(a)} = t_{i_j}^{(a)} \quad \forall a \in [s], j \in [n], \\ \bar{\mathbf{t}}^{(a)} = \text{commit}(\text{pp}, \mathbf{t}^{(a)}), \\ \bar{\mathbf{f}}^{(a)} = \text{commit}(\text{pp}, \mathbf{f}^{(a)}) \quad \forall a \in [s]. \end{array} \right\}.$$

In particular, a public-I/O mask may be included as one additional ordinary table/query pair in the batch.

3.6 Polynomials

Definition 5 (Lagrange Polynomials). For domain $D \subseteq \mathbb{F}$ and $x \in D$, we define the univariate Lagrange polynomial $\lambda_x^D(X) \in \mathbb{F}^{D-1}[X]$ such that for all $x' \in D$, $\lambda_x(x')$ is 1 if $x' = x$ and 0 otherwise. In particular,

$$\lambda_x^D(X) = \prod_{x' \in D, x' \neq x} \frac{X - x'}{x - x'}.$$

Definition 6 (Polynomial Interpolation). For a vector $\mathbf{v} \in \mathbb{F}^N$, and ordered domain $D = \{x_1, \dots, x_N\}$ we define the polynomial $\tilde{v}^D(X)$ as a polynomial in $\mathbb{F}^{N-1}[X]$ such that $\tilde{v}^D(x_s) = v_s$. That is, $\tilde{v}^D(x) = \sum v_s \lambda_s^D(X)$.

We omit D when clear from context.

Definition 7 (Polynomial Commitment Scheme). A polynomial commitment scheme for $\mathbb{F}_{<d}[X]$ is a tuple of algorithms ($\text{gen}, \text{commit}, \text{open}, \text{verify}$) such that:

- $\text{pp} \leftarrow \text{gen}(1^\lambda, d)$: On input the security parameter and a degree bound d , it outputs a structured reference string.
- $\bar{P} \leftarrow \text{commit}(\text{pp}, P)$: On input $P \in \mathbb{F}_{<d}[X]$, it deterministically outputs a commitment $\bar{P}$.
- $(y, \pi) \leftarrow \text{open}(\text{pp}, P, a)$: On input P and an evaluation point $a \in \mathbb{F}$, the prover computes and outputs $y = P(a)$ together with a proof π .
- $1/0 \leftarrow \text{verify}(\text{pp}, \bar{P}, a, y, \pi)$: On input the commitment, point a , claimed value y , and proof π , the verifier accepts or rejects.

A polynomial commitment scheme must satisfy Binding (as in Definition 1) and Extractability:

- **Extractability:** For every PPT adversary $\mathcal{A}$ that outputs a commitment $\bar{P}$, there exists an expected PPT extractor $\mathcal{E}^{\mathcal{A}}$ that outputs a polynomial $P \in \mathbb{F}_{<d}[X]$ such that any accepting opening of $\bar{P}$ at a point a is to $P(a)$, except with negligible probability.

A polynomial commitment scheme for $\mathbb{F}_{<N}[X]$ yields a vector commitment scheme for dimension N (Definition 2) over any domain $D = x_1, \dots, x_N \subset \mathbb{F}$: commit to $\mathbf{v}$ by committing to its interpolant $\tilde{v}^D(X)$, and open position i by opening $\tilde{v}^D(X)$ at the point x_i . Binding and extractability transfer directly, since $\tilde{v}^D$ is the unique polynomial of degree less than N agreeing with $\mathbf{v}$ on D .

Construction 2 (KZG). Given pairing-friendly groups $\mathbb{G}_1, \mathbb{G}_2$ and a degree bound d , we construct a polynomial commitment scheme for $\mathbb{F}_{<d}[X]$.

$\text{gen}(1^\lambda, d)$: Choose random $x \in \mathbb{F}$ and output

$$\text{pp} = (\{[x^i]_1\}_{i=0}^{d-1}, [x]_2).$$

$\text{commit}(\text{pp}, P)$: For $P(X) = \sum_{i=0}^{d-1} p_i X^i$, output

$$\bar{P} = [P(x)]_1 = \sum_{i=0}^{d-1} p_i [x^i]_1.$$

$\text{open}(\text{pp}, P, a)$: Compute $y = P(a)$ and

$$Q(X) = \frac{P(X) - y}{X - a} \in \mathbb{F}_{<d-1}[X],$$

which is a polynomial since $P(a) = y$. Output (y, π) for $\pi = [Q(x)]_1$.

$\text{verify}(\text{pp}, \bar{P}, a, y, \pi)$: Accept if and only if

$$e(\bar{P} - [y]_1, [1]_2) = e(\pi, [x - a]_2).$$

Theorem 2 (KZG [8, 13]). KZG is a polynomial commitment scheme satisfying Binding and Extractability as in Definition 7 in the AGM, under the dlog assumption.

4 Key-Transforming Reductions of Knowledge

Recall that a reduction of knowledge is an interactive protocol between a prover and a verifier that reduces the task of checking a statement in a relation $\mathcal{R}_1$ to the task of checking a (simpler) statement in a relation $\mathcal{R}_2$. In this section, we introduce *key-transforming* reductions of knowledge to capture reductions where the prover and verifier key are also soundly updated to reflect a change in the underlying computation being checked without rerunning the preprocessing step. This notion is inspired by a similar notion due to Zheng et al. [34], albeit for arguments of knowledge. In our particular use case, the key updating property is used capture the process of producing updated prover and verifier keys to a particular subcircuit. We recall the standard definition of a reduction of knowledge in Appendix A.

Definition 8 (Key-Transforming Reductions of Knowledge [19, 34]).

Consider relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance, and witness tuples. A key-transforming reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic polynomial-time algorithms $\mathcal{K}_1$, $\mathcal{K}_2$, and φ , called the generator, the prover, the verifier, the input key encoder, the output key encoder, and the transcript decoder respectively with the following interface.

- $\mathcal{G}(\lambda, n) \rightarrow (\mathbf{pp}, \kappa)$: Takes as input security parameter λ , size parameters n . Outputs the public parameters $\mathbf{pp}$ and trapdoor randomness κ ⁴.
- $\mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1; \kappa) \rightarrow (\mathbf{pk}_1, \mathbf{vk}_1)$: Takes as input public parameters $\mathbf{pp}$, structure $\mathbf{s}_1$, and explicit trapdoor randomness κ . Outputs prover key $\mathbf{pk}_1$ and verifier key $\mathbf{vk}_1$.
- $\mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2; \kappa) \rightarrow (\mathbf{pk}_2, \mathbf{vk}_2)$: Takes as input public parameters $\mathbf{pp}$, structure $\mathbf{s}_2$, and the same setup trapdoor κ . Outputs prover key $\mathbf{pk}_2$ and verifier key $\mathbf{vk}_2$.⁵
- $\mathcal{P}(\mathbf{pk}_1, u_1, w_1) \rightarrow (\mathbf{pk}_2, u_2, w_2)$: Takes as input prover key $\mathbf{pk}_1$, and an instance-witness pair (u_1, w_1) . Interactively reduces the task of checking $(\mathbf{pp}, \mathbf{s}_1, u_1, w_1) \in \mathcal{R}_1$ to the task of checking $(\mathbf{pp}, \mathbf{s}_2, u_2, w_2) \in \mathcal{R}_2$, while outputting the reduced prover key $\mathbf{pk}_2$. Here $\mathbf{s}_1$ is the structure used to generate the input keys and $\mathbf{s}_2$ is determined by the decoder φ ; neither is an explicit input or output of $\mathcal{P}$.
- $\mathcal{V}(\mathbf{vk}_1, u_1) \rightarrow (\mathbf{vk}_2, u_2)$: Takes as input verifier key $\mathbf{vk}_1$ and an instance u_1 in $\mathcal{R}_1$. Interactively reduces the task of checking instance u_1 to the task of checking a new instance u_2 in $\mathcal{R}_2$ under verifier key $\mathbf{vk}_2$, while outputting the reduced verifier key $\mathbf{vk}_2$.
- $\varphi(\mathbf{pp}, \mathbf{s}_1, u_1, \mathbf{tr}) \rightarrow (\mathbf{s}_2, u_2)$: Takes as input public parameters $\mathbf{pp}$, input structure $\mathbf{s}_1$, input instance u_1 , and transcript $\mathbf{tr}$. Outputs output structure $\mathbf{s}_2$ and output instance u_2 .

⁴ When the key encoders are computable from $\mathbf{pp}$ alone, as in the universal or transparent setting, $\kappa = \perp$.

⁵ When the trapdoor is clear from context, we write $\mathcal{K}_i(\mathbf{pp}, \mathbf{s}_i)$ in place of $\mathcal{K}_i(\mathbf{pp}, \mathbf{s}_i; \kappa)$, and similarly omit κ from an encoding evaluated inside $\mathcal{K}_i$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. We treat $\langle \mathcal{P}, \mathcal{V} \rangle$ as a function that takes as input $((\mathbf{pk}_1, \mathbf{vk}_1), u_1, w_1)$ and runs the interaction on the prover's input $(\mathbf{pk}_1, u_1, w_1)$ and the verifier's input $(\mathbf{vk}_1, u_1)$. At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the verifier's tuple $(\mathbf{vk}_2, u'_2)$ and the prover's tuple $(\mathbf{pk}_2, u_2, w_2)$ with transcript tr . A key-transforming reduction of knowledge $(\mathcal{G}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{P}, \mathcal{V}, \varphi)$ satisfies the following conditions.

- (i) *Public reducibility:* For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, given

$$\begin{aligned} (\mathbf{pp}, \kappa) &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathbf{s}_1, u_1, \text{st}) &\leftarrow \mathcal{A}(\mathbf{pp}), \\ (\mathbf{pk}_1, \mathbf{vk}_1) &\leftarrow \mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1; \kappa), \\ ((\mathbf{vk}_2, u'_2), (\mathbf{pk}_2, u_2, w_2), \text{tr}) &\leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathbf{pk}_1, \mathbf{vk}_1), u_1, \text{st}), \\ (\widehat{\mathbf{s}}_2, \widehat{u}_2) &\leftarrow \varphi(\mathbf{pp}, \mathbf{s}_1, u_1, \text{tr}), \\ (\widehat{\mathbf{pk}}_2, \widehat{\mathbf{vk}}_2) &\leftarrow \mathcal{K}_2(\mathbf{pp}, \widehat{\mathbf{s}}_2; \kappa) \end{aligned}$$

we have that

$$(\mathbf{vk}_2, u'_2) = (\widehat{\mathbf{vk}}_2, \widehat{u}_2).$$

- (ii) *Completeness:* For any PPT adversary $\mathcal{A}$, given

$$\begin{aligned} (\mathbf{pp}, \kappa) &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathbf{s}_1, u_1, w_1) &\leftarrow \mathcal{A}(\mathbf{pp}), \\ (\mathbf{pk}_1, \mathbf{vk}_1) &\leftarrow \mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1; \kappa), \\ ((\mathbf{vk}_2, u'_2), (\mathbf{pk}_2, u_2, w_2), \text{tr}) &\leftarrow \langle \mathcal{P}, \mathcal{V} \rangle((\mathbf{pk}_1, \mathbf{vk}_1), u_1, w_1), \\ (\mathbf{s}_2, \widehat{u}_2) &\leftarrow \varphi(\mathbf{pp}, \mathbf{s}_1, u_1, \text{tr}), \\ (\mathbf{pp}, \mathbf{s}_1, u_1, w_1) &\in \mathcal{R}_1 \end{aligned}$$

we have that

$$\begin{aligned} u_2 &= u'_2 \\ (\mathbf{pk}_2, \mathbf{vk}_2) &= \mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2; \kappa) \\ (\mathbf{pp}, \mathbf{s}_2, u_2, w_2) &\in \mathcal{R}_2. \end{aligned}$$

- (iii) *Knowledge soundness:* For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that given $(\mathbf{pp}, \kappa) \leftarrow \mathcal{G}(\lambda, n)$, $(\mathbf{s}_1, u_1, \text{st}) \leftarrow \mathcal{A}(\mathbf{pp})$, and $(\mathbf{pk}_1, \mathbf{vk}_1) \leftarrow \mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1; \kappa)$, we have that

$$\begin{aligned} &\Pr[(\mathbf{pp}, \mathbf{s}_1, u_1, \mathcal{E}((\mathbf{pp}, \mathbf{pk}_1, \mathbf{vk}_1), \mathbf{s}_1, u_1, \text{st})) \in \mathcal{R}_1] \\ &\approx \Pr \left[\begin{array}{l} ((\mathbf{vk}_2, u'_2), (\mathbf{pk}_2, u_2, w_2), \text{tr}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathbf{pk}_1, \mathbf{vk}_1), u_1, \text{st}), \\ (\widehat{\mathbf{s}}_2, \widehat{u}_2) \leftarrow \varphi(\mathbf{pp}, \mathbf{s}_1, u_1, \text{tr}), \\ u_2 = u'_2, \\ (\mathbf{pk}_2, \mathbf{vk}_2) = \mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2; \kappa), \\ (\mathbf{pp}, \mathbf{s}_2, u_2, w_2) \in \mathcal{R}_2 \end{array} \right]. \end{aligned}$$

We use the following terminology for reductions of knowledge, including key-transforming reductions.

