On the Power of Polynomial Preprocessing:
Proving Computations in Sublinear Time, and More

Matteo Campanelli[1], Mario Carrillo[2], Ignacio Cascudo[3], Dario Fiore[3], Danilo Francati[4⋆], and Rosario Gennaro[5]

[1] Offchain Labs and University of Tartu
[2] Universitat Rovira i Virgili
[3] IMDEA Software Institute
[4] Sapienza University of Rome
[5] The City College of New York

Abstract. Cryptographic proof systems enable a verifier to be convinced of a computation's correctness without re-executing it; common efficiency requirements include both succinct proofs and fast verification. In this work we put forth the general study of cryptographic proof systems with *sublinear* proving time (after a preprocessing).

Prior work has achieved sublinear proving only for limited computational settings (e.g., vector commitments and lookup arguments), relying on specific assumptions or through non-black-box use of cryptographic primitives. In this work we lift many of these limitations through the systematic study of a specific object: polynomial commitments (PC) with sublinear proving time, a choice motivated by the crucial role that PC play in the design of efficient cryptographic schemes.

We show a simple construction of a PC with sublinear prover based on any vector commitment scheme (VC) and any preprocessing technique for fast polynomial evaluation. We prove that this PC satisfies *evaluation binding*, which is the standard security notion for PC, but show it does not satisfy stronger notions of security.

Next, we study applications of PCs with sublinear prover. The first one is to construct *index-efficient* SNARKs, i.e., where the prover is sublinear, after preprocessing, in the size of the index (i.e., the NP-relation describing the proven statement). We propose a method to transform a class of standard Polynomial Interactive Oracle Proofs (PIOPs) into index-efficient PIOPs, obtaining two results that advance the state of the art: the first lookup argument for unstructured tables in which the prover is sublinear in the size of the table, while making only black-box use of a VC and thus allowing instantiations from generic assumptions such as collision-resistant hash functions, and the first transparent SNARK for circuits where the prover is sublinear in the number of addition gates. Finally, our last application is a transformation that builds *UC-secure* SNARKs from simulation-extractable ones, with an approximately linear overhead in proving time (as opposed to quadratic in prior work).

⋆ Work done in part while at Royal Holloway University of London, Egham, UK.

Table of Contents

On the Power of Polynomial Preprocessing: . . . 1
Matteo Campanelli, Mario Carrillo, Ignacio Cascudo, Dario Fiore, Danilo Francati, and Rosario Gennaro

1 Introduction . . . 3
- 1.1 Our Results . . . 4
- 1.2 Overview of our techniques . . . 5
- 1.3 Related Work . . . 9

2 Preliminaries . . . 10
- 2.1 Polynomial Commitment . . . 11
- 2.2 Vector Commitments . . . 12
- 2.3 SNARKs for Indexed Relations . . . 13

3 Polynomial Preprocessing Schemes . . . 14
- 3.1 Kedlaya-Umans technique as PP scheme . . . 15

4 Polynomial Commitments with Fast Opening . . . 17
- 4.1 Beyond Evaluation Binding . . . 19

5 Index-Efficient SNARKs . . . 20
- 5.1 Index-Efficient PIOPs and Applications to Index-Efficient SNARKs . . . 20
- 5.2 Our Construction of Index-Efficient PIOPs . . . 22
- 5.3 Application to Table-Efficient Lookup Arguments . . . 24
- 5.4 Application to Index-Efficient SNARKs for Circuits . . . 25

6 Witness-Succinct UC SNARKs with Nearly Optimal Proving Time . . . 29
- 6.1 Putting it all together: instantiations and main result . . . 31

A Further Preliminaries . . . 37
- A.1 Definitions of Polynomial Commitments . . . 37
- A.2 Definitions of Vector Commitments . . . 38

B Lack of Strong Evaluation Binding of $\mathsf{PC}_{\mathsf{VC}}$ . . . 39

C Knowledge Sound (SNARK-based) Polynomial Commitment with Fast Opening . . . 41

D Supporting Material for UC SNARKs . . . 44
- D.1 UC Framework . . . 44
- D.2 (Succinct) Non Interactive Zero-Knowledge Proofs . . . 45
- D.3 Special Properties of Polynomial Commitments and Polynomial Encoding Schemes [GKO+23, BCC+24] . . . 47
- D.4 Additional Relevant Cryptographic Preliminaries . . . 50
- D.5 The compiler Π_{GKOPTT} of [GKO+23] . . . 52

E Supporting Proofs . . . 53
- E.1 Proof of Theorem 4 . . . 53
- E.2 Proof of Lemmas 5 and 6 . . . 55
- E.3 Proof of Lemma 3 . . . 59

F Why Our "Hybrid" Construction is Necessary in Light of Other Instantiations of the [GKO+23] Compiler . . . 59

1 Introduction

Cryptographic proof systems allow a weak machine $\mathcal{V}$ to be persuaded by a (potentially malicious) prover $\mathcal{P}$ that a computation C produces a specific output on some input x. Efficient proofs typically require both *low* communication complexity (e.g., that the "certificate" of the computation be short) and verification efficiency. In particular, we are usually interested in the verifier working sublinearly in the complexity T of the computation C.[6] This efficiency profile is possible, naturally, only after the verifier has performed (once and for all) a preprocessing stage of the computation C, of which it keeps a short digest (e.g., a verification key).

In this work, we tackle the following research direction:

Is it possible to build proof systems where the prover can also run in time sublinear in T (after a preprocessing)?

This seemingly paradoxical property *already* appears in the literature, albeit with some limitations. For example, in *vector commitments* (VCs) [CF13], which can be viewed as succinct arguments that a certain v is the i-th value of a committed vector, one can precompute the opening proofs for all positions in such a way that, at proving time, only the correct proofs needs to be fetched. Some schemes even support a nontrivial form of this functionality to generate succinct proofs for arbitrary subsets of positions [BBF19, TAB+20, CFG+20]. However, VCs are succinct arguments with a limited expressivity because their statements are the positions, i.e., integers from a polynomial-size set. In *lookup arguments*, which allow proving that each entry of a committed vector $\mathbf{v}$ is in a large committed table $\mathbf{t}$, it is a desirable feature that the proving time be sublinear in $|\mathbf{t}|$. However, in the state of the art, this efficiency profile is only achieved by constructions that make non-black box use of cryptographic schemes (e.g., due to encoding Merkle trees verification in a circuit [BFR+13]), by pairing-based schemes [ZBK+22, ZGK+22, EFG22, CFF+24, CFG25], or in the special case of structured tables [STW24]. The same limitations of lookup arguments apply to existing SNARKs for RAM computation with space complexity S and time complexity $T = o(S)$, in which one wishes the proving time to be sublinear in S.

This state of affairs shows that more is to be understood about the feasibility of succinct arguments with sub-linear prover. In particular, a desirable goal is to achieve sublinear proving for relatively general functionalities, with constructions that make black-box use of cryptographic primitives and can be instantiated from a variety of assumptions.

In this work, we achieve this goal by focusing on a specific type of computation: *proving evaluations of committed polynomials*. Polynomial commitment schemes (PC) [KZG10] are a cryptographic primitive that allows a prover to commit to a polynomial f with a string cm (commitment) and subsequently produce a proof of the correctness π_x of its evaluation $y = f(x)$ at any point x. Consider the case of a polynomial f with m variables of degree less than d in each variable, which can be described by a vector of $|f| = d^m$ coefficients. The crucial property of polynomial commitments is succinctness: both the commitment string and the evaluation proofs must be "short", i.e., sublinear in $|f|$ (and ideally, also the verification time should be sublinear in $|f|$).

PCs have already proved to have a central role in the design of SNARKs, and have many other applications (e.g., verifiable secret sharing). For this reason, we believe that the study of PCs with sublinear proving time can be a stepping stone towards the achievement of sublinear

[6] For example, if C is a circuit of size $T = n^2$ where $n = |x|$, we would like verification to require—*at most*—subquadratic time. In the setting of arguments *of knowledge* the verifier may even be required to work sublinearly in n, and instead of reading x explicitly the verifier is simply convinced that $\mathcal{P}$ *knows* such x.

proving for more general computations, and a *valuable tool* with applications beyond the specifics of cryptographic proofs.

Research in PC has been focused on improving the efficiency of the verifier, whereas it is widely accepted that the prover must run at least linearly in the size $|f| = d^m$ of the m-variate polynomial. In fact, this linear-time behavior appears unavoidable in order to, at the very least, compute the output $y = f(x)$.

In this work, we challenge this state of affairs by investigating the *design* and the *applications* of polynomial commitments with a sublinear prover. In such a scheme, at commitment time the prover also performs a preprocessing of the polynomial f which results in some auxiliary information aux. Later, when asked to prove an evaluation of f at an input x, the prover can use aux to quickly produce the evaluation $y = f(x)$ and its proof in sublinear time. Preprocessing schemes to quickly evaluate polynomials have been known in the complexity literature for some time now, e.g. the work by Kedlaya and Umans [KU11, KU08] (KU in short) that we use in this paper. Our challenge was to make sure we could quickly compute the proofs too, and achieve provable and meaningful notions of security for the resulting PC. In contrast to vector commitments or lookup arguments, where one could think of computing a proof for each position in the preprocessing stage, the challenge in PC is that the number of evaluation points is exponential, that is $|R^m|$ for a polynomial defined over a ring R.

1.1 Our Results

Constructing PCs with sublinear prover: Towards finding constructions of PCs with sublinear proving time, we observe that the recent work of Ateniese et al. [ABC+24] proposes a novel cryptographic scheme, dubbed verifiable data structure for univariate polynomial evaluations, which can also be seen as a PC for univariate polynomials satisfying a weaker security notion—*weak evaluation binding*—which states that an adversary should not be able to produce valid proofs for a false evaluation $y \neq f(x)$ and with respect to *honestly* generated commitments to f. Their construction achieves sublinear proving time by combining the preprocessing-based fast evaluation techniques for univariate polynomials of Kedlaya and Umans [KU11, KU08] together with a vector commitment.

In our work, we formalize a construction of PC with sublinear proving time by generalizing the construction of [ABC+24] to the case of multivariate polynomials (this is crucial for some of our applications) and by proving that it satisfies the standard security notion of PC, *evaluation binding*. The latter guarantees that it is hard to generate two (possibly malicious) proofs that are accepted with respect to the same (possibly malicious) commitment, the same evaluation point x, but claiming distinct outputs $y \neq y'$. Since evaluation binding may be insufficient in some applications, we also discuss how to extend the PC construction to achieve stronger security, specifically knowledge soundness, albeit this comes at the price of employing a SNARK.

This PC scheme has fast openings, meaning that opening proofs can be computed in sublinear time in the description of the polynomial f. In particular, for a univariate polynomial f of degree at most $d-1$ (i.e., $|f| = d$), this means proving in time $\mathsf{poly}(\log d)$, whereas, for an m-variate polynomial f of degree at most $d-1$ in each variable (i.e., $|f| = d^m$), it means proving in time $\mathsf{poly}(d, m)$.

See Section 1.2 for a description of the challenges involved in formalizing the construction in [ABC+24] as a PC and extending it to the multivariate case.

Index-Efficient SNARKs: We present an application of PCs with sublinear prover to construct SNARKs where the prover can be sublinear, after preprocessing, in the size of the NP relation—

what is usually referred to as the "index". We call this property *index-efficiency*. To build index-efficient SNARKs, we refine the popular framework to build SNARKs from polynomial interactive oracle proofs (PIOPs) and PCs [CHM+20, BFS20] showing how index-efficient SNARKs can be constructed black-box from index-efficient PIOPs and PCs with sublinear prover. Next, to instantiate this framework, our main technical contribution is a method to transform a class of standard PIOPs into index-efficient PIOPs. Interestingly, the combination of all our results yields two novel applications.

