Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees

Matteo Campanelli[1], Antonio Faonio[2(✉)], Dario Fiore[3], Tianyu Li[4(✉)], and Helger Lipmaa[5]

[1] Protocol Labs, Aarhus, Denmark
matteo.campanelli@gmail.com
[2] EURECOM, Sophia Antipolis, France
faonio@eurecom.fr
[3] IMDEA Software Institute, Madrid, Spain
dario.fiore@imdea.org
[4] Delft University of Technology, Delft, Netherlands
tianyu.li@tudelft.nl
[5] University of Tartu, Tartu, Estonia

Abstract. Lookup arguments allow to prove that the elements of a committed vector come from a (bigger) committed table. They enable novel approaches to reduce the prover complexity of general-purpose zkSNARKs, implementing "non-arithmetic operations" such as range checks, XOR and AND more efficiently. We extend the notion of lookup arguments along two directions and improve their efficiency: (1) we extend vector lookups to matrix lookups (where we can prove that a committed matrix is a submatrix of a committed table). (2) We consider the notion of zero-knowledge lookup argument that keeps the privacy of both the sub-vector/sub-matrix and the table. (3) We present new zero-knowledge lookup arguments, dubbed cq+, zkcq+ and cq++, more efficient than the state of the art, namely the recent work by Eagen, Fiore and Gabizon named cq. Finally, we give a novel application of zero-knowledge matrix lookup argument to the domain of zero-knowledge decision tree where the model provider releases a commitment to a decision tree and can prove zero-knowledge statistics over the committed data structure. Our scheme based on lookup arguments has succinct verification, prover's time complexity asymptotically better than the state of the art, and is secure in a strong security model where the commitment to the decision tree can be malicious.

1 Introduction

General-purpose zero-knowledge succinct arguments of knowledge (zkSNARKs) promise to efficiently and succinctly prove any kind of NP-statement while keeping privacy, integrity and verifiability guarantees. Thanks to their generality, a great number of real-world applications can be performed with built-in security. The two-step recipe for building a brand new zero-knowledge application typically consists of first describing the application in a low-level constraint system

Q. Tang and V. Teague (Eds.): PKC 2024, LNCS 14602, pp. 337–369, 2024.
https://doi.org/10.1007/978-3-031-57722-2_11

(for example, Rank-1 Constraint System [4] or Plonk arithemization [19]) and then use the latest fully-developed zkSNARK as *backend*. Unfortunately, most often, the *unfolded circuit* of the applications at hand becomes huge and, thus, the proving time could become unfeasible for real-world applications.

Lookup arguments [6,14,32,38,39] are a novel approach to reducing the size of unfolded circuits, bringing back to the real world many interesting applications. Briefly and informally, a lookup argument allows to trade *sub-circuits* evaluations for lookup into their truth tables. For example, instead of having n different sub-circuits describing the computation of a hash function in the final unfolded circuit, the protocol designer could define n different *custom gates* that perform efficient lookup operations in the truth table of such a hash function. More concretely, lookup arguments are used in current zkSNARKs for representing "non-arithmetic operations" that cannot be expressed efficiently through the finite field operations supported by the zkSNARK, such as range checks, XOR and AND (see for example [6,18]). Very recently, the work of Arun, Setty and Thaler [3] shows how to use lookup arguments to create SNARKs for virtual-machine executions, namely a new SNARK scheme, called Jolt, that allows verification of the correct execution of a computer program specified with an assembly language. Informally, in Jolt, the truth table of each assembly instruction is encoded in a (predefined and highly structured) table. Then, lookup arguments enforce the correct instructions execution, namely checking the inputs and outputs described by their truth tables.

In this work, we advance on lookup arguments in multiple ways. We propose new lookup arguments that improve over the state of the art [14]. One of our schemes enjoys, almost for free, an extended notion of zero-knowledge, which we call fully zero-knowledge, which protects the privacy of arbitrary commitments to the tables. Orthogonally, we consider two natural extensions from vectors to matrices and give constructions for such extensions. Finally, we motivate the extensions to matrix and to fully zero-knowledge by giving a new application to privacy-preserving machine learning that crucially relies on them.

New Lookup Arguments Based on cq. In a lookup argument, the prover aims to show that each coefficient of a (short) committed vector $\mathbf{f}$ of size n belongs to the (large) table $\mathbf{t}$ of size $N \gg n$. Since $N \gg n$, one of the desiderata of lookup arguments is that the prover's computation does not depend on N. Following a fast-pace line of recent works, Eagen, Fiore, and Gabizon [14] proposed an efficient lookup argument called cq (cq for *cached quotients*). Notably, cq's prover's computation is quasi-linear in n, while the proof size and verifier's computation are constant (e.g., proofs are 3840 bits, when using the standard BLS12-381 elliptic curve). In spite of appearing nearly optimal in efficiency, cq comes with two shortcomings. The first one is that it is not designed to have zero knowledge in mind. The second, more technical, one is that its use in larger protocols likely requires additional proof elements and pairing computations.[1] In this

[1] This is due to the fact that cq assumes an SRS of the same size as the table $\mathbf{t}$, and this allows avoiding a degree check. This condition, though, is often not guaranteed (e.g., in a SNARK for constraint systems larger than such a table).

work, we propose a new lookup argument, dubbed cq^+, that addresses all these shortcomings of cq and even achieves better efficiency. Namely, cq^+ achieves (standard) zero-knowledge at no overhead: it has the same prover's computation of cq and shorter proofs (3328 bits, and 2944 bits without ZK). Additionally, we consider two variations of cq^+: the first, dubbed zkcq^+, is fully zero-knowledge, while the second, dubbed cq^{++}, has shorter proofs. Both schemes require in verification only one pairing computation more than cq^+.

Lookup Arguments for Matrices. A lookup argument could be used to show that a database $\mathbf{f}$ is a selection of the rows of a database $\mathbf{t}$. However, to naively use lookup arguments for such an application, each row of the database must be efficiently encoded in one single field element (supported by the lookup argument). We consider two natural extensions to matrices. We focus on Kate *et al.* [25] polynomial commitment (also known as KZG commitment scheme) adapted to matrices. We give two lookup arguments for matrices that internally call a lookup argument for KZG commitments. The first scheme allows proving that a committed database $\mathbf{f}$ is a selection of the rows of a committed database $\mathbf{t}$, the second one allows proving that $\mathbf{f}$ is a selection of a projection of a database $\mathbf{t}$.

A New Approach to Zero-Knowledge for Decision Trees. We show an application of fully zero-knowledge matrix lookup arguments to zero-knowledge for decision trees (zkDT). We improve over the framework of Zhang *et al.* [40], which showed zkSNARKs for evaluations of committed decision trees and zkSNARKs for accuracy of committed decision trees. The former kind of zero-knowledge protocols can prove that a committed decision tree T, on input a vector $\mathbf{x}$, outputs a label v, while the latter schemes enable to validate the accuracy (namely, the ratio of true positives) of a decision tree on a given dataset.

Our framework can instantiate different kinds of statistics over committed decision trees, including evaluation and accuracy. Our design decouples the computation of the committed decision tree and the performed statistics. This allows for a plug-and-play approach. For security, we extend the notion of security from [40] considering possibly maliciously generated commitments to decision trees.

1.1 Technical Overview

Our Zero-Knowledge Lookup Arguments. Similarly to cq, cq^+ uses the technique of logarithmic derivates of Haböck [23]. However, we diverge from cq early, introducing several novel ideas that allow us to improve on cq's efficiency. One of the differences is that, while cq uses Aurora's sumcheck [5] twice, our cq^+ only runs it once. Nicely, this technique allows us to kill two birds with one stone, in fact, cq^+ does not require any additional low-degree tests. We give a more detailed technical overview in Sect. 4.1.

Matrix Lookup from Vector Lookup. To commit to a matrix, we can commit the concatenation of the rows of the matrix. Our matrix lookup arguments label all the entries of such a vectorization with the coordinates of each cell of the sub-matrix $\mathbf{F}$ into the bigger table $\mathbf{T}$. Similarly, in the precomputation phase,

they label each cell in the big table $\mathbf{T}$ with its coordinate. To prove that the k-th row of $\mathbf{F}$ appears in $\mathbf{T}$, we show that the *labelled* matrix $\mathbf{F}^* = (i_j, j, \mathbf{F}_{k,j})_{j\in[d]}$ is a sub-matrix of labeled table $\mathbf{T}^* = (i, j, \mathbf{T}_{i,j})_{i,j}$ and that $i_1 = i_2 = \cdots = i_d$ (in particular $i_j = k$), where d is the number of columns of the matrices. Notice that the first claim can be proved efficiently with a (non-succinct) matrix commitment for matrices with $N \cdot d$ rows and 3 columns following techniques from [6], while the second claim can be efficiently expressed through polynomial equations following techniques from [10]. In particular, for the first part, given a challenge $\rho \leftarrow \mathbb{F}$, the prover hashes $h(\mathbf{F}^*) = \sum_{i=1}^{3} \rho^{i-1} \cdot \mathbf{F}_i^*$ to a single column (where $\mathbf{F}_i^*$ are the columns of $\mathbf{F}^*$). Since $h(\cdot)$ is a universal hash function, if $h(\mathbf{F}^*)$ is a subvector of $h(\mathbf{T}^*)$, then with overwhelming probability, $\mathbf{F}^*$ is a submatrix of $\mathbf{T}^*$, thus reducing matrix lookup argument to vector lookup. For the second part, we notice that the first column $\mathbf{F}_1^*$ of $\mathbf{F}^*$ is a *step* function, thus we first commit to the shift of $\mathbf{F}_1^*$ and then show that the difference between the shifted column and the column $\mathbf{F}_1^*$ is a function that has zeros in well-defined positions. More details in Sect. 5.2. The second scheme goes even further and proves that a matrix $\mathbf{F}$ with d' columns and $d' < d$ is a submatrix of $\mathbf{T}$. As before, we set $\mathbf{F}^* = (i, j, \mathbf{F}_{i,j})_{i\in R, j\in D}$ for subset $R = \{r_1, \ldots, r_{d'}\} \subset [N]$ and $D \subset [d]$. Additionally, using the technique of shifted polynomials, $\mathbf{F}^*_{2,id'+j} = \mathbf{F}^*_{2,(i+1)d'+j} = r_j$ for any i, j. More details in in our full version [7, Appendix D].

Our Approach to ZK for Decision Trees. A decision tree is an algorithm that performs a sequence of adaptive queries reading from its input and outputs a value. At each query, the algorithm moves from a node in the tree to one of its children, and the output is defined by the label of the reached leaf. Two important parameters are the total number of nodes N_{tot} and the number of features d of the inputs. Following the work of Chen *et al.* [11], we can efficiently (although redundantly) encode a decision tree as a matrix with N_{tot} rows and $2d + 1$ columns. An evaluation of a decision tree under this alternative representation consists of locating the row corresponding to the correct leaf and then showing that the input vector matches all the constraints described by such a row. Thus, we can commit to a decision tree by committing to its matrix encoding, and to prove correct evaluation, we can commit to the single row corresponding to the correct leaf and prove with a matrix lookup argument that the committed row is indeed a leaf of the committed decision tree. Once isolated such a row, we can then prove that the input vector matches all the constraints described by the row. Notice that our strategy scales well with the number of different evaluations. In fact, to prove statements which involve multiple input vectors for the decision tree, we can commit at proving time to a matrix whose rows correspond to the entries of the leaves reached by the evaluations (instead of committing to a single row). Thanks to the efficiency property of the matrix lookup argument, the prover time complexity is independent of the size of decision trees.

Beyond a Trusted Commitment to the Tree. A malicious committer could commit to a matrix that contains a row that matches a leaf with a label, let's say, 0, and another row that matches the same leaf but where it maliciously assigns the label 1. Now, such a bogus commitment to a decision tree could allow the

malicious prover to show both $\mathsf{T}(\mathbf{x}) = 0$ and $\mathsf{T}(\mathbf{x}) = 1$. The problem is that the committed matrix does not *encode* a decision tree. To solve this problem, we show a set of sufficient algebraic conditions (cf. Sect. 6.2) for a matrix to *encode* a decision tree. We can check efficiently these algebraic conditions through a general-purpose zkSNARKs for R1CS (see for example [5,8,21,29,31,33,34]). However, the number of constraints is $O(dN_{\mathsf{tot}}^2)$, and thus the prover time complexity is quadratic in the number of nodes. The algebraic constraints we propose are essentially linear equations between matrices and Hadamard-product equations, which are the kind of equation checks performed in R1CS-based zkSNARKs. In fact, if we gave up on the privacy of the decision tree[2], we could define an R1CS circuit that depends on the tree-structure of the decision tree, and we would go down to $O(dN_{\mathsf{tot}})$ number of constraints. We can restore zero-knowledge using this approach, by privately committing to such an R1CS-like circuit and prove in zero-knowledge that the circuit belongs to a well-defined family of circuits (defined in Sect. 6.2). We use the techniques from Zapico *et al.* [38] for committing to a *basic* matrix, whose rows are elementary vectors, and to prove its basic-matrix structure and the permutation argument from Plonk [19] to prove the rows of the matrix are all different.

1.2 Related Work

Lookup Arguments. Lookup arguments were introduced by Bootle, Cerulli, Groth, Jakobsen and Maller [6]. The state-of-art for lookup arguments for arbitrary tables[3] is the recent work of Eagen, Fiore, Gabizon [14] named cq and based on the technique of logarithmic derivates of Haböck [23]. cq has prover complexity proportional only to the size of the smaller vector and independent of the bigger table assuming pre-processing for table. To our knowledge, all lookup arguments with similar efficiency properties are based on the Kate *et al.* (commonly known as KZG) polynomial commitment scheme [25]. Among these, we mention Caulk+ by Posen and Kattis [32] (based on Caulk [38] by Zapico *et al.*) and Baloo [39] by Zapico *et al.*. The latter work introduces the notion of Commit-and-Prove Checkable Subspace Argument (extending over [33]) that we use for our (extractable) commitment scheme (cf. Sect. 6.3).

Comparison with [14]. As previously mentioned, we diverge from cq, introducing several novel ideas that allow us to improve on cq's efficiency. As the end result, cq^+'s communication is about 14% (or even 23% in a variant without the ZK) better than cq's. All other efficiency parameters of cq^+ are similar to cq's. Moreover, we propose cq^{++}, a batched variant of cq^+. cq^{++} saves 23% (or 33%,

[2] Specifically, giving up only to the privacy of the *structure of* the decision tree while keeping private the values of the thresholds and labels.

[3] Recently, Setty, Thaler and Wahby [35] introduced a new lookup argument for a restricted subclass of tables. Their work is extremely efficient, and in particular more efficient than cq, for such a restricted class of tables. On the other hand, cq can handle arbitrary tables. For this reason, we refer to cq as the state-of-art for arbitrary tables.

in a variant without ZK) communication compared to cq. A slight drawback of cq^{++} is that the verifier has to execute one more pairing. We emphasize that cq is already almost optimally efficient, and thus improving on it is non-trivial.

Concurrent Work. Choudhuri *et al.* [13] very recently introduced the notion of *segment lookup arguments* which, besides some syntactical differences, matches the simpler of our notions of matrix lookup arguments. Additionally, in [13], they show, in our lingo, a matrix lookup argument based on cq. Their matrix lookup argument is less efficient than ours; we defer to Table 1 for more details. Interestingly, in the same paper, the authors build a general-purpose zkSNARK based on Plonk and matrix lookup, which they call Sublonk, showing another application for matrix lookup arguments. The main feature of Sublonk is that the prover's running time grows with the size of the *active part* of the circuit, namely the part of the circuit activated by its execution on a given instance. Sublonk makes black-box use of the underlying matrix lookup argument. Thus, we can plug in our scheme to obtain a more efficient version of Sublonk.

Privacy-Preserving Machine Learning. We focus on the related work on zero-knowledge proofs for decision trees and, more in general, for machine learning algorithms. The main related work for decision trees is the paper of Zhang *et al.* [40], where they introduce the notions of zero-knowledge proofs for decision tree predictions and accuracy. Besides decision trees, zero-knowledge proofs and verifiable computation for machine learning is a vibrant area of research (see for example [1,16,24,26,30,36,37]).

Comparison with [40]. Briefly, the main techniques of [40] consist of an authenticated data structure for committing to decision trees and highly-tuned R1CS circuits to evaluate the authenticated data structure in zero-knowledge. More in detail, they commit to a decision tree with a *labelled Merkle Tree* whose labelled nodes are the nodes of the decision tree. This commitment scheme is binding and hiding and allows for path openings (with proof size proportional to the length of the path). On top of this authenticated data structure, they use general-purpose zkSNARKs for R1CS to prove, for example, the knowledge of a valid opening for a path and the labelling of the leaf. While the basic ideas are simple, the paper needs to solve many technical details and presents many optimizations which are necessary to obtain a practical scheme. The *backend* general-purpose zero-knowledge scheme they use is Aurora [5]. Thanks to this choice and because of the Merkle-Tree approach, their zero-knowledge scheme has a transparent setup and is presumably post-quantum secure.

Their security model stipulates that the decision tree is adversarially chosen, but the commitment to a decision tree is honestly generated. On the other hand, in our security model, we require the commitment scheme to be extractable, thus allowing for maliciously generated commitments. Notice that, besides improving security, our definitional choices allow for more efficient design. In fact, the (proof for the) extractable commitment is generated only once, let's say in an offline phase, while the (multiple) proofs of evaluation, in the online phase, can leverage extra security properties offered by the extractable commitment and thus faster.