Definition 9 (Succinctness). *A reduction of knowledge is succinct if the communication complexity and the verifier time complexity is at most poly-logarithmic in the size of the structure and witness.*

Definition 10 (Non-interactivity). *A reduction of knowledge is non-interactive if the interaction consists of a single message from the prover to the verifier. In this case, we denote this single message as the output of the prover, and as an input to the verifier.*

Definition 11 (Public-coin). *A reduction of knowledge is public-coin if the verifier only sends uniformly random challenges during the interaction.*

Definition 12 (Argument of Knowledge). *Consider the relation $\text{TRUE} = (\text{true}, \perp)$. An argument of knowledge for relation $\mathcal{R}$ is a reduction of knowledge from $\mathcal{R}$ to TRUE .*

Definition 13 (Sparse Reductions of Knowledge). *Consider relation $\mathcal{R}_1$ with witnesses in $\mathbb{F}^N$ and relation $\mathcal{R}_2$ with witnesses in $\mathbb{F}^M$. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is sparse if given a witness that contains n non-zero values the prover time is $O(n) \cdot o(N)$, and the output witness contains $O(n)$ non-zero values.*

Lemma 1 (Closure of Sparsity). *Sparsity is closed under sequential composition.*

Lemma 2 (Sequential composition [19]). *Let $\Pi_1 = (\mathcal{G}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{P}_1, \mathcal{V}_1, \varphi_1) : \mathcal{R}_1 \rightarrow \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}_2, \mathcal{K}_3, \mathcal{P}_2, \mathcal{V}_2, \varphi_2) : \mathcal{R}_2 \rightarrow \mathcal{R}_3$. Assume that all key encoders are publicly computable ($\kappa = \perp$). Then $\Pi_2 \circ \Pi_1 : \mathcal{R}_1 \rightarrow \mathcal{R}_3$ is a key-transforming reduction of knowledge. The composed prover and verifier run Π_1 followed by Π_2 . The composed decoder is defined by*

$$\varphi(\text{pp}, \text{s}_1, u_1, (\text{tr}_1, \text{tr}_2)) = \varphi_2(\text{pp}, \varphi_1(\text{pp}, \text{s}_1, u_1, \text{tr}_1), \text{tr}_2).$$

Proof. By completeness of Π_1 , its output keys are generated by $\mathcal{K}_2$ for the structure decoded by φ_1 , and its output witness satisfies $\mathcal{R}_2$ for that structure. These are valid inputs for Π_2 . Its completeness gives the required final keys and witness for the structure decoded by φ_2 , which is also the structure decoded by the composed φ . Public reducibility follows from the definition of φ and the public reducibility of the two component reductions. Indeed, parse the composed transcript as $\text{tr} = (\text{tr}_1, \text{tr}_2)$. By public reducibility of Π_1 ,

$$(\text{s}_2, u_2) = \varphi_1(\text{pp}, \text{s}_1, u_1, \text{tr}_1)$$

gives the actual intermediate instance, and the intermediate verifier key is the verifier-key component of $\mathcal{K}_2(\text{pp}, \text{s}_2)$. Therefore the second phase is a valid execution of Π_2 on input (vk_2, u_2) with source structure s_2 . Applying public reducibility of Π_2 shows that $\varphi_2(\text{pp}, \text{s}_2, u_2, \text{tr}_2)$ gives a final structure s_3 and the

actual final instance, with final verifier key equal to the verifier-key component of $\mathcal{K}_3(\mathbf{pp}, \mathbf{s}_3)$. This is exactly the public reducibility condition for the composed decoder φ .

It remains to prove knowledge soundness. Consider arbitrary expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$ for the composed reduction. Let $(\mathbf{pp}, \perp) \leftarrow \mathcal{G}(\lambda, n)$, $(\mathbf{s}_1, u_1, \mathbf{st}) \leftarrow \mathcal{A}(\mathbf{pp})$, and $(\mathbf{pk}_1, \mathbf{vk}_1) \leftarrow \mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1)$. Suppose that

$$\epsilon = \Pr \left[\begin{array}{l} ((\mathbf{vk}_3, u'_3), (\mathbf{pk}_3, u_3, w_3), \mathbf{tr}) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathbf{pk}_1, \mathbf{vk}_1), u_1, \mathbf{st}), \\ (\mathbf{s}_3, \hat{u}_3) \leftarrow \varphi(\mathbf{pp}, \mathbf{s}_1, u_1, \mathbf{tr}), \\ u_3 = u'_3, \\ (\mathbf{pk}_3, \mathbf{vk}_3) = \mathcal{K}_3(\mathbf{pp}, \mathbf{s}_3), \\ (\mathbf{pp}, \mathbf{s}_3, u_3, w_3) \in \mathcal{R}_3 \end{array} \right]. \quad (1)$$

We construct an expected polynomial-time extractor for $\mathcal{R}_1$ whose success probability differs from ϵ by at most a negligible function.

First, we construct adversaries $\mathcal{A}_2$ and $\mathcal{P}_2^{**}$ for Π_2 . View $\mathcal{P}^*$ as a first-stage machine, which interacts with $\mathcal{V}_1$, followed by a second-stage machine, which receives the first-stage state and interacts with $\mathcal{V}_2$.

$\mathcal{A}_2(\mathbf{pp}) \rightarrow (\mathbf{s}_2, u_2, \mathbf{st}_2)$:

1. Compute $(\mathbf{s}_1, u_1, \mathbf{st}) \leftarrow \mathcal{A}(\mathbf{pp})$ and $(\mathbf{pk}_1, \mathbf{vk}_1) \leftarrow \mathcal{K}_1(\mathbf{pp}, \mathbf{s}_1)$.
2. Run the first stage of $\mathcal{P}^*$ with $\mathcal{V}_1$ on input $((\mathbf{pk}_1, \mathbf{vk}_1), u_1, \mathbf{st})$. Let $\mathbf{tr}_1$ be the first-stage transcript and let $\mathbf{st}_2$ be the resulting state of $\mathcal{P}^*$.
3. Compute $(\mathbf{s}_2, u_2) \leftarrow \varphi_1(\mathbf{pp}, \mathbf{s}_1, u_1, \mathbf{tr}_1)$.
4. Output $(\mathbf{s}_2, u_2, \mathbf{st}_2)$.

$\mathcal{P}_2^{**}(\mathbf{pk}_2, u_2, \mathbf{st}_2)$:

1. Resume the second stage of $\mathcal{P}^*$ from state $\mathbf{st}_2$ and interact with $\mathcal{V}_2$.
2. Output the final tuple produced by $\mathcal{P}^*$.

By public reducibility of Π_1 , the instance decoded by φ_1 is exactly the instance used by the composed verifier in the second phase, and its verifier key is the verifier key output by $\mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2)$. Therefore the execution of $\mathcal{P}_2^{**}$ with $\mathcal{V}_2$ in the Π_2 experiment has the same distribution as the second phase of the composed execution. By the definition of the composed decoder, both experiments also decode the same final structure $\mathbf{s}_3$. Hence, by Equation (1),

$$\Pr \left[\begin{array}{l} (\mathbf{pk}_2, \mathbf{vk}_2) \leftarrow \mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2), \\ ((\mathbf{vk}_3, u'_3), (\mathbf{pk}_3, u_3, w_3), \mathbf{tr}_2) \leftarrow \langle \mathcal{P}_2^{**}, \mathcal{V}_2 \rangle((\mathbf{pk}_2, \mathbf{vk}_2), u_2, \mathbf{st}_2), \\ (\mathbf{s}_3, \hat{u}_3) \leftarrow \varphi_2(\mathbf{pp}, \mathbf{s}_2, u_2, \mathbf{tr}_2), \\ u_3 = u'_3, \\ (\mathbf{pk}_3, \mathbf{vk}_3) = \mathcal{K}_3(\mathbf{pp}, \mathbf{s}_3), \\ (\mathbf{pp}, \mathbf{s}_3, u_3, w_3) \in \mathcal{R}_3 \end{array} \right] = \epsilon. \quad (2)$$

By knowledge soundness of Π_2 , there exists an expected polynomial-time extractor $\mathcal{E}_2$ such that

$$\delta = \Pr[(\mathbf{pp}, \mathbf{s}_2, u_2, \mathcal{E}_2((\mathbf{pp}, \mathbf{pk}_2, \mathbf{vk}_2), \mathbf{s}_2, u_2, \mathbf{st}_2)) \in \mathcal{R}_2] \approx \epsilon. \quad (3)$$

Next, we use $\mathcal{E}_2$ to construct adversaries $\mathcal{A}_1$ and $\mathcal{P}_1^{**}$ for Π_1 .

$\mathcal{A}_1(\text{pp}) \rightarrow (s_1, u_1, \text{st}_1)$:

1. Compute $(s_1, u_1, \text{st}) \leftarrow \mathcal{A}(\text{pp})$.
2. Compute $(\text{pk}_1, \text{vk}_1) \leftarrow \mathcal{K}_1(\text{pp}, s_1)$.
3. Output (s_1, u_1, st_1) where $\text{st}_1 = (\text{st}, \text{pp}, s_1, \text{vk}_1)$.

$\mathcal{P}_1^{**}(\text{pk}_1, u_1, \text{st}_1)$:

1. Parse st_1 as $(\text{st}, \text{pp}, s_1, \text{vk}_1)$.
2. Run the first stage of $\mathcal{P}^*$ with $\mathcal{V}_1$ on input $((\text{pk}_1, \text{vk}_1), u_1, \text{st})$. Let tr_1 be the first-stage transcript and let st_2 be the resulting state of $\mathcal{P}^*$.
3. Compute $(s_2, u_2) \leftarrow \varphi_1(\text{pp}, s_1, u_1, \text{tr}_1)$ and $(\text{pk}_2, \text{vk}_2) \leftarrow \mathcal{K}_2(\text{pp}, s_2)$.
4. Compute $w_2 \leftarrow \mathcal{E}_2((\text{pp}, \text{pk}_2, \text{vk}_2), s_2, u_2, \text{st}_2)$.
5. Output (pk_2, u_2, w_2) .

By public reducibility of Π_1 , the verifier output of this Π_1 execution is (vk_2, u_2) for the keys computed above. Therefore $\mathcal{P}_1^{**}$ succeeds in the Π_1 experiment exactly when $\mathcal{E}_2$ outputs a witness for (pp, s_2, u_2) . By Equation (3),

$$\Pr \left[\begin{array}{l} ((\text{vk}_2, u'_2), (\text{pk}_2, u_2, w_2), \text{tr}_1) \leftarrow \langle \mathcal{P}_1^{**}, \mathcal{V}_1 \rangle((\text{pk}_1, \text{vk}_1), u_1, \text{st}_1), \\ (s_2, \hat{u}_2) \leftarrow \varphi_1(\text{pp}, s_1, u_1, \text{tr}_1), \\ u_2 = u'_2, \\ (\text{pk}_2, \text{vk}_2) = \mathcal{K}_2(\text{pp}, s_2), \\ (\text{pp}, s_2, u_2, w_2) \in \mathcal{R}_2 \end{array} \right] = \delta. \quad (4)$$

By knowledge soundness of Π_1 , there exists an expected polynomial-time extractor $\mathcal{E}_1$ such that

$$\Pr[(\text{pp}, s_1, u_1, \mathcal{E}_1((\text{pp}, \text{pk}_1, \text{vk}_1), s_1, u_1, \text{st}_1)) \in \mathcal{R}_1] \approx \delta \approx \epsilon.$$

More explicitly, let $\eta_2(\lambda)$ bound $|\delta - \epsilon|$ and let $\eta_1(\lambda)$ bound the difference between the success probability of $\mathcal{E}_1$ and δ . Both functions are negligible by the respective knowledge-soundness guarantees. The triangle inequality bounds the difference between the success probability of $\mathcal{E}_1$ and ϵ by $\eta_1(\lambda) + \eta_2(\lambda)$, which is negligible. The desired extractor for the composed reduction computes $\text{st}_1 = (\text{st}, \text{pp}, s_1, \text{vk}_1)$ and outputs

$$\mathcal{E}_1((\text{pp}, \text{pk}_1, \text{vk}_1), s_1, u_1, \text{st}_1).$$

□

5 Building Sparse SNARKs

In this section we make the strategy of the technical overview precise. Starting from the CCS relation, we define its committed (CCCS) and sparse committed (SCCCS) variants, and show (Theorem 3) that, for a column-indexable encoding and a lookup argument whose prover is linear in the size of the query,

sparsity can be compiled away by a key-transforming reduction of knowledge from $\text{SCCS}(M, N)$ to $\text{CCCS}(M, n)$. We track the number M of constraint rows separately: the reduction shrinks the column dimension from N to n and leaves the row dimension M unchanged. Since the result is a key-transforming reduction of knowledge, it composes (Lemma 2) with an argument of knowledge for $\text{CCCS}(M, n)$.

Concretely, consider the CCS relation below, which is known to be a representation of circuit satisfiability. Throughout, we write $\bar{N} := m + N + 1$ for the length of the extended vector $\mathbf{z}$.