- *Application 1: new general lookup arguments with $o(|T|)$ prover*: we present the first lookup argument for unstructured tables in which the prover is sublinear in the size of the table T, while making only black-box use of a VC and thus allowing instantiations from generic assumptions such as collision-resistant hash functions. As mentioned earlier in the paper, prior lookup arguments with sublinear provers make non-black-box use of cryptographic schemes [BFR+13],[7] rely on pairings [ZBK+22, ZGK+22, EFG22, CFF+24], or can only deal with structured tables [STW24].
- *Application 2: transparent SNARKs for circuits with $o\left(\#\mathsf{gates}(+)\right)$ prover*: Our second application is the first universal SNARK for circuits (with transparent setup) in which the prover is sublinear in the number of addition gates. This also gives us the first SNARK of this kind with a *transparent* setup. Prior SNARKs in which the prover running time does not depend on the number of addition gates are either circuit-specific [GGPR13, Gro16] or, if universal, ([GKM+18]) they rely on structured reference strings that need a trusted setup.

Witness-Succinct UC-SNARKs: Our next application of PCs with sublinear prover concerns UC [Can01]. UC security is essential for SNARKs deployed within larger protocols, yet prior results lifting SNARKs to UC incurred a quadratic overhead on the prover. We propose the first compiler from simulation-extractable SNARKs to UC-secure SNARKs where the final prover is *quasilinear* in the size of the witness. We obtain our result by (non-trivially) extending the techniques of Ganesh et al. [GKO+23] (Eurocrypt '23), which provided the first general compiler for UC-secure SNARKs. Our compiler achieves proving time $O(n^{1+\epsilon})$ for an arbitrary $\epsilon > 0$, in contrast to $O(n^2)$ in [GKO+23].

Our core result can be summarized as follows:

Theorem (Restatement of Theorem 11). *Under the Strong Diffie-Hellman assumption ([BB04]), if there exists a simulation-extractable NIZK with $O_\lambda(n \cdot \mathsf{polylog}(n))$ proving-time and proof size and verification time $O_\lambda(\mathsf{polylog}(n))$ (for a witness of size n), then for any $\epsilon > 0$ there exists a UC-NIZK with $O_\lambda(\mathsf{polylog}(n))$ proof size and verification time and $O(n^{1+\epsilon})$ proving time.*

1.2 Overview of our techniques

PCs with fast opening. Kedlaya and Umans (KU) [KU11, KU08] show how a univariate polynomial f with d coefficients can be preprocessed and stored as a data structure DS of $O(d^{1+\epsilon})$ elements. Following this step, it is possible to evaluate $y = f(x)$ by simply querying DS on $\mathsf{poly}(\log d)$ points (see Theorem 3 for the precise statement). In [ABC+24], Ateniese et al. show how to make the KU-based data structure verifiable by using a vector commitment (VC) on top of it. The idea is to use the VC to commit to the data structure DS encoding the preprocessing of the polynomial

[7] In [BC23], the authors construct a lookup argument with sublinear proving time using recursive techniques. However, this approach is also non-black-box, due to the use of recursion, and the verifier's runtime is linear in the size of the table, thus it does not have succinct verification.

f. At evaluation time, to prove $y = f(x)$, the prover opens the $k = \mathsf{poly}(\log d)$ values required to evaluate f at x, and the verifiers run the KU evaluation algorithm themselves using those k points and check a VC opening for each of them. If the VC has (poly)logarithmic opening proofs, then the opening proof for our PC is sublinear in $|f|$, and so is the running time of the verifier.

In this paper, we extend this PC construction to work for m-variate polynomials of degree d, relying on the KU's preprocessing result for multivariate polynomials. The complexity of KU's technique in this case has been explored in [LMW23], showing that one evaluation can be computed with $\mathsf{poly}(d, m)$ queries to the data structure DS obtained after preprocessing, and that the size of DS is $d^m \cdot \mathsf{poly}(d, m) \cdot O(m^m)$ (see Theorem 2). Observe that the query complexity $\mathsf{poly}(d, m)$ is sublinear in the size of the multivariate polynomial $|f| = d^m$. In our PC scheme, as in [ABC+24], we use a VC to commit to DS and to open the $\mathsf{poly}(d, m)$ values needed to compute the evaluation.

We highlight that, in the multivariate case, DS is not of polynomial size in $|f|$ for every choice of d and m (a necessary requirement for having a polynomial-time PC). This is because the size of the data structure depends on the factor m^m, as described above. To avoid this issue, we show a selection of parameters that yield a PC with polynomial-time preprocessing time and space. We provide more details in Section 3, where we focus on going beyond constant arity m. Concretely, we show the following consequence of Theorem 2 and our PC construction:

Corollary 2 (Informal). *Assuming collision resistance hash functions, there exists a PC scheme for polynomials in $\mathbb{Z}_q[X_1, \ldots, X_m]$, $q \approx 2^\lambda$, with the following running times*

- *For univariate polynomials ($m = 1$) of degree at most $d \in \mathsf{poly}(\lambda)$, the committing time is $\mathsf{poly}(\lambda, \log d) \cdot O(d^{1+\epsilon})$ and the opening time is $\mathsf{poly}(\lambda, \log d)$ for every positive constant $\epsilon > 0$.*
- *For multivariate polynomials f where $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$, $|f| := d^m = \mathsf{poly}(\lambda)$ and where $d = \log^{3/\epsilon}(|f|)$ (which implies $m = \frac{\epsilon}{3} \log(|f|)/(\log\log(|f|))$) for an (arbitrary) constant $\epsilon > 0$, we have committing time $\mathsf{poly}(\lambda, m, d) \cdot O(d^{m(1+\epsilon)})$ and opening time $\mathsf{poly}(\lambda, m, d)$.*

The result above does not directly cover the relevant case, for many applications, of multilinear m-variate polynomials ($d = 2$ and $|f| = 2^m = \mathsf{poly}(\lambda)$); however, we will show how it can still be used in a concrete application in such a setting, namely in our construction of index-efficient PIOPs.

One of our contributions with respect to this construction of PC with sublinear prover is to prove that it satisfies *evaluation binding*, the standard security notion for PCs, which informally says that for any committed string cm (even a maliciously generated one), it is infeasible to produce valid openings for the same evaluation point x and two different outputs. In our PC construction, this security notion is inherited by the *position binding* security property of the underlying VC, which guarantees that given a committed vector DS it is infeasible to find an index i and two different valid openings of DS at i (see Theorem 4).

Stronger notions of security for PC can be defined as follows: *strong evaluation binding* [LM19, CFT22] says that, for any sequence of valid opening proofs for evaluations $y_i = f(x_i)$, there must exist a (univariate or multivariate depending on the polynomial space) polynomial f of degree d such that $f(x_i) = y_i$. Even stronger is *knowledge soundness* which says that such polynomial can be *extracted* from such proofs.

We first show (Appendix B) that our combination of KU with VC, does *not* satisfy strong evaluation binding. Intuitively the reason is that there are many more possible tables DS than polynomials of degree d and therefore simply committing to a table does not readily identify a *single*

polynomial.[8] We then discuss (Appendix C) possible ways to achieve strong evaluation binding and/or knowledge soundness by augmenting the commitment phase with a SNARK to guarantee that the DS committed by the VC is indeed the result of a correctly executed preprocessing on the polynomial f.

Index-Efficient SNARKs. Our first step towards constructing index-efficient SNARKs is an adaptation of the popular framework for building SNARKs from PIOPs and PCs [CHM+20, BFS20]. In brief, in a PIOP a computation F, also known as the "index", is "arithmetized" and preprocessed into a small set of polynomials to which the verifier has oracle access. Next, prover and verifier interact: the prover sends oracle polynomials and the verifier sends random challenges. Finally, the verifier can query, on a small set of points, all the polynomials sent by the prover as well as the index polynomials. The compilation of the idealized PIOP into a succinct argument consists into replacing the oracles by commitments, while the queries of the verifier are replaced by having the prover evaluate the polynomials and prove these evaluations with the PC scheme.

To achieve index-efficiency, we adapt this compiler with three main changes. (i) We assume that the PIOP is index-efficient, namely the PIOP prover is sublinear in the size of the index. (ii) We commit to the index polynomials using a PC scheme with sublinear prover. (iii) For every query of the PIOP verifier to the index polynomials, we let the prover use fast-evaluation techniques to compute the result and the sub-linear proof generation to compute the evaluation proof.

Equipped with this framework and with our PC construction, then we turn our attention to constructing index-efficient PIOPs. To take advantage of existing literature, we tackle the problem by proposing a transformation from a class of standard PIOPs to index-efficient PIOPs. Roughly speaking, our transformation applies to PIOPs in which the prover algorithm can be implemented using a small number of evaluations of the index polynomials. While in the original PIOP each of these evaluations must be fully computed by the prover (in at least linear time), in our case we exploit preprocessing techniques and replace them by fast evaluations, e.g., using [KU08]. This step is clearly natural. However, the main technical obstacle is that PIOPs normally work with multilinear polynomials (which is the most popular case in the literature) but the known results for preprocessing schemes only capture either univariate polynomials or multivariate polynomials with a very specific choice of parameters m and d, which in particular requires a degree larger than 1 to achieve efficiency.

For this reason, we additionally make the assumption that our starting class of standard PIOPs satisfy the property that, when we replace each multilinear polynomial produced by the indexer algorithm by a polynomial of degree $d > 1$ that agrees with it when evaluated on the boolean hypercube, the PIOP is still correct and knowledge sound. This assumption seems to be a natural property satisfied by most PIOP constructions, e.g., [Set20, CBBZ23, STW24]. See Theorem 6 for the concrete statement of these requirements.

Then, the idea of the transformation (Construction 2) is that we can use a "change of basis" technique as follows. We encode a multilinear n-variate polynomial $\tilde{f}$ with a degree-d m-variate polynomial $\hat{f}$ where $m < n$ and $(d+1)^m = 2^n$. We show how to construct a degree-d polynomial f in n variables such that: (a) f agrees with $\tilde{f}$ on the boolean hypercube; (b) an evaluation $f(\mathbf{z})$ can be efficiently obtained from an evaluation $\hat{f}(G(\mathbf{z}))$, where G is a linear function. These two properties yield the desired outcome. By (a) and our assumptions on the PIOP we can replace $\tilde{f}$ by f in the PIOP while preserving correctness and soundness (albeit with a larger loss $d/|\mathbb{F}|$).

[8] Think about a large data structure such that a part encodes evaluations of f and the second part encodes evaluations to f' where $f \neq f'$.

By property (b) we get fast evaluations since we have freedom to choose the parameters m and d of $\hat{f}$ in a suitable way to allow fast evaluation techniques.

We refer to Section 5.2 for more details.

Table-Efficient Lookup Arguments. Our application to table-efficient lookup arguments derives by observing that (a slight generalization of) the GeneralizedLasso protocol from [STW24] is a PIOP that satisfies all the conditions needed for our transformation. In particular, in this protocol the indexer produces a multilinear polynomial encoding of the (possibly unstructured) table. This observation, in combination with all the results described earlier yields the first table-efficient lookup argument for unstructured tables that makes only black-box use of a VC and thus allows instantiations from generic assumptions such as collision-resistant hash functions.