For comparison with our work, we consider the extractability of their scheme for decision tree evaluation. This is not immediate: the main reason is that the witness for the zkSNARK is a single path from the root to the evaluated leaf (which could be extracted) while, to obtain our notion of extractability, it would be required to extract the full decision tree. Additionally, their authenticated data structure could allow to commit (and prove statements) to 2-fan-in direct-acyclic graphs (DAGs), which are more general than trees[4]. We believe their second scheme for the accuracy of decision trees can be proved secure in our model. In fact, proposed as an efficiency optimization, their second scheme computes a consistency check over the full decision tree. Thanks to this, we could extract the full tree from the zkSNARK. We also believe that our techniques could be integrated into theirs. Our approach separating the extractable commitment from the "online-stage" of the zero-knowledge proof could be adapted to their scheme for accuracy (thus improving its efficiency). Interestingly, by using our approach, their scheme could be interpreted as an application of a lookup argument based on [5,6] to decision trees. The main difference is this: our scheme runs lookup arguments over the leaves associated with the evaluation vectors, while the scheme in [40] requires lookups for paths from the root to the leaves associated with the evaluation vectors.

For other points of comparison efficiency-wise (we refer the reader to Sect. 6.5 for more details), we mention that their commitments require hashing only, while ours requires multiexponentations in a group. Therefore, their commitment stage is faster than ours. Our proof size is concretely smaller (few kilobytes vs hundreds of kilobytes). To compare proving time, we start from observing the asymptotic advantages of our solutions: their prover is linear in the size of the tree and in the complexity of a hash function; ours is sublinear in all these dimensions. This results in *concretely* faster proving times *despite* the fact that our prover requires group operations and theirs only field operations. This is a consequence of removing the constants deriving from the hash function size, the sublinearity in the tree and of the efficient lookup argument instantiations[5]. Our improvements also translate to a better verification time. Our estimates show improvements of almost one order of magnitude for proving time (regardless of the underlying backend proof systems for [40]; see [7, Appendix A]) and two orders of magnitude for verification time.

2 Preliminaries

We denote matrices with capital and bold, for example, $\mathbf{M}$, and vectors with lowercase and bold, for example, $\mathbf{v}$. We denote with $\circ$ the Hadamard product

[4] We believe that this does not pose any problems neither for correctness nor for soundness, as indeed, one could argue this is a feature rather than a bug.

[5] As a bottleneck, the dependency [40] has on the hash function is one that is hard to remove. Applying a hash function optimized for SNARK constraints, e.g. the one we used to experimentally run [40]—SWIFFT—*nonetheless* yields high constants in practice *regardless* of the proof system used as a backend.

between two matrices/vectors of the same size, while $\cdot$ is reserved for the matrix-vector/vector-vector multiplication. Given two vectors $\mathbf{a}, \mathbf{b}$ we define $\mathbf{a} < \mathbf{b}$ if and only if $\forall i : \mathbf{a}_i < \mathbf{b}_i$ (and similarly for $\leq$). We denote $\|$ the concatenation by columns of two matrices. We denote by $\mathbb{F}$ a finite field, by $\mathbb{F}[X]$ the ring of univariate polynomials, and by $\mathbb{F}_{<d}[X]$ (resp. $\mathbb{F}_{\leq d}[X]$) the set of polynomials in $\mathbb{F}[X]$ of degree $< d$ (resp. $\leq d$). For any subset $S \subseteq \mathbb{F}$, we denote by $\nu_S(X) \stackrel{\text{def}}{=} \prod_{s \in S}(X - s)$ the *vanishing polynomial* of S, and by $\lambda_s^S(X)$ the *s-th Lagrange basis polynomial*, which is the unique polynomial of degree at most $|S| - 1$ such that for any $s' \in S$, it evaluates to 1 if $s = s'$ and to 0 otherwise. Any multiplicative subgroup of a finite field is cyclic. Thus, given a group $\mathbb{H}$, we can find an element ω that generates the subgroup $\mathbb{H}$. For convenience, given a subgroup $\mathbb{H}$ of order n we denote with ω_n a fixed generator of $\mathbb{H}$. If $\mathbb{H} \subseteq \mathbb{F}$ is a multiplicative subgroup of order n, then its vanishing polynomial has a compact representation $\nu_{\mathbb{H}}(X) = (X^n - 1)$ and $\lambda_i^{\mathbb{H}}(X) = \nu_{\mathbb{H}}(X)\omega_n^{i-1}/(n(X - \omega_n^{i-1}))$. Both $\nu_{\mathbb{H}}(X)$ and $\lambda_i^{\mathbb{H}}(X)$ can be evaluated in $O(\log n)$ field operations. For any vector $\mathbf{v} \in \mathbb{F}^n$, we denote by $v_{\mathbb{H}}(X)$ the *low degree encoding* (LDE) in $\mathbb{H}$ of $\mathbf{v}$, i.e., the unique degree-$(|\mathbb{H}| - 1)$ polynomial such that, $v_{\mathbb{H}}(\omega_n^{i-1}) = v_i$, when the subgroup $\mathbb{H}$ is clear from the context, we simply write $v(X)$. Similarly, we consider the k-degree *randomized low-degree encoding* (RLDE) in $\mathbb{H}$ of a vector $\mathbf{v} \in \mathbb{F}^n$ to be a randomized polynomial of the form $\hat{v}_{\mathbb{H}}(X) = v_{\mathbb{H}}(X) + \nu_{\mathbb{H}}(X)\rho_v(X)$ for a random polynomial ρ_v of degree k. Sometimes, we will not explicitly mention the degree of the randomizer. In this case, the reader should assume that the degree is set to be the minimum degree necessary to keep zero-knowledge of v in the presence of evaluations (on points outside of $\mathbb{H}$) of the polynomial $\hat{v}$.

A type-3 bilinear group $\mathbb{G}$ is a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$. $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q. P_1, P_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$. $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently-computable non-degenerate bilinear map, and there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the implicit notation $[a]_i := aP_i$, for elements in $\mathbb{G}_i, i \in \{1, 2, T\}$ and set $P_T := e(P_1, P_2)$.

Definition 1 (Power Discrete Logarithm [27]). *Let* $d_1(\lambda), d_2(\lambda) \in \mathsf{poly}(\lambda)$. *A bilinear group generator* $\mathsf{GroupGen}$ *is* (d_1, d_2)-PDL (Power Discrete Logarithm) secure *if for any non-uniform PPT* $\mathcal{A}$, $\mathsf{Adv}^{\text{pdl}}_{d_1,d_2,\mathsf{GroupGen},\mathcal{A}}(\lambda) :=$

$$\Pr\left[\mathsf{pp} \leftarrow \mathsf{GroupGen}(1^\lambda); s \leftarrow\!\$\, \mathbb{F}^* : \mathcal{A}\left(\mathsf{pp}, \left[(s^i)_{i=0}^{d_1}\right]_1, \left[(s^i)_{i=0}^{d_2}\right]_2\right) = s\right] = \mathsf{negl}(\lambda).$$

2.1 Commit-and-Prove SNARKs

A commitment scheme is a tuple of algorithm $\mathsf{CS} = (\mathsf{KGen}, \mathsf{Com})$ where the first algorithm samples a commitment key ck and the second algorithm, upon input of the commitment key, a message p and opening material ρ, outputs a commitment c. The basic notions of security for the commitment scheme are (perfect) *hiding* and (computational) *binding*.

Following Groth *et al.* [22], we define a relation $\mathcal{R}$ verifying triple $(\mathsf{pp}; x; w)$. We say that w is a witness to the instance x being in the relation defined

by the parameters pp when $(\mathsf{pp}; x; w) \in \mathcal{R}$ (equivalently, we sometimes write $\mathcal{R}(\mathsf{pp}; x; w) = 1$). For example, the parameters pp could be the description of a bilinear group, or additionally contain a commitment key for a commitment scheme or a common reference string. Whenever it is clear of the context, we will write $\mathcal{R}(x; w)$ as a shortcut for $\mathcal{R}(\mathsf{pp}; x; w)$.

Briefly speaking, Commit-and-Prove SNARKs (CP-SNARKs) are zkSNARKs whose relations verify predicates over commitments [9]. Given a commitment scheme CS, we consider relations $\mathcal{R}$ whose instances are of the form $x = ((\mathsf{c}_j)_{j\in[\ell]}, \hat{x})$, where we can un-ambiguously parse the witness $w = ((p_j)_{j\in[\ell]}, (\rho_j)_{j\in[\ell]})$ for some $\ell \in \mathbb{N}$ with $\forall j : p_j$ is in the domain of a commitment scheme CS, and such that there exists a PT relation $\hat{\mathcal{R}}$ such that let $\hat{w} = (p_j)_{j\in[\ell]}$:

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

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

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

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

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

Zero-Knowledge in the SRS (and RO) Model. The zero-knowledge simulator $\mathcal{S}$ of a CP-SNARK is a stateful PPT algorithm that can operate in three modes. $(\mathsf{srs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(0, 1^{\lambda}, d)$ takes care of generating the parameters and the simulation trapdoor (if necessary). $(\pi, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(1, \mathsf{st}_{\mathcal{S}}, x)$ simulates the proof for a statement x. $(a, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(2, \mathsf{st}_{\mathcal{S}}, s)$ takes care of answering random oracle queries. The state $\mathsf{st}_{\mathcal{S}}$ is shared and updated after each operation. We define zero-knowledge similarly to [15,20]:

Definition 2 (Zero-Knowledge). *We say that a CP-SNARK* CP *for a CP-relation* $\hat{\mathcal{R}}$ *and commitment scheme* CS *is (perfect) zero-knowledge if there exists a PPT simulator* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$ *and for all* $d \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l}\mathsf{ck} \leftarrow \mathsf{CS.KGen}(1^\lambda, d)\\ \mathsf{srs} \leftarrow \mathsf{CP.KGen}(\mathsf{ck})\\ \mathcal{A}^{\mathsf{Prove}(\mathsf{srs},\cdot,\cdot),\mathsf{RO}(\cdot)}(\mathsf{srs}) = 1\end{array}\right] \approx \Pr\left[\begin{array}{l}(\mathsf{srs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(0, \mathsf{pp}_{\mathbb{G}})\\ \mathcal{A}^{\mathcal{S}_1(\cdot,\cdot),\mathcal{S}_2(\cdot)}(\mathsf{srs}) = 1\end{array}\right]$$

where $\mathcal{S}_1, \mathcal{S}_2$ *are stateful (wrapper) algorithms that share their state* $\mathsf{st} = (\mathsf{st}_{\mathcal{S}}, \mathcal{Q}_{\mathsf{RO}})$ *where* $\mathsf{st}_{\mathcal{S}}$ *is initially set to be the empty string, and* $\mathcal{Q}_{\mathsf{RO}}$ *is initially set to be the empty set, such that:*

- $\mathcal{S}_1(x, w)$ *denotes an oracle that first checks* $(\mathsf{pp}, x, w) \in \mathcal{R}$ *where* pp *is part of* srs *and then runs the first output of* $\mathcal{S}(1, \mathsf{st}_{\mathcal{S}}, x)$*.*
- $\mathcal{S}_2(s)$ *denotes an oracle that first checks if the query* s *is already present in* $\mathcal{Q}_{\mathsf{RO}}$ *and in case answers accordingly, otherwise it returns the first output* a *of* $\mathcal{S}(2, \mathsf{st}_{\mathcal{S}}, s)$*. The oracle updates* $\mathcal{Q}_{\mathsf{RO}}$ *by adding the tuple* (s, a) *to the set.*

Knowledge Soundness. Our definition of knowledge soundness is in the algebraic group model [17]. An algorithm $\mathcal{A}$ is called *algebraic* if for all group elements that $\mathcal{A}$ outputs, it additionally provides the representation relative to all previously received group elements. That is, if elems is the list of group elements that $\mathcal{A}$ has received, then for any group element z in output, the adversary must also provide a vector $\mathbf{r}$ such that $\mathsf{z} = \langle \mathbf{r}, \mathsf{elems} \rangle$. We define the notion of knowledge soundness in the algebraic model.

Definition 3 (Knowledge Soundness in the AGM). *A CP-SNARK is knowledge extractable in the Algebraic Group Model if for any PT algebraic adversary, there exists a PT extractor* $\mathcal{E}$ *that receives in input the algebraic representations* $\mathbf{r}_1, \ldots, \mathbf{r}_l$ *of* $\mathcal{A}$ *and such that:*

$$\Pr\left[\begin{array}{l}\mathsf{ck} \leftarrow \mathsf{CS.KGen}(1^\lambda, d); \mathsf{srs} \leftarrow \mathsf{CP.KGen}(\mathsf{ck});\\ (x, \pi, \mathbf{r}_1, \ldots, \mathbf{r}_l) \leftarrow \mathcal{A}(\mathsf{srs}); w \leftarrow \mathcal{E}(\mathsf{srs}, \mathbf{r}_1, \ldots, \mathbf{r}_l)\\ \mathsf{Verify}(\mathsf{srs}, x, \pi) \wedge \neg\mathcal{R}(\mathsf{pp}, x, w)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Indexed Relations and Universal CP-SNARKs. We extend the notion of relations to indexed relations [12]. We define a PT indexed relation $\mathcal{R}$ verifying tuple (pp, ind, x, w). We say that w is a witness to the instance x being in the relation defined by the pp and index ind when $(\mathsf{pp}, ind, x, w) \in \mathcal{R}$ (equivalently, we sometimes write $\mathcal{R}(\mathsf{pp}, ind, x, w) = 1$).

Briefly, we say that a CP-SNARK is *universal* if there exists a deterministic algorithm Der that takes as input an srs and an index ind, and outputs a specialized reference string $\mathsf{srs}_{ind} = (\mathsf{vk}_{ind}, \mathsf{ek}_{ind})$ where vk_{ind} is a succinct verification key and ek_{ind} is a proving key for such an index. Moreover, we require that the verifier Verify (resp. the P) of a Universal CP-SNARK takes as additional input the specialized verification key vk_{ind} (resp. the specialized ek_{ind}). We refer to [7, Appendix B] for more details.

2.2 Extractable Commitment Schemes

An extractable commitment scheme for a domain $\mathcal{D} = \{\mathcal{D}_\lambda\}_\lambda$ is a commitment scheme equipped with a CP-SNARK that proves the knowledge of an opening of the commitments.

Definition 4. *Given a domain $\mathcal{D}$, $\mathsf{CS} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{VerCom})$ is an extractable commitment scheme for the domain $\mathcal{D}$ if there exist two algorithms $\mathsf{Com}', \mathsf{Prove}'$ such that $\mathsf{Com}(\mathsf{ck}, p, \rho)$ executes (1) $\mathsf{c} \leftarrow \mathsf{Com}'(\mathsf{ck}, p, \rho)$ and (2) $\pi \leftarrow \mathsf{Prove}'(\mathsf{ck}, \mathsf{c}, (p, \rho))$ and outputs (c, π), and $(\mathsf{Prove}', \mathsf{VerCom})$ is a CP-SNARK for the commitment scheme $(\mathsf{KGen}, \mathsf{Com}')$ and for the CP-Relation $\hat{\mathcal{R}}_{\mathsf{open}}$ defined below:*

$$\hat{\mathcal{R}}_{\mathsf{open}} = \{\mathsf{pp}; \varepsilon; p : p \in \mathcal{D}_\lambda\}.$$

2.3 Polynomial, Vector and Matrix Commitment Schemes

We use the KZG polynomial commitment scheme of [25] described below:

$\mathsf{KGen}(1^\lambda, d_1, d_2)$ samples a type-3 pairing group with security level λ and outputs commitment key $\mathsf{ck} := ((\left[s^i\right]_1)_{i\in[d_1]}, (\left[s^i\right]_2)_{i\in[d_2]},)$ for random secrets $s \in \mathbb{Z}_q$.
$\mathsf{Com}(\mathsf{ck}, p)$ outputs $[p(s)]_1$.

We notice that the above commitment scheme is not hiding and it is extractable[6] for the domain of polynomial of degree d_1 in the algebraic group model of [17] under the power discrete logarithm assumption (PDL), which informally states that find s is hard given a freshly sampled commitment key, see Definition 1 for details. The commitment scheme allows for a very efficient CP-SNARK $\Pi_{\mathsf{eval}} = (\mathsf{Prove}_{\mathsf{eval}}, \mathsf{Verify}_{\mathsf{eval}})$ for the CP-relation $\hat{\mathcal{R}}_{\mathsf{eval}} = \{(x, y; p) \colon p(x) = y\}$. In particular, the prover $\mathsf{Prove}_{\mathsf{eval}}$ upon input the SRS ck, an instance $([p(s)]_1, x, y)$ and the witness p, computes the unique polynomial $w(X) = (p(X) - y)/(X - x)$ and outputs $[w(s)]_1$ as its proof. On the other hand, the verifier $\mathsf{Verify}_{\mathsf{eval}}$ upon input the SRS ck, an instance (c, x, y) and a proof π, checks $e(\mathsf{c} - [y]_1, [1]_2) = e(\pi, [s - x]_2)$.

Vector and Matrix Commitment Schemes. From a polynomial commitment scheme, we can define a *vector* commitment. Specifically, let $\mathbb{H}$ be multiplicative subgroup of $\mathbb{F}$ with order N, and let ω_N be a fixed generator of $\mathbb{H}$. We can commit to vector $\mathbf{v}$ by committing to the low degree encoding of v over $\mathbb{H}$. Namely, $[v_{\mathbb{H}}(s)]_1$ is a commitment to $\mathbf{v}$. The commitment key should additionally contain the description of the subgroup $\mathbb{H}$ to allow for verification. Notice that

[6] As argued in [8], we can define a *vacuous* CP-SNARK for opening in the AGM where the prover does nothing and the verifier checks that the commitment is a valid group element. However, Lipmaa et al. [28] recently defined AGMOS, a more realistic variant of the AGM where the algebraic adversary can obliviously sample group elements. They pointed out that KZG is only extractable after the prover has successfully opened the commitment at some point. In this case, such a vacuous CP-SNARK is not sufficient. We leave it to further work to prove the security of our protocols in AGMOS.

such a commitment scheme is not hiding. We can make it hiding by committing to a RLDE of v over $\mathbb{H}$ instead of its LDE.