Definition 14 (Customizable Constraint System). *We define the customizable constraint system relation CCS, parametrized by the number of constraint rows M and the witness size N , as follows.*

$$\text{CCS}(M, N) = \left\{ \begin{array}{ll} \text{params} & \perp, \\ \text{structure} & S, S^k \subseteq [t], \\ & \mathbf{T}_i \in \mathbb{F}^{M \times \bar{N}} \text{ for } i \in [t], \\ & c \in \mathbb{F}^q, \\ \text{instance} & \mathbf{x} \in \mathbb{F}^m, \\ \text{witness} & \mathbf{w} \in \mathbb{F}^N \end{array} \middle| \begin{array}{l} \mathbf{z} \leftarrow (\mathbf{x}, \mathbf{w}, 1), \\ \sum_{k \in [q]} c_k \cdot \bigcirc_{i \in S^k} (\mathbf{T}_i \cdot \mathbf{z}) = \mathbf{0} \end{array} \right\}.$$

Let $I \subseteq [m + N + 1]$ be such that $z_i = 0$ for all $i \in I$, and let $J = [m + N + 1] \setminus I$ denote the remaining coordinates. Let $\mathbf{z}_J$ be the vector obtained by keeping the coordinates of $\mathbf{z}$ indexed by J , in the same order. Similarly, let $\mathbf{T}_{i,J}$ be the matrix obtained from $\mathbf{T}_i$ by keeping only the columns indexed by J . Then

$$\mathbf{T}_i \cdot \mathbf{z} = \mathbf{T}_{i,J} \cdot \mathbf{z}_J.$$

Therefore, for this sparse witness, the original CCS equation is equivalent to the smaller equation

$$\sum_{k \in [q]} c_k \bigcirc_{i \in S^k} \mathbf{T}_{i,J} \cdot \mathbf{z}_J = \mathbf{0}.$$

It remains to let the verifier check that every $\mathbf{T}_i$ was restricted to the same J ; the relations below and Definition 17 capture this.

Definition 15 (Committed Customizable Constraint System). *We define the committed customizable constraint system relation CCCS, parametrized by the number of constraint rows M and the witness size N , as follows.*

$$\text{CCCS}(M, N) = \left\{ \begin{array}{ll} \text{params} & \perp, \\ \text{structure} & S, S^k \subseteq [t], \\ & c \in \mathbb{F}^q, \\ & \mathcal{E} : \mathbb{F}^{M \times \bar{N}} \rightarrow \mathbb{F}^{\ell \times \bar{N}}, \\ \text{instance} & \mathbf{x} \in \mathbb{F}^m, \\ & \bar{\mathbf{T}}_i \text{ for } i \in [t], \\ \text{witness} & \mathbf{T}_i \in \mathbb{F}^{M \times \bar{N}} \text{ for } i \in [t], \\ & \mathbf{w} \in \mathbb{F}^N \end{array} \middle| \begin{array}{l} \mathbf{z} \leftarrow (\mathbf{x}, \mathbf{w}, 1), \\ \bar{\mathbf{T}}_i = \text{commit}(\mathcal{E}(\mathbf{T}_i)) \ \forall i \in [t], \\ \sum_{k \in [q]} c_k \cdot \bigcirc_{i \in S^k} (\mathbf{T}_i \cdot \mathbf{z}) = \mathbf{0} \end{array} \right\}.$$

Definition 16 (Sparse Committed Customizable Constraint System).

We define the sparse committed customizable constraint system relation SCCCS , parametrized by the number of constraint rows M and the witness size N , as follows.

$$\text{SCCCS}(M, N) = \left\{ \begin{array}{l|l} \begin{array}{l} \text{params} \quad \perp, \\ \text{structure} \quad S, S^k \subseteq [t], \\ \quad c \in \mathbb{F}^q, \\ \quad \mathcal{E} : \mathbb{F}^{M \times \bar{N}} \rightarrow \mathbb{F}^{\ell \times \bar{N}}, \\ \text{instance} \quad \mathbf{x} \in \mathbb{F}^m, \\ \quad \bar{\mathbf{T}}_i \text{ for } i \in [t], \\ \text{witness} \quad \mathbf{T}_i \in \mathbb{F}^{M \times \bar{N}} \text{ for } i \in [t], \\ \quad \mathbf{w} \in \mathbb{F}^N, \\ \quad J \subseteq [N] \end{array} & \begin{array}{l} \mathbf{z} \leftarrow (\mathbf{x}, \mathbf{w}, 1), \\ w_i = 0 \quad \forall i \in [N] \setminus J, \\ \bar{\mathbf{T}}_i = \text{commit}(\mathcal{E}(\mathbf{T}_i)) \quad \forall i \in [t], \\ \sum_{k \in [q]} c_k \cdot \bigcirc_{i \in S^k} (\mathbf{T}_i \cdot \mathbf{z}) = \mathbf{0} \end{array} \end{array} \right\}.$$

In the relation SCCCS above, the verifier holds commitments $\bar{\mathbf{T}}_i$ to the encodings of the *full* matrices, while the prover only works with the columns in J . Reducing to $\text{CCCS}(M, n)$ therefore requires the encoding to be compatible with column restrictions: the encoding of the restricted matrix $\mathbf{T}_{i,J}$ must be obtainable as a subvector of $\mathcal{E}(\mathbf{T}_i)$, so that consistency of the new commitments with $\bar{\mathbf{T}}_i$ is a subvector (lookup) statement in the sense of Definition 4. The following definition of *column-indexability* captures this by asking that every element of the encoding is computed from a single fixed column of the input matrix, or from the setup randomness alone. It is a property of the encoding and not of the matrix; this is what allows one index vector to be applied to all the $\mathbf{T}_i$ at once. We verify it for the Groth16 encoder in Example 1, and show in Example 2 that the compact (row, col, val) encoding of Marlin and Spartan does not satisfy it. Conversely, an accepting proof for the restricted relation yields a witness for the original relation by embedding $\mathbf{z}_J$ back into the ambient vector and setting all coordinates outside J to zero.

Definition 17 (Column-Indexable Encoding). We say that an encoding function $\mathcal{E} : \mathbb{F}^{m \times n} \times (\mathbb{F}^k \cup \{\perp\}) \rightarrow \mathbb{F}^{\ell \times n}$ is column-indexable if there exist functions $\mathcal{S}_1, \dots, \mathcal{S}_\ell : (\mathbb{F}^m \cup \{\perp\}) \times (\mathbb{F}^k \cup \{\perp\}) \rightarrow \mathbb{F}$ such that, for every output column $j \in [n]$, there is a fixed $i \in [n]$ for which, for every matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, setup trapdoor $\boldsymbol{\kappa} \in \mathbb{F}^k \cup \{\perp\}$, and $r \in [\ell]$, we have

$$\mathcal{E}(\mathbf{A}, \boldsymbol{\kappa})_{r,j} = \mathcal{S}_r(\mathbf{A}_i, \boldsymbol{\kappa}) \quad \text{or} \quad \mathcal{E}(\mathbf{A}, \boldsymbol{\kappa})_{r,j} = \mathcal{S}_r(\perp, \boldsymbol{\kappa}).$$

The index i , as well as which of the two cases applies at each output position, is determined only by the encoding and the output position; it is independent of $\mathbf{A}$ and $\boldsymbol{\kappa}$. Here $\boldsymbol{\kappa}$ is the trapdoor output by $\mathcal{G}$ and supplied to the key algorithm $\mathcal{K}$, and the second case represents encoding elements that depend only on this randomness. The notation $\mathcal{E}(\mathbf{A}, \boldsymbol{\kappa})_{r,j}$ denotes the element in row r and column j of the encoding, and $\mathbf{A}_j$ denotes the j th column of $\mathbf{A}$.

Example 1: the Groth16 encoding. Consider the encoder of Groth16 [17],

$$\mathcal{E}_{\text{G16}} : \mathbb{F}^{3M \times (m+N+1)} \times \mathbb{F}^5 \rightarrow \mathbb{F}^{\ell \times (m+N+1)}, \quad \boldsymbol{\kappa} = (\alpha, \beta, \gamma, \delta, x),$$

which takes the stacked RICS matrix $\mathbf{T} = (\mathbf{U}^\top, \mathbf{V}^\top, \mathbf{W}^\top)^\top$ and the setup randomness $\boldsymbol{\kappa}$ to

$$\mathcal{E}_{G16}(\mathbf{T}, \boldsymbol{\kappa}) = \left\{ \begin{array}{l} \alpha, \beta, \gamma, \delta, \{x^i\}_{i=0}^{M-1}, \left\{ \frac{\beta u_j(x) + \alpha v_j(x) + w_j(x)}{\gamma} \right\}_{j=0}^m, \\ \left\{ \frac{\beta u_j(x) + \alpha v_j(x) + w_j(x)}{\delta} \right\}_{j=m+1}^{m+N}, \left\{ \frac{x^i t(x)}{\delta} \right\}_{i=0}^{M-2} \end{array} \right\},$$

where u_j, v_j, w_j interpolate the j th columns of $\mathbf{U}, \mathbf{V}, \mathbf{W}$ over the constraint domain and t is that domain's vanishing polynomial.

Lemma 3. *The encoder of Groth16 [17] is column-indexable.*

Proof. Let $\mathbf{T}_j = (\mathbf{U}_j, \mathbf{V}_j, \mathbf{W}_j)$ denote the j th column of the stacked matrix, that is, the vector encoding u_j, v_j, w_j . Set

$$\begin{aligned} \mathcal{S}_\gamma(\mathbf{T}_j, \boldsymbol{\kappa}) &= \frac{\beta u_j(x) + \alpha v_j(x) + w_j(x)}{\gamma}, \\ \mathcal{S}_\delta(\mathbf{T}_j, \boldsymbol{\kappa}) &= \frac{\beta u_j(x) + \alpha v_j(x) + w_j(x)}{\delta}, \\ \mathcal{S}_{\text{rand}}(\perp, \boldsymbol{\kappa}) &= (\alpha, \beta, \gamma, \delta), \quad \mathcal{S}_{\text{pow}}(\perp, \boldsymbol{\kappa}) = x^i, \quad \mathcal{S}_{\text{quot}}(\perp, \boldsymbol{\kappa}) = \frac{x^i t(x)}{\delta}. \end{aligned}$$

Each entry of $\mathcal{E}_{G16}(\mathbf{T}, \boldsymbol{\kappa})$ is obtained by applying one of $\mathcal{S}_\gamma, \mathcal{S}_\delta, \mathcal{S}_{\text{rand}}, \mathcal{S}_{\text{pow}}$ or $\mathcal{S}_{\text{quot}}$ to the corresponding column and $\boldsymbol{\kappa}$, or to $(\perp, \boldsymbol{\kappa})$. Hence $\mathcal{E}_{G16}$ is column-indexable. $\square$

Groth16 exercises both cases of Definition 17: the per-wire entries are determined by a single column of $\mathbf{T}$ together with $\boldsymbol{\kappa}$, while $\alpha, \beta, \gamma, \delta$, the powers $\{x^i\}$ and the quotient terms $\{x^i t(x)/\delta\}$ depend on $\boldsymbol{\kappa}$ alone.

Example 2: the compact (row, col, val) encoding. The standard sparse-matrix encoding, used in Marlin [8] and Spartan [24], instead lists the non-zero entries of $\mathbf{A}$ in a fixed order and records the row, column and value of the k th of them in the k th slot. Such an encoding is not column-indexable: its slots are budgeted per non-zero entry rather than per column, so inserting one extra non-zero entry in an early column shifts every later slot by one, and which column a given slot describes depends on the matrix rather than on the encoding alone. Restricting the columns of $\mathbf{A}$ then does not correspond to reading off a fixed set of positions of the encoding. The obstruction is the compaction and not the (row, col, val) format itself: budgeting ν slots per column instead, as Basilisk [23], restores column-indexability, at the cost of an encoding with one column per column of $\mathbf{A}$ rather than one slot per non-zero entry. We prove this in Lemma 4.

A lookup argument only guarantees that each restricted entry is *some* entry of the original table; it does not say which one. We discuss below, why this is

already enough for the witness coordinates and what additionally has to be done for the public-input and constant coordinates.

To ensure that the prover does not remove any of the public I/O columns or the constant column, we use the mask vectors

$$\mathbf{mask}_N = (1, \dots, m, \underbrace{0, \dots, 0}_N, m+1), \quad \mathbf{mask}_n = (1, \dots, m, \underbrace{0, \dots, 0}_n, m+1).$$

The mask value r uniquely identifies the r th public-input column, the value $m+1$ identifies the constant column, and zero masks the witness columns. The source mask vector is preprocessed exactly like any other table and is not circuit specific.

Why a lookup argument is enough, except on public I/O. Knowledge soundness of a lookup argument only extracts an index vector $I = (i_1, \dots, i_n)$ such that each restricted entry equals the original table entry at position i_j ; it need not be injective, and two positions $j \neq j'$ may extract the same index $i_j = i_{j'}$. For a witness coordinate this is harmless: since $\mathbf{T}_i \cdot \mathbf{z}$ is linear in $\mathbf{z}$, if several restricted-witness entries w'_j trace back to the same original coordinate w_i (i.e. $i_j = i$ for every such j), setting $w_i := \sum_{j:i_j=i} w'_j$ reproduces $\mathbf{T}_i \cdot \mathbf{z}$ exactly, so the extracted witness remains valid however the index vector folds coordinates together.