Index-Efficient SNARKs for Circuits. The main ingredient that yields our application to index-efficient SNARKs for circuits, where the prover is sublinear in the number of addition gates, is an index-efficient PIOP for linear checks. In this PIOP, after a preprocessing of the "index" matrix $\mathbf{M}$ of dimension $N \times N$, one can prove that $\mathbf{M} \cdot \mathbf{z} = \mathbf{v}$ by running in time $N\mathsf{polylog}(N)$, that is sublinear in N^2. We construct this index-efficient PIOP for linear checks through a "change of basis" technique similarly to our generic transformation. For the offline phase, we encode the matrix $\mathbf{M}$ with a degree-d $2m$-variate polynomial $\hat{M}$ that interpolates $\mathbf{M}$ such that $(d+1)^m = 2^n = N$ and (d, m) admit fast preprocessing. For the online phase, we adapt Thaler's matrix multiplication protocol [Tha13] into a PIOP where, after a verifier's random challenge $\mathbf{s}$, it is possible to implement the prover using $N\mathsf{polylog}(N)$ evaluation calls to the degree-d n-variate polynomial $\hat{M}(\mathbf{s}, G(\mathbf{X}))$, where $G : \mathbb{F}^n \to \mathbb{F}^m$ is a linear map. Then we can replace each of these calls with a fast evaluation that costs $\mathsf{polylog}(N)$. Finally, we obtain the index-efficient SNARK for circuits by using the index-efficient PIOP for linear checks to build an index-efficient PIOP for R1CS where the prover is sublinear in the number of nonzero entries of the index matrices[9], and then applying our index-efficient compiler that uses PCs with fast openings.

Efficient SE-to-UC SNARK compiler The Universal Composability (UC) framework is considered the "gold-standard" of security for cryptographic protocols. It ensures that protocols remain secure even when composed with others in arbitrary environments, which is a crucial property for SNARKs since they are almost always used in the context of larger protocols. However, a fundamental challenge arises when trying to prove UC-security for SNARKs: the "knowledge extractor" has to be "straight-line" (non-rewinding) and black-box (i.e., having only access to the I/O of the adversary). Prior work has tackled this problem but commonly it suffers from losing succinctness or being restricted to arguments with a very specific structure[10].

Our starting point instead is the generic compiler by Ganesh et al. [GKO+23] which builds a UC-secure witness-succinct[11] SNARK generically from any simulation-extractable one. Their approach can be informally described as follows

[9] In the reduction from circuits to R1CS, the number of nonzero entries correspond to the number of addition gates.

[10] Here we are referring to two lines of works. The first adopts some form of encryption of the witness, making the proof size at least linear in witness size [ARS20, BS21, AGRS23, KZM+15]. Another line of work, like the recent [CF24], leverages the straight-line-extractability properties of Kilian-like constructions but is applicable only to arguments stemming from Interactive Oracle Proofs (an interactive form of PCPs [BCS16]).

[11] A SNARK is circuit-succinct if the proof is sublinear in the circuit size, and witness-succinct—which is the most stringent succinctness property—if it also sublinear in the witness size. Circuit-succinct UC-secure SNARKs were known before [GKO+23].

– The witness w is represented as a polynomial f_w via a redundant randomized encoding, which is committed with a PC $\mathsf{cm}_w = \mathsf{PC.Com}(f_w)$;
– The SE-SNARK is used to prove that w is a valid witness and that cm_w is a valid PC commitment to the correct randomized polynomial for w;
– r evaluations of f_w are provided to the verifier together with their evaluation proofs with respect to f_w

In the last step r is chosen small enough to guarantee succinctness of the final proof. Additionally the r evaluations must satisfy a particular property meant to enforce a "proof of work" by the prover (generalizing Fischlin's approach [Fis05]). This step requires the prover to actually evaluate f_w at a linear number of points (to find the r ones that satisfy the property—only this small fraction of them is revealed), and from this high number of queries, the extractor (which sees all the queries the prover makes, not just the r revealed ones) can reconstruct the witness. No knowledge about w is leaked due to two facts: (i) w is encoded via a randomized polynomial, and r is small enough; (ii) the PC they use (KZG [KZG10]) has the "evaluation-hiding" property that an opening proof for a particular point reveals no information about the value of the polynomial at other points.

This approach however blows up the complexity of the prover: evaluating f_w at a linear number of points, without preprocessing, yields a quadratic total running time. A naive application of our sublinear opening PC does not work because the KU technique does not guarantee that our PC is evaluation-hiding (see Section 6). Therefore, we modify the [GKO+23] compiler in two major ways:

1. To achieve hiding, we use a different *hybrid* encoding for the witness. We first encrypt it $\mathsf{ct} = \mathsf{Enc}(\mathsf{sk}, w)$ using any symmetric encryption scheme. We then encode sk with a randomized polynomial g, while the ciphertext ct is instead encoded deterministically as a polynomial f (e.g. by parsing it as coefficients over a field)
2. We use two different PCs to commit to our hybrid encoding of the witness. The randomized polynomial g is committed via KZG [KZG10], while f is committed using our KU-based PC with fast opening.

Note that our scheme is approximately linear for the Prover since the openings of f take sublinear time with our PC (note that the degree of g is independent of the size of w so those openings do not count), with the additional cost for the prover of $O(n^{1+\epsilon})$ for preprocessing. Zero-Knowledge is preserved because (as in [GKO+23]) the randomization of the encoding of sk means that no knowledge is leaked about sk even after r evaluations are revealed, due also to the fact that sk is committed with an evaluation-hiding PC. The r evaluations may leak knowledge of the value ct (since that is encoded deterministically, and the PC is not evaluation-hiding), but these amount to encryptions of w under sk; therefore, they hide w since no further knowledge is leaked on sk.

The reader can find an expanded version of our technical overview on our UC-SNARK applications at the beginning of Section 6.

1.3 Related Work

As we discussed earlier, the KU polynomial preprocessing technique was introduced as a tool in the design of cryptographic protocols by Lin et al. who built a *Doubly Efficient Private Information Retrival (DEPIR)* scheme [LMW23]. A Private Information Retrieval (PIR) scheme, is a protocol between a Server holding a database DB and a Client who queries DB on an index i. At the end of the protocol the Client learns $\mathsf{DB}(i)$ while the Server has no information on what index was

queried. The crucial property of PIR is that communication between the parties be sublinear in $N = |\mathsf{DB}|$ (otherwise the solution is trivial by having the Server send the entire DB to the Client). Before [LMW23] it was widely believed that any PIR scheme required the Server to work in $\Omega(N)$ steps, since if the Server did not read every item in the database, he would get information about i. Surprisingly [LMW23] shows that by preprocessing DB into a data ‘ looking at a $O(\log^c N)$ entries in $\mathsf{DS}_{\mathsf{DB}}$ yielding a "doubly efficient" PIR for both Server and Client.

Ateniese et al. [ABC+24] showed that the KU (univariate) technique can be used to improve the efficiency of *proofs of storage*. Assume that a storage provider needs to prove to a client that it is storing a file F. One approach is to encode F as a univariate polynomial f. Then, by querying the polynomial at a random point, the client can be assured that the file is still stored (indeed, the file can theoretically be retrieved by querying a sufficient number of points to allow interpolation of f). However, this approach requires the provider to read the entire file every time in order to evaluate f. By using KU, the file is encoded into a larger data structure DS, which permits to answer in sublinear time.

Discussion and Future Work *On practical instantiations:* We acknowledge that our work is unlikely to be *immediately* practical due to the large constants in the specific preprocessing scheme [KU11] that we use. Given the many practical applications that motivated this work, and that we show can be built from PC with fast opening, it is an important research question to investigate different approaches for polynomial preprocessing, with the goal of getting schemes that are more practically feasible.
More functionalities with sublinear proving? We showcase our framework for general relations provable in sublinear time, *index-efficient* PIOPs, by obtaining a new construction of table-efficient lookup arguments and index-efficient SNARK for circuits. Yet interesting open questions remain: Can we build index-efficient PIOPs for other relations that would be surprising to prove in sublinear time (and from relatively simple tools)? For what kinds of relations is it even conceivable to achieve sublinear proving?
More properties/applications for commitments with fast openings? To go beyond evaluation binding, our PC construction must be augmented with a SNARK. Is that inherent, or can strong binding/extractability for PC with fast opening be achieved with simpler tools/weaker assumptions? What about hiding (which as we point out, does not readily hold for our scheme)? Can we build such a commitment *directly* for *multilinear* polynomials, rather than go through the workaround we use for index-efficient PIOPs? What are other settings, beyond the one in this work—UC-NIZK compiler—where amortizing an expensive preprocessing yields a better efficiency overall?

2 Preliminaries

Notation. For any integer $n \geq 1$, define $[n] = \{1, \ldots, n\}$. We use $\lambda \in \mathbb{N}$ to denote the security parameter. A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is said to be *negligible*, denoted $\epsilon(\lambda) \in \mathsf{negl}(\lambda)$ if for every positive polynomial $p(\cdot)$ and all sufficiently large λ it holds that $\epsilon(\lambda) < 1/p(\lambda)$. A function $p(\lambda)$ is polynomial, denoted $p(\lambda) \in \mathsf{poly}(\lambda)$, if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. In the context of cryptographic primitives we use a special notation to emphasize asymptotic behavior "independent" of the security parameter λ: we write $f = O_\lambda(g(n))$ if there exists a fixed polynomial λ^c such that $f = O(\lambda^c \cdot g(n))$. We denote by $\widetilde{O}_\lambda(f(n))$ quasi-linear running time, i.e. $O_\lambda\big(f(n)\mathsf{polylog}(f(n))\big)$. We say that an algorithm is probabilistic polynomial time (abbreviated PPT) if its running time

is bounded by a polynomial function $p(\lambda)$. For a finite set S, we write $a \leftarrow\!\!\$\ S$ to mean that a is sampled uniformly at random from S. For an algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$ to denote executing $\mathcal{A}$ on input x and assigning its output to y. Given $f : X \to Y$and $\mathbf{v} = (v_1, \ldots, v_n) \subseteq X^n$, we denote by $f(\mathbf{v})$ the vector in Y^n obtained through element-wise evaluation of f on $\mathbf{v}$. An *indexed relation* $\mathcal{R}$ is a set of tuples $(\mathbb{i}, \mathbb{x}; \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ the statement and $\mathbb{w}$ the witness. The language $\mathcal{L}(\mathcal{R})$ associated to a relation $\mathcal{R}$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$.

Approximately-linear time. If an algorithm runs in time $O_\lambda(n^{1+\epsilon})$ in a parameter n for any positive constant $\epsilon > 0$, we say that it runs in *approximately-linear time*[12]. For sake of brevity, we often simply write that "an algorithm runs in time $O_\lambda(n^{1+\epsilon})$" meaning implicitly "for any (unspecified) constant $\epsilon > 0$".

RAM model. Our work focuses on the application of preprocessing techniques to build fast and efficient schemes. Even if not directly specified, all efficiency properties and running times will be measured in the RAM model of computation, which allows an algorithm to run in time sublinear its input size.

2.1 Polynomial Commitment

A polynomial commitment scheme allows a prover to commit to a (possibly multivariate) polynomial $f \in R[X_1, \ldots, X_m]$, producing a commitment value cm, and then compute proofs π_x demonstrating that y is the correct evaluation of the committed f at an input $x \in R^m$.