We define the *vectorization* of a matrix $\mathbf{M} \in \mathbb{F}^{n \times d}$ to be the vector $\bar{\mathbf{m}} \in \mathbb{F}^{n \cdot d}$ which is the concatenation of the rows of $\mathbf{M}$. Namely, for any $i \in [n], j \in [d]$, we define $\bar{\mathbf{m}}_{d \cdot i + j} = \mathbf{M}_{i,j}$. To commit to a matrix $\mathbf{M}$, we commit to its vectorization $\bar{\mathbf{m}}$. Notice that, additionally, the commitment key should contain the values n and d, and the subgroup $\mathbb{H}$ should be of cardinality $n \cdot d$.

3 Zero-Knowledge Matrix Lookup Arguments

Given two vectors $\mathbf{f}, \mathbf{t}$, we say that $\mathbf{f}$ is a *sub-vector* of $\mathbf{t}$ if there exits a (multi) set $K = \{k_1, \ldots, k_n\}$ such that $\mathbf{f}_j = \mathbf{t}_{k_j}$ for any j. We write $\mathbf{f} \prec \mathbf{t}$ to denote that $\mathbf{f}$ is a sub-vector of $\mathbf{t}$. Notice we diverge from the usual notion of sub-vector. Namely, we assume that a sub-vector $\mathbf{f}$ may contain multiple copies of an element in $\mathbf{t}$ and, moreover, any permutation of $\mathbf{f}$ is a sub-vector of $\mathbf{t}$. We extend the notion of sub-vectors to matrices. We say that a matrix $\mathbf{F} \in \mathbb{F}^{n \times d}$ is a (rows) sub-matrix of a matrix $\mathbf{T} \in \mathbb{F}^{N \times d}$ if $\mathbf{F}$ parsed as a $\mathbb{F}^d$-vector of length n is a sub-vector of $\mathbf{T}$ parsed as a $\mathbb{F}^d$-vector of length N. In other words, $\mathbf{F}$ is a matrix whose rows are also rows in $\mathbf{T}$. Similarly, given a multi set $K = \{k_1, \ldots, k_l\}$ we can define the sub-matrix $\mathbf{F}_{|K}$ as the sub-matrix of $\mathbf{F}$ which j-th row is the row $\mathbf{F}_{k_j}$. Notice that our notion of sub-matrix is not standard. Besides the differences mentioned for the notion of sub-vector, we consider the special case where the number of columns of $\mathbf{F}$ and $\mathbf{T}$ are the same. This is sufficient for our application. However, for completeness, in [7, Appendix D.1], we consider the more general case where $\mathbf{F}$ may be a selection of a projection of $\mathbf{T}$. We call the latter the rows-columns sub-matrix relationship. We consider the following indexed CP-relation, where we will refer to $\mathbf{T}$ as the table and to $\mathbf{F}$ as the sub-vector (or sub-matrix):

$$\hat{\mathcal{R}}_{\mathsf{zklkp}} := \{\mathsf{pp}; (N, d, n); \varepsilon; (\mathbf{T}, \mathbf{F}) : \mathbf{F} \prec \mathbf{T}, |\mathbf{T}| = N \times d, |\mathbf{F}| = n \times d\}, \quad (1)$$

Previous work focuses on $d = 1$, namely the lookup argument for vector commitments, where the table $\mathbf{T}$ is public. Moreover, some of the previous work did not focus on zero-knowledge. Namely, previous work focused on (ZK or not) CP-SNARKs for the following CP-relation:

$$\hat{\mathcal{R}}_{\mathsf{lkp}} := \{\mathsf{pp}; (\mathbf{t}, n); \varepsilon; \mathbf{f} : \mathbf{f} \prec \mathbf{t}, |\mathbf{f}| = n\}. \quad (2)$$

A *fully* zero-knowledge lookup argument for a commitment scheme CS is a CP-SNARK for the CP-relation $\hat{\mathcal{R}}_{\mathsf{zklkp}}$ and for the commitment scheme CS. We use the adjective fully zero-knowledge to distinguish our definition from the definition from previous work. State-of-the-art lookup arguments have prover time complexity independent of the length of the table and quasi-linear (or even linear) on the length of the sub-vector. To obtain such a property, all the lookup arguments for arbitrary tables in previous work precompute the table $\mathbf{T}$, producing auxiliary material that is then used during the proving phase. Thus, using the notational framework of Universal SNARK, the precomputation is handled by the Der algorithm (since $\mathbf{t}$ is in the index).

Definition 5. *A tuple of algorithm* $\mathsf{CP} = (\mathsf{KGen}, \mathsf{Der}, \mathsf{Prove}, \mathsf{Verify})$ *is a lookup argument for a commitment scheme* CS *if (1)* CP *forms a CP-SNARK for* $\hat{\mathcal{R}}_{\mathsf{lkp}}$ *and* CS*, (2)* Der *is a* $\mathbb{F}$*-linear function (with respect to the proving key in its output) and the commitment scheme is linearly homomorphic and (3)* Prove *has running time* $\mathsf{poly}(n, \lambda)$.

We define an additional algorithm $\mathsf{Preproc}$ to handle our stronger privacy requirement. Similarly to Der, the algorithm $\mathsf{Preproc}$ performs an offline preprocessing — both algorithms are necessary *only* for speeding up the proving and verification algorithms. The difference is that Der works over *public* information, meanwhile $\mathsf{Preproc}$ works over *private* information[7].

Definition 6. *A tuple of algorithm* $\mathsf{CP} = (\mathsf{KGen}, \mathsf{Der}, \mathsf{Preproc}, \mathsf{Prove}, \mathsf{Verify})$ *is a fully zero-knowledge lookup argument for a matrix commitment scheme* CS *if (1)* $(\mathsf{KGen}, \mathsf{Der}, \mathsf{Prove}', \mathsf{Verify})$ *forms a CP-SNARK for* $\hat{\mathcal{R}}_{\mathsf{zklkp}}$ *and* CS *where* Prove' *is the algorithm that upon witness* $(\mathbf{T}, \mathbf{F}, \rho_T, \rho_M)$ *such that* $\mathbf{T}_{|K} = \mathbf{F}$ *first runs* $(\mathsf{aux}_j)_{j\in[N]} \leftarrow \mathsf{Preproc}(\mathsf{srs}, \mathbf{T}, \rho_T)$ *and then runs* Prove *with witness* $(\mathbf{F}, \rho_M, (\mathsf{aux}_j)_{j\in K})$*; (2)* $\mathsf{Preproc}$ *is a* $\mathbb{F}$*-linear function and the commitment scheme is linearly homomorphic and (3)* Prove *has running time* $\mathsf{poly}(nd, \lambda)$.

4 Our New Zero-Knowledge Lookup Arguments

In this section, we present our new lookup arguments for KZG-based vector commitments. Let the commitment $\mathsf{c}_\mathbf{t}$ and $\mathsf{c}_\mathbf{f}$, to the vectors $\mathbf{t}$ and $\mathbf{f}$ respectively, be KZG commitments to randomized low-degree encodings of $\mathbf{t}$ and $\mathbf{f}$. We denote these polynomials $\mathsf{T}(X)$ and $\mathsf{F}(X)$, respectively. Since $\mathbf{t}$ and $\mathbf{f}$ have different sizes, we interpolate them over two multiplicative subgroups of $\mathbb{F}$: $\mathbb{K}$ of order N and $\mathbb{H}$ of order $n \leq N$. In our construction, we need $n \mid N$; however, this usually holds in practice where both n and N are powers of two. Hence, we have

$$\mathsf{T}(X) := \sum_{j=1}^{N} \mathbf{t}_j \lambda_j^{\mathbb{K}}(X) + \rho_\mathsf{T} \cdot \nu_{\mathbb{K}}(X), \quad \mathsf{F}(X) := \sum_{i=1}^{n} \mathbf{f}_i \lambda_i^{\mathbb{H}}(X) + \rho_\mathsf{F}(X) \cdot \nu_{\mathbb{H}}(X)$$

Above, $\rho_\mathsf{F}(X)$ is a random polynomial of degree $< \mathsf{b}_F$ so that $\mathsf{c}_\mathbf{f} = [\mathsf{F}(s)]_1$ is perfectly hiding. Furthermore, our lookup arguments work (and are zero-knowledge) for any choice of $\mathsf{b}_F \geq 0$; this property matters whenever the commitment $\mathsf{c}_\mathbf{f}$ is generated by other protocols with their own zero-knowledge requirements (e.g., $\mathsf{c}_\mathbf{f}$ may come from a SNARK construction where b_F is carefully set to meet the number of leaked evaluations of $\mathsf{F}(X)$ in that protocol). Our lookup arguments achieve zero knowledge without leaking additional evaluations of $\mathsf{F}(X)$.

On the other hand, if $\rho_\mathsf{T} \leftarrow\!\!\$\ \mathbb{F}$ is a random field element, then $\mathsf{c}_\mathbf{t} = [\mathsf{T}(s)]_1$ is a perfectly hiding commitment to $\mathbf{t}$. Otherwise, if $\rho_\mathsf{T} = 0$, we capture the case of public tables (that is the common use case of lookup arguments).

[7] Alternatively, one could define one single algorithm Der that handles both public and private data. In this case, one needs to redefine the Universal SNARK's framework to handle zero knowledge correctly. Our definition instead is only functional as we require that $\mathsf{Preproc}$, Prove form a two-step prover algorithm for a Universal SNARK.

Lemma 1 (Set inclusion, [23]). *Let $\mathbb{F}$ be a field of characteristic $p > N$, and suppose that $(a_i)_{i=1}^N$, $(b_i)_{i=1}^N$ are arbitrary sequences of field elements. Then $\{a_i\} \subseteq \{b_i\}$ as sets (with multiples of values removed), if and only if there exists a sequence $(m_i)_{i=1}$ of field elements from $\mathbb{F}_p \subseteq \mathbb{F}$ such that*

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

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

Roadmap. For the sake of presentation, we first describe our main lookup argument cq^+, which works for a public table $\mathbf{t}$, thus meeting Definition 5. This protocol is fully described in Fig. 1 and explained in the next section. Next, we discuss an optimized variant, cq^{++}. Finally, in Sect. 4.2 we show how to obtain the protocol meeting the fully zero-knowledge notion of Definition 6.

4.1 cq^+ Lookup Argument

For ease of exposition, we present our protocol as a public coin interactive argument. We can compile it into a CP-SNARK using the Fiat-Shamir heuristic.

Setup. We assume a universal $\mathsf{srs} = ((\llbracket s^j \rrbracket_1)_{j=0}^{N_1}, (\llbracket s^j \rrbracket_2)_{j=0}^{N_2}))$ for any $N_1 \geq N + \max(\mathsf{b}_F, 1) - 1$ and $N_2 \geq N + \max(\mathsf{b}_F, 1) + 1$, where b_F is the degree of the randomization polynomial $\rho_{\mathsf{F}}(X)$ explained earlier.

Round 1. Our interactive lookup protocol cq^+ starts the same as cq [14]. Namely, based on Lemma 1, the prover computes the multiplicities vector $\mathbf{m}$ such that $\sum_{j=1}^N \frac{m_j}{\mathbf{t}_j + X} = \sum_{i=1}^n \frac{1}{\mathbf{f}_i + X}$, and sends to the verifier a commitment $[m(s)]_1$ to a randomized low-degree encoding $m(X)$ of $\mathbf{m}$ over $\mathbb{K}$.

Round 2. The verifier sends a random challenge β. At this point, the goal of the prover is to convince the verifier that

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

which, by Schwartz-Zippel, implies the polynomial identity over $\mathbb{F}[X]$ and thus $\mathbf{f} \prec \mathbf{t}$ by Lemma 1. To this end, the prover commits to randomized low-degree encodings of the two vectors containing the terms of the two sums, i.e.,

$$A(X), B(X) \text{ s.t. } A_j = A(\omega_N^{j-1}) = \frac{m_j}{\mathbf{t}_j + \beta} \quad \text{and} \quad B_i = B(\omega_n^{i-1}) = \frac{1}{\mathbf{f}_i + \beta} \ . \tag{5}$$

In order to prove the well-formedness of $A(X)$ and $B(X)$, as in cq, the prover commits to the polynomials $Q_A(X) = (A(X)(\mathsf{T}(X) + \beta) - m(X))/\nu_{\mathbb{K}}(X)$ and $Q_B(X) = (B(X)(\mathsf{F}(X) + \beta) - 1)/\nu_{\mathbb{H}}(X)$. As we discuss later, we compute a commitment to $Q_A(X)$ using the cached quotients technique of [14] to meet the efficiency requirement (3) of Definition 5.

From this Point, Our Protocol Diverges from cq. At this point of the protocol, cq would proceed by applying Aurora's univariate sumcheck on both $A(X)$ and

$B(X)$ to prove the correctness of results $A(0) = \sum_j A(\omega_N^{j-1})/N$ and $B(0) = \sum_i B(\omega_n^{i-1})/n$ and then the verifier would check that the results are equal.

In cq^+, we instead apply Aurora's univariate sumcheck on a scaled sum of $A(X)$ and $B(X)$ and prove that the result is zero. More precisely, we define $C(X) := A(X) - \vartheta^{-1}B(X)\mathsf{z}(X)$ where we denote $\vartheta := N/n$ and $\mathsf{z}(X) := \nu_{\mathbb{K}\setminus\mathbb{H}}(X)$ and use the following lemma (see [7, Appendix C] for its proof).

Lemma 2. $\sum_{j=0}^{N} A(\omega_N^{j-1}) = \sum_{i=0}^{n} B(\omega_n^{i-1})$ *iff* $\sum_{j=0}^{N} C(\omega_N^{j-1}) = 0$.

The lemma relies on the observation that the polynomial $\Delta(X) := \vartheta^{-1}B(X)\mathsf{z}(X)$ encodes over $\mathbb{K}$ the same vector encoded by $B(X)$ over $\mathbb{H}$, i.e., $\left(\frac{1}{\mathbf{f}_i+\beta}\right)_i$, but in different positions; while in the rest of positions it encodes zeros. Thus, $\sum_{j=0}^{N} \Delta(\omega_N^{j-1}) = \sum_{i=0}^{n} B(\omega_n^{i-1})$. Moreover, multiplying $B(X)$ by $\mathsf{z}(X)$ gives us for free a low-degree test on $B(X)$.

Thus, towards proving (4), we use Aurora's sumcheck on $C(X)$ to show

$$\exists R_C(X) \in \mathbb{F}_{\leq N-2}[X],\ Q_C(X) \text{ s.t. } C(X) = R_C(X)X + Q_C(X)\nu_{\mathbb{K}}(X)\ . \quad (6)$$

However, we do not send commitments to these two polynomials but use alternative techniques that allow us to obtain both zero knowledge and an efficient degree check on $R_C(X)$. More precisely, to obtain zero-knowledge, we use the sparse ZK sumcheck technique from Lunar [8]: the prover commits to a polynomial $S(X) := R_S X + \rho_S \nu_{\mathbb{K}}(X)$, with the idea that in the next round we perform a sumcheck on $C(X) + \eta^2 S(X)$, for a random challenge η to be chosen by the verifier in the following round. Actually, although for ease of expositions we introduced the use of $S(X)$ here; this polynomial is computed and committed as $[S(s)]_1$ in round 1. In summary, in round 2, the prover sends $[A(s), B(s), Q_B(s)]_1$.

Round 3. The verifier sends random challenges γ, η. In this round, the prover's goal is to show that

$$A(X)(\mathsf{T}(X) + \beta) - m(X) = Q_A(X)\nu_{\mathbb{K}}(X), \quad (7)$$
$$B(X)(\mathsf{F}(X) + \beta) - 1 = Q_B(X)\nu_{\mathbb{H}}(X), \quad (8)$$
$$A(X) - \vartheta^{-1}B(X)\mathsf{z}(X) + \eta^2 S(X) - (R_C(X) + \eta^2 R_S)X = Q_{C,S}(X)\nu_{\mathbb{K}}(X) \quad (9)$$

where $Q_{C,S}(X) = Q_C(X) + \eta^2\rho_S$ in (9). To prove Eq. (7), we use the cached quotient technique of [14] to compute a commitment $[Q_A(s)]_1$ using n scalar group multiplications (see below).

To prove Eq. (8), notice that we already sent $Q_B(X)$; thus, using a linearization trick and random point evaluations, we set $B_\gamma = B(\gamma)$ and we show $B(X)$ evaluates to B_γ on γ, $D(X) := B_\gamma(\mathsf{F}(X) + \beta) - 1 - Q_B(X)\nu_{\mathbb{H}}(\gamma)$ evaluates at 0 on γ. We batch these claims using the verifier's challenge η. Namely, we send the KZG-evaluation proof $P(X) := ((B(X) - B_\gamma) + \eta D(X))/(X - \gamma)$.