This slack cannot be tolerated on the public-input and constant coordinates of $\mathbf{z}$, whose values the verifier supplies directly at fixed positions: the reduction must guarantee that the r th restricted public-input column is genuinely the r th original public-input column, not an index vector that merges or permutes two such columns while still satisfying the linear equation. A lookup argument alone does not enforce this. Our fix is to include the public-I/O mask pair $(\mathbf{mask}_N, \mathbf{mask}_n)$ as one further table/query pair inside the same common-index batch (Definition 4), so that every table/query pair – the matrices and the mask alike – is certified against the *same* extracted index vector I . Because $\mathbf{mask}_N$ assigns a value unique to each public-input column and to the constant column, and the value 0, shared by every witness column, the shared index vector I cannot map a public-input or constant position to any other column: it is forced to reproduce the identity there. On the witness columns, where the mask value is repeated, I remains free to fold coordinates together, exactly where the summation argument above shows this is safe.

Theorem 3. *If $\mathcal{E}$ is column-indexable and Π_{LU} is a lookup argument as in Definition 4 such that the prover is linear in the size of the query, then the protocol in Figure 1 is a key-transforming reduction of knowledge from $\text{SCCS}(M, N)$ with a witness $\mathbf{w}$ whose support is contained in $J \subseteq [N]$ to $\text{CCCS}(M, n)$, for $n = |J|$. Its online reduction cost is linear in the size of the restricted encoding and is independent of N ; the reduction does not change the row dimension M .*

Proof. We now prove that the construction is a key-transforming reduction of knowledge. Completeness follows directly from the construction. Let $(\mathbf{pp}, \mathbf{s}_1, u_1, w_1) \in$

$\mathcal{R}_1$. Then $w_j = 0$ for all $j \notin J$, and therefore, for every $i \in [t]$,

$$\mathbf{T}_i \cdot (\mathbf{x}, \mathbf{w}, 1) = \mathbf{T}'_i \cdot (\mathbf{x}, \mathbf{w}', 1).$$

Hence the CCS equation for $\mathcal{R}_1$ is equal to the CCS equation for the restricted instance. By completeness of the lookup argument, the verifier accepts the restriction proofs. The prover and verifier output the same instance u_2 , the keys satisfy $(\mathbf{pk}_2, \mathbf{vk}_2) = \mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2)$ by construction, and $(\mathbf{pp}, \mathbf{s}_2, u_2, w_2) \in \mathcal{R}_2$.

For knowledge soundness, consider an adversary $\mathcal{A}$ that outputs π_{LU} that makes the verifier output an accepting tuple for $\mathcal{R}_2$. By knowledge soundness of the common-index batched lookup argument, the extractor obtains a common index vector $I = (i_1, \dots, i_n)$ and witnesses showing that the commitments $\{\overline{\mathbf{T}'_i}\}_{i \in [t]}$ are column restrictions of $\{\overline{\mathbf{T}_i}\}_{i \in [t]}$ under I . As argued above, the public-I/O mask forces I to fix the public-input and constant columns in their prescribed output positions J^* , while possibly folding together repeated witness coordinates. From the accepting target relation, the extractor obtains $w_2 = (\{\mathbf{T}'_i\}_{i \in [t]}, \mathbf{w}')$. It builds a length- N vector $\mathbf{w}$ by setting, for every witness coordinate $i \in [N]$, $w_i := \sum_{j: i_j = m+i} w'_j$ (zero if no such j exists), and setting all remaining coordinates to zero. Since the restricted matrices are certified to be the corresponding column restrictions of the original committed matrices under I , and $\mathbf{T}_i \cdot \mathbf{z}$ is linear in $\mathbf{z}$, the equality above lifts the target CCS equation to the source CCS equation. Thus the extractor outputs a witness for $(\mathbf{pp}, \mathbf{s}_1, u_1) \in \mathcal{R}_1$.

Finally, for the **cq** instantiation in the AGM, the algebraic explanations of the input and query commitments determine their polynomial encodings, and the common-index lookup argument identifies a compatible column restriction. The decoder φ reads the reduced size and query commitments from the transcript and returns $\mathbf{s}_2 = (M, n, S, c, \mathcal{E})$ and $u_2 = (\mathbf{x}, \{\overline{\mathbf{T}'_i}\}_{i \in [t]})$. The decoded instance is exactly the verifier's output instance, and the output verification key agrees with $\mathcal{K}_2$ for the decoded structure, as specified in Figure 1. The algebraic explanations are used only in the security analysis and do not add messages to the protocol. $\square$

5.1 Efficiency

The Prover time is the time it takes to commit to the encodings of matrices $\mathbf{T}'_i$, fold the preprocessed keys at the queried positions, and run one LU prover. If there are s encoded table rows in total (including the public-I/O mask), key folding costs $O(sn)$ and is independent of the ambient column dimension N . Any dependence of s or of the target argument on the unchanged row dimension M is accounted for by the particular encoding and target argument. The verifier time is that of the LU verifier. The proof size is the size of π_{LU} .

5.2 Instantiating with cq

In Appendix B we describe **cq**, a lookup argument introduced in [11]. We show **cq** satisfies Definition 4 and thus instantiates our reduction in Theorem 3.

Let $\mathcal{R}_1 = \text{SCCCS}(M, N)$ and $\mathcal{R}_2 = \text{CCCS}(M, n)$, where $n = |J|$.

- $\mathcal{G}(\lambda, M, N) \rightarrow (\text{pp}, \kappa)$: Output public parameters pp and any trapdoor κ required by the key encoders for the commitment scheme and the lookup argument, including the reusable preprocessing of the public-I/O mask mask_N . For a universal or transparent encoder, set $\kappa = \perp$.
- $\mathcal{K}_1(\text{pp}, \text{s}_1; \kappa) \rightarrow (\text{pk}_1, \text{vk}_1)$: Parse $\text{s}_1 = (M, N, S, c, \mathcal{E})$ and, using κ , output the corresponding prover and verifier keys $(\text{pk}_1, \text{vk}_1)$.
- $\mathcal{K}_2(\text{pp}, \text{s}_2; \kappa) \rightarrow (\text{pk}_2, \text{vk}_2)$: Parse $\text{s}_2 = (M, n, S, c, \mathcal{E})$ and, using the same κ , output the corresponding prover and verifier keys $(\text{pk}_2, \text{vk}_2)$.
- $\mathcal{P}((\text{pk}_1, \text{s}_1), u_1, w_1) \rightarrow ((\text{pk}_2, \text{s}_2), u_2, w_2)$:
 - Parse

$$(M, N, S, c, \mathcal{E}) \leftarrow \text{s}_1, \quad (\text{x}, \{\overline{\mathbf{T}}_i\}_{i \in [t]}) \leftarrow u_1, \quad (\{\mathbf{T}_i\}_{i \in [t]}, \mathbf{w}, J) \leftarrow w_1.$$

- Let $n = |J|$ and define

$$J^* = [m] \cup \{m + j : j \in J\} \cup \{m + N + 1\}.$$

- For every $i \in [t]$, compute the restricted matrix $\mathbf{T}'_i = (\mathbf{T}_i)_{J^*}$ and the commitment

$$\overline{\mathbf{T}}'_i = \text{commit}(\mathcal{E}(\mathbf{T}'_i)).$$

Compute the restricted witness $\mathbf{w}' = (w_j)_{j \in J}$.

- After fixing all query commitments, run one common-index batched lookup-argument proof π_{LU} for every row pair $(\mathcal{E}(\mathbf{T}_i), \mathcal{E}(\mathbf{T}'_i))$ and the additional pair $(\text{mask}_N, \text{mask}_n)$. mask_n is computed by the verifier given the size of the restricted witness n .
- Define

$$\text{s}_2 = (M, n, S, c, \mathcal{E}), \quad u_2 = (\text{x}, \{\overline{\mathbf{T}}'_i\}_{i \in [t]}), \quad w_2 = (\{\mathbf{T}'_i\}_{i \in [t]}, \mathbf{w}').$$

Let $(\text{pk}_2, \text{vk}_2) = \mathcal{K}_2(\text{pp}, \text{s}_2)$, send π_{LU} to the verifier and output $(\text{pk}_2, \text{s}_2, u_2, w_2)$.

- $\mathcal{V}(\text{vk}_1, u_1, \pi_{\text{LU}}) \rightarrow (\text{vk}_2, u_2)$:
 - Parse $\text{s}_1 = (M, N, S, c, \mathcal{E})$ from vk_1 and $u_1 = (\text{x}, \{\overline{\mathbf{T}}_i\}_{i \in [t]})$.
 - Verify the single common-index batched lookup-argument proof for all rows of $\{\mathcal{E}(\mathbf{T}_i)\}_{i \in [t]}$ and for the public-I/O mask instance $(\text{mask}_N, \text{mask}_n)$. Reject unless the proof accepts.
 - If the lookup checks accept, define

$$\text{s}_2 = (M, n, S, c, \mathcal{E}), \quad u_2 = (\text{x}, \{\overline{\mathbf{T}}'_i\}_{i \in [t]}).$$

Let $(\text{pk}_2, \text{vk}_2) = \mathcal{K}_2(\text{pp}, \text{s}_2)$ and output (vk_2, u_2) .

Fig. 1: Reduction from $\mathcal{R}_1 = \text{SCCCS}(M, N)$ to $\mathcal{R}_2 = \text{CCCS}(M, n)$

6 The Sparse SNARK Construction

In this section, we instantiate Theorem 3 with Basilisk [23] and the **cq** [11] lookup argument. We refer the reader to Appendix C for a detailed Basilisk protocol.

Why Basilisk. Theorem 3 applies to any column-indexable encoding, and we know that Groth16’s is by Lemma 3, so instantiating the framework with $\mathcal{E}_{G16}$ and **cq** already yields a partially sparse Groth16. The reduction restricts the Groth16 CRS to the n wires on which the witness is supported, the restricted key pk_2 is a genuine Groth16 CRS for a circuit with n wires and M constraints, and the prover finishes by running the ordinary Groth16 prover and sending its three group elements. Offline preprocessing is $O(N \log N)$, and the lookup proof costs $O(n \log n)$ field and $O(n)$ group operations, independently of N .

Unfortunately, Groth16’s prover must still enforce every constraint in full as it computes and commits to a quotient polynomial of degree $M - 2$, at a cost of $O(M \log M)$ field and $O(M + n)$ group operations. The reduction therefore removes the N -dependence of the wire-indexed work but leaves an M -dependence, which is no gain when $M = \Theta(N)$. Groth16 also needs a circuit-specific trapdoor, so the resulting SRS is neither universal nor updatable. Basilisk avoids both problems: under a bounded fan-out its prover is determined by the size of the witness rather than by the number of constraints, and its encoder needs no trapdoor, so the construction below inherits a universal and updatable SRS.

Lemma 4 (Basilisk’s encoder is column-indexable). *The encoding $\mathcal{E}_{\text{Bas}} : \mathbb{F}^{2m \times K} \rightarrow \mathbb{F}^{(3\nu+1) \times K}$ in Construction 4 is column-indexable.*

Proof. Recall that on input $\mathbf{C} = ((\mathbf{C}^1)^\top, (\mathbf{C}^2)^\top)^\top$, $\mathcal{E}_{\text{Bas}}$ output has rows indexed by $(\hat{R}, r, j)$ for $r \in [2]$ and $j \in \{0, \dots, \nu - 1\}$, and by $(\hat{I}, j)$ for $j \in \{0, \dots, \nu\}$, with

$$\mathcal{E}_{\text{Bas}}(\mathbf{C})_{(\hat{R}, r, j), \ell} = \hat{R}_{\ell j}^r, \quad \mathcal{E}_{\text{Bas}}(\mathbf{C})_{(\hat{I}, j), \ell} = \hat{I}_{\ell j}.$$

For a column $\mathbf{c} \in \mathbb{F}^{2m}$ and $r \in [2]$, set

$$T^r(\mathbf{c}) = \{q \in [m] : c_{(r-1)m+q} \neq 0\},$$

$$T(\mathbf{c}) = T^1(\mathbf{c}) \cup T^2(\mathbf{c}).$$

Define the column functions, for $r \in [2]$, $j \in \{0, \dots, \nu - 1\}$, and $j' \in \{0, \dots, \nu\}$, by

$$\mathcal{S}_j^{\hat{R}, r}(\mathbf{c}) = [Y^j] \left(\frac{1}{m} \sum_{q \in T^r(\mathbf{c})} c_{(r-1)m+q} \mathbf{h}_q \prod_{s \in T(\mathbf{c}) \setminus \{q\}} (Y - \mathbf{h}_s) \right),$$

$$\mathcal{S}_{j'}^{\hat{I}}(\mathbf{c}) = [Y^{j'}] \prod_{s \in T(\mathbf{c})} (Y - \mathbf{h}_s),$$

where $[Y^j]f(Y)$ denotes the coefficient of Y^j in f . For the ℓ th column $\mathbf{c}_\ell$ of $((\mathbf{C}^1)^\top, (\mathbf{C}^2)^\top)^\top$, these functions return exactly $\hat{R}_{\ell j}^r$ and $\hat{I}_{\ell j'}$ in the encoder below. Every entry of $\mathcal{E}_{\text{Bas}}$ therefore depends only on its corresponding input column, as required. $\square$

6.1 Setup

Let

$$\mathbf{W} = \begin{pmatrix} \mathbf{A} & \mathbf{0} & -\mathbf{C}^1 \\ \mathbf{0} & \mathbf{B} & -\mathbf{C}^2 \end{pmatrix} \in \mathbb{F}^{2m \times 3K}$$

be the rectangular $\text{R1CS}_{\text{lite}}$ matrix from Appendix C. The first l coordinates contain the public input. Write $\mathbf{C} = ((\mathbf{C}^1)^\top, (\mathbf{C}^2)^\top)^\top$ and

$$s_{\text{Bas}} = 3\nu + 1, \quad \mathcal{E}_{\text{Bas}}(\mathbf{C}) = (\mathbf{t}^{(1)}, \dots, \mathbf{t}^{(s_{\text{Bas}})}) \in \mathbb{F}^{s_{\text{Bas}} \times K}$$

for the row vectors defined in Lemma 4.