Definition 1 (Polynomial Commitment [KZG10]). *A polynomial commitment scheme* PC *for multivariate polynomials over a ring R is defined by the following algorithms:*

$\mathsf{Setup}(1^\lambda, m, d) \to \mathsf{ck}$**:** *On input the security parameter λ, the number of variables m, and a bound d for the degree of each variable, the setup algorithm outputs a public commitment key* ck*, which implicitly defines the message space of the commitment scheme $\mathcal{P}_{m,d} = \{f \in R[X_1, \ldots, X_m] \mid \mathsf{deg}_i(f) < d, \forall i \in [m]\}$.*

$\mathsf{Com}(\mathsf{ck}, f; r) \to (\mathsf{cm}, \mathsf{aux})$**:** *On input the commitment key* ck*, a polynomial $f \in \mathcal{P}_{m,d}$, and (possibly) a randomness r, the commit algorithm outputs a commitment* cm *and an auxiliary information* aux*.*

$\mathsf{Open}(\mathsf{ck}, x, \mathsf{aux}) \to \pi_x$**:** *On input the commitment key* ck*, a point $x \in R^m$, and auxiliary information* aux*, the open algorithm outputs a proof π_x.*

$\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) \to b$**:** *On input the commitment key* ck*, the commitment* cm*, $x \in R^m, y \in R$, and a proof π_x, the verification algorithm outputs $b \in \{0, 1\}$.*

Correctness ensures that honestly generated proofs always pass verification. See Appendix A.1 for the syntax of PC and the correctness definition. Furthermore, we are interested in (fully) succinct schemes, i.e., the sizes of commitments and proofs, as well as the running time of Ver, are sublinear in the size of the polynomial.

Definition 2 (Succinctness of PC). *We say that a polynomial commitment scheme* PC *is succinct if the size of the commitments produced by* Com*, the proofs produced by* Open *and the running time of* Ver *are sublinear (e.g., polylogarithmic) in the size of the polynomial.*

[12] To the best of our knowledge, this is not a standard notion. The phrase "superlinear" usually refers to a similar asymptotic behavior but in the setting of lower bounds.

A polynomial commitment scheme must satisfy at least one of the following security properties, which we define in order from weaker to stronger:

Binding. It is computationally infeasible to find two different polynomials $f \neq f'$ and two randomnesses r and r' that produce the same commitment cm.

Weak evaluation binding. It is computationally infeasible to produce a verifying proof π_x for an incorrect evaluation $y \neq f(x)$ when the commitment cm of f (required for verification) is honestly computed.

Evaluation binding. It extends the guarantees of weak evaluation binding even in the presence of a maliciously computed commitment cm.

Strong evaluation binding. It guarantees that, for any maliciously chosen sequence of verifying proofs π_i for evaluations y_i and points x_i, there must exist a polynomial $f \in \mathcal{P}_{m,d}$ (where $\mathcal{P}_{m,d}$ is the supported domain of the polynomial commitment scheme) such that $f(x_i) = y_i$.

Knowledge soundness. This last property says that it exists an efficient non-black-box extractor able to extract the committed polynomial f if the corresponding adversary is able to provide verifying proofs.

Below, we provide the formal definition of evaluation binding. We refer the reader to Appendix A.1 for the remaining ones.

Definition 3 (Evaluation Binding of PC). *A polynomial commitment scheme PC satisfies evaluation binding if for any PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(\mathsf{ck},\mathsf{cm},x,y,\pi_x)=1\wedge \\ \mathsf{Ver}(\mathsf{ck},\mathsf{cm},x,y',\pi'_x)=1\wedge\ y\neq y'\end{array}\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,m,d)\\ (\mathsf{cm},x,y,\pi_x,y',\pi'_x)\leftarrow\mathcal{A}(\mathsf{ck})\end{array}\right]\in\mathsf{negl}(\lambda).$$

The following theorem relates the above five security definitions. Such relations are standard and we refer the reader to [CFT22, BCFL23] for more details.

Theorem 1. *Let PC be a correct polynomial commitment scheme. Then,*

1. *If PC is weak evaluation binding then PC is also binding.*
2. *If PC is evaluation binding then PC is weak evaluation binding.*
3. *If PC is strong evaluation binding then PC is evaluation binding.*
4. *If PC satisfies non-black-box (resp. black-box) knowledge soundness then PC is strong evaluation binding.*

2.2 Vector Commitments

A vector commitment (VC) scheme permits to commit on vector of messages and later open individual positions. We make use of the standard definition of VC composed of algorithms (Setup, Com, Open, Ver), satisfying correctness and position binding (see Appendix A.2 for more details.). We are interested in *succinct* vector commitments schemes where the sizes of commitments and proofs, as well as the running time of Ver, are sublinear in the size of the vector.

Definition 4 (Succinctness of VC). *We say that VC is succinct if the size of the commitments produced by Com, the proofs produced by Open and the running time of Ver are $\mathsf{poly}(\lambda, \log n)$ where n is the size of the vector.*

In addition to the above, we formalize the notion of vector commitments in which the opening algorithm is "fast", i.e., polylogarithmic after a one-time preprocessing at commitment time. In order to model that the cost of this preprocessing can be amortized, we require Com to run in quasilinear time.

Definition 5 (VC with Fast Opening). *We say that a vector commitment scheme VC has fast openings if the running time $\mathsf{T}_{\mathsf{Com}}(\lambda, n)$ of Com is $\mathsf{poly}(\lambda, \log n) \cdot n$, and the running time $\mathsf{T}_{\mathsf{Open}}(\lambda, n)$ of Open is $\mathsf{poly}(\lambda, \log n)$.*

A natural instantiation of VC satisfying position binding, succinctness, and fast openings, is Merkle tree [Mer88], which can be built from collision resistant hash functions (CRHF). Other instantiations are discussed in Appendix A.2.

2.3 SNARKs for Indexed Relations

We recall the definition of succinct non-interactive arguments of knowledge (SNARKs) for indexed relations.

Definition 6 (SNARK). *A SNARK for an indexed relation $\mathcal{R}$ is defined by the following algorithms:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{srs}$**:** *On input the security parameter λ, the randomized setup algorithm returns a structured reference srs.*

$\mathsf{Index}(\mathsf{srs}, \mathbb{i}) \to (\mathsf{pk}_\mathbb{i}, \mathsf{vk}_\mathbb{i})$ *On input the structured references srs and an index $\mathbb{i}$, the deterministic indexing algorithm outputs a proving key $\mathsf{pk}_\mathbb{i}$ and a verification key $\mathsf{vk}_\mathbb{i}$.*

$\mathsf{Prove}(\mathsf{pk}_\mathbb{i}, \mathbb{x}, \mathbb{w}) \to \pi$ *On input the proving key $\mathsf{pk}_\mathbb{i}$ (for index $\mathbb{i}$), a statement-witness pair $(\mathbb{x}, \mathbb{w})$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the randomized proving algorithm outputs a proof π.*

$\mathsf{Ver}(\mathsf{vk}_\mathbb{i}, \mathbb{x}, \pi) \to b$ *On input the verification key $\mathsf{vk}_\mathbb{i}$ (for index $\mathbb{i}$), a statement $\mathbb{x}$, and proof π, the deterministic verification algorithm outputs a bit $b \in \{0, 1\}$.*

Definition 7 (Correctness of SNARK). *A SNARK for an indexed relation $\mathcal{R}$ is correct if for any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ we have:*

$$\Pr\left[\mathsf{Ver}(\mathsf{vk}_\mathbb{i}, \mathbb{x}, \pi) = 1 \;\middle|\; \begin{matrix}(\mathsf{pk}_\mathbb{i}, \mathsf{vk}_\mathbb{i}) \leftarrow \mathsf{Index}(\mathsf{Setup}(1^\lambda), \mathbb{i}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{pk}_\mathbb{i}, \mathbb{x}, \mathbb{w})\end{matrix}\right] = 1.$$

Definition 8 (Succinctness of SNARK). *A SNARK is succinct if Prove outputs proofs of size $\mathsf{poly}(\lambda, \log |\mathbb{w}|)$, the running time of Prove is $\mathsf{poly}(\lambda, |\mathbb{x}|, |\mathbb{w}|)$, and the running time of Ver is $\mathsf{poly}(\lambda, |\mathbb{x}|, \log |\mathbb{w}|)$.*

Definition 9 (Knowledge soundness of SNARK). *A SNARK for an indexed relation $\mathcal{R}$ satisfies knowledge soundness if for any PPT adversary $\mathcal{A}$, there exists a polynomial-time extractor $\mathsf{Ext}_\mathcal{A}$ (running on the same input of $\mathcal{A}$, including its random coins) such that:*

$$\Pr\left[\begin{matrix}\mathsf{Ver}(\mathsf{vk}_\mathbb{i}, \mathbb{x}, \pi) = 1 \\ \wedge \;\; (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}\end{matrix} \;\middle|\; \begin{matrix}\mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda) \\ ((\mathbb{i}, \mathbb{x}, \pi), \mathbb{w}) \leftarrow (\mathcal{A}(\mathsf{srs}), \mathsf{Ext}_\mathcal{A}(\mathsf{srs})), \\ (\mathsf{pk}_\mathbb{i}, \mathsf{vk}_\mathbb{i}) \leftarrow \mathsf{Index}(\mathsf{srs}, \mathsf{vk}, \mathbb{i})\end{matrix}\right] \in \mathsf{negl}(\lambda).$$

The above definition assumes a non-black-box extractor. Sometimes we make us of the *black-box knowledge soundness* flavour, which is identical to the above except that the extractor Ext is universal.

3 Polynomial Preprocessing Schemes

In this section, we define a building block called the polynomial preprocessing scheme (PP). Intuitively, a PP scheme abstracts algorithmic methods (such as [KU08, KU11]) that enable the fast evaluation of a polynomial $f \in R[X_1, \ldots, X_m]$ at any given point $x \in R^m$, after performing a one-time preprocessing.[13] The main objective of a PP is to achieve an evaluation time complexity that is significantly lower (e.g., sublinear) than the natural polynomial evaluation, which requires reading the entire description of f, whose size is $O(d^m \cdot \log |R|)$.

Definition 10 (Polynomial Preprocessing Scheme). *A polynomial preprocessing scheme* PP *is defined by the following algorithms:*

$\mathsf{Params}(R, m, d) \to \mathsf{pp}$**:** *On input the description of a ring R, the number of variables $m \in \mathbb{N}$, and the degree $d \in \mathbb{N}$, the randomized parameter generation algorithm outputs the parameters* pp *that can be used for the execution of the following algorithms on m-variate polynomials of individual degree at most d. We assume the public parameters* pp *includes an upper bound S on the size of the data structures produced by the preprocessing algorithm below.*

$\mathsf{PreProcess}(\mathsf{pp}, f) \to \mathsf{DS}$**:** *On input a m-variate polynomial $f \in R[X_1, \ldots, X_m]$ of individual degree at most d, the randomized preprocess algorithm outputs a data structure $\mathsf{DS} = (v_1, \ldots, v_S)$ (where S is the data structure upper bound contained in* pp*). We model* DS *as a vector in $\mathcal{M}^S$ for a message space $\mathcal{M}$.*

$\mathsf{Lookup}(\mathsf{pp}, x) \to (i_j)_{j \in [k]}$**:** *On input the public parameters* pp *and a point $x \in R^m$, the deterministic lookup algorithm outputs k indices $i_1, \ldots, i_k \in [S]$.*

$\mathsf{Reconstruct}(\mathsf{pp}, x, v_1, ..., v_k) \to y$**:** *On input the public parameters* pp*, an evaluation point $x \in R^m$, and k values $v_1, \ldots, v_k \in \mathcal{M}$, the deterministic reconstruct algorithm outputs a value $y \in R$.*

Looking ahead, we are interested into efficient PP schemes which access only a "small" subset $(v_{i_k})_{i \in [k]}$ of the data structure DS (i.e., $k \ll |\mathsf{DS}|$). To accomplish this, it is necessary to work in the RAM model of computation to efficiently retrieve $(v_{i_k})_{i \in [k]}$ from DS.