To prove Eq. (9), we apply a novel idea that allows obtaining, for free, a degree check on $R_C(X)$. We set the polynomial $U(X) = (X^\mu - 1)$ where $\mu = N_1 - N + 2$

and ask the prover to send $R^*_C(X) = (R_C(X) + \eta^2 R_S)U(X)$. To balance this, we multiply the rest of Eq. (9) by $U(X)$, obtaining

$$(A(X) - \vartheta^{-1}B(X)\mathsf{z}(X) + \eta^2 S(X))U(X) - R^*_C(X)X = Q_{C,S}(X)\nu_{\mathbb{K}}(X)U(X) \quad (10)$$

To further optimize this, we batch Eqs. (7) and (10) by using the verifier's random challenge η (and multiplying (7) by $U(X)$), finally obtaining:

$$A(X) \cdot \mathsf{T}(X)U(X) + ((\beta + \eta)A(X) - m(X) + \eta^2 S(X)) \cdot U(X) - \frac{\eta}{\vartheta}B(X) \cdot \mathsf{z}(X)U(X) - Q(X)\nu_{\mathbb{K}}(X)U(X) = \eta R^*_C(X) \cdot X \quad (11)$$

The idea of this batching is that after multiplying (7) by $U(X)$, both equations aim to prove that the left-hand side is divisible by $\nu_{\mathbb{K}}(X)$ and thus we can send a single quotient polynomial $Q(X) = Q_A(X) + \eta Q_C(X) + \eta^2 \rho_S$.

To summarize, in round 3, the prover sends $[P(s), R^*_C(s), Q(s)]_1$ and B_γ.

Verification. The verifier proceeds as described in Verify of Fig. 1. The verification Item (ii) is a standard technique to check the batched evaluation proof $[P(s)]_1$. The verification Item (i) instead implements the check of Eq. (11) using pairings. Doing this requires the verifier to have in the verification key the $\mathbb{G}_2$ elements $[\mathsf{T}(s)U(s)]_2$ as well as $[U(s), \mathsf{z}(s)U(s), \nu_{\mathbb{K}}(s)U(s)]_2$. Therefore, we let Der compute all these elements and include them in the verification key.

Prover Efficiency. We discuss how the prover algorithm can be implemented with $O(n)$ scalar multiplications in $\mathbb{G}_1$ and $O(n \log n)$ $\mathbb{F}$ operations. First, one can easily see that by preprocessing the computation of the elements $\left[\lambda^{\mathbb{K}}_j(s)\right]_1$ and $[\nu_{\mathbb{K}}(s)]_1$ and by using the n-sparsity of $\mathbf{m}$, it is possible to compute $[m(s), A(s)]_1$ using $2(n+1)$ scalar multiplications. Computing $Q_B(X)$ is the only step that requires time $O(n \log n)$ (in field operations). Computing $[B(s), Q_B(s), P(s)]_1$ requires $\approx 3n$ scalar multiplications.

Computing the commitments $[R^*_C(s)]_1$ and $[Q_A(s)]_1$ with $\approx 2n$ and n scalar multiplications, respectively, can be achieved thanks to the cached quotients and, again, the sparseness of $\mathbf{m}$. Following [14], in Der for $\mathbf{t}$, we compute and store

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

Then, we use this auxiliary input to compute, with $n+1$ scalar multiplications,

$$[Q_A(s)]_1 \leftarrow \textstyle\sum_{m_j \neq 0} A_j [Q_j(s)]_1 + [\rho_A(T(s) + \beta) - \rho_m]_1 . \quad (12)$$

The correctness of $Q_A(s)$ is due to

$$\begin{aligned}
\textstyle\sum_{j=1}^N A_j Q_j(X) &= \textstyle\sum_{j=1}^N \frac{A_j(\mathsf{T}(X) - \mathbf{t}_j)\lambda^{\mathbb{K}}_j(X)}{\nu_{\mathbb{K}}(X)} \\
&= \textstyle\sum_{j=1}^N \frac{A_j(\mathsf{T}(X) + \beta)\lambda^{\mathbb{K}}_j(X)}{\nu_{\mathbb{K}}(X)} - \sum_{j=1}^N \frac{A_j(\mathbf{t}_j + \beta)\lambda^{\mathbb{K}}_j(X)}{\nu_{\mathbb{K}}(X)} \\
&\overset{(5)}{=} (\mathsf{T}(X) + \beta) \textstyle\sum_{j=1}^N \frac{A_j \lambda^{\mathbb{K}}_j(X)}{\nu_{\mathbb{K}}(X)} - \sum_{j=1}^N \frac{m_j \lambda^{\mathbb{K}}_j(X)}{\nu_{\mathbb{K}}(X)} \\
&= \tfrac{(A(X) - \rho_A \nu_{\mathbb{K}}(X))(\mathsf{T}(X) + \beta) - m(X) + \rho_m \nu_{\mathbb{K}}(X)}{\nu_{\mathbb{K}}(X)} \\
&= Q_A(X) - \rho_A(T(X) + \beta) + \rho_m .
\end{aligned}$$

Using a similar technique, in Der we can precompute $\left[(r_j^{\mathbb{K}}(s))_{j=1}^{N}, (r_i^{\mathbb{H}}(s))_{i=1}^{n}\right]_1$ where $\left\{r_j^{\mathbb{K}}(X) = \frac{\lambda_j^{\mathbb{K}}(X)-\lambda_j^{\mathbb{K}}(0)}{X}U(X)\right\}_{j\in[N]}$, and $\left\{r_i^{\mathbb{H}}(X) = \frac{\lambda_i^{\mathbb{H}}(X)\mathsf{z}(X)-\lambda_i^{\mathbb{H}}(0)}{X}U(X)\right\}_{i\in[n]}$, and use them to compute $[R_C^*(s)]_1$ in $2n$ scalar multiplications.

Thus, the prover's computation is dominated by $8n$ scalar multiplications, which was also the case in cq that did not achieve zero-knowledge and assumed $A(X)$ to be of degree $< N$.

cq^{++}: A Variant with a Shorter Proof. We can further optimize cq^{+} by applying one more batching technique that consists of sending a single group element $[P^*(s)]_1 = [P(s) + R_C^*(s)]_1$ and in merging the two verification equations ((i)) and ((ii)) as follows:

$$\begin{aligned} &e\left([A(s)]_1, [\mathsf{T}(s)U(s)(s-\gamma)]_2\right) \cdot e\left(\left[(\beta+\eta)A(s) - m(s) + \eta^2 S(s)\right]_1, [U(s)(s-\gamma)]_2\right) \cdot \\ &\quad e\left(\tfrac{\eta}{\vartheta}[B(s)]_1, [\mathsf{z}(s)U(s)(s-\gamma)]_2\right)^{-1} \cdot e\left([Q(s)]_1, [\nu_{\mathbb{K}}(s)U(s)(s-\gamma)]_2\right)^{-1} \cdot \\ &\quad e\left(\eta[B(s) + \eta D(s) - B_\gamma]_1, [s]_2\right) = e\left(\eta[P^*(s)]_1, [s(s-\gamma)]_2\right) \ . \end{aligned}$$

This change also requires some small changes. First, we require in the srs to have $N_2 \geq N + \max(\mathsf{b}_F, 1) + 2$. Second, the verification key $\mathsf{vk}_{N,n}$ computed by Der must include $\left[(s^k U(s), s^k \mathsf{z}(s)U(s), s^k \nu_{\mathbb{K}}(s)U(s))_{k=0}^{1}\right]_2$. Third, the table-dependent verification key for $\mathbf{t}$ should include $\left[(s^k \mathsf{T}(s)U(s))_{k=0}^{1}\right]_2$.

Overall Efficiency. Assume that we use a standard curve like BLS12-381, where elements of $\mathbb{G}_1$ (resp., $\mathbb{F}$) are $\mathfrak{g}_1 = 384$ (resp., $\mathfrak{f} = 256$) bits long. Then, in cq^{+}, the communication is $8\mathfrak{g}_1 + 1\mathfrak{f}$ (3328 bits) and in cq^{++}, $7\mathfrak{g}_1 + 1\mathfrak{f}$ (2944 bits). The prover executes $\approx 8n$ scalar multiplications. Verifier has to execute 5 pairings in cq^{+} or 6 in cq^{++}. Importantly, two or three of the pairings are with the standard $\mathbb{G}_2$ element (depending on the variant, $[1, x]_2$ or $\left[1, x, x^2\right]_2$). Hence they can be batched with other pairings in the master protocol and essentially come for free.

If one does not wish ZK, we can remove $[S(s)]_1$ from the argument, and proof size is $7\mathfrak{g}_1 + 1\mathfrak{f}$ (2944 bits) in cq^{+}, and $6\mathfrak{g}_1 + 1\mathfrak{f}$ (2560 bits) in cq^{++}.

To compare, in cq [14] (that is not ZK), the communication is $8\mathfrak{g}_1 + 3\mathfrak{f}$ (3840 bits), the prover's computation is $\approx 8n$ scalar multiplications, and the verifier has to execute 5 pairings. Hence, even cq^{+} (*with* ZK) has better communication than cq (*without* ZK) while having the same cost in the rest of the parameters.

Security. In the following theorem, we argue the security of cq^{+} (see [7, Appendix C] for the proof and the definition of the Power Discrete Logarithm (PDL) assumption); the proof of cq^{++} is very similar.

Theorem 1. *The protocol cq^{+} from Fig. 1 is a lookup argument according to Definition 5. Specifically, cq^{+} is knowledge-sound in the AGM and ROM under the (N_1, N_2)-PDL assumption (see Definition 1), and, furthermore, the protocol is zero-knowledge.*

4.2 Our Fully Zero-Knowledge Lookup Argument

In this setting we have $\mathsf{T}(X) = \sum_{j=1}^{N} = \mathbf{t}_j \lambda_j^{\mathbb{K}}(X) + \rho_{\mathsf{T}} \cdot \nu_{\mathbb{K}}(X)$ where $\rho_{\mathsf{T}} \leftarrow\$ \mathbb{F}$ and $\mathsf{c_t} = [\mathsf{T}(s)]_1$. We need only slight modifications to turn cq^+ to a fully zero-knowledge lookup argument. We refer to the modified lookup argument as zkcq^+. First, we defer, from Der to $\mathsf{Preproc}$, the computation of all the *table-dependent* group elements. Namely, $\mathsf{Preproc}(\mathsf{srs}, \mathbf{t}, \rho_{\mathsf{T}})$ computes $([Q_j(s))]_1)_{j=1}^{N}$ and $\tilde{\mathsf{c}}_{\mathbf{t}} \leftarrow [\mathsf{T}(s)U(s)]_2$. The latter group element is included as part of the proof at proving time by the algorithm Prove. As consequence, Verify needs to additionally run the pairing check $e([1]_1, \tilde{\mathsf{c}}_{\mathbf{t}}) = e(\mathsf{c_t}, [U(s)]_2)$ to verify the well-formedness of the commitment $\mathsf{c_t}$. In the proof of knowledge soundness, this check allows us to

$\underline{\mathsf{Der}(\mathsf{srs}, \mathbf{t}, n)}$: // Assume that $|\mathbf{t}| = N = |\mathbb{K}|$, $n = |\mathbb{H}|$ and $n \mid N$, $\mathsf{srs} = ([(s^j)_{j\in[N_1]}]_1, [(s^j)_{j\in[N_2]}]_2)$ for any $N_1, N_2 \geq N + \max(\mathsf{b}_F, 1) - 1$.
Set $\mu = N_1 - N + 2$; define $U(X) := (X^\mu - 1)$, $\vartheta = N/n$, and $\mathsf{z}(X) = \nu_{\mathbb{K}\setminus\mathbb{H}}(X)$;
Define $\mathsf{T}(X) := \sum_{j=1}^{N} = \mathbf{t}_j \lambda_j^{\mathbb{K}}(X)$;
Let $\left\{ r_j^{\mathbb{K}}(X) = \frac{\lambda_j^{\mathbb{K}}(X) - \lambda_j^{\mathbb{K}}(0)}{X} U(X) \right\}_{j\in[N]}$, $\left\{ r_i^{\mathbb{H}}(X) = \frac{\lambda_i^{\mathbb{H}}(X)\mathsf{z}(X) - \lambda_i^{\mathbb{H}}(0)}{X} U(X) \right\}_{i\in[n]}$,
and $\left\{ Q_j(X) = \frac{(\mathsf{T}(X) - \mathbf{t}_j)\lambda_j^{\mathbb{K}}(X)}{\nu_{\mathbb{K}}(X)} \right\}_{j\in[N]}$.
Compute $\mathsf{ek}_{\mathbf{t},n} := \left[(r_j^{\mathbb{K}}(s))_{j=1}^{N}, (r_i^{\mathbb{H}}(s))_{i=1}^{n}, U(s), \nu_{\mathbb{K}}(s), s\nu_{\mathbb{K}}(s), (Q_j(s))_{j=1}^{N}, \mathsf{T}(s) \right]_1$;
Compute $\mathsf{vk}_{\mathbf{t},n} := [1, U(s), \mathsf{z}(s)U(s), \nu_{\mathbb{K}}(s)U(s), \mathsf{T}(s)U(s)]_2$;
Return $(\mathsf{ek}_{\mathbf{t},n}, \mathsf{vk}_{\mathbf{t},n})$.

$\underline{\mathsf{Prove}(\mathsf{ek}_{N,n}, \mathsf{c_f}, (\mathbf{f}, \rho_{\mathsf{F}}(X)))}$: //$\mathsf{c_f} = \left[\sum_i \mathbf{f}_i \lambda_i^{\mathbb{H}}(s) + \rho_{\mathsf{F}}(s)\nu_{\mathbb{H}}(s) \right]_1$, $deg(\rho_{\mathsf{F}}) = \mathsf{b}_F$.
Compute $\mathbf{m} = (m_1, \ldots, m_N)$ s.t. $\forall j : \mathbf{t}_j$ appears m_j times in $\mathbf{f}$; samples $\rho_m \leftarrow\$ \mathbb{F}$;
Compute $[m(s)]_1 \leftarrow \sum_{j=1}^{N} m_j \cdot \left[\lambda_j^{\mathbb{K}}(s) \right]_1 + \rho_m \cdot [\nu_{\mathbb{K}}(s)]_1$; // n scalar mults
Sample $R_S, \rho_S \leftarrow\$ \mathbb{F}$ and compute $[S(s)]_1 \leftarrow R_S \cdot s + \rho_S \cdot \nu_{\mathbb{K}}(s)$;
$\beta \leftarrow \mathsf{RO}(\mathsf{vk}_{N,n} \| (\mathsf{c_t}, \mathsf{c_f}) \| [m(s)]_1)$ //Fiat-Shamir challenge.
Sample $\rho_A \leftarrow\$ \mathbb{F}$, $\rho_B(X) \leftarrow\$ \mathbb{F}_{\leq 1}[X]$;
Let $A_j \leftarrow m_j/(\mathbf{t}_j + \beta)$ $\forall j = 1, \ldots, N$ and $B_i \leftarrow 1/(\mathbf{f}_i + \beta)$ $\forall i = 1, \ldots, n$;
Compute $[A(s)]_1 \leftarrow \sum_{j=1}^{N} A_j \left[\lambda_j^{\mathbb{K}}(s) \right]_1 + \rho_A \cdot [\nu_{\mathbb{K}}(s)]_1$;
Compute $[B(s)]_1 \leftarrow \sum_{i=1}^{n} B_i \left[\lambda_i^{\mathbb{H}}(s) \right]_1 + \rho_B(s) \cdot [\nu_{\mathbb{H}}(s)]_1$;
Compute $Q_B(X) \leftarrow (B(X)(\mathsf{F}(X) + \beta) - 1)/\nu_{\mathbb{H}}(X)$ and $[Q_B(s)]_1$;
$(\gamma, \eta) \leftarrow \mathsf{RO}(\beta \| [A(s), B(s), Q_B(s), S(s)]_1)$;//Fiat-Shamir challenge.
Compute $B_\gamma \leftarrow B(\gamma)$, $D(X) \leftarrow B_\gamma \cdot (\mathsf{F}(X) + \beta) - 1 - Q_B(X)\nu_{\mathbb{H}}(\gamma)$;
Compute $P(X) \leftarrow ((B(X) - B(\gamma)) + \eta D(X))/(X - \gamma)$ and $[P(s)]_1$; // KZG-proof for (8).
Compute $[R_C^*(s)]_1 \leftarrow \sum_{m_j \neq 0} A_j \cdot \left[r_j^{\mathbb{K}}(s) \right]_1 - \vartheta^{-1} \sum_{i=1}^{n} B_i \cdot \left[r_i^{\mathbb{H}}(s) \right]_1 + \eta^2 R_S \cdot [U(s)]_1$
Compute $[Q_A(s)]_1 \leftarrow \sum_{m_j \neq 0} A_j \cdot [Q_j(s)]_1 + [\rho_A(T(s) + \beta) - \rho_m]_1$;
Compute $[Q_C(s)]_1 \leftarrow \left[\rho_A + \vartheta^{-1} \rho_B(s) \right]_1$;
Compute $[Q(s)]_1 \leftarrow [Q_A(s)]_1 + \eta [Q_C(s)]_1 + \eta^2 [\rho_S]_1$;
Return $\pi = ([m(s), S(s), A(s), B(s), Q_B(s), P(s), R_C^*(s), Q(s)]_1, B_\gamma)$.