The universal setup produces the KZG SRS of degree $d \geq 2K - 3$, shared by Basilisk and cq , together with the Lagrange-form SRS for the supported target domains. Circuit-specific indexing computes these rows and the public-I/O mask

$$\mathbf{t}^{(\text{mask})} = (1, \dots, l, 0, \dots, 0) \in \mathbb{F}^K.$$

For $a \in [s_{\text{Bas}}] \cup \{\text{mask}\}$, it computes

$$(\text{pk}_{\text{cq}}^{(a)}, \text{vk}_{\text{cq}}^{(a)}) \leftarrow \text{cq}.\mathcal{K}(\text{pp}, \mathbf{t}^{(a)}), \quad \overline{\mathbf{t}^{(a)}} = ([T^{(a)}(x)]_1, [T^{(a)}(x)]_2),$$

for $T^{(a)}(X) = \sum_{\ell \in [K]} t_\ell^{(a)} \mu_\ell(X)$. For $a \in [s_{\text{Bas}}]$ both components are output by $\mathcal{K}(\text{pp}, \mathbf{W})$: the $\mathbb{G}_1$ components are vk_{Bas} , and the $\mathbb{G}_2$ components are exactly the table commitments $\overline{\mathbf{t}}$ that $\text{cq}.\mathcal{K}$ requires, so they are not computed twice. The mask row is not part of Basilisk's encoding and is used only as a cq table, so only its $\mathbb{G}_2$ component is needed.

6.2 Applying Theorem 3

Write $\mathbf{a} = (x, \mathbf{a}')$ and $\mathbf{b} = (1, \mathbf{b}')$. Let $J = \text{supp}(\mathbf{a}') \cup \text{supp}(\mathbf{b}')$ and set

$$J^\star = [l] \cup \{l + j : j \in J\}, \quad n = |J|.$$

Thus the reduced domain has size $l + n$. Let $\mathbb{H}_{l+n}$ be the target domain and, for $a \in [s_{\text{Bas}}]$, let

$$\mathbf{f}^{(a)} = (t_i^{(a)})_{i \in J^\star}, \quad \mathbf{f}^{(\text{mask})} = (1, \dots, l, 0, \dots, 0) \in \mathbb{F}^{l+n},$$

and define

$$v'^{(a)}(X) = \sum_{j=1}^{l+n} f_j^{(a)} \lambda_j^{\mathbb{H}_{l+n}}(X).$$

- $\mathcal{G}(\lambda, K) \rightarrow (\text{pp}, \perp)$: Generate the universal SRS above; the encoder requires no circuit-specific trapdoor.
- $\mathcal{K}_1(\text{pp}, \mathbf{s}_1; \perp) \rightarrow (\text{pk}_1, \text{vk}_1)$: Run Basilisk's indexer $\mathcal{K}(\text{pp}, \mathbf{W})$ to obtain $(\text{pk}_{\text{Bas}}, \text{vk}_{\text{Bas}})$, and set

$$\begin{aligned} \text{pk}_1 &= \left(\text{pk}_{\text{Bas}}, \{\mathbf{t}^{(a)}, \text{pk}_{\text{cq}}^{(a)}\}_{a \in [s_{\text{Bas}}] \cup \{\text{mask}\}} \right), \\ \text{vk}_1 &= \left(\text{vk}_{\text{Bas}}, \{\overline{\mathbf{t}^{(a)}}, \text{vk}_{\text{cq}}^{(a)}\}_{a \in [s_{\text{Bas}}] \cup \{\text{mask}\}} \right). \end{aligned}$$

– $\mathcal{K}_2(\mathbf{pp}, \mathbf{s}_2; \perp) \rightarrow (\mathbf{pk}_2, \mathbf{vk}_2)$: Define

$$\begin{aligned}\mathbf{W}' &= \begin{pmatrix} \mathbf{A}_{J^*} & \mathbf{0} & -\mathbf{C}_{J^*}^1 \\ \mathbf{0} & \mathbf{B}_{J^*} & -\mathbf{C}_{J^*}^2 \end{pmatrix}, \\ \mathbf{pk}_2 &= \left(\mathbf{W}', \{v'^{(a)}\}_{a \in [s_{\text{Bas}}]} \right), \\ \mathbf{vk}_2 &= \{[v'^{(a)}(x)]_1\}_{a \in [s_{\text{Bas}}]}.\end{aligned}$$

The public-input Lagrange commitments for $\mathbb{H}_{l+n}$ are read from $\mathbf{pp}$.

– $\mathcal{P}((\mathbf{pk}_1, \mathbf{s}_1), u_1, w_1) \rightarrow (\mathbf{pk}_2, \mathbf{s}_2, u_2, w_2)$: For $a \in [s_{\text{Bas}}] \cup \{\text{mask}\}$, compute

$$\overline{\mathbf{f}^{(a)}} = [v'^{(a)}(x)]_1.$$

After fixing these commitments, the verifier samples $\theta \xleftarrow{\$} \mathbb{F}$, and the parties set

$$\begin{aligned}\mathbf{t}^* &= \sum_{a=1}^{s_{\text{Bas}}} \theta^{a-1} \mathbf{t}^{(a)} + \theta^{s_{\text{Bas}}} \mathbf{t}^{(\text{mask})}, \\ \mathbf{f}^* &= \sum_{a=1}^{s_{\text{Bas}}} \theta^{a-1} \mathbf{f}^{(a)} + \theta^{s_{\text{Bas}}} \mathbf{f}^{(\text{mask})}.\end{aligned}$$

Fold the corresponding preprocessing keys and run one **cq** proof for $(\mathbf{t}^*, \mathbf{f}^*)$.
Output

$$u_2 = \mathbf{x}, \quad w_2 = (\mathbf{a}_{J^*}, \mathbf{b}_{J^*}),$$

together with $(\mathbf{pk}_2, \mathbf{s}_2)$ for $\mathbf{W}'$.

– $\mathcal{V}(\mathbf{vk}_1, u_1, \pi_{\text{LU}}) \rightarrow (\mathbf{vk}_2, u_2)$: Verify the folded **cq** proof, reading the $\mathbb{G}_2$ components of $\{\overline{\mathbf{t}^{(a)}}\}$, and output

$$\mathbf{vk}_2 = \{\overline{\mathbf{f}^{(a)}}\}_{a \in [s_{\text{Bas}}]} = \{[v'^{(a)}(x)]_1\}_{a \in [s_{\text{Bas}}]}, \quad u_2 = \mathbf{x}.$$

The mask fixes the first l positions, so no public-input column can be deleted or moved.

The parties then run Construction 4 on $(\mathbf{pk}_2, \mathbf{vk}_2, u_2, w_2)$.

Theorem 4 (Sparse Basilisk). *Assume the knowledge soundness of **cq** and of Construction 4. The construction above is a sparse argument of knowledge with a universal and updatable SRS. Its online prover time is independent of the ambient dimension K .*

Proof. This follows from Theorem 3, Lemma 4, Lemma 5, and the knowledge soundness of Basilisk. $\square$

6.3 Efficiency

- **Universal SRS:** $O(K)$ group elements, independent of the circuit and updatable.
- **Circuit preprocessing:** $O(\nu K \log K)$ field operations, $3\nu + 1$ size- K MSMs in each of $\mathbb{G}_1$ and $\mathbb{G}_2$ for Basilisk’s indexer (the $\mathbb{G}_2$ ones being the **cq** table commitments), one size- K $\mathbb{G}_2$ MSM for the mask, and $3\nu + 2$ **cq** table preprocessings.
- **Online proving:** $O((l + n) \log(l + n) + \nu(l + n))$ field operations; $3\nu + 2$ MSMs of size at most $l + n$ for the query commitments; $l + n$ MSMs of size $3\nu + 2$ for folding the table-dependent prover keys; seven **cq** MSMs (six of size at most $l + n$ and one of size at most $2(l + n)$); and eight Basilisk MSMs (six of size at most $l + n$, one of size at most $2(l + n) - 2$, and one of size at most $2(l + n) - 3$).
- **Verification:** Three MSMs of size $3\nu + 2$ to fold the **cq** verifier data, one size- l public-input MSM, two Basilisk MSMs with $O(\nu)$ terms, and thirteen pairings: eight for **cq** and five for Basilisk.
- **Proof size:** The **cq** and Basilisk proofs contain fifteen $\mathbb{G}_1$ elements and five field elements. The reduction also outputs the $3\nu + 1$ query commitments forming vk_2 (and the mask query commitment if it is not derived from **pp**).

References

- [1] Arun, A., Setty, S.T.V.: Nebula: Proving machine executions via folding schemes. In: 2026 IEEE Symposium on Security and Privacy. pp. 3223–3242. IEEE Computer Society Press (May 2026). <https://doi.org/10.1109/SP63933.2026.00099>
- [2] Arun, A., Setty, S.T.V., Thaler, J.: Jolt: SNARKs for virtual machines via lookups. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 3–33. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_1
- [3] Batista, M., Montes, Á., Paslis, N., Ràfols, C.: Dynasaurs: Efficient universal dynamic zkSNARKs from sparse linear arguments. Cryptology ePrint Archive, Paper 2026/1892 (2026), <https://eprint.iacr.org/2026/1892>
- [4] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). <https://doi.org/10.1109/SP.2014.36>
- [5] Bünz, B., Chen, B.: Protostar: Generic efficient accumulation/folding for special-sound protocols. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part II. LNCS, vol. 14439, pp. 77–110. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8724-5_3
- [6] Campanelli, M., Faonio, A., Fiore, D., Li, T., Lipmaa, H.: Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part II. LNCS, vol. 14602, pp. 337–369. Springer, Cham (Apr 2024). https://doi.org/10.1007/978-3-031-57722-2_11
- [7] Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Berlin, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_5
- [8] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45721-1_26
- [9] Choudhuri, A.R., Garg, S., Goel, A., Sekar, S., Sinha, R.: SublonK: Sublinear prover PlonK. PoPETs **2024**(3), 314–335 (Jul 2024). <https://doi.org/10.56553/popets-2024-0080>
- [10] Di, Z., Xia, L., Nguyen, W.D., Tyagi, N.: MuxProofs: Succinct arguments for machine computation from vector lookups. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part V. LNCS, vol. 15488, pp. 236–265. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0935-2_8

- [11] Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763 (2022), <https://eprint.iacr.org/2022/1763>
- [12] Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
- [13] Gabizon, A., Williamson, Z.J.: The Turbo-PLONK program syntax for specifying SNARK programs. In: Proceedings of the 2019 ZKProof Workshop (2019), <https://docs.zkproof.org/pages/standards/proposals/turbo-plonk.pdf>
- [14] Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315 (2020), <https://eprint.iacr.org/2020/315>
- [15] Goel, A., Hall-Andersen, M., Kaptchuk, G., Spooner, N.: Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 347–378. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30617-4_12
- [16] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989)
- [17] Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
- [18] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Berlin, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11
- [19] Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 669–701. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_21
- [20] Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758 (2022), <https://eprint.iacr.org/2022/1758>
- [21] Ozdemir, A., Laufer, E., Boneh, D.: Volatile and persistent memory for zkSNARKs via algebraic interactive proofs. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) 2025 IEEE Symposium on Security and Privacy. pp. 3309–3327. IEEE Computer Society Press (May 2025). <https://doi.org/10.1109/SP61157.2025.00054>
- [22] Posen, J., Kattis, A.A.: Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957 (2022), <https://eprint.iacr.org/2022/957>
- [23] Ràfols, C., Zapico, A.: An algebraic framework for universal and updatable SNARKs. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS,

- vol. 12825, pp. 774–804. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_27
- [24] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_25
 - [25] Tzialla, I., Kothapalli, A., Parno, B., Setty, S.T.V.: Transparency dictionaries with succinct proofs of correct operation. In: NDSS 2022. The Internet Society (Apr 2022)
 - [26] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_1
 - [27] Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943. IEEE Computer Society Press (May 2018). <https://doi.org/10.1109/SP.2018.00060>
 - [28] Wang, W., Papamanthou, C., Srinivasan, S., Papadopoulos, D.: Designated-verifier dynamic zk-SNARKs with applications to dynamic proofs of index. Cryptology ePrint Archive, Paper 2026/144 (2026), <https://eprint.iacr.org/2026/144>
 - [29] Wang, W., Papamanthou, C., Srinivasan, S., Papadopoulos, D.: Dynamic zk-SNARKs (with applications to sparse zk-SNARKs and IVC). In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part VII. LNCS, vol. 16547, pp. 329–358. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25336-1_12
 - [30] Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press (Nov 2021). <https://doi.org/10.1145/3460120.3484556>
 - [31] Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: Lookup arguments in sublinear time. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 3121–3134. ACM Press (Nov 2022). <https://doi.org/10.1145/3548606.3560646>
 - [32] Zapico, A., Gabizon, A., Khovratovich, D., Maller, M., Ràfols, C.: Baloo: Algebraic lookup arguments. CiC **2**(2), 1 (2025). <https://doi.org/10.62056/ae890l5vt>
 - [33] Zhang, T., Ouyang, Y., Zhang, Y.: Dynark: Making Groth16 dynamic. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part VII. LNCS, vol. 16547, pp. 178–208. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25336-1_7
 - [34] Zheng, T., Gao, S., Guo, Y., Xiao, B.: KiloNova: Non-uniform PCD with zero-knowledge property from generic folding schemes. Cryptology ePrint Archive, Report 2023/1579 (2023), <https://eprint.iacr.org/2023/1579>

A Standard Reductions of Knowledge

Definition 18 (Reduction of Knowledge [19]). Consider relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance, and witness tuples. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic algorithm $\mathcal{K}$, called the generator, the prover, the verifier and the encoder respectively with the following interface.