Correctness says that the reconstruction algorithm computes $f(x)$ when $(v_1, \ldots, v_k) = (\mathsf{DS}_{i_1}, \ldots, \mathsf{DS}_{i_k})$, for DS the data structure computed by the preprocess algorithm and $(i_1, \ldots, i_k)$ the indices outputted by the lookup algorithm.

Definition 11 (Correctness of PP). *A polynomial preprocessing scheme* PP *is correct for a ring R if for any integers $m, d \in \mathbb{N}$, any polynomial $f \in R[X_1, \ldots, X_m]$ such that $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$, and for any point $x \in R^m$, we have $\mathsf{Reconstruct}(\mathsf{pp}, x, v_{i_1}, \ldots, v_{i_k}) = f(x)$ where $\mathsf{pp} \leftarrow \mathsf{Params}(R, m, d)$, $(v_i)_{i \in [S]} \leftarrow \mathsf{PreProcess}(\mathsf{pp}, f)$, and $(i_1, \ldots, i_k) \leftarrow \mathsf{Lookup}(\mathsf{pp}, x)$.*

The property below capture the main benefit of using a PP scheme, i.e., enabling fast polynomial evaluation (compared to reading the entire polynomial) at the price of having a one-time preprocessing. We observe that the definition below allows PreProcess to be "slow" and produce data structures of size larger than $|f| = d^m$. This is because, to enable fast evaluation, such schemes (e.g., [KU11, KU08]) encode additional and redundant information into DS.

[13] Alternatively, a PP scheme can be seen as the algorithmic process of building a data structure for evaluating polynomials.

Definition 12 (Efficiency of PP). *A polynomial preprocessing scheme* PP *satisfies efficiency for* R *if, for every* $d, m \in \mathbb{N}$*, for every polynomial* $f \in R[X_1, \ldots, X_m]$ *such that* $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$*, we have:*

1. *the running time of both* Lookup *and* Reconstruct *is* $\mathsf{poly}(\log |R|, \log d)$ *(resp.* $\mathsf{poly}(\log |R|, d, m)$*) for the univariate case* $m = 1$ *(resp. multivariate case* $m > 1$*).*[14]
2. *the running time of* PreProcess *and the size* S *of the data structure* DS *that it produces are* $\mathsf{poly}(\log |R|, d^m)$ *(ideally slightly superlinear in* d^m*).*

Remark 1 (Comparison with [ABC+24]). In [ABC+24], Ateniese et al. propose an abstraction with the same intent as ours: abstracting any algorithmic method of preprocessing polynomials. Despite both abstractions ([ABC+24] and ours) having the same final objective, there are crucial differences.

The abstraction by Ateniese et al., named the data structure for univariate polynomial evaluation,[15] is composed of two algorithms: (i) GenData, which computes the data structure given the polynomial to be evaluated, and (ii) Eval, which outputs $f(x)$ given the *entire* data structure and the point x. This abstraction does not explicitly highlight that these algorithms leverage the RAM model to achieve efficiency. In particular, Eval hides inside the fact that a subset of the large data structure (provided as input) is retrieved to compute the final output y. For this reason, [ABC+24] formalizes, alongside the abstraction, the notion of localized RAM computation, which captures the idea that if a RAM computation outputs y by reading a subset of a large input (e.g., a data structure), then there exists a method (the localized RAM computation) to compute the same output using only the subset accessed by the original computation. In other words, the original computation depends only on a subset of the large input.

In this regard, our abstraction is simpler and makes this aspect more explicit. In a nutshell, Definition 10 decouples the RAM algorithm Eval of [ABC+24] into two separate ones: Lookup entitled to identify the subset (determined by the indexes $(i_1, \ldots, i_k)$) of the data structure $\mathsf{DS} = (v_1, \ldots, v_S)$ that must be accessed to compute the correct output, and Reconstruct which computes the actual output y given the subset $(v_{i_1}, \ldots, v_{i_k}) \subseteq \mathsf{DS}$. We emphasize that our abstraction captures PP schemes such that indexes computed by Lookup depend only on the point x, which is indeed the case for the data structures in [KU11, KU08].

3.1 Kedlaya-Umans technique as PP scheme

In [KU08], Kedlaya and Umans describe an elegant and efficient method for the fast evaluation of polynomials with preprocessing. Their approach will be main focus of our work, in particular for the case of the ring of integers $\mathbb{Z}_q$ (see also [LMW23, LMW22, Theorem A.4]). The method in [KU08] falls squarely into our PP abstraction.

Theorem 2 (Kedlaya-Umans PP scheme [LMW22, Theorem A.4]). *Let* $q \in \mathbb{N}$*. There exists an efficient polynomial preprocessing scheme for the ring* $R = \mathbb{Z}_q$ *and for* m*-variate polynomials* $f \in R[X_1, \ldots, X_m]$ *of individual degree* $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$*. Its efficiency profile is as follows:*

[14] We explicate a different bound (i.e., $\mathsf{poly}(\log |R|, \log d)$) for univariate polynomial since applying the multivariate one (for $m = 1$) would result in having a meaningless efficiency definition since $|f| \in \mathsf{poly}(\log |R|, d)$.

[15] The term "univariate" comes from the fact that [ABC+24]'s abstraction only covers univariate polynomials over $\mathbb{Z}_p$. In contrast, we consider m-variate polynomials over arbitrary rings R.

1. *the running time of both* Lookup *and* Reconstruct *is* $\mathsf{poly}(\log q, d, m)$*;*
2. *the running time of* PreProcess *and the size* S *of the data structure* DS *that it produces are* $d^m \cdot \mathsf{poly}(\log q, m, d) \cdot O(m(\log m + \log d + \log\log q))^m$*.*

Henceforth, we refer to the aforementioned PP scheme as *KU-based PP*.

When focusing on polynomial-time algorithms (as in this work), we can easily observe that parameters matter when considering Theorem 2. Indeed, preprocessing some $\mathsf{poly}(\lambda)$-size polynomials using the KU-based PP scheme of Theorem 2 can cause the preprocessing algorithm PreProcess (and the related size S of the output data structure) to be exponential in the security parameter λ. For instance, PreProcess runs in exponential time if we consider: (i) $\mathsf{poly}(\lambda)$-variate polynomials (due to the $O(m^m)$ term in Theorem 2), or (ii) $O(\log \lambda)$-variate polynomials of individual degree $d \in \mathsf{poly}(\lambda)$ (due to the d^m term in Theorem 2).

Unfortunately, even the case of multilinear m-variate polynomials, where $2^m = N = \mathsf{poly}(\lambda)$, implies PreProcess to run in superpolynomial time due to $O(m^m) = O(N^{\log N})$. This is the use case arising in most applications of polynomial commitments to SNARKs where multilinear polynomials are used to encode N-long vectors via interpolation and thus we cannot apply the KU preprocessing technique in a straightforward way to this important use case.

For this reason, in the following corollary we show a selection of parameters to instantiate Theorem 2 in such a way that the preprocessing (resp. evaluation) time is slightly *superlinear* (resp. *polylogarithmic*) in the size of the polynomial. These will be the parameters of interest in this work.

Corollary 1 (KU-based PP scheme for multivariate polynomials). *Let* $\lambda \in \mathbb{N}$ *be a security parameter, and let* q, d, N *such that* $q \approx 2^\lambda$*,* $N = d^m \in \mathsf{poly}(\lambda)$*, and* $d = \log^{3/\epsilon}(N)$ *for some constant* $\epsilon > 0$*. Then, there exists a (KU-based) PP scheme for* m*-variate polynomials* $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ *of individual degree* $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$ *achieving the following efficiency property:*

1. *the running time of both* Lookup *and* Reconstruct *is* $\mathsf{poly}(\log q, d, m)$*;*
2. *the running time of* PreProcess *and the size* S *of the data structure* DS *that it produces are* $O(N^{1+\epsilon}) \cdot \mathsf{poly}(\log q, \log N)$*.*

Proof. Observe that our parameters' choice gives $m = \frac{\epsilon}{3}\log(N)/(\log\log N)$. Hence, the proof follows by leveraging Theorem 2 by using the following bounds (similar to [LMW23, LMW22]):[16] $m^m \leq (\log N)^{\frac{\epsilon}{3}\log(N)/(\log\log N)} = N^{\epsilon/3}$, and $O(\log m + \log d + \log\lambda)^m \leq O(\log N)^m \leq O(1)^m(\log N)^m \leq O(N^{\epsilon/3})N^{\epsilon/3}$. □

As observed, the parameters in Theorem 2 do not result in a PP scheme for univariate polynomials with sublinear evaluation time. Specifically, when $m = 1$, the evaluation time becomes polynomial in the degree d, which is at least as large (or potentially larger) than the size of the polynomial itself. However, for the univariate case, we can obtain polylogarithmic efficiency by encoding the degree d univariate polynomial into a multivariate one. This is essentially the result shown in [KU11, Theorem 5.1], which we report below.

Theorem 3 (KU-based PP scheme for univariate polynomials [KU11, Theorem 5.1]). *Let* $\lambda \in \mathbb{N}$ *be a security parameter, and let* q, d *such that* $q \approx 2^\lambda$*,* $d \in \mathsf{poly}(\lambda)$*. Then, there exists a (KU-based) PP scheme for univariate polynomials* $f \in \mathbb{Z}_q[X]$ *of degree* $\mathsf{deg}(f) < d$ *achieving the following efficiency property:*

[16] Contrarily to [LMW23, LMW22], we do not need d to be a prime number.

1. *the running time of both* Lookup *and* Reconstruct *is* $\mathsf{poly}(\log q, \log d)$*;*
2. *the running time of* PreProcess *and the size* S *of the data structure* DS *that it produces are* $O(d^{1+\epsilon}) \cdot \mathsf{poly}(\log q, \log d)$*.*

4 Polynomial Commitments with Fast Opening

We show how to construct polynomial commitment schemes with a sublinear opening time. Below we define this efficiency notion, dubbed *fast openings*; it can be seen as an adaptation of its VC counterpart (Definition 5) to polynomials.

Definition 13 (PC with Fast Opening). *We say that a polynomial commitment scheme* PC *for multivariate (resp. univariate) polynomials over a ring* R *has fast openings if the running time* $\mathsf{T}_{\mathsf{Com}}(\lambda, d, m, |R|)$ *of* Com *is* $\mathsf{poly}(\lambda, \log|R|, d^m)$ *(resp.* $\mathsf{poly}(\lambda, \log|R|, d)$*), and the running time* $\mathsf{T}_{\mathsf{Open}}(\lambda, d, m, |R|)$ *of* Open *is* $\mathsf{poly}(\lambda, \log|R|, d, m)$ *(resp.* $\mathsf{poly}(\lambda, \log|R|, \log d)$*), where the parameters* m *(number of variables) and* d *(maximum individual degree) are chosen on setup.*[17]

We highlight that achieving fast openings in PC is much more challenging than in the vector commitment setting. Given a committed vector of polynomial size, there are only a polynomial number of indices that can be opened. Thus, fast opening can be achieved by preprocessing all those proofs and leveraging the RAM model to retrieve each of these proofs when needed. However, in PC, the number of possible proofs corresponds to the cardinality of the point space R^m of the target polynomials $f \in R[X_1, \ldots, X_m]$. For some choices of parameters the cardinality of R^m is exponential in λ. Hence, simply preprocessing all proofs is not feasible and would result in exponential memory usage. This is essentially the reason why research focuses on building PP schemes, i.e., efficient algorithms for polynomial evaluation at all points (such as [KU08, KU11]).[18]

Evaluation Binding Polynomial Commitment with Fast Opening from VC. Our main construction, dubbed $\mathsf{PC}_{\mathsf{VC}}$, is obtained by combining a PP scheme (Definition 10) with a VC that enjoys fast opening time (Definition 5). The basic idea of the construction $\mathsf{PC}_{\mathsf{VC}}$ is that the commitment algorithm runs the preprocessing algorithm of the PP scheme and uses the VC to commit to the blocks of resulting data structure DS. Next, in order to prove an evaluation in a point x, one can give to the verifier the blocks of DS at the k indices identified by PP.Lookup for the point x, along with opening proofs for those indices. The verifier can check this proof and runs the PP.Reconstruct algorithm to recover $f(x)$. The prove that this construction $\mathsf{PC}_{\mathsf{VC}}$ satisfies evaluation binding, yet it is provably not strong evaluation binding.