$\underline{\mathsf{Verify}(\mathsf{vk}_{\mathbf{t},n}, \mathsf{c_f}, \pi)}$:
Compute $[D(s)]_1 \leftarrow B_\gamma(\mathsf{c_f} + [\beta]_1) - [1]_1 - \nu_{\mathbb{H}}(\gamma)[Q_B(s)]_1$.
Return 1 if and only if the following holds:
(i) $e([A(s)]_1, \mathsf{c_t}) \cdot e((\beta + \eta) \cdot [A(s)]_1 - [m(s)]_1 + \eta^2 [S(s)]_1, [U(s)]_2) \cdot e(\eta/\vartheta \cdot [B(s)]_1, [\mathsf{z}(s)U(s)]_2)^{-1} \cdot e([Q(s)]_1, [\nu_{\mathbb{K}}(s)U(s)]_2)^{-1} = e(\eta \cdot [R_C^*(s)]_1, [x]_2)$,
(ii) $e([B(s)]_1 + \eta [D(s)]_1 - [B_\gamma]_1, [1]_2) = e([P(s)]_1, [s - \gamma]_2)$

Fig. 1. Our zero-knowledge lookup argument cq^+.

ensure that the polynomials extracted from $\mathsf{c_t}$ and $\tilde{\mathsf{c}}_\mathbf{t}$ are of the form $\mathsf{T}^*(X)$ and $\mathsf{T}^*(X)U(X)$ for some $\mathsf{T}^*(X)$; thus, after verifying this we can apply virtually the same proof of Theorem 1.

5 Our Matrix Lookup Argument

We show a compiler from a fully zero-knowledge vector lookup argument for KZG-based vector commitment to a fully zero-knowledge matrix lookup for the (succinct) KZG-based matrix commitment from Sect. 2.3. The same construction applies for lookup argument as in Definition 5.

5.1 The Straw Man Solution

An alternative approach to commit to a matrix is to one-by-one vector commit to its columns. The obvious shortcoming is that the commitment scheme is not succinct in the number of columns. Nonetheless, this approach already results in a matrix lookup argument (under the assumption that the vector commitment is linearly homomorphic). In particular, consider the lookup argument that hashes together the columns $\mathbf{t}_j$ of the table $\mathbf{T}$ and the columns $\mathbf{f}_j$ of the sub-matrix $\mathbf{F}$ using a random challenge ρ computing vectors

$$\mathbf{t}^* = \sum_j \mathbf{t}_j \rho^{j-1} \qquad\qquad \mathbf{f}^* = \sum_j \mathbf{f}_j \rho^{j-1}.$$

Notice that by Schwartz-Zippel lemma we that $\mathbf{f}^* \prec \mathbf{t}^*$ implies $\mathbf{F} \prec \mathbf{T}$ with overwhelming probability. Thus, we could run a vector lookup argument over $(\mathbf{f}^*, \mathbf{t}^*)$, thanks to the linear homomorphic property of the commitment scheme the verifier can compute commitments to $\mathbf{f}^*$ and $\mathbf{t}^*$ and verify the proof. Notice the prover time complexity is $\mathsf{poly}(n, d, \lambda)$ thanks to the $\mathbb{F}$-linearity of the pre-computation algorithm. However, the verification time is linear in in the number of columns. We show in the next section how to restore succinct verification time and commitment size.

5.2 Our Scheme

In Fig. 2 we describe our scheme $\mathsf{mtx}[\mathsf{CP}]$ that runs internally a lookup argument CP for KZG-based vector commitment scheme. The proof of the following theorem is in [7, Appendix D]. In the description of the scheme, we let $\mathbb{K}$ (resp. $\mathbb{H}$) be a multiplicative subgroup of $\mathbb{F}$ of order $N \cdot d$ (resp. of order $n \cdot d$), we let $\omega := \omega_{n \cdot d}$ be the fixed generator for $\mathbb{H}$ and we consider the following matrices and polynomial:

1. the matrix $\mathbf{R} \in \mathbb{F}^{N \times d}$ where $R_{i,j} = i$,
2. for any k the matrix $\mathbf{C}^{(k)} \in \mathbb{F}^{k \times d}$ where $C_{i,j} = j$.
3. Let $\nu_{\mathbb{H}}(X)$ be the vanishing polynomial of $\mathbb{H} = \{\omega^{d \cdot i + j} : j \in [1, d-1], i \in [n]\}$.

Theorem 2. *The lookup argument* $\mathsf{mtx}[\mathsf{CP}]$ *defined in Fig. 2 is knowledge-sound in the AGM and ROM under the* $(N{\cdot}d, N{\cdot}d)$*-PDL assumption and assuming that* CP *is knowledge-sound. Furthermore, the protocol is zero-knowledge assuming* CP *is zero-knowledge.*

A Row-Column Matrix Lookup Argument. In [7, Appendix D.1] we consider the rows-columns sub-matrix relation where $\mathbf{F} \prec \mathbf{T}$ if and only if there exist (multi)sets $R = \{r_1, \ldots, r_n\}$ and $C = \{c_1, \ldots, c_d\}$ with $\mathbf{F}_{i,j} = \mathbf{T}_{r_i, c_j}$, and give an *rows-columns* matrix-lookup argument system $\mathsf{mtx}^*[\mathsf{CP}]$ for such a relation. Briefly, the main difference with the scheme in this section is that we commit to an additional vector $\bar{\boldsymbol{\sigma}}^C$ which is the concatenation of the vector $(c_1, \ldots, c_d)$ for n times, prove in zero-knowledge its tensor structure, and show that $\bar{\mathbf{f}}^* = \bar{\mathbf{f}} + \rho \cdot \bar{\boldsymbol{\sigma}}^C + \rho^2 \cdot \bar{\boldsymbol{\sigma}}$ is a sub-vector of $\bar{\mathbf{t}}^*$.

$\underline{\mathsf{Der}(\mathsf{srs}, N, d, n)}$**:**

Let $\bar{\mathbf{f}}, \bar{\mathbf{r}}_N, \bar{\mathbf{c}}_N$ and $\bar{\mathbf{c}}_n$ be vectorizations of the matrices $\mathbf{F}, \mathbf{R}, \mathbf{C}^{(N)}$ and $\mathbf{C}^{(n)}$.
Compute $\mathsf{c}_{r,N} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{r}}_N), \mathsf{c}_{c,N} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{c}}_N)$ and $\mathsf{c}_{c,n} \leftarrow \mathsf{Com}(\bar{\mathbf{c}}_n)$.
Compute $(\mathsf{ek}', \mathsf{vk}') \leftarrow \mathsf{CP.Der}(\mathsf{srs}, Nd, nd)$.
Return $(\mathsf{ek}', \mathsf{vk}_n)$ where $\mathsf{vk}_n = (\mathsf{c}_{r,N}, \mathsf{c}_{c,N}, \mathsf{c}_{c,n}, [\nu_{\bar{\mathbb{H}}}(s)]_2, \mathsf{vk}')$.

$\underline{\mathsf{Preproc}(\mathsf{srs}, \mathbf{T}, \rho_T)}$**:**

Let $\bar{\mathbf{t}}$ be vectorization of the matrix $\mathbf{T}$.
Compute $(\mathsf{aux}_{T,j})_{j\in[Nd]} \leftarrow \mathsf{CP.Preproc}(\mathsf{srs}, \bar{\mathbf{t}}, \rho_T)$,
$(\mathsf{aux}_{R,j})_{j\in[Nd]} \leftarrow \mathsf{CP.Preproc}(\mathsf{srs}, \bar{\mathbf{r}}_N)$,
$(\mathsf{aux}_{C,j})_{j\in[Nd]} \leftarrow \mathsf{CP.Preproc}(\mathsf{srs}, \bar{\mathbf{c}}_N)$.
Let $\mathsf{aux}_i = (\mathsf{aux}_{T,di+j}, \mathsf{aux}_{R,di+j}, \mathsf{aux}_{C,di+j})_{j\in[d]}$.
Return $(\mathsf{aux}_i)_{i\in[N]}$.

$\underline{\mathsf{Prove}(\mathsf{ek}, (\mathsf{c}_\mathbf{T}, \mathsf{c}_\mathbf{F}), \mathbf{F}, (\mathsf{aux}_j)_{j\in K})}$**:** //$\mathbf{T}_{|K} = \mathbf{F}$, $K = \{k_1, \ldots, k_n\}$.
Let $\mathbf{S}$ be s.t. $\mathbf{S}_{i,j} = k_i$ for $i \in [n], j \in [d]$.
Let $\sigma(X)$ be the randomized low-degree encoding over $\mathbb{H} = \langle\omega\rangle$ of the vectorization of $\mathbf{S}$.
Compute $w(X)$ such that $\sigma(\omega \cdot X) - \sigma(X) = w(X) \cdot \nu_{\bar{\mathbb{H}}}(X)$.
$(\rho, \zeta) \leftarrow \mathsf{RO}(\mathsf{vk}_n \| (\mathsf{c}_\mathbf{T}, \mathsf{c}_\mathbf{F}) \| (\mathsf{c}_{R,n}, \mathsf{c}_{R',n}, \mathsf{c}_w))$. //Fiat-Shamir challenge.
Compute $z \leftarrow \sigma(\omega \cdot \zeta)$.
Compute proofs π_R and $\pi_{R'}$ for $\hat{\mathcal{R}}_{\mathsf{eval}}(\omega \cdot \zeta, z; \sigma(X)) = 1$ and $\hat{\mathcal{R}}_{\mathsf{eval}}(\zeta, z; \sigma(\omega \cdot X)) = 1$;
Let π^* proof for $\hat{\mathcal{R}}_{\mathsf{zklkp}}((N \cdot d, n \cdot d); \varepsilon; (\bar{\mathbf{t}}^*, \bar{\mathbf{f}}^*)) = 1$ where

$$\bar{\mathbf{t}}^* = \bar{\mathbf{t}} + \rho \cdot \bar{\mathbf{c}}_N + \rho^2 \cdot \bar{\mathbf{r}}_N \qquad \bar{\mathbf{f}}^* = \bar{\mathbf{f}} + \rho \cdot \bar{\mathbf{c}}_n + \rho^2 \cdot \bar{\boldsymbol{\sigma}} \tag{13}$$

Return $\pi = ([\sigma(s)]_1, [\sigma(\omega \cdot s)]_1, [w(s)]_1, \pi_R, \pi_{R'}, \pi^*, z)$.

$\underline{\mathsf{Verify}(\mathsf{vk}_n, (\mathsf{c}_\mathbf{T}, \mathsf{c}_\mathbf{F}), \pi)}$**:**

Parse the proof $\pi = (\mathsf{c}_{R,n}, \mathsf{c}_{R',n}, \mathsf{c}_w, \pi_R, \pi_{R'}, \pi^*, z)$.
$(\rho, \zeta) \leftarrow \mathsf{RO}(\mathsf{vk}_n \| (\mathsf{c}_\mathbf{T}, \mathsf{c}_\mathbf{F}) \| (\mathsf{c}_{R,n}, \mathsf{c}_{R',n}, \mathsf{c}_w))$. //Fiat-Shamir challenge.
Compute $\mathsf{c}^*_\mathbf{T} \leftarrow \mathsf{c}_\mathbf{T} + \rho \mathsf{c}_{c,N} + \rho^2 \mathsf{c}_{r,N}$ and $\mathsf{c}^*_\mathbf{F} \leftarrow \mathsf{c}_\mathbf{F} + \rho \mathsf{c}_{c,n} + \rho^2 \mathsf{c}_{R,n}$.
Return 1 if the following checks hold (else 0):

(i) $\mathsf{Verify}_{\mathsf{eval}}(\mathsf{ck}, (\mathsf{c}_{R,n}, \omega \cdot \zeta, z)) = 1$,
(ii) $\mathsf{Verify}_{\mathsf{eval}}(\mathsf{ck}, (\mathsf{c}_{R',n}, \zeta, z)) = 1$,
(iii) $e(\mathsf{c}_{R',n} - \mathsf{c}_{R,n}, [1]_2) = e(\mathsf{c}_w, [\nu_{\bar{\mathsf{k}}}(s)]_2)$,
(iv) $\mathsf{CP.Verify}(\mathsf{srs}, \mathsf{vk}', (\mathsf{c}^*_\mathbf{T}, \mathsf{c}^*_\mathbf{F}), \pi^*) = 1$.

Fig. 2. Our Matrix Lookup Argument $\mathsf{mtx}[\mathsf{CP}]$.

5.3 Concrete Efficiency

In Table 1, we describe the complexity of proving a matrix lookup in a table T described by a matrix of size $N \times d$. The size of the submatrix we are looking up in the larger table is $n \times d$. In [7, Appendix F], we describe a breakdown of efficiency measurements for our fully zero-knowledge construction ($\mathsf{mtx}[\mathsf{zkcq}^+]$). The values for [13] are taken directly from the paper, the number of pairings in verification is computed by simple inspection of the protocol, the extra $O(\log nd)$ factor in the number of exponentiations in $\mathbb{G}_1$ for the prover arises from their sub-protocol adapted from [38].

Table 1. Summary of efficiency of our constructions for matrix lookups. The relation considered is parametrized with table size of size $N \times d$ and looked-up submatrix of size $n \times d$. $\mathbb{P}$ is the cost of one pairing. Proof size includes commitment to the witness.

Scheme	Preprocessing	Proof size	Time (P)	Time (V)
$\mathsf{mtx}^{\mathsf{longprf}}[\mathsf{zkcq}^+]$ (Sect. 5.1)	$O(dN \log N)\mathbb{F}, \mathbb{G}$	$(d+9)\mathfrak{g}_1 + 1\mathfrak{f}$	$O(nd)\mathbb{G}_1 + O(nd \log n)\mathbb{F}$	$d\mathbb{G}_1 + 7\mathbb{P}$
$\mathsf{mtx}[\mathsf{zkcq}^+]$ (Fig. 2)	$O(dN \log dN)\mathbb{F}, \mathbb{G}$	$16\mathfrak{g}_1 + 2\mathfrak{f}$	$O(nd)\mathbb{G}_1 + O(nd \log(nd))\mathbb{F}$	$13\mathbb{P}$
[13]	$O(dN \log dN)\mathbb{F}, \mathbb{G}$	$20\mathfrak{g}_1 + 6\mathfrak{f}$	$O(nd \log nd)\mathbb{G}_1 + O(nd \log(nd))\mathbb{F}$	$23\mathbb{P}$

6 Zero-Knowledge Decision Tree Statistics

A decision tree is an algorithm that, upon an input, performs a finite sequence of adaptive queries on the input and eventually outputs a value. Concretely, we consider binary decision trees where the inputs are vectors in $[B]^d$ for natural numbers d and B, where the queries are comparisons and the outputs (often called the labels) are in $[B]$. We let N_{tot} be the number of nodes in a decision tree T, and we index the root node with 1. A binary tree with N_{tot} nodes and where each node has either zero children or exactly two children, has $N_{\mathsf{leaf}} := (N_{\mathsf{tot}} + 1)/2$ leaf nodes, and the remaining $N_{\mathsf{int}} = N_{\mathsf{tot}} - N_{\mathsf{leaf}}$ nodes are called internal nodes (including the root node). We index the internal nodes of the decision tree with numbers in $[N_{\mathsf{int}}]$. The computation of a decision tree T upon input $\mathbf{x}$, denoted as $\mathsf{T}(\mathbf{x})$, consists of a traversal of the tree from the root node to a leaf. During the traversal, the computation fetches, from each internal node i, a threshold t_i and a feature index $e_i \in [d]$. If $x_{e_i} < t_i$, the computation continues recurring on the left child of node i, and otherwise, to the right child. Once reaches a leaf, the computation outputs the label v_i assigned to the leaf i as the final output.

Therefore, seen as a data structure, a decision tree T is made by a binary tree (namely, the *structure* of the tree), by the values d_i, t_i for each internal node i, and by the label v_i for each leaf node i. We refer to this encoding of a tree as the *standard encoding*. We define $\mathcal{T}_{N_{\mathsf{tot}},B,d}$ to be the set of decision trees with N_{tot} nodes that maps vector in $[B]^d$ to the co-domain $\mathbb{F}$.

Quasi-Complete Decision Tree. We define the notion of *quasi-complete* decision tree. The difference with a standard tree is that during the traversal, the

computation fetches from each internal node i two vectors $\mathbf{E}_i$ and $\mathbf{T}_i$, we call the vector $\mathbf{E}_i \in \{0,1\}^d$ the feature vector associated to the node i and vector $\mathbf{T}_i \in [B]^d$ the threshold vector associated to the node i. The computation continues recurring on the left child of node i if $\forall j \in [d] : \mathbf{E}_{i,j} = 1 \Rightarrow x_j < \mathbf{T}_{i,j}$, on to the right child of the node i if $\forall j \in [d] : \mathbf{E}_{i,j} = 1 \Rightarrow x_j \geq \mathbf{T}_{i,j}$, or outputs $\perp$ if neither of the two conditions holds.

Similarly to decision trees, we define $\mathcal{T}^*_{N_{\mathsf{tot}},B,d}$ to be the set of quasi-complete decision trees with N_{tot} nodes that maps feature vector in $[B]^d$ to the co-domain $\mathbb{F}$. Notice that when for any node j the (row) vector $\mathbf{E}_j$ is an elementary vector (namely with only one position set to 1) then the quasi-complete decision tree is indeed a standard decision tree thus $\mathcal{T}_{N_{\mathsf{tot}},B,d} \subset \mathcal{T}^*_{N_{\mathsf{tot}},B,d}$.

The class of quasi-complete decision trees defines a correct but not complete computational model. In fact, every input is either correctly labelled to one label or to the error message $\perp$. Being a more general class of computation than standard decision trees, it is easier to decide whether a data structure is a quasi-complete decision tree than to decide if it is a standard decision tree. This allows for faster prover time. On the other hand, an adversary that commits to (strictly) quasi-complete decision tree (namely, a decision tree in $\mathcal{T}^*_{N_{\mathsf{tot}},B,d} \backslash \mathcal{T}_{N_{\mathsf{tot}},B,d}$) cannot prove contradicting statements, in particular, we require that it cannot prove any statistics on an input $\mathbf{x}$ whenever $\mathsf{T}(\mathbf{x}) = \perp$.