- $\mathcal{G}(\lambda, n) \rightarrow \text{pp}$: Takes as input security parameter λ , size parameters n . Outputs the public parameters pp .
- $\mathcal{K}(\text{pp}, \mathbf{s}) \rightarrow (\text{pk}, \text{vk})$: Takes as input public parameters pp and structure $\mathbf{s}$. Outputs prover key pk and verifier key vk .
- $\mathcal{P}(\text{pk}, u_1, w_1) \rightarrow (u_2, w_2)$: Takes as input the prover key pk , and an instance-witness pair (u_1, w_1) . Interactively reduces the task of checking $(\text{pp}, \mathbf{s}, u_1, w_1) \in \mathcal{R}_1$ to the task of checking $(\text{pp}, \mathbf{s}, u_2, w_2) \in \mathcal{R}_2$.
- $\mathcal{V}(\text{vk}, u_1) \rightarrow u_2$: Takes as input public parameters pp , and an instance u_1 in $\mathcal{R}_1$. Interactively reduces the task of checking instance u_1 to the task of checking a new instance u_2 in $\mathcal{R}_2$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. We treat $\langle \mathcal{P}, \mathcal{V} \rangle$ as a function that takes as input $((\text{pk}, \text{vk}), u_1, w_1)$ and runs the interaction on the prover's input (pk, u_1, w_1) and the verifier's input (pp, u_1) . At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the verifier's instance u_2 and the prover's witness w_2 . A reduction of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies the following conditions.

- (i) *Completeness*: For any PPT adversary $\mathcal{A}$, given $\text{pp} \leftarrow \mathcal{G}(\lambda, n, \mathbf{s})$, $(\text{pk}, \text{vk}) \leftarrow \mathcal{K}(\text{pp}, \mathbf{s})$, and $(u_1, w_1) \leftarrow \mathcal{A}(\text{pp})$ such that $(\text{pp}, \mathbf{s}, u_1, w_1) \in \mathcal{R}_1$ we have that the prover's output instance is equal to the verifier's output instance u_2 , and that

$$(\text{pp}, \mathbf{s}, \langle \mathcal{P}, \mathcal{V} \rangle((\text{pk}, \text{vk}), u_1, w_1)) \in \mathcal{R}_2.$$

- (ii) *Knowledge soundness*: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that given $\text{pp} \leftarrow \mathcal{G}(\lambda, n, \mathbf{s})$, $(\text{pk}, \text{vk}) \leftarrow \mathcal{K}(\text{pp}, \mathbf{s})$, and $(u_1, \text{st}) \leftarrow \mathcal{A}(\text{pp})$, we have that

$$\Pr[(\text{pp}, \mathbf{s}, u_1, \mathcal{E}(\text{pp}, \mathbf{s}, u_1, \text{st})) \in \mathcal{R}_1] \approx \Pr[(\text{pp}, \mathbf{s}, \langle \mathcal{P}^*, \mathcal{V} \rangle((\text{pk}, \text{vk}), u_1, \text{st})) \in \mathcal{R}_2].$$

- (iii) *Public reducibility*: There exists a deterministic polynomial-time function φ such that for any PPT adversary $\mathcal{A}$ and expected polynomial-time adversary $\mathcal{P}^*$, given

$$\begin{aligned} \text{pp} &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathbf{s}_1, u_1, \text{st}) &\leftarrow \mathcal{A}(\text{pp}), \\ (\text{pk}, \text{vk}, \mathbf{s}_2) &\leftarrow \mathcal{K}(\text{pp}, \mathbf{s}_1), \end{aligned}$$

and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\text{pk}, \text{vk}), u_1, \text{st})$ with the interaction transcript tr , we have that $\varphi(\text{pp}, \mathbf{s}, u_1, \text{tr}) = u_2$.

B Instantiating Lookup Arguments with **cq**

cq is a lookup argument described in [11] that allows a prover to prove that every entry of a vector $\mathbf{f}$ occurs in a vector/table $\mathbf{t}$. The table has size N , the queried vector has size n , and entries of $\mathbf{f}$ may repeat. Its main property for our purposes is *table independence*: after preprocessing the table, the online prover runtime is independent of N and linear in n .

For completeness, we present the **cq** protocol from [11] below. The protocol does not provide zero knowledge as is; follow-up work by [6] shows how to make it zero knowledge without much additional prover overhead.

We first reproduce the relation instantiated here. The table is committed in $\mathbb{G}_2$ and the queried vector is committed in $\mathbb{G}_1$, where T and $\mathbf{f}$ denote the interpolations of $\mathbf{t}$ and $\mathbf{f}$ over the respective domains of Construction 3:

$$\text{cLU}_{\text{KZG}}(N, n) = \left\{ \begin{array}{ll} \text{params} & \text{pp}, \\ \text{structure} & \perp, \\ \text{instance} & (\bar{\mathbf{t}}, \bar{\mathbf{f}}), \\ \text{witness} & (\mathbf{t}, \mathbf{f}, I) \end{array} \middle| \begin{array}{l} \mathbf{t} \in \mathbb{F}^N, \quad \mathbf{f} \in \mathbb{F}^n, \\ I = (i_1, \dots, i_n) \in [N]^n, \\ f_j = t_{i_j} \quad \forall j \in [n], \\ \bar{\mathbf{t}} = [T(x)]_2, \\ \bar{\mathbf{f}} = [f(x)]_1 \end{array} \right\}.$$

Construction 3 (Lookup Argument). Given pairing-friendly groups $\mathbb{G}_1, \mathbb{G}_2$, a maximum SRS degree d , and multiplicative subgroups $\mathbb{V} = \{\mathbf{k}_i\}_{i=1}^N$ and $\mathbb{H} = \{\mathbf{h}_j\}_{j=1}^n$ of $\mathbb{F}^*$, of sizes $N < d$ and n , respectively, we construct an argument of knowledge for $\text{cLU}_{\text{KZG}}(N, n)$.

$\mathcal{G}(d)$:

1. Choose random $x \in \mathbb{F}$ and output

$$\text{pp} = (\{[x^i]_1\}_{i=0}^d, \{[x^i]_2\}_{i=0}^d).$$

$\mathcal{K}(\text{pp}, \mathbf{t})$:

1. Compute $Z_{\mathbb{V}}(X) = \prod_{i \in [N]} (X - \mathbf{k}_i)$ and $[Z_{\mathbb{V}}(x)]_2$.
2. Compute

$$T(X) = \sum_{i \in [N]} t_i \lambda_{\mathbf{k}_i}^{\mathbb{V}}(X), \quad \bar{\mathbf{t}} = [T(x)]_2, \quad [xT(x)]_2.$$

3. For every $i \in [N]$, compute
 - (a) $q_i = [Q_i(x)]_1$ such that

$$\lambda_{\mathbf{k}_i}^{\mathbb{V}}(X)T(X) = t_i \lambda_{\mathbf{k}_i}^{\mathbb{V}}(X) + Z_{\mathbb{V}}(X)Q_i(X);$$

- (b) $[\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x)]_1$;
- (c) $[(\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x) - \lambda_{\mathbf{k}_i}^{\mathbb{V}}(0))/x]_1$.
- (d) $[x^{d-(N-2)}(\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x) - \lambda_{\mathbf{k}_i}^{\mathbb{V}}(0))/x]_1$.

4. Output

$$\begin{aligned} \mathbf{pk} &= \left(\{q_i\}_{i \in [N]}, \{[\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x)]_1\}_{i \in [N]}, \left\{ \left[\frac{\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x) - \lambda_{\mathbf{k}_i}^{\mathbb{V}}(0)}{x} \right]_1 \right\}_{i \in [N]}, \right. \\ &\quad \left. \left\{ \left[x^{d-(N-2)} \frac{\lambda_{\mathbf{k}_i}^{\mathbb{V}}(x) - \lambda_{\mathbf{k}_i}^{\mathbb{V}}(0)}{x} \right]_1 \right\}_{i \in [N]} \right), \\ \mathbf{vk} &= ([Z_{\mathbb{V}}(x)]_2, \bar{\mathbf{t}}, [xT(x)]_2). \end{aligned}$$

$\langle \mathcal{P}, \mathcal{V} \rangle((\mathbf{pk}, \mathbf{vk}), (\bar{\mathbf{t}}, \bar{\mathbf{f}}), (\mathbf{t}, \mathbf{f}, I))$: Let $f(X) \in \mathbb{F}_{<n}[X]$ interpolate $\mathbf{f}$ over $\mathbb{H}$, with $\bar{\mathbf{f}} = [f(x)]_1$.

1. $\mathcal{P}$: Compute $m_i = |\{j \in [n] : i_j = i\}|$ for every $i \in [N]$, and send $\mathbf{m} = [m(x)]_1$.
2. $\mathcal{V}$: Send random $\beta \in \mathbb{F}$.
3. $\mathcal{P}$: Compute and send $\mathbf{a}_0, \mathbf{q}_a, \mathbf{b}_0, \mathbf{q}_b$:

$$\begin{aligned} A(X) &\in \mathbb{F}_{<N}[X] & \text{s.t. } A(\mathbf{k}_i) &= \frac{m_i}{t_i + \beta} \quad \forall i \in [N], \\ Q_A(X) &\text{ s.t. } A(X)(T(X) + \beta) - m(X) = Q_A(X)Z_{\mathbb{V}}(X), & \mathbf{q}_a &= [Q_A(x)]_1 = \sum_{i \in [N]} A(\mathbf{k}_i)q_i, \\ A_0(X) &= \frac{A(X) - A(0)}{X}, & \mathbf{a}_0 &= [A_0(x)]_1, \\ B(X) &\in \mathbb{F}_{<n}[X] & \text{s.t. } B(\mathbf{h}_j) &= \frac{1}{f(\mathbf{h}_j) + \beta} \quad \forall j \in [n], \\ B_0(X) &= \frac{B(X) - B(0)}{X}, & \mathbf{b}_0 &= [B_0(x)]_1, \\ Q_B(X) &\text{ s.t. } B(X)(f(X) + \beta) - 1 = Q_B(X)Z_{\mathbb{H}}(X), & \mathbf{q}_b &= [Q_B(x)]_1. \end{aligned}$$

4. $\mathcal{V}$: Send random $\rho \in \mathbb{F}$.
5. $\mathcal{P}$: Compute and send $\mathbf{p}_{\text{deg}} = [P_{\text{deg}}(x)]_1$ for

$$P_{\text{deg}}(X) = X^{d-(N-2)}A_0(X) + \rho X^{d-(n-2)}B_0(X) + \rho^2 X^{d-(n-1)}f(X).$$

6. $\mathcal{V}$: Check

$$\begin{aligned} e(\mathbf{a}_0, [x^{d-(N-2)}]_2) \cdot e(\rho \mathbf{b}_0, [x^{d-(n-2)}]_2) \cdot e(\rho^2 \bar{\mathbf{f}}, [x^{d-(n-1)}]_2) \\ = e(\mathbf{p}_{\text{deg}}, [1]_2). \end{aligned}$$