We highlight that this technique is used in [ABC+24] to build a verifiable data structure for polynomial evaluation. Also, [ABC+24] mentions in the introduction the possibility of viewing their data structure as a polynomial commitment. However, our objective is to carefully analyze and provide a formal treatment of this statement in the setting of polynomial commitments, aiming at clarifying the extent to which security is guaranteed.

[17] Observe that being polynomial in the variables d and m (the complexity of Open) implies being sublinear in the size d^m of the multivariate polynomial.

[18] While considering PC with fast openings, a crucial requirement is the balance between memory usage (to store the precomputed data) and running time of Open.

Construction 1 *Let* PP *be a PP scheme (Definition 10) and* VC *be a vector commitment (Definition 23) scheme for a message space* $\{0,1\}^*$. *We build a PC scheme* $\mathsf{PC}_{\mathsf{VC}}$ *for multivariate polynomials over a ring* R *as follows:*

$\mathsf{Setup}(1^\lambda, m, d) \to \mathsf{ck}$**:** *On input the security parameter* λ, *the number of variables* m, *and a bound* d *for the degree of each variable, the setup algorithm computes* $\mathsf{pp} \leftarrow \mathsf{PP.Params}(R, m, d)$ *and* $\mathsf{ck}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, S)$, *where* S *is the upper bound on the size of the data structure included in* pp *(see Definition 10). Finally, it outputs* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}})$.

$\mathsf{Com}(\mathsf{ck}, f; r) \to (\mathsf{cm}, \mathsf{aux})$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}})$, *a polynomial* $f \in \mathcal{P}_{m,d}$, *and a randomness* r, *the commit algorithm computes* $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(\mathsf{pp}, f)$ *and* $(\mathsf{cm}_{\mathsf{DS}}, \mathsf{aux}_{\mathsf{DS}}) \leftarrow \mathsf{VC.Com}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{DS}; r)$ *(recall that* DS *is a vector, i.e.,* $\mathsf{DS} = (v_1, \ldots, v_S)$*). Finally, it outputs* $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$ *and* $\mathsf{aux} = (\mathsf{aux}_{\mathsf{DS}}, \mathsf{DS})$.

$\mathsf{Open}(\mathsf{ck}, x, \mathsf{aux}) \to \pi_x$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}})$, *a point* $x \in R^m$, *and auxiliary information* $\mathsf{aux} = (\mathsf{aux}_{\mathsf{DS}}, \mathsf{DS})$ *where* $\mathsf{DS} = (v_1, \ldots, v_S)$, *the open algorithm retrieves the indexes* $(i_1, ..., i_k) \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, x)$ *and compute an opening proof* $\pi_j \leftarrow \mathsf{VC.Open}(\mathsf{ck}_{\mathsf{VC}}, i_j, \mathsf{aux}_{\mathsf{DS}})$ *for every* $j \in [k]$. *Finally, it outputs* $\pi_x = (v_{i_j}, \pi_j)_{j\in[k]}$.

$\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) \to b$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}})$, *the commitment* $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$, $x \in R^m$, *an evaluation* $y \in R$, *and a proof* $\pi_x = (v_j, \pi_j)_{j\in[k]}$, *the verification algorithm outputs* $b \in \{0,1\}$ *equals to* $\bigwedge_{j\in[k]} \mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v_j, \pi_j) \stackrel{?}{=} 1 \ \wedge \ y \stackrel{?}{=} \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v_1, \ldots, v_k)$.

Our $\mathsf{PC}_{\mathsf{VC}}$ has the following properties (formal proofs are in Appendix E.1):

Theorem 4. *Let* PP, VC, *and* $\mathsf{PC}_{\mathsf{VC}}$ *as defined in Construction 1.*

1. *If both* PP *and* VC *satisfy correctness (Definitions 11 and 24), then* $\mathsf{PC}_{\mathsf{VC}}$ *satisfies correctness (Definition 18).*
2. *If* VC *satisfies position binding (Definition 25), then* $\mathsf{PC}_{\mathsf{VC}}$ *satisfies evaluation binding (Definition 3).*
3. *If* PP *satisfies efficiency (Definition 12) and* VC *satisfies succinctness (Definition 4), then* $\mathsf{PC}_{\mathsf{VC}}$ *satisfies succinctness (Definition 2).*
4. *If* PP *satisfies efficiency (Definition 12) and* VC *satisfies fast openings (Definition 5), then* $\mathsf{PC}_{\mathsf{VC}}$ *satisfies fast openings (Definition 13).*

Instantiations. One can obtain several instantiations of $\mathsf{PC}_{\mathsf{VC}}$ by using the KU-based PP schemes for either multivariate (Corollary 1) or univariate (Corollary 1 and Theorem 3) polynomials, and a VC with fast openings such as Merkle trees [Mer88], KZG [KZG10] or [CFG+20]. Corollary 2 provides an example of instantiation considering $R = \mathbb{Z}_q$ and using Merkle trees [Mer88]. For the multivariate case, we focus on particular choices of parameters d and m to remain in polynomial time.

Corollary 2 (Polynomial-time instantiations of $\mathsf{PC}_{\mathsf{VC}}$). *Assuming collision resistance hash functions, there exists a PC scheme* $\mathsf{PC}_{\mathsf{VC}}$ *for multivariate (resp. univariate) polynomials over the ring* $R = \mathbb{Z}_q$ *for* $q \approx 2^\lambda$ *that satisfies correctness, succinctness, evaluation binding, and fast opening where the running times of* $\mathsf{PC}_{\mathsf{VC}}.\mathsf{Com}$ *and* $\mathsf{PC}_{\mathsf{VC}}.\mathsf{Open}$ *are defined as follows:*

– *For univariate polynomials of degree at most* $d \in \mathsf{poly}(\lambda)$, *we have* $\mathsf{T}_{\mathsf{PC.Com}} = \mathsf{poly}(\lambda, \log d) \cdot O(d^{1+\epsilon})$ *and* $\mathsf{T}_{\mathsf{PC.Open}} = \mathsf{poly}(\lambda, \log d)$ *for every constant* $\epsilon > 0$.

– *For m-variate polynomials s.t.* $\mathsf{deg}_1(f), \ldots, \mathsf{deg}_m(f) < d$, $|f| := d^m = \mathsf{poly}(\lambda)$, *and* $d = \log^{3/\epsilon}(|f|)$ *(which in turn implies* $m = \frac{\epsilon}{3}\log(|f|)/(\log\log|f|)$*), we have* $\mathsf{T}_{\mathsf{PC.Com}} = \mathsf{poly}(\lambda, m, d) \cdot O(d^{m(1+\epsilon)})$ *and* $\mathsf{T}_{\mathsf{PC.Open}} = \mathsf{poly}(\lambda, m, d)$ *for any positive constant* $\epsilon > 0$. *Note: our choices of* d, m *permit* $|f| = d^m \in \mathsf{poly}(\lambda)$.[19]

Lack of Strong Evaluation Binding. Theorem 4 states the evaluation binding property of $\mathsf{PC}_{\mathsf{VC}}$. A natural question is whether the same construction achieves stronger binding notions, such as strong evaluation binding or knowledge soundness. In Appendix B, we give a negative answer to this question, showing that $\mathsf{PC}_{\mathsf{VC}}$ is not strongly evaluation binding. In a nutshell, the core functioning of $\mathsf{PC}_{\mathsf{VC}}$ relies on the usage of data structures DS, whose space is larger than that of the polynomial space $\mathcal{P}_{m,d}$. This allows an adversary to commit to an inconsistent data structure (with one block sampled at random) whose resulting evaluations (output of $\mathsf{PP.Reconstruct}$) can be interpolated only from polynomials of larger degrees (lying outside $\mathcal{P}_{m,d}$). We provide more details in Appendix B.

4.1 Beyond Evaluation Binding

During the execution of Com, our approach is to compute a commitment $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$ (where $\mathsf{cm}_{\mathsf{DS}}$ is computed as defined in Construction 1). In addition, we include into the auxiliary information a proof of correctness π demonstrating that DS has been computed by running $\mathsf{PP.PreProcess}$. This can be accomplished by using a SNARK for the following relation:

$$\mathcal{R}_{\mathsf{VC},\mathsf{PP}} = \{((\mathsf{pp}, \mathsf{ck}, \mathsf{cm}), (f, r)) \mid (\mathsf{cm}, \cdot) = \mathsf{VC.Com}(\mathsf{ck}, \mathsf{PP.PreProcess}(\mathsf{pp}, f); r)\} \tag{1}$$

Each time an opening proof is computed, we include π into it. Then, when verifying an opening proof, we simply perform all steps defined in Construction 1 and, in addition, verify the correctness of the commitment by checking π. Observe that the resulting PC still achieves the succinctness and fast opening properties (the latter thanks to the succinctness property of SNARKs). We refer the reader to Appendix C for more details.

Engineering the (SNARK) relation through algebraic approaches. The aforementioned approach requires a SNARK supporting $\mathcal{R}_{\mathsf{VC},\mathsf{PP}}$, which, in turn, relies on the entire computation of $\mathsf{PP.PreProcess}(\mathsf{pp}, f)$. By opening the box, we observe that the PP scheme of Kedlaya and Umans [KU08, KU11] has an algebraic structure. For example, the data structure DS (output by $\mathsf{PP.PreProcess}$) consists of multiple small tables, each containing all evaluations of a multivariate polynomial f_i (derived by manipulating the original one) modulo a small prime p_i. Due to the algebraic nature of this PP scheme, we envision the possibility of engineering and simplifying the SNARK relation to achieve a faster PC. However, this direction is outside the scope of this paper, and we leave it as a possible direction for future work.

[19] We are interested only in m-variate polynomials of polynomial size $d^m \in \mathsf{poly}(\lambda)$. Hence, it is fundamental to balance out d and m (as defined here and in Corollary 1) to enforce $d^m \in \mathsf{poly}(\lambda)$ and $m^m \in \mathsf{poly}(\lambda)$ (so that pre-processing can run in polynomial time) while obtaining non-trivial choices of d and m (i.e., choices which are different from just fixing $O(1)$ degree or $O(1)$ variables).

5 Index-Efficient SNARKs

We give applications of polynomial preprocessing schemes and PCs with fast opening to construct *index-efficient SNARKs*, i.e., preprocessing SNARKs in which the prover runs in time which is sub-linear in the size of the index.[20]

Definition 14 (Index-Efficient SNARKs). *We say that a preprocessing SNARK (Definition 6) for an indexed relation $\mathcal{R}$ is index-efficient if the running time of the prover is polylogarithmic in the index size, i.e.,* $\mathsf{poly}(\lambda, \log|\mathtt{i}|, |\mathtt{x}|, |\mathtt{w}|)$.