6.1 Security Model

We consider the scenario where a *model producer* commits to a decision tree T, the model producer can delegate the computation of statistics on a set of data points and predictions over T to a server, a user can obtain such statistics. Informally, we require integrity of the computation, namely the statistics are correctly computed over the set of data points and predictions over the committed decision tree T, and privacy, namely the user does not learn anything more than the validity of such statistics.

We consider an adversarial model where either the model producer and the server can be corrupted, or the user is corrupted. Previous work considered only the case where the model producer is honest [40] (and either the server or user are corrupted). Notice that a corrupted model producer could commit to a *useless/bogus* decision tree. Unfortunately, we cannot do anything to prevent that. On the other hand, we would like to prevent the corrupted model producer and corrupted server can convince the user of the validity of *incoherent* statistics. For example, an attacker should not be able to convince the user that simultaneously $\mathsf{T}(\mathbf{x}) = 1$ and $\mathsf{T}(\mathbf{x}) = 0$ for a data point $\mathbf{x}$.

To formalize such property, we use the notion of knowledge soundness for argument systems. In particular, we require that whenever the verifier is convinced (w.r.t. a commitment c) of the statistic over a set of data points, there must exist an extractor that outputs an opening of the commitment to a decision tree T where such a statistic over such data is correct. Notice the commitment to the decision tree is binding. Thus we must obtain coherent statistics over many

queries on the same committed decision tree. To optimize the efficiency of the statistic evaluations, we split in two parts the generation of a valid commitment from the evaluation of a proof for a given tuple statistic/data points.

Definition 7. *Let $\mathcal{S}$ be an arbitrary set of tuples (S, m) such that $S : [B]^m \to \{0,1\}^*$ and $m \in \mathbb{N}$ where S is an efficiently computable function (a statistic). A (commit-and-prove)* decision-tree-statistic argument *for a set of statistics $\mathcal{S}$ is a tuple* $\mathsf{zkDT} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Der}, \mathsf{Prove}, \mathsf{Verify})$ *where:*

(i) $\mathsf{CS}_{DT} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{VerCom})$ *define an extractable commitment scheme for the domain $\mathcal{T}^*$ of (quasi-complete) decision tree. In particular,* KGen *takes in input a natural number N_{tot} the maximum number of nodes, and the natural numbers B and d, besides the security parameter and generates a commitment key for the set $\mathcal{T}^*_{B,d,N_{\mathsf{tot}}}$.*

(ii) $\mathsf{CP}_{DT} = (\mathsf{Der}, \mathsf{Prove}, \mathsf{Verify})$ *define a Universal CP-SNARK for the indexed CP-relation $\hat{\mathcal{R}}_{\mathsf{DTstat}}$ defined below.*

$$\hat{\mathcal{R}}_{\mathsf{DTstat}} = \left\{ \mathsf{pp}; (S, m); y, (\mathbf{x}_j)_{j\in[m]}); \mathsf{T} : \begin{array}{c} y = S(\mathsf{T}(\mathbf{x}_1), \ldots, \mathsf{T}(\mathbf{x}_m)), \\ \forall i : \mathsf{T}(\mathbf{x}_i) \neq \perp, \quad (S, m) \in \mathcal{S} \end{array} \right\}.$$

6.2 The Extended Encoding of Decision Trees

We introduce an alternative encoding of a decision tree as a data structure before presenting our zero-knowledge decision-tree statistics argument. We follow the work of Chen *et al.* [11]. In particular, we define a d-dimensional *box* as a tuple of vectors in $[B+1]^d$, where the first vector defines the *left bounds* and the second vector defines the *right bounds*. We say that a vector $\mathbf{x} \in [B]^d$ *is contained* in a box $(\mathbf{b}^{\vdash}, \mathbf{b}^{\dashv})$ if $\mathbf{b}^{\vdash} \leq \mathbf{x} < \mathbf{b}^{\dashv}$. We can assign to each node of a decision tree a $d-$dimensional box. In particular, we denote with $(\mathbf{N}_i^{\vdash}, \mathbf{N}_i^{\dashv})$ the box assigned to the i-th node in the tree and with $\mathbf{N}^{\vdash}, \mathbf{N}^{\dashv}$ the tuple of matrices of all the boxes of a decision tree (mapping the i-th row to the box of i-th node).

We can associate a (quasi-complete) decision tree to a tuple of matrices, below we define such a relation:

Definition 8. *Given a quasi-complete decision tree T with N_{tot} nodes and given matrices $\mathbf{N}^{\vdash}, \mathbf{N}^{\dashv}$, we say that $(\mathbf{N}^{\vdash}, \mathbf{N}^{\dashv})$ is a* boxes-encoding *of T if*

1. *$\mathbf{N}_1^{\vdash} = \mathbf{0}$ and $\mathbf{N}_1^{\dashv} = \mathbf{B} + \mathbf{1}$, where $\mathbf{0}$ (resp. $\mathbf{1}$ and $\mathbf{B}$) is the vector of all 0 (resp. of all 1 and of all B).*
2. *Let $p \in [N_{\mathsf{int}}]$ be the index of a node and let l and r respectively be the indexes of the left child and right child of the node with index p.*

$$\mathbf{N}_l^{\vdash} - \mathbf{N}_p^{\vdash} = \mathbf{0} \qquad \mathbf{N}_r^{\vdash} - \mathbf{N}_p^{\vdash} = \mathbf{0} \tag{14}$$

$$\mathbf{E}_p \circ (\mathbf{N}_l^{\vdash} - \mathbf{T}_p) = \mathbf{0} \qquad \mathbf{E}_p \circ (\mathbf{N}_r^{\vdash} - \mathbf{T}_p) = \mathbf{0} \tag{15}$$

$$(\mathbf{1} - \mathbf{E}_p) \circ (\mathbf{N}_l^{\dashv} - \mathbf{N}_p^{\dashv}) = \mathbf{0} \qquad (1 - \mathbf{E}_p) \circ (\mathbf{N}_r^{\dashv} - \mathbf{N}_p^{\dashv}) = \mathbf{0} \tag{16}$$

The computation, through a boxes-encoding, of a decision tree $\mathsf{T}(\mathbf{x})$ consists in finding the index k of the leaf whose box contains $\mathbf{x}$ and outputs the label associated with such a leaf. For a quasi-complete decision tree, such an index k might not exist. We formalize this in the next definition and prove such a computational equivalence in the next lemma whose proof is in [7, Appendix E].

Definition 9. *Let* T *be a quasi-complete decision tree with* N_{tot} *nodes (with domain* $[B]^d$*) and* $(\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow})$ *be a boxes-encoding of* T*. For any* $\mathbf{x} \in [B]^d$*, if* $\mathbf{x}$ *is contained in the box of a leaf of* T *define the index of the leaf as* $k_{\mathsf{T}}(\mathbf{x})$ *such that* $\mathbf{x}$ *is contained in* $(\mathbf{N}^{\leftarrow}_{k_{\mathsf{T}}(\mathbf{x})}, \mathbf{N}^{\rightarrow}_{k_{\mathsf{T}}(\mathbf{x})})$ *else* $k_{\mathsf{T}}(\mathbf{x})$ *is set to* $\perp$*.*

Whenever it is clear from the context, we will omit the subscript T and write $k(\mathbf{x})$ to refer to such an index.

Lemma 3. *Let* T *be a quasi-complete decision tree with* N_{tot} *nodes and* $(\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow})$ *be a boxes-encoding of* T*. Let* $\mathbf{v}$ *be the vector of the labels assigned to the leaf nodes of* T*, namely for any* $i \in [N_{\mathsf{int}} + 1, N_{\mathsf{tot}}]$*, we have* v_i *as the label assigned to the* i*-th leaf. For any* $\mathbf{x} \in [B]^d$*,* $\mathsf{T}(\mathbf{x}) = v_{k(\mathbf{x})}$ *or* $\mathsf{T}(\mathbf{x}) = \perp$*.*

As corollary of the above lemma, we have that the boxes of leaf do not overlap because no vector $\mathbf{x}$ can be contained in more than one of the boxes of the leaves.

Before giving the next definition, we set some notation: given a decision tree, we say that node p splits at coordinate $i^* \in [d]$ if i^* is a coordinate where p's left and right child boundaries are different, namely, $\mathbf{N}^{\rightarrow}_{p,i} \neq \mathbf{N}^{\rightarrow}_{\ell,i}$ and $\mathbf{N}^{\leftarrow}_{p,i} \neq \mathbf{N}^{\leftarrow}_{r,i}$ where ℓ and r are the left and right child of p. We are ready to describe our (more redundant but ZKP-friendly) encoding of a quasi-complete decision tree.

Definition 10. *Let* T *be a quasi-complete decision tree with* N_{tot} *nodes. Let* $\mathcal{T} = (\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow}, \mathbf{v}, \mathbf{L}, \mathbf{R}, \mathbf{E})$ *be a tuple of matrices (described below). We say that* $\mathcal{T}$ *is an extended encoding of* T *if the following conditions hold:*

(i) $(\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow})$ *is a boxes-encoding of* T*;*
(ii) $\mathbf{v}$ *is the vector of the labels assigned to the leaf nodes of* T*;*
(iii) $\mathbf{L}$ *(resp.* $\mathbf{R}$*) is the* $N_{\mathsf{int}} \times N_{\mathsf{tot}}$ *bit matrix whose* p*-th row is the elementary vector* $\mathbf{e}_{\ell}^{\mathsf{T}}$ *(resp.* $\mathbf{e}_{r}^{\mathsf{T}}$*) if* ℓ *is the left (resp.* r *is the right) child of node* p*'s in* T*,*
(iv) $\mathbf{E} \in \{0,1\}^{N_{\mathsf{int}} \times d}$ *is the bit matrix such that its* p*-th row and* i *column is* 1 *iff the node* p *splits at coordinate* i*.*

Let Encode *be the algorithm that, given a quasi-complete decision tree* T*, computes the extended encoding of* T*.*

Let the matrices $\mathbf{P}^{\leftarrow}, \mathbf{P}^{\rightarrow} \in \mathbb{F}^{N_{\mathsf{int}} \times d}$ describe the boxing encodings of the internal nodes, and $\mathbf{F}^{\leftarrow}, \mathbf{F}^{\rightarrow} \in \mathbb{F}^{N_{\mathsf{leaf}} \times d}$ describe the boxing encodings of the leaves. Thus:

$$\mathbf{N}^{\leftarrow} = \begin{pmatrix} \mathbf{P}^{\leftarrow} \\ = \\ \mathbf{F}^{\leftarrow} \end{pmatrix} \text{ and } \mathbf{N}^{\rightarrow} = \begin{pmatrix} \mathbf{P}^{\rightarrow} \\ = \\ \mathbf{F}^{\rightarrow} \end{pmatrix}.$$

The function Encode in Definition 10 is injective but not surjective. In the next lemma (whose proof is in [7, Appendix E]), we give sufficient conditions for belonging in the image of Encode.

Lemma 4. *Consider a tuple* $(\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow}, \mathbf{L}, \mathbf{R}, \mathbf{E}, \mathbf{v})$ *such that the following constraints hold:*

a) The following equations hold:

$$\mathbf{N}_1^{\leftarrow} = \mathbf{0}, \mathbf{N}_1^{\rightarrow} = \mathbf{B} + \mathbf{1}, \tag{17}$$
$$\mathbf{L} \cdot \mathbf{N}^{\leftarrow} = \mathbf{P}^{\leftarrow}, \mathbf{R} \cdot \mathbf{N}^{\rightarrow} = \mathbf{P}^{\rightarrow}, \tag{18}$$
$$\mathbf{E} \circ (\mathbf{L} \cdot \mathbf{N}^{\rightarrow} - \mathbf{R} \cdot \mathbf{N}^{\leftarrow}) = \mathbf{0} \tag{19}$$
$$(\mathbf{1} - \mathbf{E}) \circ (\mathbf{P}^{\leftarrow} - \mathbf{R} \cdot \mathbf{N}^{\leftarrow}) = \mathbf{0}, \qquad (\mathbf{1} - \mathbf{E}) \circ (\mathbf{P}^{\rightarrow} - \mathbf{L} \cdot \mathbf{N}^{\rightarrow}) = \mathbf{0} \tag{20}$$

b) All the boxes are not empty. Namely, for all i, j *we have* $\mathbf{N}^{\leftarrow}_{i,j} < \mathbf{N}^{\rightarrow}_{i,j}$.

c) The matrix $\begin{pmatrix} \mathbf{L} \\ \mathbf{R} \end{pmatrix}$ *is a (row) permutation of the (squared) matrix* $(\mathbf{0} \| \mathbf{I}_{N_{\mathsf{tot}}-1})$ *(the matrix whose rows are the row vectors* $(\mathbf{e}_i)_{i \in [2, N_{\mathsf{tot}}]}$ *of length* N_{tot}*).*

Then there exists a quasi-complete decision tree T *with* N_{tot} *nodes such that* $\mathsf{Encode}(\mathsf{T}) = (\mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow}, \mathbf{L}, \mathbf{R}, \mathbf{E}, \mathbf{v})$.

6.3 Extractable Commitment to Decision Trees

In a nutshell our commitment procedure on input a decision tree computes the encoding described in Sect. 6.2, then it commits to the matrices $\mathbf{F}^{\leftarrow}, \mathbf{F}^{\rightarrow}$ and $\mathbf{v}$ and prove in zero-knowledge the constraints from Lemma 4. We can implement the latter zero-knowledge proof using a general-purpose R1CS circuit describing the constraints of the lemma, however, the size of the circuit would be $O(dN_{\mathsf{tot}}^2)$, in fact, we would need to commit to the remaining matrices $\mathbf{P}^{\leftarrow}, \mathbf{P}^{\rightarrow}, \mathbf{L}, \mathbf{R}$ and $\mathbf{E}$ and we would need already $O(dN_{\mathsf{tot}}^2)$ multiplication gates for Eq. (18). We show how to remove the quadratic dependency from the number of total nodes. The main idea is to notice that $\mathbf{L}$ and $\mathbf{R}$ have sparsity linear in N_{tot}, thus we can use techniques from [39] to commit to such sparse matrices and then prove in zero-knowledge that the constraints in Item c) of Lemma 4 hold for the committed matrices. The remaining constraints can be proved in $O(dN_{\mathsf{tot}} \log(dN_{\mathsf{tot}}))$.

The Building Blocks. Consider the following (indexed) CP-relations:

$$\hat{\mathcal{R}}_{\mathsf{lin}} = \{\mathsf{pp}; \varepsilon; (\mathbf{M}, \mathbf{N}, \mathbf{R}) : \mathbf{M} \cdot \mathbf{N} = \mathbf{R}\} \tag{21}$$
$$\hat{\mathcal{R}}_{\mathsf{had}} = \{\mathsf{pp}; \varepsilon; (\mathbf{M}, \mathbf{N}) : \mathbf{M} \circ \mathbf{N} = \mathbf{0}\} \tag{22}$$
$$\hat{\mathcal{R}}_{\mathsf{perm}} = \left\{\mathsf{pp}; (N, i(X)); \varepsilon; p(X) : \exists \pi, \forall j \in [N] : i(\pi(\omega^j)) = p(\omega^j)\right\} \tag{23}$$
$$\hat{\mathcal{R}}_{\mathsf{shift}} = \left\{\mathsf{pp}; S; \varepsilon; (\mathbf{v}, \mathbf{u}) : \mathbf{v}_i = \mathbf{u}_{(i+S \pmod{|\mathbf{u}|})}\right\} \tag{24}$$
$$\hat{\mathcal{R}}_{\mathsf{rng}} = \left\{\mathsf{pp}; (B, n, d); \varepsilon; \mathbf{X} : \mathbf{X} \in [B]^{n \times d}\right\} \tag{25}$$
$$\hat{\mathcal{R}}_{\mathsf{sm}} = \left\{\mathsf{pp} : K; \varepsilon; \mathbf{M} : \mathbf{M}_{|K} = \mathbf{0}\right\} \tag{26}$$

Our scheme uses CP-SNARKs for all the relations above as building blocks. The first three relations are standard, and CP-SNARKs for them can be found in the related work. Given a CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{lin}}$, we can define a CP-SNARK for

$\hat{\mathcal{R}}_{\mathsf{shift}}$ in fact that the shifting operator can be described through a linear transformation. The latter linear transformation can be public, thus the underlying CP-SNARK (for $\hat{\mathcal{R}}_{\mathsf{lin}}$) does not need to be zero-knowledge w.r.t. the first matrix $\mathbf{M}$, in particular, a commitment to such a matrix could be part of the index polynomials. A CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{rng}}$ can be realized using our lookup argument and considering the table $\mathbf{b} = (j)_{j\in[B]}$ and proving that the vectorization $\bar{\mathbf{x}}$ of $\mathbf{X}$ is such that $\bar{\mathbf{x}} \prec \mathbf{b}$. Finally, a CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{sm}}$ can be easily realized by committing to a matrix $\bar{\mathbf{T}}$ such that $\bar{\mathbf{T}}_K = \mathbf{T}$ and 0 everywhere else and to the vanishing polynomial in ν_K in $\mathbb{G}_2$ as part of the index. At proving time, the prover returns as proof a commitment to the quotient polynomial q such that $f'(X) = q(X) \cdot \nu_K(X)$ where $f'(X)$ is the polynomial associated to the matrix $\mathbf{M} - \bar{T}$. At verification time the verifier checks $e(\mathsf{c}_{\mathbf{M}} - \mathsf{c}_{\bar{\mathbf{T}}}, [1]_2) = e(\pi, [\nu_K]_2)$.