7. $\mathcal{V}$: Send random $\gamma \in \mathbb{F}$.
8. $\mathcal{P}$: Send $b_{0,\gamma} = B_0(\gamma)$, $f_{\gamma} = f(\gamma)$, and $a_0 = A(0)$.
9. $\mathcal{V}$: Compute

$$b_0 = \frac{Na_0}{n}, \quad b_{\gamma} = \gamma b_{0,\gamma} + b_0, \quad Q_{b,\gamma} = \frac{b_{\gamma}(f_{\gamma} + \beta) - 1}{Z_{\mathbb{H}}(\gamma)},$$

and check

$$\begin{aligned} e(\mathbf{a}_0, [xT(x)]_2 + \beta[x]_2) \cdot e([a_0]_1, \bar{\mathbf{t}} + \beta[1]_2) \\ = e(\mathbf{q}_a, [Z_V(x)]_2) \cdot e(\mathbf{m}, [1]_2). \end{aligned}$$

10. $\mathcal{V}$: Send random $\eta \in \mathbb{F}$.
11. $\mathcal{P}, \mathcal{V}$: Compute $f^* = b_{0,\gamma} + \eta f_\gamma + \eta^2 Q_{b,\gamma}$.
12. $\mathcal{P}$: Send $\pi_\gamma = [h(x)]_1$ for

$$h(X) = \frac{B_0(X) + \eta f(X) + \eta^2 Q_B(X) - f^*}{X - \gamma}.$$

13. $\mathcal{V}$: Compute $\mathbf{c} = \mathbf{b}_0 + \eta \bar{\mathbf{f}} + \eta^2 \mathbf{q}_b$ and check

$$e(\mathbf{c} - [f^*]_1 + \gamma \pi_\gamma, [1]_2) = e(\pi_\gamma, [x]_2).$$

Theorem 5. *Construction 3 is an argument of knowledge for $\text{cLU}_{\text{KZG}}(N, n)$ in the algebraic group model with the following efficiency guarantees:*

- $\mathcal{G}(d)$ takes $O(d)$ operations in each of $\mathbb{G}_1$ and $\mathbb{G}_2$.
- $\mathcal{K}(\mathbf{pp}, \mathbf{t})$ takes $O(N \log N)$ field and $\mathbb{G}_1$ operations and $O(N)$ $\mathbb{G}_2$ operations.
- Prover time = $O(n \log n)$ field operations and seven MSMs in $\mathbb{G}_1$: six with at most n terms and one with at most $2n$ terms.
- Verifier time = $O(1)$ field and group operations and eight pairings (after batching).
- Proof size = seven $\mathbb{G}_1$ elements and three field elements.

Proof. Completeness follows by inspection. For knowledge soundness, since the prover is algebraic, along with

$$\mathbf{m}, \mathbf{a}_0, \mathbf{q}_a, \mathbf{b}_0, \mathbf{q}_b, \mathbf{p}_{\text{deg}}, \pi_\gamma,$$

it supplies the corresponding polynomials

$$m, A_0, Q_A, B_0, Q_B, P_{\text{deg}}, h \in \mathbb{F}_{<d+1}[X].$$

Except with negligible probability, the verifier checks imply

$$X^{d-(N-2)} A_0(X) + \rho X^{d-(n-2)} B_0(X) + \rho^2 X^{d-(n-1)} f(X) = P_{\text{deg}}(X), \quad \text{by Step 6,} \quad (5)$$

$$(X A_0(X) + a_0)(T(X) + \beta) = Q_A(X) Z_V(X) + m(X), \quad \text{by Step 9,} \quad (6)$$

$$B_0(X) + \eta f(X) + \eta^2 Q_B(X) - f^* = (X - \gamma) h(X), \quad \text{by Step 13.} \quad (7)$$

The polynomials A_0, B_0, f are fixed before ρ is sampled. Comparing the coefficients above degree d in (5), the randomness of ρ therefore gives, except with negligible probability,

$$\deg A_0 \leq N - 2, \quad \deg B_0 \leq n - 2, \quad \deg f \leq n - 1.$$

Define

$$A(X) = X A_0(X) + a_0, \quad b_0 = \frac{N a_0}{n}, \quad B(X) = X B_0(X) + b_0.$$

Then $\deg A < N$, $\deg B < n$, and (6) gives, for every $i \in [N]$,

$$A(\mathbf{k}_i) = \frac{m(\mathbf{k}_i)}{t_i + \beta}. \quad (8)$$

Evaluating (7) at $X = \gamma$ gives

$$(B_0(\gamma) - b_{0,\gamma}) + \eta(f(\gamma) - f_\gamma) + \eta^2(Q_B(\gamma) - Q_{b,\gamma}) = 0.$$

The three claimed values are fixed before η is sampled, so except with probability at most $2/|\mathbb{F}|$,

$$B_0(\gamma) = b_{0,\gamma}, \quad f(\gamma) = f_\gamma, \quad Q_B(\gamma) = Q_{b,\gamma}.$$

Consequently, by the verifier's definitions of b_γ and $Q_{b,\gamma}$,

$$B(\gamma)(f(\gamma) + \beta) - 1 = Q_B(\gamma)Z_{\mathbb{H}}(\gamma).$$

The committed polynomials are fixed before γ is sampled, and (6) fixes a_0 and hence B . Thus, except with negligible probability,

$$B(X)(f(X) + \beta) - 1 = Q_B(X)Z_{\mathbb{H}}(X). \quad (9)$$

Evaluating (9) on $\mathbb{H}$ yields

$$B(\mathbf{h}_j) = \frac{1}{f(\mathbf{h}_j) + \beta} \quad \forall j \in [n].$$

Using the root-of-unity identities and (8),

$$\begin{aligned} \sum_{i \in [N]} \frac{m(\mathbf{k}_i)}{t_i + \beta} &= \sum_{i \in [N]} A(\mathbf{k}_i) = N A(0) = N a_0 = n b_0 \\ &= \sum_{j \in [n]} B(\mathbf{h}_j) = \sum_{j \in [n]} \frac{1}{f(\mathbf{h}_j) + \beta}. \end{aligned}$$

Since β is sampled after m , f , and the table are fixed, [11, Lemma 2.4] implies

$$f|_{\mathbb{H}} \subseteq \mathbf{t}.$$

It remains to extract the index vector. The extractor evaluates the polynomial corresponding to $\mathbf{m}$ on $\mathbb{V}$ and sets $m_i = m(\mathbf{k}_i)$. The same rational-function identity gives, for every $y \in \mathbb{F}$,

$$\sum_{i: t_i = y} m_i = |\{j \in [n] : f(\mathbf{h}_j) = y\}|.$$

Hence, for every position $j \in [n]$, there is an $i_j \in \text{supp}(\mathbf{m})$ such that $t_{i_j} = f_j$; choose one such index and set $I = (i_1, \dots, i_n)$. Repeated positions may use the same index. Together with the extracted vectors $\mathbf{t}$ and $\mathbf{f} = f|_{\mathbb{H}}$, this gives

$$(\text{pp}, \perp, (\bar{\mathbf{t}}, \bar{\mathbf{f}}), (\mathbf{t}, \mathbf{f}, I)) \in \text{cLU}_{\text{KZG}}(N, n).$$

□

Folding and batching. To batch s table/query pairs having a common index vector, first fix all query commitments, sample $\theta \xleftarrow{\$} \mathbb{F}$, and set

$$\mathbf{t}^* = \sum_{a=1}^s \theta^{a-1} \mathbf{t}^{(a)}, \quad \mathbf{f}^* = \sum_{a=1}^s \theta^{a-1} \mathbf{f}^{(a)}.$$

The parties derive the preprocessing key and commitments for these folded vectors by the same linear combination and run one copy of Construction 3. Each base table is preprocessed only once. Online batching adds $O(sn)$ group operations for local key folding and uses one **cq** proof, so its proof size and verifier pairing count are independent of s .

Lemma 5 (Foldable preprocessing and common-index batching). *Construction 3 satisfies the foldable-preprocessing and common-index-batching requirements of Definition 4, with batching soundness error at most $nN(s-1)/|\mathbb{F}|$.*

Proof. Write $T^{(a)}$ for the polynomial interpolating $\mathbf{t}^{(a)}$. The only table-dependent prover-key elements in Construction 3 are $q_i^{(a)} = [Q_i^{(a)}(x)]_1$, where

$$\lambda_{\mathbf{k}_i}^{\mathbb{V}}(X) T^{(a)}(X) = t_i^{(a)} \lambda_{\mathbf{k}_i}^{\mathbb{V}}(X) + Z_{\mathbb{V}}(X) Q_i^{(a)}(X).$$

Both interpolation and this identity are linear in the table. Hence

$$T^* = \sum_{a=1}^s \theta^{a-1} T^{(a)}, \quad Q_i^* = \sum_{a=1}^s \theta^{a-1} Q_i^{(a)}, \quad q_i^* = \sum_{a=1}^s \theta^{a-1} q_i^{(a)}.$$

The table-dependent verifier-key elements satisfy the same property:

$$\bar{\mathbf{t}}^* = \sum_{a=1}^s \theta^{a-1} \overline{\mathbf{t}^{(a)}}, \quad [xT^*(x)]_2 = \sum_{a=1}^s \theta^{a-1} [xT^{(a)}(x)]_2.$$

All Lagrange-basis elements and $[Z_{\mathbb{V}}(x)]_2$ are table-independent and are retained once. Finally, in an honest execution $A(\mathbf{k}_i) = 0$ outside the support of the multiplicity vector. Thus the prover needs q_i^* only for the at most n queried table positions and can fold those entries in $O(sn)$ group operations, independently of N .

It remains to show that the folded lookup enforces a common index vector. By knowledge soundness of Construction 3, extract $I^* = (i_1^*, \dots, i_n^*)$ satisfying $f_j^* = t_{i_j^*}^*$ for every j . For fixed $i \in [N]$ and $j \in [n]$, define

$$p_{i,j}(Y) = \sum_{a=1}^s Y^{a-1} (f_j^{(a)} - t_i^{(a)}).$$

Because all table and query commitments are fixed before θ is sampled, this is a fixed polynomial of degree at most $s-1$. The Schwartz–Zippel lemma and a union bound over i and j show that except with probability at most $nN(s-1)/|\mathbb{F}|$, each extracted i_j^* satisfies

$$f_j^{(a)} = t_{i_j^*}^{(a)} \quad \forall a \in [s],$$

which is a witness for the common-index batched relation. $\square$

C Basilisk: A SNARK for $\mathbf{R1CS}_{\text{lite}}$ with column-indexable encoding and universal SRS

Construction 4 is the non-zero-knowledge variant of Basilisk, introduced in [23], for the following rectangular $\mathbf{R1CS}_{\text{lite}}$ instance:

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} & -\mathbf{C}^1 \\ \mathbf{0} & \mathbf{B} & -\mathbf{C}^2 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{a} \circ \mathbf{b} \end{pmatrix} = \mathbf{0},$$

where $\mathbf{W} \in \mathbb{F}^{2m \times 3K}$, $\mathbf{a}, \mathbf{b} \in \mathbb{F}^K$, and $((\mathbf{C}^1)^\top, (\mathbf{C}^2)^\top)^\top$ is a sum of at most ν simple matrices.

In Theorem 6 we prove the construction inherits Basilisk's security guarantees. In Lemma 4 we prove Basilisk's encoding is column-indexable. Thus, we instantiate Theorem 3 with Basilisk and cq (Construction 3) to obtain a universal and updatable sparse SNARK.

Construction 4 (SNARK). Let $\mathbb{H} = \{\mathbf{h}_j\}_{j=1}^m$ and $\mathbb{V} = \{\mathbf{k}_\ell\}_{\ell=1}^K$ be multiplicative subgroups of $\mathbb{F}$, with vanishing polynomials $Z_{\mathbb{H}}, Z_{\mathbb{V}}$ and Lagrange bases $\{\lambda_j\}_{j=1}^m, \{\mu_\ell\}_{\ell=1}^K$, respectively. Let ν bound the number of non-zero entries per column, let $\mathbb{V}_l = \{\mathbf{k}_1, \dots, \mathbf{k}_l\}$ index the public-input coordinates, and let $d \geq 2K - 3$.

$\mathcal{G}(d)$:

1. Choose random $x \in \mathbb{F}$ and output

$$\mathbf{pp} = (\{[x^i]_1\}_{i=0}^d, \{[x^i]_2\}_{i=0}^d, \{[\mu_\ell(x)]_1\}_{\ell \in [K]}, \{[\mu_\ell(x)]_2\}_{\ell \in [K]}).$$

The last two sets are Lagrange-form representations of the same universal KZG SRS; the $\mathbb{G}_1$ set is used to process the public input, and the $\mathbb{G}_2$ set lets the indexer commit each encoding row in $\mathbb{G}_2$ as well. Both are derived publicly from the monomial powers and introduce no circuit dependence.

$\mathcal{K}(\mathbf{pp}, \mathbf{W})$:

1. Parse

$$\mathbf{W} = \begin{pmatrix} \mathbf{A} & \mathbf{0} & -\mathbf{C}^1 \\ \mathbf{0} & \mathbf{B} & -\mathbf{C}^2 \end{pmatrix} \in \mathbb{F}^{2m \times 3K},$$

where $\mathbf{A}, \mathbf{B}$ are public column restrictions of $\mathbf{I}_m$ and $\mathbf{C}^1, \mathbf{C}^2 \in \mathbb{F}^{m \times K}$.