To construct index-efficient SNARKs, we proceed as follows. First, we formalize the notion of index-efficient PIOPs, which are a class of PIOPs in which the prover's running time depends polylogarithmically on the size of the index. Second, we show how to use index-efficient PIOPs along with PCs with fast openings in order to construct an index-efficient SNARK. Third, in Section 5.2 we study the feasibility of instantiating the above result, namely the existence of index-efficient PIOPs. Our result is a technique to transform a class of (non-index-efficient) PIOPs into index-efficient ones. Finally, in Sections 5.3 and 5.4 we show how the combination of the three results above can be used to obtain new table-efficient lookup arguments and index-efficient SNARKs for circuits.

5.1 Index-Efficient PIOPs and Applications to Index-Efficient SNARKs

We start by recalling the definition of polynomial interactive oracle proofs (PIOP) for indexed relations.

Definition 15 (Polynomial IOPs). *Let R be a ring and $\mathcal{R}$ be an indexed relation. A polynomial IOP for $\mathcal{R}$ is defined by a tuple $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ such that:*

Offline phase: *$\mathcal{I}(\mathtt{i})$ is a deterministic algorithm that on input the index $\mathtt{i}$ outputs i oracle polynomials $p_1^{\mathtt{i}}, \ldots, p_{\mathsf{i}}^{\mathtt{i}}$.*

Online phase: *the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ run an interactive protocol, compactly denoted by $\langle \mathcal{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathcal{V}^{\mathcal{I}(\mathtt{i})}(\mathtt{x})\rangle$, where in each round the prover $\mathcal{P}$ sends oracle polynomials and/or ring elements, and the verifier $\mathcal{V}$ sends random challenges. Each oracle $[\![p]\!]$ specifies the number of variables m and the degree in each variable. The verifier can query all the oracles, the ones sent by the prover and the ones generated by $\mathcal{I}$, at arbitrary points $\mathbf{r} \in R^m$. At the end of the execution, $\mathcal{V}$ accepts or rejects.*

Correctness: *If $(\mathtt{i}, \mathtt{x}; \mathtt{w}) \in \mathcal{R}$, the verifier accepts execution $\langle \mathcal{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathcal{V}^{\mathcal{I}(\mathtt{i})}(\mathtt{x})\rangle$.*

δ-Soundness: *A PIOP is δ-sound if for any unbounded adversary $\mathcal{A}$ and any $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$, we have $\Pr\left[\langle \mathcal{A}(\mathtt{i}, \mathtt{x}), \mathcal{V}^{\mathcal{I}(\mathtt{i})}(\mathtt{x})\rangle = 1\right] \leq \delta$.*

δ-Knowledge Soundness: *A PIOP is δ-knowledge-sound if there exists a PPT oracle machine Ext (which can query all the oracle polynomials at arbitrary points) such that for any adversary $\mathcal{A}$ and any $(\mathtt{i}, \mathtt{x})$, the following probability holds:*

$$\Pr\left[\langle \mathcal{A}(\mathtt{i}, \mathtt{x}), \mathcal{V}^{\mathcal{I}(\mathtt{i})}(\mathtt{x})\rangle = 1 \ \wedge \ (\mathtt{i}, \mathtt{x}; \mathtt{w}) \notin \mathcal{R} \quad |\mathtt{w} \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathtt{i}, \mathtt{x})\right] \leq \delta.$$

[20] We remark that our notion of index-efficiency is different from [CHM+20] that uses the term "index-efficient" for an argument to indicate that the running time of the prover is polynomial in the size of the index, but independent of the size of the universal relation and the structured reference string.

We measure the efficiency of a PIOP according to the following metrics: the running times $\mathsf{T}_{\mathcal{P}}$ and $\mathsf{T}_{\mathcal{V}}$ of prover and verifier, the number of rounds r, the query complexity q (the number of queries performed by the verifier), the number of oracles sent by the prover o, and the size s of the proof oracles (the sum of the sizes of all the polynomials sent by the prover).

Next we define a class of PIOPs in which the prover's running time depends polylogarithmically on the size of the index. This is possible after a preprocessing of the index performed by the indexer. More formally:

Definition 16 (Index-Efficient PIOPs). *A PIOP for an indexed relation $\mathcal{R}$ is called index-efficient if it satisfies the following conditions:*

- *The indexer $\mathcal{I}(\mathbb{i})$ outputs oracle polynomials $p_1^{\mathbb{i}}, \ldots, p_{\mathsf{i}}^{\mathbb{i}}$ and a preprocessed index $\mathsf{hk}_{\mathbb{i}}$.*
- *The prover $\mathcal{P}(\mathsf{hk}_{\mathbb{i}}, \mathbb{x}, \mathbb{w})$ works as in a standard PIOP except that it takes as input $\mathsf{hk}_{\mathbb{i}}$ instead of the plain index $\mathbb{i}$.*
- ***Efficiency:** the running time of the prover $\mathsf{T}_{\mathcal{P}}$ is $\mathsf{poly}(\lambda, \log|\mathbb{i}|, |\mathbb{x}|, |\mathbb{w}|)$.*

Compiling Index-Efficient PIOPs into Index-Efficient SNARKs. We show that given an index-efficient PIOP and a PC with fast openings it is possible to construct an index-efficient SNARK.

Theorem 5 (Index-Efficient Compilation of PIOPs). *Let Π be a PIOP for a ring R and an indexed relation $\mathcal{R}$, let PP be a polynomial preprocessing scheme, and let PC and subPC be two polynomial commitment schemes for polynomials with coefficients in$R$. If Π has $\mathsf{negl}(\lambda)$-knowledge soundness, PC is knowledge-sound (Definition 22) and subPC is weak evaluation binding (Definition 20) then there exists a public-coin interactive argument Π for the indexed relation $\mathcal{R}$. Furthermore, if Π is index-efficient and subPC has fast openings, then Π is index-efficient.*

The construction Π is obtained via a slight variant of the usual PIOP compilation appeared in previous work [CHM+20, BFS20]. The main difference is that the polynomials produced by the PIOP indexer are committed using the PC scheme subPC with fast opening; this occurs in the preprocessing phase of the argument system. Later, for every PIOP verifier's query to such polynomials, the prover uses the PP scheme to fast-evaluate the polynomial and the PC's fast opening to create a proof. Since this compiler is similar to previous work, we give an informal presentation emphasizing the main differences in more detail.

$\Pi.\mathsf{Setup}$ runs the setup algorithms of both PC and subPC and store in srs their commitment keys. Given an index $\mathbb{i}$, $\Pi.\mathsf{Index}(\mathsf{srs}, \mathbb{i})$ calls the PIOP indexer $(p_1^{\mathbb{i}}, \ldots, p_{\mathsf{i}}^{\mathbb{i}}, \mathsf{hk}_{\mathbb{i}}) \leftarrow \mathcal{I}(\mathbb{i})$ and then, for each polynomial $p_i^{\mathbb{i}}$ it runs the PP preprocessing, $\mathsf{DS}_i \leftarrow \mathsf{PP.PreProcess}(p_i^{\mathbb{i}})$, commits to it using subPC, and stores all this preprocessed information in $\mathsf{pk}_{\mathbb{i}}$ and the commitments in $\mathsf{vk}_{\mathbb{i}}$.

The prover $\Pi.\mathsf{Prove}(\mathsf{pk}_{\mathbb{i}}, \mathbb{x}, \mathbb{w})$ runs the PIOP prover $\mathcal{P}(\mathsf{hk}_{\mathbb{i}}, \mathbb{x}, \mathbb{w})$ until the end of the interactive protocol with $\Pi.\mathcal{V}$ by sending to the verifier, in each round, commitments to the polynomials generated by $\Pi.\mathcal{P}$ computed using PC. Finally, for every query of $\mathcal{V}$ to a polynomial the prover of Π evaluates the polynomial and produces an evaluation for it. In particular, for polynomials produced by $\mathcal{P}$ this is done using standard evaluation and by using PC for the evaluation proofs; this is the same as in previous compilers. For polynomials produced by $\mathcal{I}$, the evaluations are performed using the PP scheme and the proofs are generated using $\mathsf{subPC.Open}$ algorithm.

The verifier $\Pi.\mathsf{Ver}$ runs the PIOP verifier $\Pi.\mathcal{V}$ and checks the validity of all the evaluations and the associated proofs using PC or subPC.

Compared to previous work, the main difference in the proof of knowledge soundness of Π is that our compiler uses two different polynomial commitments. The crucial observation is that we commit the polynomials produced by the PIOP indexer $\mathcal{I}$ as part of the $\Pi.\mathsf{Index}$ algorithm[21] and thus any incorrectly proven evaluation of these polynomials (which can be checked without extraction) can be reduced to the weak evaluation binding property of subPC.

The efficiency of the resulting argument depends on the efficiency of Π, PP, PC, and subPC as follows. For Π, let $\mathsf{T}_{\Pi.\mathcal{P}}$ and $\mathsf{T}_{\Pi.\mathcal{V}}$ be its prover and verifier running times, and let $\mathsf{r}, \mathsf{q}_{\mathcal{I}}, \mathsf{q}_{\mathcal{P}}, \mathsf{o}, \mathsf{s}$ be, respectively, its round complexity, number of verifier's queries to $\mathcal{I}$'s and $\mathcal{P}$'s oracles respectively, number of prover's oracles, and total size of the prover's oracles. For PC, let $\mathsf{T}_{\mathsf{PC.Com}}, \mathsf{T}_{\mathsf{PC.Open}}, \mathsf{T}_{\mathsf{PC.Ver}}$ be the running times of the corresponding algorithms, and let $|\mathsf{cm}|$ and $|\pi_{\mathsf{PC}}|$ be the size of a commitment and a proof respectively, and consider analogous notations for subPC complexity. Then the argument Π enjoys the following efficiency:

- Proving time is $\mathsf{T}_{\Pi.\mathcal{P}} + \mathsf{T}_{\mathsf{PC.Com}} \cdot \mathsf{s} + (\mathsf{T}_{\mathsf{subPC.Open}} + \mathsf{T}_{\mathsf{PP.Lookup}} + \mathsf{T}_{\mathsf{PP.Reconstruct}}) \cdot \mathsf{q}_{\mathcal{I}} + \mathsf{T}_{\mathsf{PC.Open}} \cdot \mathsf{q}_{\mathcal{P}}$
- Verifier time is $\mathsf{T}_{\Pi.\mathcal{V}} + \mathsf{T}_{\mathsf{subPC.Ver}} \cdot \mathsf{q}_{\mathcal{I}} + \mathsf{T}_{\mathsf{PC.Ver}} \cdot \mathsf{q}_{\mathcal{P}}$
- Proof size is $|\mathsf{cm}| \cdot \mathsf{o} + |\pi_{\mathsf{PC}}| \cdot \mathsf{q}_{\mathcal{P}}$

One can see that the index-efficiency of Π's prover follows by the facts that the following terms are all polylogarithmic in $|\mathbb{i}|$: $\mathsf{T}_{\Pi.\mathcal{P}}$, and thus s and $\mathsf{q}_{\mathcal{P}}$, due to the index-efficiency of Π, $\mathsf{q}_{\mathcal{I}}$ due to the succinctness of the PIOP, $\mathsf{T}_{\mathsf{subPC.Open}}$ due to the fast opening property of subPC, $\mathsf{T}_{\mathsf{PP.Lookup}} + \mathsf{T}_{\mathsf{PP.Reconstruct}}$ due to the efficiency of PP, and $\mathsf{T}_{\mathsf{PC.Open}}$ since it is a function of the size of the polynomials produced by $\Pi.\mathcal{P}$ which is polylogarithmic in $|\mathbb{i}|$ due to the index-efficiency of Π.