For the CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{lin}}$, we require two different commitment schemes, one for the first matrix and one for the other two. In particular, we consider an alternative way to commit to matrices following the work of [33,39]. Let $\mathbf{M}$ be a *basic matrix*, namely a matrix whose rows are elementary vectors. Let $\mathbb{H}$ be any fixed subgroup with $|\mathbb{H}| \geq N_{\mathsf{tot}}$[8] of $\mathbb{F}$ with generator ω. For any basic matrix $\mathbf{M} \in \{0,1\}^{n\times k}$ and $n, k \in \mathbb{N}$, let $col_{\mathbf{M}}(X)$ be the (low-degree) polynomial such that $col_{\mathbf{M}}(\omega^i) = \omega^j$ where the i-th row of $\mathbf{M}$ is the vector $\mathbf{e}_j^\top$ (notice that $col_{\mathbf{M}}$ is the LDE of the vector whose i-th element is the value ω^j). We define the sparse (hiding) commitment of a matrix $\mathbf{M}$ as a (hiding) polynomial commitment of $col_{\mathbf{M}}$. Namely, we define:

$$\mathsf{sparseCom}(\mathsf{ck}, \mathbf{M}, \rho) := \mathsf{Com}(\mathsf{ck}, col_{\mathbf{M}}, \rho).$$

Notice that, by the above definition, a sparse commitment to a basic matrix $\mathbf{M}$ has a dual interpretation (as a sparse matrix or as a vector *col*).

Let $\mathsf{CP}_{\mathsf{lin}}$ be a CP-SNARK for the $\hat{\mathcal{R}}_{\mathsf{lin}}$ relation where the first matrix is committed using $\mathsf{sparseCom}$ while the other matrices are committed with the matrix commitment scheme from Sect. 2.3. An instantiation of such a scheme can be found for the matrix-times-vector case (namely, $\mathbf{N} \in \mathbb{F}^{n\times 1}$) in Baloo by [39] (see Sects. 5.2, 5.3 and 5.4 of the paper). We show a generalization to matrix-times-matrix case in [7, Appendix E.4]. We write $\boxed{\mathbf{M}}$ to underline that the matrix $\mathbf{M}$ is committed with a sparse matrix commitment. For example, we can write $(\mathsf{pp}, \varepsilon; \boxed{\mathbf{M}}, \mathbf{N}, \mathbf{R}) \in \hat{\mathcal{R}}_{\mathsf{lin}}$ to identify the statement that there are commitments $\mathsf{c}_M, \mathsf{c}_N, \mathsf{c}_R$ where the first is a sparse matrix commitment and that open to $\mathbf{M}, \mathbf{N}$ and $\mathbf{R}$ with $\mathbf{M} \cdot \mathbf{N} = \mathbf{R}$.

Let $\mathsf{CP}_{\mathsf{had}}$ be a CP-SNARK for the $\hat{\mathcal{R}}_{\mathsf{had}}$ relation where all the matrices are committed using the commitment scheme from Sect. 2.3. Notice that a CP-SNARK for our matrix commitment scheme for such a CP-relation derives directly from CP-SNARK for vector commitment. Finally, let $\mathsf{CP}_{\mathsf{perm}}$ be a CP-SNARK for the CP-relation $\hat{\mathcal{R}}_{\mathsf{perm}}$. The permutation argument of Plonk [19] is a CP-SNARK for such a relation.

[8] Alternatively, we can consider the same subgroup used for the matrix commitment and thus $|\mathbb{H}| = N_{\mathsf{tot}} \cdot d$.

The Extractable Commitment to Decision Tree. We define our extractable commitment scheme for the domain of quasi-complete decision trees. The main idea is, as part of the proof of opening, to commit to the matrices $\mathbf{L}$ and $\mathbf{R}$ through sparse commitments to basic matrices and then prove the linear relations from Lemma 4 in zero-knowledge with a complexity that is linear in the sparsity of the matrices and the dimension d. The additional constraints on the two matrices $\mathbf{L}$ and $\mathbf{R}$ are proved using the permutation argument. To improve readability, we list below shortcuts used in the protocol's description.

$$\bar{\mathbf{F}}_{\leftarrow} := \begin{pmatrix} \mathbf{0} \\ \hline \mathbf{F}^{\leftarrow} \end{pmatrix},\ \bar{\mathbf{F}}_{\rightarrow} := \begin{pmatrix} \mathbf{0} \\ \hline \mathbf{F}^{\rightarrow} \end{pmatrix},\ \bar{\mathbf{P}}_{\leftarrow} := \begin{pmatrix} \mathbf{P}^{\leftarrow} \\ \hline \mathbf{0} \end{pmatrix},\ \bar{\mathbf{P}}_{\rightarrow} := \begin{pmatrix} \mathbf{P}^{\rightarrow} \\ \hline \mathbf{0} \end{pmatrix},$$
$$\bar{\mathbf{L}} := \begin{pmatrix} \mathbf{L} \\ \hline \mathbf{0} \end{pmatrix},\ \bar{\mathbf{R}} := \begin{pmatrix} \mathbf{R} \\ \hline \mathbf{0} \end{pmatrix},\ \underline{\mathbf{R}} := \begin{pmatrix} \mathbf{0} \\ \hline \mathbf{R} \end{pmatrix},\ \bar{\mathbf{E}} := \begin{pmatrix} \mathbf{E} \\ \hline \mathbf{0} \end{pmatrix}$$

The padding for the matrices make them all to have N_{tot} rows. Moreover, we let $\mathbf{B}$ be the matrix whose first row is the vector $(B+1, \ldots, B+1)$ and the remaining rows are set to $\mathbf{0}$, and we let $b(X)$ be the LDE of the vectorization of such a

$\underline{\mathsf{KGen}(1^\lambda, (N_{\mathsf{tot}}, B, d))}$:
Sample a type-3 pairing group pp_G with security level λ.
Set $\mathsf{ck}' \leftarrow (\mathsf{pp}_G, ([s^i]_1)_{i\in[N_1]}, ([s^i]_2)_{i\in[N_2]}$ for random secrets $s \leftarrow\!\$\ \mathbb{Z}_q$.
Let $\mathcal{N}_1 := [N_{\mathsf{int}}], \mathcal{N}_2 := (N_{\mathsf{int}}, N_{\mathsf{tot}}], \mathcal{N}_3 := \{1\}$, for $i \in [3]$ compute $\mathsf{srs}_{\mathsf{sm},1} \leftarrow \mathsf{CP}_{\mathsf{sm}}.\mathsf{Der}(\mathsf{ck}', \mathcal{N}_i)$.
Compute $\mathsf{srs}_{\mathsf{perm}} \leftarrow \mathsf{CP}_{\mathsf{perm}}.\mathsf{Der}(\mathsf{ck}', (N_{\mathsf{tot}} - 1, id))$.
Compute $\mathsf{srs}_{\mathsf{rng}} \leftarrow \mathsf{CP}_{\mathsf{rng}}.\mathsf{Der}(\mathsf{ck}', (B, N_{\mathsf{tot}}, d))$.
Compute $\mathsf{srs}_{\mathsf{shift}} \leftarrow \mathsf{CP}_{\mathsf{shift}}.\mathsf{Der}(\mathsf{ck}', N_{\mathsf{int}})$.
Return $\mathsf{ck} := (\mathsf{ck}', [b(s)]_1, \mathsf{srs}_{\mathsf{perm}}, \mathsf{srs}_{\mathsf{rng}}, \mathsf{srs}_{\mathsf{shift}}, (\mathsf{srs}_{\mathsf{sm},j})_{j\in[3]})$.

$\underline{\mathsf{Com}(\mathsf{ck}, \mathsf{T}, \rho_{\mathsf{T}})}$:
Compute $(\mathbf{L}, \mathbf{R}, \mathbf{E}, \mathbf{N}^{\leftarrow}, \mathbf{N}^{\rightarrow}, \mathbf{v}) \leftarrow \mathsf{Encode}(\mathsf{T})$, parses ρ_{T} as $(\rho_v, \rho_{\leftarrow}, \rho_{\rightarrow})$.
$\mathsf{c}_v \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{v}, \rho_v)$. // Parse $\mathbf{v}$ as a $N_{\mathsf{tot}} \times d$ matrix whose last $d-1$ columns are empty.
$\mathsf{c}_{\leftarrow} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{F}}_{\leftarrow}, \rho_{\leftarrow}), \mathsf{c}_{\rightarrow} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{F}}_{\rightarrow}, \rho_{\rightarrow})$, $\mathsf{c}'_{\leftarrow} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{P}}_{\leftarrow}, \rho_{\leftarrow}), \mathsf{c}'_{\rightarrow} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{P}}_{\rightarrow}, \rho_{\rightarrow})$.
$\mathsf{c}_{ln} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{L}} \cdot \mathbf{N}^{\rightarrow},), \mathsf{c}_{rn} \leftarrow \mathsf{Com}(\mathsf{ck}, \bar{\mathbf{R}} \cdot \mathbf{N}^{\leftarrow})$ and $\mathsf{c}_E \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{E})$.
$\mathsf{c}_L \leftarrow \mathsf{sparseCom}(\mathsf{ck}, \bar{\mathbf{L}}), \mathsf{c}_R \leftarrow \mathsf{sparseCom}(\mathsf{ck}, \bar{\mathbf{R}})$ and $\mathsf{c}'_R \leftarrow \mathsf{sparseCom}(\underline{\mathbf{R}})$.
Let $col_{\bar{\mathbf{L}}}, col_{\bar{\mathbf{R}}}$ and $col_{\underline{\mathbf{R}}}$ be the underlying polynomials.
Prove the following statements, let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_{16})$ be the proofs.

$$\pi_1, \ldots, \pi_4 : \quad (\boxed{\bar{\mathbf{L}}}, \mathbf{N}^{\leftarrow}, \bar{\mathbf{P}}_{\leftarrow}),\ (\boxed{\bar{\mathbf{L}}}, \mathbf{N}^{\rightarrow}, \bar{\mathbf{L}} \cdot \mathbf{N}^{\rightarrow}),\ (\boxed{\bar{\mathbf{R}}}, \mathbf{N}^{\rightarrow}, \bar{\mathbf{P}}_{\rightarrow}),\ (\boxed{\bar{\mathbf{R}}}, \mathbf{N}^{\leftarrow}, \bar{\mathbf{R}} \cdot \mathbf{N}^{\leftarrow}) \in \hat{\mathcal{R}}_{\mathsf{lin}},$$
$$\pi_5, \pi_6, \pi_7 : \quad (\bar{\mathbf{E}}, \bar{\mathbf{L}} \cdot \mathbf{N}^{\rightarrow} - \bar{\mathbf{R}} \cdot \mathbf{N}^{\leftarrow}),\ (\mathbf{1} - \bar{\mathbf{E}}, \mathbf{P}^{\leftarrow} - \bar{\mathbf{R}} \cdot \mathbf{N}^{\leftarrow}),\ (\mathbf{1} - \bar{\mathbf{E}}, \mathbf{P}^{\rightarrow} - \bar{\mathbf{L}} \cdot \mathbf{N}^{\rightarrow}) \in \hat{\mathcal{R}}_{\mathsf{had}},$$
$$\pi_8, \pi_9 : \quad ((B, N_{\mathsf{tot}}, d); \mathbf{N}^{\rightarrow} - \mathbf{N}^{\leftarrow} - \mathbf{1}) \in \hat{\mathcal{R}}_{\mathsf{rng}}, \quad (N_{\mathsf{tot}} - 1, id; col_{\bar{\mathbf{L}}}(X) + col_{\underline{\mathbf{R}}}(X)) \in \hat{\mathcal{R}}_{\mathsf{perm}},$$
$$\pi_{10} : \quad (N_{\mathsf{int}}, col_{\bar{\mathbf{R}}}, col_{\underline{\mathbf{R}}}) \in \hat{\mathcal{R}}_{\mathsf{shift}},$$
$$\pi_{11}, \ldots, \pi_{16} : \quad (\mathcal{N}_1; \bar{\mathbf{F}}_{\leftarrow}), (\mathcal{N}_1; \bar{\mathbf{F}}_{\rightarrow}), (\mathcal{N}_2; \bar{\mathbf{P}}_{\leftarrow}), (\mathcal{N}_2; \bar{\mathbf{P}}_{\rightarrow}), (\mathcal{N}_3; \bar{\mathbf{P}}_{\leftarrow}), (\mathcal{N}_3; \bar{\mathbf{P}}_{\rightarrow} - \mathbf{B}) \in \hat{\mathcal{R}}_{\mathsf{sm}}.$$

Return $(\mathsf{c}_{\leftarrow}, \mathsf{c}_{\rightarrow}, \mathsf{c}_v), \pi$ where $\pi = (\mathsf{c}'_{\leftarrow}, \mathsf{c}'_{\rightarrow}, \mathsf{c}_{ln}, \mathsf{c}_{rn}, \mathsf{c}_E, \mathsf{c}_L, \mathsf{c}_R, \mathsf{c}'_R, \boldsymbol{\pi})$.

$\underline{\mathsf{Verify}(\mathsf{ck}, \mathsf{c}_{\mathsf{T}})}$:
Let $\mathsf{c}_{\mathsf{T}} = (\mathsf{c}_{\leftarrow}, \mathsf{c}_{\rightarrow}, \mathsf{c}_v, \pi)$ and parse π. Let $\mathsf{c}_{N,\leftarrow} \leftarrow \mathsf{c}_{\leftarrow} + \mathsf{c}'_{\leftarrow}$ and $\mathsf{c}_{N,\rightarrow} \leftarrow \mathsf{c}_{\rightarrow} + \mathsf{c}'_{\rightarrow}$.
1. Verify $\pi_1, \pi_2, \pi_3, \pi_4$ w.r.t. $(\mathsf{c}_L, \mathsf{c}_{N,\leftarrow}, \mathsf{c}'_{\leftarrow}), (\mathsf{c}_L, \mathsf{c}_{N,\rightarrow}, \mathsf{c}_{ln}), (\mathsf{c}_R, \mathsf{c}_{N,\rightarrow}, \mathsf{c}'_{\rightarrow}), (\mathsf{c}_R, \mathsf{c}_{N,\leftarrow}, \mathsf{c}_{rn})$.
2. Verify π_5, π_6, π_7 w.r.t. $(\mathsf{c}_E, \mathsf{c}_{ln} - \mathsf{c}_{rn}), ([1]_1 - \mathsf{c}_E, \mathsf{c}'_{\leftarrow} - \mathsf{c}_{rn}), ([1]_1 - \mathsf{c}_E, \mathsf{c}'_{\rightarrow} - \mathsf{c}_{ln})$.
3. Verify π_8, π_9 w.r.t. $((B, N_{\mathsf{tot}}, d); \mathsf{c}_{N,\rightarrow} - \mathsf{c}_{N,\leftarrow} - [1]_1)$ and $(N_{\mathsf{tot}} - 1, id; \mathsf{c}_L + \mathsf{c}'_R)$.
4. Verify π_{10} w.r.t. $(N_{\mathsf{int}}; (\mathsf{c}_R, \mathsf{c}'_R))$.
5. Verify $\pi_{11}, \ldots, \pi_{16}$ w.r.t. $(\mathcal{N}_1, \mathsf{c}_{\leftarrow}), (\mathcal{N}_1, \mathsf{c}_{\rightarrow}), (\mathcal{N}_2, \mathsf{c}'_{\leftarrow}), (\mathcal{N}_2, \mathsf{c}'_{\rightarrow}), (\mathcal{N}_3, \mathsf{c}'_{\leftarrow}), (\mathcal{N}_3, \mathsf{c}'_{\rightarrow} - [b(s)]_1)$.

Fig. 3. Our extractable commitment CS_{DT}. The value $N_1 \geq N_{\mathsf{tot}} \cdot d$, N_1 and N_2 are big enough to support all the building-block.

matrix. This polynomial can be computed in $O(d \log d)$ operations, however, for simplicity, we commit to the polynomial at key-generation phase. We let id be the low-degree polynomial that evaluates $id(\omega^i) = \omega^{i+1}$ for $i \in [N_{\mathsf{tot}} - 1]$ (equivalently, the commitment $[id(s)]_1$ is a sparse-matrix commitment to the matrix $(\mathbf{0} \| \mathbf{I}_{N_{\mathsf{tot}}-1})$).

Theorem 3. *The commitment scheme CS_{DT} defined in Fig. 3 is hiding and it is an extractable commitment scheme for the domain $\{\mathcal{T}^*_{N_{\mathsf{tot}},B,d}\}_{N_{\mathsf{tot}},d,B}$ in the AGM and assuming the building blocks are knowledge-sound and zero-knowledge.*

Efficiency. The extractable commitment in this section has constant proof size when the CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{lin}}$ is instantiated with the building block described in [7, Appendix E.4]. Its proving time is $O(dN_{\mathsf{tot}} \log(dN_{\mathsf{tot}}))$ when applied to a decision tree with d features and N_{tot} nodes. Notice that N_{tot} is usually at least one order of magnitude larger than d.