2. Parse

$$\begin{pmatrix} \mathbf{C}^1 \\ \mathbf{C}^2 \end{pmatrix} = \sum_{i=1}^{\nu} \begin{pmatrix} \mathbf{C}_i^1 \\ \mathbf{C}_i^2 \end{pmatrix},$$

using a fixed columnwise ordering of the non-zero entries and zero-padding to ν slots, where every summand has at most one non-zero entry per column. Let $\text{val}_i : \mathbb{V} \rightarrow \mathbb{F}$ and $\text{row}_i : \mathbb{V} \rightarrow [2m]$ be such that when evaluated in $\mathbf{k}_\ell$,

output the value and row of the non-zero element in the ℓ th column of matrix $\begin{pmatrix} \mathbf{C}_i^1 \\ \mathbf{C}_i^2 \end{pmatrix}$, assigning any fixed row to a zero-padding slot. For $r \in [2]$, define

$$V_\ell^r = \{i \in [\nu] : \text{val}_i(\mathbf{k}_\ell) \neq 0, (r-1)m < \text{row}_i(\mathbf{k}_\ell) \leq rm\},$$

$$S_\ell = \{\text{row}_i(\mathbf{k}_\ell) - (r-1)m : r \in [2], i \in V_\ell^r\}.$$

That is, V_ℓ^r is the set of matrices $\mathbf{C}_i^r$ that have a non-zero element in column ℓ , and S_ℓ is the set of rows of such matrices.

3. For $r \in [2]$ and $\ell \in [K]$, compute

$$\widehat{R}_\ell^r(Y) = \frac{1}{m} \sum_{i \in V_\ell^r} \text{val}_i(\mathbf{k}_\ell) \mathbf{h}_{\text{row}_i(\mathbf{k}_\ell) - (r-1)m} \prod_{s \in S_\ell \setminus \{\text{row}_i(\mathbf{k}_\ell) - (r-1)m\}} (Y - \mathbf{h}_s),$$

$$\widehat{I}_\ell(Y) = \prod_{s \in S_\ell} (Y - \mathbf{h}_s).$$

Write $\widehat{R}_\ell^r(Y) = \sum_{j=0}^{\nu-1} \widehat{R}_{\ell j}^r Y^j$ and $\widehat{I}_\ell(Y) = \sum_{j=0}^{\nu} \widehat{I}_{\ell j} Y^j$.

4. Define

$$v_j^{\widehat{R},r}(X) = \sum_{\ell=1}^K \widehat{R}_{\ell j}^r \mu_\ell(X), \quad v_j^{\widehat{I}}(X) = \sum_{\ell=1}^K \widehat{I}_{\ell j} \mu_\ell(X).$$

5. Output

$$\mathbf{pk} = \left(\mathbf{W}, \{v_j^{\widehat{R},r}\}_{j=0, r \in [2]}^{\nu-1}, \{v_j^{\widehat{I}}\}_{j=0}^{\nu} \right),$$

$$\mathbf{vk} = \left(\{[v_j^{\widehat{R},r}(x)]_1, [v_j^{\widehat{R},r}(x)]_2\}_{j=0, r \in [2]}^{\nu-1}, \{[v_j^{\widehat{I}}(x)]_1, [v_j^{\widehat{I}}(x)]_2\}_{j=0}^{\nu} \right),$$

that is, every encoding row is committed in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Each v_j has degree at most $K-1 \leq d$, so both commitments are available from $\mathbf{pp}$. The verifier below uses only the $\mathbb{G}_1$ commitments; the $\mathbb{G}_2$ commitments are used in Section 6, where these rows serve as $\mathbf{cq}$ tables.

$\langle \mathcal{P}, \mathcal{V} \rangle((\mathbf{pk}, \mathbf{vk}), ()):$

1. $\mathcal{P}$: Define $\mathbf{a} = (\mathbf{x}, \mathbf{a}')$, $\mathbf{b} = (\mathbf{1}, \mathbf{b}')$ and compute $A(X) = \sum_{j=1}^K a_j \mu_j(X)$, $B(X) = \sum_{j=1}^K b_j \mu_j(X)$, $B'(X) = (B(X) - 1)/Z_{\mathbf{v}_l}(X)$ and

$$A'(X) = \frac{A(X) - \sum_{j=1}^l \mathbf{x}_j \mu_j(X)}{Z_{\mathbf{v}_l}(X)} = \frac{\sum_{j=l+1}^K a_j \mu_j(X)}{Z_{\mathbf{v}_l}(X)}.$$

Output polynomial commitments $[A']_1, [B']_1$

2. $\mathcal{V}$: Send evaluation randomness $\alpha_1 \in \mathbb{F}$ and aggregation randomness $\beta_1 \in \mathbb{F}$.
3. $\mathcal{P}$: Compute polynomials
 - $D_a(X) = P_a(X, \alpha_1)$, $D_b(X) = P_b(X, \alpha_1)$, $D_c(X) = P_c(X, \alpha_1) = D_c^1(X) + \beta_1 D_c^2(X)$

- For $\widehat{R}_{\alpha_1}(X) = \sum_{j=0}^{\nu-1} \alpha_1^j (v_j^{\widehat{R},1}(X) + \beta_1 v_j^{\widehat{R},2}(X))$, $\widehat{I}_{\alpha_1}(X) = \sum_{j=0}^{\nu} \alpha_1^j v_j^{\widehat{I}}(X)$, $H_2(X)$ s.t.

$$D_c(X) \widehat{I}_{\alpha_1}(X) = Z_{\mathbb{H}}(\alpha_1) \widehat{R}_{\alpha_1}(X) + H_2(X) Z_{\mathbb{V}}(X)$$

- $H_1(X), R(X)$ s.t.

$$A(X) D_a(X) + \beta_1 B(X) D_b(X) - D_c(X) A(X) B(X) = X R(X) + Z_{\mathbb{V}}(X) H_1(X)$$

and output commitments $[D_c]_1, [R]_1, [H_1]_1, [H_2]_1$.

4. $\mathcal{V}$: Send random $\rho \in \mathbb{F}$.
5. $\mathcal{P}$: Compute and send $\mathbf{p}_{\text{deg}} = [P_{\text{deg}}(x)]_1$ for

$$P_{\text{deg}}(X) = X^{d-(K-1)} (A'(X) + \rho B'(X) + \rho^2 D_c(X)) + \rho^3 X^{d-(K-2)} R(X).$$

6. $\mathcal{V}$: Check

$$e([A']_1 + \rho[B']_1 + \rho^2[D_c]_1, [x^{d-(K-1)}]_2) \cdot e(\rho^3[R]_1, [x^{d-(K-2)}]_2) = e(\mathbf{p}_{\text{deg}}, [1]_2).$$

7. $\mathcal{V}$: Send evaluation randomness $\alpha_2 \in \mathbb{F}$.
8. $\mathcal{P}$: Compute and send $a = A(\alpha_2)$ and $d_c = D_c(\alpha_2)$.
9. $\mathcal{V}$: Send aggregation randomness $\beta_2, \beta_3 \in \mathbb{F}$.
10. $\mathcal{P}$: Compute

$$\begin{aligned} E(X) &= a D_a(\alpha_2) + \beta_1 B(X) D_b(\alpha_2) + d_c(-a B(X) + \beta_2^2 \widehat{I}_{\alpha_1}(X)) \\ &\quad - \alpha_2 R(X) - \beta_2^2 Z_{\mathbb{H}}(\alpha_1) \widehat{R}_{\alpha_1}(X) - Z_{\mathbb{V}}(\alpha_2)(H_1(X) + \beta_2^2 H_2(X)) \end{aligned}$$

Open $p(X) = A(X) + \beta_3 D_c(X) + \beta_3^2 E(X)$ at α_2 and output π_{KZG} .

11. $\mathcal{V}$:
 - Define $D_a = P_a(\alpha_2, \alpha_1)$ and $D_b = P_b(\alpha_2, \alpha_1)$.
 - Compute $[A] = [A'] Z_{\mathbb{V}_t}(\alpha_2) + \sum_{i=1}^l \mathbf{x}_i [\mu_i(x)]_1$, $[B] = [B'] Z_{\mathbb{V}_t}(\alpha_2) + 1$
 - Compute $[\widehat{R}_{\alpha_1}]_1 = \sum_{j=0}^{\nu-1} \alpha_1^j ([v_j^{\widehat{R},1}]_1 + \beta_1 [v_j^{\widehat{R},2}]_1)$ and $[\widehat{I}_{\alpha_1}]_1 = \sum_{j=0}^{\nu} \alpha_1^j [v_j^{\widehat{I}}]_1$, from the $\mathbb{G}_1$ components of $\mathbf{vk}$

Output 1 if and only if

$$e([p]_1 - (a + \beta_3 d_c), [1]_2) = e(\pi_{\text{KZG}}, [x - \alpha_2]_2)$$

for

$$\begin{aligned} [p]_1 &= [A]_1 + \beta_3 [D_c]_1 + \beta_3^2 (a D_a + \beta_1 [B]_1 D_b + d_c(-a[B]_1 + \beta_2^2 [\widehat{I}_{\alpha_1}]_1) \\ &\quad - \alpha_2 [R]_1 - \beta_2^2 Z_{\mathbb{H}}(\alpha_1) [\widehat{R}_{\alpha_1}]_1 - Z_{\mathbb{V}}(\alpha_2)([H_1]_1 + \beta_2^2 [H_2]_1)) \end{aligned}$$

Differences from Basilisk. Construction 4 is the non-zero-knowledge rectangular specialization of Basilisk. Removing Basilisk's masking coordinates gives the simpler public-input quotient A' above. We also send the claimed evaluations a, d_c before sampling the aggregation challenges β_2, β_3 , so that coefficient comparison in those challenges is sound. Finally, we use a universal SRS of degree $d \geq 2K - 3$ to accommodate H_1 ; this changes the concrete MSM sizes but not the asymptotic prover time.

Theorem 6. *Construction 4 is an argument of knowledge for the $\text{R1CS}_{\text{lite}}$ relation above in the algebraic group model, with the following efficiency guarantees:*

- $\mathcal{G}(d)$ takes $O(d)$ operations in each of $\mathbb{G}_1$ and $\mathbb{G}_2$.
- $\mathcal{K}(\mathbf{pp}, \mathbf{W})$ takes $O(\nu K \log K)$ field operations, $3\nu + 1$ MSMs of size at most K in $\mathbb{G}_1$, and, for the $\mathbb{G}_2$ components of $\mathbf{vk}$, $3\nu + 1$ MSMs of size at most K in $\mathbb{G}_2$.
- Prover time is $O(K \log K + \nu K)$ field operations and eight MSMs in $\mathbb{G}_1$: six of size at most K , one of size at most $2K - 2$, and one of size at most $2K - 3$.
- Verifier time is $O(l + \nu + \log K)$ field and group operations and five pairings.
- Proof size is eight $\mathbb{G}_1$ elements and two field elements.

Proof sketch. We follow the Basilisk proof; its AGM extraction, simple-matrix CSS soundness, and univariate sumcheck argument apply without change. We only record the points affected by this specialization.

First, $A(X) - \sum_{j=1}^l x_j \mu_j(X)$ vanishes on $\mathbb{V}_l$, so the displayed definition of A' is a polynomial and binds the public coordinates of $\mathbf{a}$. The same argument applies to B' because the corresponding coordinates of $\mathbf{b}$ equal one.

Second, the prover sends a, d_c after α_2 but before β_2, β_3 . Thus the final batched KZG opening implies $a = A(\alpha_2)$, $d_c = D_c(\alpha_2)$, and $E(\alpha_2) = 0$, except with negligible probability. Since the claims and all polynomial commitments are fixed before β_2 is sampled, comparing the constant and quadratic coefficients in β_2 yields the usual Basilisk sumcheck and CSS identities. The remainder of the original extraction therefore produces a witness for the stated $\text{R1CS}_{\text{lite}}$ relation.

Finally, the random linear combination in Step 6 enforces $\deg A', \deg B', \deg D_c \leq K - 1$ and $\deg R \leq K - 2$. The highest-degree remaining polynomial is H_1 : the numerator defining it has degree at most $3K - 3$, so $\deg H_1 \leq 2K - 3$. Hence $d \geq 2K - 3$ suffices. Committing to H_1 uses an MSM of size at most $2K - 2$, while the final opening quotient uses one of size at most $2K - 3$; all other prover MSMs have size at most K .

Public reducibility in the AGM follows by viewing Basilisk as a reduction to TRUE. Given the input structure, instance, and transcript, φ recomputes the verification key using the public indexer and replays the verifier's checks. On acceptance it returns the fixed terminal structure and instance $(\perp, \text{true})$, whose key encoder outputs $(\perp, \perp)$; otherwise it follows the verifier's rejection. No witness extraction or reconstruction of an updated circuit is needed for this terminal reduction. $\square$

D AI Disclosure

Generative AI tools were used throughout the drafting and revision of this manuscript as interactive writing, research, and typesetting assistants.

The assistance included proposing and revising technical exposition, reorganizing text, improving grammar and clarity, editing LaTeX, locating and normalizing bibliography entries, checking claims against cited papers, and auditing the related-work comparison table.

All generated suggestions were subject to author guidance and review. The underlying research contributions and final technical judgments are those of the authors, who take full responsibility for all content in the manuscript, including its accuracy, originality, and correctness.