6.4 CP-SNARK for Statistics on Decision Trees

Consider the scheme CP_{DT} in Fig. 4 based on the following building blocks:

1. Let $\mathsf{CP}_{\mathsf{lkp}*}$ be a CP-SNARK for the indexed CP-relation:

$$\hat{\mathcal{R}}_{\mathsf{lkp}*} = \left\{ \mathsf{pp}; (N, d, n); \varepsilon; (\mathbf{T}_j)_{j\in[m]}, (\mathbf{F}_j)_{j\in[m]} : \begin{array}{l} (\mathbf{F}_1 \| \dots \| \mathbf{F}_m) \prec (\mathbf{T}_1 \| \dots \| \mathbf{T}_m) \\ \forall j : |\mathbf{T}_j| = N \times d, |\mathbf{F}_j| = n \times d \end{array} \right\}$$

2. Let $\mathsf{CP}_{\mathsf{rng}}$ be a CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{rng}}$ in Eq. (25).
3. Let $\mathsf{CP}_{\mathsf{stat}}$ be a CP-SNARK for the following indexed CP-relation:

$$\hat{\mathcal{R}}_{\mathsf{stat}} = \{\mathsf{pp}, (S, m); y; \mathbf{v} : S(\mathbf{v}) = y \wedge |\mathbf{v}| = m\}$$

Notice, we can easily define a CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{lkp}*}$ on top of our compiler from Sect. 5. Namely, we batch together the matrices $\mathbf{T}_j$ and the matrices $\mathbf{F}_j$ using a random challenge, as described in Sect. 5.1, and then we run our matrix lookup argument. As corollary of Theorem 3 and the theorem below, we have that the CP_{DT} and the commitment scheme CS_{DT} from the previous section define a decision-tree statistic argument.

Theorem 4. $\mathsf{CP}_{DT} = (\mathsf{Der}, \mathsf{Prove}, \mathsf{Verify})$ *in Fig. 4 defines an Universal CP-SNARK for the indexed CP-relation $\hat{\mathcal{R}}_{\mathsf{DTstat}}$.*

6.5 Efficiency and Concrete Instantiations

We discuss how to instantiate our scheme above, the resulting system has a universal trusted setup.

– $\mathsf{CP}_{\mathsf{lkp}*}$ can be instantiated with our construction $\mathsf{mtx}[\mathsf{zkcq}^+]$ from Sect. 3;

$\underline{\mathsf{Der}(\mathsf{srs}, (S,m))}$:

Compute $\mathsf{srs}_{(S,m)} \leftarrow \mathsf{CP}_{\mathsf{stat}}.\mathsf{Der}(\mathsf{ck}, (S,m))$, $\mathsf{srs}_m \leftarrow \mathsf{CP}_{\mathsf{lkp}*}.\mathsf{Der}(\mathsf{ck}, N_{\mathsf{tot}}, d, m)$.
Compute $\mathsf{srs}_{(B,m,d)} \leftarrow \mathsf{CP}_{\mathsf{rng}}.\mathsf{Der}(\mathsf{ck}, (B,m,d))$ with values B, d contained in srs.
Return the specialized SRSs.

$\underline{\mathsf{Prove}(\mathsf{srs}, (\mathsf{c}_\mathsf{T}, y, (\mathbf{x}_j)_{j\in[m]}), (\mathsf{T}, \rho_\mathsf{T}))}$:

Parse $\mathsf{c}_\mathsf{T} = (\mathsf{c}_{\leftarrow}, \mathsf{c}_{\rightarrow}, \mathsf{c}_v)$ and $\rho_\mathsf{T} = (\rho_{\leftarrow}, \rho_{\rightarrow}, \rho_v)$.
Let $k_i = k_\mathsf{T}(\mathbf{x}_i)$ and $K = \{k_1, \ldots, k_m\}$, where $k_\mathsf{T}(\cdot)$ as defined in Definition 9.
Compute matrix commitments $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3$ to the matrices $(\bar{\mathbf{F}}_{\leftarrow})_{|K}, (\bar{\mathbf{F}}_{\rightarrow})_{|K}, \mathbf{v}_{|K}$.
Compute a proof π_{zklkp} that

$$(m; \varepsilon; ((\bar{\mathbf{F}}_{\leftarrow}, \bar{\mathbf{F}}_{\rightarrow}, \mathbf{v}), (\bar{\mathbf{F}}_{\leftarrow})_{|K}, (\bar{\mathbf{F}}_{\rightarrow})_{|K}, \mathbf{v}_{|K})) \in \hat{\mathcal{R}}_{\mathsf{lkp}*}.$$

Compute a (not hiding) commitment to the matrix $\mathbf{X}$ whose rows are the vectors $(\mathbf{x}_j)_{j\in[m]}$.
Compute proofs $\pi^{\leftarrow}_{\mathsf{rng}}$ and $\pi^{\rightarrow}_{\mathsf{rng}}$ for the following two statements:

$$((B,m,d); \mathbf{X} - (\bar{\mathbf{F}}_{\leftarrow})_{|K}) \in \hat{\mathcal{R}}_{\mathsf{rng}}, \qquad ((B,m,d); (\bar{\mathbf{F}}_{\rightarrow})_{|K} - \mathbf{X} - \mathbf{1}) \in \hat{\mathcal{R}}_{\mathsf{rng}}.$$

Compute a proof π_{stat} that $((S,m); y; \mathbf{v}_{|K})) \in \hat{\mathcal{R}}_{\mathsf{stat}}$.
Return $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi_{\mathsf{zklkp}}, \pi^{\leftarrow}_{\mathsf{rng}}, \pi^{\rightarrow}_{\mathsf{rng}}, \pi_{\mathsf{stat}})$.

$\underline{\mathsf{Verify}(\mathsf{srs}, \mathsf{vk}_{(S,m)}, (\mathsf{c}_\mathsf{T}, y, (\mathbf{x}_j)_{j\in[m]}), \pi_\mathsf{T})}$:

Parse the proof $\pi_\mathsf{T} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi_{\mathsf{zklkp}}, \pi^{\leftarrow}_{\mathsf{rng}}, \pi^{\rightarrow}_{\mathsf{rng}}, \pi_{\mathsf{stat}})$.
Compute $\mathsf{c}_\mathbf{X} \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{X})$ ($\mathbf{X}$ computed from $(\mathbf{x}_j)_{j\in[m]}$) . Return 1 if the following statements hold (else 0):

1. $\mathsf{CP}_{\mathsf{lkp}*}.\mathsf{Verify}(\mathsf{srs}, \mathsf{vk}_m, (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_{\leftarrow}, \mathsf{c}_{\rightarrow}, \mathsf{c}_v), \pi_{\mathsf{zklkp}}) = 1$
2. $\mathsf{CP}_{\mathsf{rng}}.\mathsf{Verify}(\mathsf{srs}, \mathsf{vk}_{(B,m,d)}, (\mathsf{c}_\mathbf{X} - \mathsf{c}_1), \pi^{\leftarrow}_{\mathsf{rng}}) = 1$ and
   $\mathsf{CP}_{\mathsf{rng}}.\mathsf{Verify}(\mathsf{srs}, \mathsf{vk}_{(B,m,d)}, (\mathsf{c}_2 - \mathsf{c}_\mathbf{X} - [1]_1), \pi^{\rightarrow}_{\mathsf{rng}}) = 1$
3. $\mathsf{CP}_{\mathsf{stat}}.\mathsf{Verify}(\mathsf{srs}, \mathsf{vk}_{(S,m)}, (\mathsf{c}_3, y), \pi_{\mathsf{stat}}) = 1$
4. $\mathsf{VerCom}(\mathsf{ck}, \mathsf{c}_\mathsf{T}) = 1$.

Fig. 4. Our CP-SNARK CP_{DT}. The pre-processing algorithm runs the preprocessing of the matrix lookup argument on $\bar{\mathbf{F}}_{\leftarrow}, \bar{\mathbf{F}}_{\rightarrow}, \mathbf{v}$ and openings $\rho_T = (\rho_{\leftarrow}, \rho_{\rightarrow}, \rho_v)$.

- $\mathsf{CP}_{\mathsf{rng}}$ can be implemented through a (vector) lookup in a table of size B where the subvector being looked up is of size m[9];
- $\mathsf{CP}_{\mathsf{stat}}$ can be implemented through a general-purpose commit-and-prove SNARK, such as [2,8]. For concreteness, and to minimize proof size, in the remainder of this document, we consider the proof scheme CP-LunarLite from [8] (Sect. 9.4).

We can provide an upper bound on the total proof size for the instantiations above to $20\mathbb{G}_1$ elements[10] per each of the proof above (this is a loose upper bound)—see Table 1 in this work, Table 1 and Sect. 9.4 in [8]. On a concrete curve like BLS12-381 this yields a total proof size of *at most* approximately 3.84 KB (this is a generous lower bound). For comparison, the proof size in [40] is of the order of hundreds of kilobytes.

[9] The idea is to consider the table $\mathbf{b} = (j)_{j\in[B]}$ and prove, through a lookup argument, that that $\bar{\mathbf{x}} \prec \mathbf{b}$ where $\bar{\mathbf{x}}$ is the vectorization of $\mathbf{X}$.

[10] We approximate the size of field elements with that of $\mathbb{G}_1$ elements.

Decision Tree Accuracy. In the specific case of proving decision tree accuracy we prove that a decision tree is able to correctly estimate a specific fraction of a given data sample. Namely we consider the statistic that upon input $(v_j)_{j\in[m]}, (y_j)_{j\in[m]}$ computes $\sum_j \mathsf{eq}_k(v_j, y_j)/m$, $v_j = \mathsf{T}(\mathbf{x}_j)$ for $j \in [m]$ where $k \in \mathbb{N}$ is a small constant and eq_k is the function returning 1 when its two arguments, of size k, are equal[11]; otherwise it returns 0. Thanks to Theorem 4 this can be reduced to a CP-SNARK for the following relation[12]:

$$\mathcal{R}_{\mathsf{acc}} = \left\{ (m,k); \left((y_j)_{j\in[m]}, n^*\right); (v_j)_{j\in[m]} : n^* = \sum_j \mathsf{eq}_k(v_j, y_j) \right\} \tag{27}$$

Even with an R1CS-based (Rank-1 Constraint System) general purpose SNARK, the relation above can be implemented very efficiently.

Our estimates show improvements of almost one order of magnitude for proving time and two orders of magnitude for verification time for representative choices of parameters (see full version for details). Our prover runs in the order of a few seconds; our verifier in the order of 100 ms. The construction in [40] in contrast has a prover running in the order of minutes (2–5 m) and a verifier running in the order of 10 s[13].

Table 2. Comparison between our solution and [40] for zero-knowledge decision tree accuracy. Parameters are d (number of attributes), m (size of sample), $|\mathcal{H}|$ is the cost of hash function invocation (such as SHA256); $|\mathcal{H}_{\text{circ}}|$ is the cost of a hash function invocation as a circuit; $\mathbb{P}$ is the cost of one pairing. Notation $\tilde{O}(f)$ refers to $O(f \log f)$. This table does not include the one-time cost of preprocessing for the prover (see Table 1 for concrete costs). Notice that the asymptotics in the row for our construction account for just the commitment algorithm and the *extractability* proof. The asymptotics reported for [40] are actually a lower bound and do not include some additional factors in their complexity, such as tree height. Dominated factors, such as B and k (input and output size of decision tree respectively), are also not included in the asymptotics.

Scheme	Commit Time	Prover Time	Verifier Time	Proof Size
[40]	$O(N_{\text{tot}})\lvert\mathcal{H}\rvert$	$\tilde{O}(md + N_{\text{tot}} \log m + N_{\text{tot}}\lvert\mathcal{H}_{\text{circ}}\rvert)\mathbb{F}$	$O(md)\mathbb{F}$	$O(\log^2(md))\mathfrak{f}$
Our solution	$\tilde{O}(dN_{\text{tot}})(\mathbb{G}+\mathbb{F})$	$\tilde{O}(md)(\mathbb{G}+\mathbb{F})$	$O(m)\mathbb{G} + O(1)\mathbb{P}$	$O(1)(\mathfrak{g}_1 + \mathfrak{f})$

Acknowledgements. This work has received funding from the MESRI-BMBF French-German joint project named PROPOLIS (ANR-20-CYAL-0004-01), the Dutch Research Council (NWO) under Project Spark! Living Lab (439.18.453B), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under project PICOCRYPT (grant agreement No. 101001283),

[11] In typical applications of decision trees the labels are integer values belonging to a small domains, for example, either booleans or bytes.

[12] Here expressed as a sum instead of a fraction. Since the size of the sample is public this is equivalent.

[13] These estimates refer to running times on an AWS EC2 c5.9xlarge. This architecture is comparable to the one used in [40].

and from the Spanish Government MCIN/AEI/ 10.13039/501100011033/ under projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00). The last two projects are co-funded by European Union FEDER and NextGenerationEU/PRTR funds.

We thank Melek Onën for her contributions during the early stages of this project.

References

1. Ali, R.E., So, J., Avestimehr, A.S.: On polynomial approximations for privacy-preserving and verifiable RELU networks. arXiv preprint arXiv:2011.05530 (2021)
2. Aranha, D.F., Bennedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: enhanced compiling method for Pedersen-committed zkSNARK engines. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part I. LNCS, vol. 13177, pp. 584–614. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-030-97121-2_21
3. Arun, A., Setty, S., Thaler, J.: Jolt: Snarks for virtual machines via lookups. Cryptology ePrint Archive, Paper 2023/1217 (2023). https://eprint.iacr.org/2023/1217
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-17653-2_4
6. Bootle, J., Cerulli, A., Groth, J., Jakobsen, S.K., Maller, M.: Arya: nearly linear-time zero-knowledge proofs for correct program execution. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part I. LNCS, vol. 11272, pp. 595–626. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-030-03326-2_20
7. Campanelli, M., Faonio, A., Fiore, D., Li, T., Lipmaa, H.: Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees. Cryptology ePrint Archive, Paper 2023/1518 (2023). https://eprint.iacr.org/2023/1518
8. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 3–33. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-92078-4_1
9. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2075–2092. ACM Press (2019). https://doi.org/10.1145/3319535.3339820
10. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 499–530. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30617-4_17
11. Chen, H., Zhang, H., Si, S., Li, Y., Boning, D.S., Hsieh, C.: Robustness verification of tree-based models. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) NeurIPS 2019, pp. 12317–12328. Curran Associates, Inc., Red Hook (2019). https://proceedings.neurips.cc/paper/2019/hash/cd9508fdaa5c1390e9cc329001cf1459-Abstract.html

12. 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, Heidelberg (2020). https://doi.org/10.1007/978-3-030-45721-1_26
13. Choudhuri, A.R., Garg, S., Goel, A., Sekar, S., Sinha, R.: Sublonk: sublinear prover plonk. Cryptology ePrint Archive, Paper 2023/902 (2023). https://eprint.iacr.org/2023/902
14. 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
15. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
16. Feng, B., Qin, L., Zhang, Z., Ding, Y., Chu, S.: ZEN: an optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Report 2021/087 (2021). https://eprint.iacr.org/2021/087
17. 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, Heidelberg (2018). https://doi.org/10.1007/978-3-319-96881-0_2
18. 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
19. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). https://eprint.iacr.org/2019/953
20. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-Shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_14
21. 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, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
22. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-96878-0_24
23. Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530 (2022). https://eprint.iacr.org/2022/1530
24. Kang, D., Hashimoto, T., Stoica, I., Sun, Y.: Scaling up trustless DNN inference with zero-knowledge proofs. arXiv preprint arXiv:2210.08674 (2022)
25. 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, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
26. Lee, S., Ko, H., Kim, J., Oh, H.: vcnn: Verifiable convolutional neural network based on zk-snarks. IEEE Trans. Depend. Secur. Comput. 1–17 (2023). https://doi.org/10.1109/TDSC.2023.3348760
27. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 169–189. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_10

28. Lipmaa, H., Parisella, R., Siim, J.: Algebraic group model with oblivious sampling. In: Rothblum, G., Wee, H. (eds.) TCC 2023 (4). LNCS, vol. 14372, pp. 363–392. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-48624-1_14
29. Lipmaa, H., Siim, J., Zajac, M.: Counting vampires: from univariate sumcheck to updatable ZK-SNARK. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 249–278. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22966-4_9
30. Liu, T., Xie, X., Zhang, Y.: zkCNN: zero knowledge proofs for convolutional neural network predictions and accuracy. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 2968–2985. ACM Press (2021). https://doi.org/10.1145/3460120.3485379
31. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2111–2128. ACM Press (2019). https://doi.org/10.1145/3319535.3339817
32. Posen, J., Kattis, A.A.: Caulk+: table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957 (2022). https://eprint.iacr.org/2022/957
33. 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, Heidelberg, Virtual Event (2021). https://doi.org/10.1007/978-3-030-84242-0_27
34. 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, Heidelberg (2020). https://doi.org/10.1007/978-3-030-56877-1_25
35. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. Cryptology ePrint Archive, Paper 2023/1216 (2023). https://eprint.iacr.org/2023/1216
36. Wang, H., Hoang, T.: ezdps: an efficient and zero-knowledge machine learning inference pipeline. PoPETs **2023**(2), 430–448 (2023). https://doi.org/10.56553/popets-2023-0061
37. Weng, J., Weng, J., Tang, G., Yang, A., Li, M., Liu, J.: PVCNN: privacy-preserving and verifiable convolutional neural network testing. IEEE Trans. Inf. Forens. Secur. **18**, 2218–2233 (2023). https://doi.org/10.1109/TIFS.2023.3262932
38. 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 (2022). https://doi.org/10.1145/3548606.3560646
39. Zapico, A., Gabizon, A., Khovratovich, D., Maller, M., Ràfols, C.: Baloo: nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565 (2022). https://eprint.iacr.org/2022/1565
40. Zhang, J., Fang, Z., Zhang, Y., Song, D.: Zero knowledge proofs for decision tree predictions and accuracy. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 2039–2053. ACM Press (2020). https://doi.org/10.1145/3372297.3417278