5.2 Our Construction of Index-Efficient PIOPs

In the following theorem we show how to transform a class of (non-index-efficient) PIOPs into index-efficient ones. The main technical obstacle in this transformation is that we want to transform PIOPs that work with generic multivariate polynomials whose configuration for number of variables and degree is not covered by known preprocessing schemes, which only capture either univariate polynomials or multivariate polynomials with very specific configurations. The most popular example are PIOPs working with multilinear polynomials. Our contribution is a technique that solves this issue by encoding the (possibly multilinear) polynomial produced by the indexer via a low-degree multivariate polynomial for which we can use a PP.

Theorem 6 (Index-Efficient PIOPs from PP schemes). *Let PP be a polynomial preprocessing scheme for m-variate polynomials of degree $\leq d$ such that $(d+1)^m = 2^n$ and d is at most $\mathsf{poly}(n)$. Let Π be a PIOP for a finite field $\mathbb{F} = \mathbb{Z}_q$ for a prime $q \approx 2^\lambda$ and for an indexed relation $\mathcal{R}$, where:*

1. *$\Pi.\mathcal{I}$ produces polynomials $\bar{p}_1^{\mathbb{i}}, \ldots, \bar{p}_{\mathsf{i}}^{\mathbb{i}} \in \mathbb{Z}_q[X_1, \ldots, X_n]$ of degree $\bar{d}$.*

There is an implementation of Π such that:

2. *it remains correct if one replaces each indexer's polynomial $\bar{p}_j^{\mathbb{i}}$ with a polynomial $p_j^{\mathbb{i}}$ of different degree $d > 1$ that agrees on the boolean hypercube, i.e., $\forall \mathbf{b} \in \{0,1\}^n : \bar{p}_j^{\mathbb{i}}(\mathbf{b}) = p_j^{\mathbb{i}}(\mathbf{b})$;*
3. *the prover's algorithm has only oracle access to the indexer's polynomials and its running time is $\mathsf{T}_{\Pi.\mathcal{P}} = \mathsf{T}_{\mathbb{F}} + \mathsf{T}_{\mathbb{i}}$, where $\mathsf{T}_{\mathbb{F}}$ is the number of field operations and $\mathsf{T}_{\mathbb{i}}$ the number of oracle calls to the indexer's polynomials;*

[21] In the real world, this algorithm is supposed to be executed by the verifier during preprocessing; in the knowledge-soundness experiment it is executed by the challenger.

4. *the prover's time* $\mathsf{T}_{\mathbb{F}}, \mathsf{T}_{\mathbb{i}}$ *and the remaining efficiency measures of* Π *depend only polynomially on the degree* $\bar{d}$ *and the number of variables* n *of the indexer's polynomials (namely, we can express them as functions* $\mathsf{T}_{\mathbb{F}}(\bar{d}, n), \mathsf{T}_{\mathbb{i}}(\bar{d}, n), \mathsf{r}(\bar{d}, n)$, $\mathsf{q}(\bar{d}, n)$, $\mathsf{o}(\bar{d}, n)$, $\mathsf{s}(\bar{d}, n)$, *ignoring for simplicity their dependence on* $|\mathbb{x}|, |\mathbb{w}|$*);*
5. *for* $\bar{d} = \mathsf{poly}(\lambda)$ *it achieves* $\delta(\bar{d}, n)$*-knowledge-soundness with* $\delta(\bar{d}, n) = \mathsf{negl}(\lambda)$.

Then, there exists an index-efficient PIOP $\hat{\Pi}$ *for the indexed relation* $\mathcal{R}$.

We now provide the required preliminaries on Lagrange interpolation before describing our index-efficient PIOP construction $\hat{\Pi}$ (see Construction 2) stated in Theorem 6 above. Then, we provide the corresponding proof.

Preliminaries on Lagrange interpolation. For any $\mathbf{b} \in \{0,1\}^n$, let $\tilde{eq}(\mathbf{X}; \mathbf{b}) = \prod_{i=1}^{n}(X_i b_i + (1 - X_i)(1 - b_i))$ be the multilinear polynomial in $\mathbb{F}[\mathbf{X}]$ such that, for every $\mathbf{x} \in \{0,1\}^n$, $\tilde{eq}(\mathbf{x}; \mathbf{b}) = 1$ if $\mathbf{x} = \mathbf{b}$, and is 0 otherwise. The set $\{\tilde{eq}(\mathbf{X}; \mathbf{b}) : \mathbf{b} \in \{0,1\}^n\}$ is the set of Lagrange basis polynomials for interpolation domain $\{0,1\}^n$. Here we shortly call it a $\{0,1\}^n$-basis.

Given a set $\mathbb{H} \subset \mathbb{F}$ of size $d + 1$, we define the m-variate polynomials $\{L(\mathbf{Y}; \mathbf{h}) : \mathbf{h} \in \mathbb{H}^m\}$ of individual degree $\leq d$ such that, for every $\mathbf{y} \in \mathbb{H}^m$, $L(\mathbf{y}; \mathbf{h}) = 1$ if $\mathbf{y} = \mathbf{h}$, and is 0 otherwise. These polynomials are the set of Lagrange basis polynomials for interpolation domain $\mathbb{H}^m$, or shortly an $\mathbb{H}^m$-basis. A useful property of Lagrange polynomials is that, since they form a basis, for any m-variate polynomial p of degree $\leq d$ we have that $p(\mathbf{Y}) = \sum_{\mathbf{h} \in \mathbb{H}^m} p(\mathbf{h}) \cdot L(\mathbf{Y}; \mathbf{h})$.

For any $h \in \mathbb{H}$, let $L'(Y, h)$ be the h-th Lagrange univariate polynomial for interpolation domain $\mathbb{H}$, i.e., $L'(Y; h) = \prod_{h' \in \mathbb{H}, h' \neq h} \frac{Y - h'}{h - h'}$, that is of degree $d = |\mathbb{H}| - 1$. Then, L is the product of the m Lagrange polynomials $L(Y_1, \ldots, Y_m; h_1, \ldots, h_m) = \prod_{i=1}^{m} L'(Y_i; h_i)$.

For $\mathbb{F} = \mathbb{Z}_q$, let $\mathbb{H} = \{0, 1, \ldots, d\}$ and let $m, n, \delta \in \mathbb{Z}$ such that $(d+1)^m = 2^n$ and $\delta \cdot m = n$ (i.e. $\delta = \log_2(d+1)$). We define the linear map: $\mathsf{G} : \mathbb{F}^n \to \mathbb{F}^m$, $(x_1, \ldots, x_n) \mapsto \left(\sum_{\eta=1}^{\delta} x_{\delta(k-1)+\eta} \cdot 2^{\eta-1}\right)_{k=1}^{m}$.[22]

Construction 2 (Index-efficient PIOP) *Let* Π *be a PIOP (Definition 15) and* PP *be a polynomial preprocessing scheme (Definition 12). Then, we build an index-efficient PIOP* $\hat{\Pi}$ *as follows:*

Indexer $\hat{\Pi}.\mathcal{I}(\mathbb{i})$**:** *Run the indexer of* Π *to get the polynomials* $(\bar{p}_1^{\mathbb{i}}, \ldots, \bar{p}_{\mathsf{i}}^{\mathbb{i}}) \leftarrow \mathcal{I}(\mathbb{i})$*. Then, for each polynomial* $\bar{p}_j^{\mathbb{i}}$*, proceed as follows.*

- *Choose integers* m, d *such that* $(d+1)^m = 2^n$ *and for which there exists an efficient PP scheme, and generate* $\mathsf{PP.Params}(\mathbb{Z}_q, m, d)$.
- *Build the* m*-variate polynomial* $\hat{p}_j^{\mathbb{i}}(Y_1, \ldots, Y_m) = \sum_{\mathbf{b} \in \{0,1\}^n} \bar{p}_j^{\mathbb{i}}(\mathbf{b}) \cdot L(Y_1, \ldots, Y_m; \mathsf{G}(\mathbf{b}))$ *of individual degree* $\leq d$ *such that* $(d+1)^m = 2^n$.
- *Compute* $\mathsf{hk}_{\mathbb{i}} = \mathsf{DS}_j \leftarrow \mathsf{PP.PreProcess}(\mathsf{pp}, \hat{p}_j^{\mathbb{i}})$.
- *Build* $p_j^{\mathbb{i}}(X_1, \ldots, X_n) = \hat{p}_j^{\mathbb{i}}(\mathsf{G}(X_1, \ldots, X_n)) = \hat{p}_j^{\mathbb{i}}\big(\big(\sum_{\eta \in [\delta]} 2^{\eta-1} X_{\delta(k-1)+\eta}\big)_{k \in [m]}\big)$.

Finally, output $p_1^{\mathbb{i}}, \ldots, p_{\mathsf{i}}^{\mathbb{i}}$ *and* $\mathsf{hk}_{\mathbb{i}} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_{\mathsf{i}})$.

Prover $\hat{\Pi}.\mathcal{P}(\mathsf{hk}_{\mathbb{i}}, \mathbb{x}, \mathbb{w})$**:** *It runs* $\Pi.\mathcal{P}$ *on* $(\mathbb{x}, \mathbb{w})$ *and for every oracle call of* $\Pi.\mathcal{P}$ *to* $p_j^{\mathbb{i}}$ *on a point* $\mathbf{z} \in \mathbb{F}^n$*,* $\hat{\Pi}.\mathcal{P}$ *answers as follows:*

- *Compute* $\hat{\mathbf{z}} = (\hat{z}_i)_{i \in [m]} = \mathsf{G}(\mathbf{z})$*, the evaluation* $y_j = \hat{p}_j^{\mathbb{i}}(\hat{\mathbf{z}})$ *where* $(i_j)_{j \in [k]} \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, \hat{\mathbf{z}})$ *and* $y_j \leftarrow \mathsf{PP.Reconstruct}(\mathsf{pp}, \mathsf{DS}_{j,i_1}, \ldots, \mathsf{DS}_{j,i_k})$.

[22] This is almost the same as the gadget matrix of lattice-based cryptosystems.

– *Return* y_j *to* $\Pi.\mathcal{P}$.

Verifier $\hat{\Pi}.\mathcal{V}(\mathbb{x})$**:** *This is identical to* $\Pi.\mathcal{V}$.

Correctness of $\hat{\Pi}$. Notice that $p_j^{\mathbb{i}}$ is an n-variate polynomial of individual degree $\leq d$ that agrees with $\bar{p}_j^{\mathbb{i}}$ over the n-dimensional hypercube since

$$\forall \mathbf{b} \in \{0,1\}^n : \bar{p}_j^{\mathbb{i}}(\mathbf{b}) = \hat{p}_j^{\mathbb{i}}(\mathsf{G}(\mathbf{b})) = p_j^{\mathbb{i}}(\mathbf{b})$$

where the second equality follows by construction of $p_j^{\mathbb{i}}$ from $\hat{p}_j^{\mathbb{i}}$, and the second equality by construction of $\hat{p}_j^{\mathbb{i}}$ from $\bar{p}_j^{\mathbb{i}}$ and the fact that $\forall \mathbf{b}, \mathbf{b}' \in \{0,1\}^n$, $L(\mathsf{G}(\mathbf{b}), \mathsf{G}(\mathbf{b}')) = 1$ if $\mathbf{b}' = \mathbf{b}$ and is 0 otherwise. Therefore by property (2) in the Theorem statement, we get correctness from that of Π.

Efficiency of $\hat{\Pi}$. Let us start by analyzing the efficiency of the prover. $\hat{\Pi}.\mathcal{P}$ executes $\Pi.\mathcal{P}$ while simulating each of the $\mathsf{T}_{\mathbb{i}}$ oracle calls of $\Pi.\mathcal{P}$ to the indexer polynomials by first computing $\mathbf{z}$, which takes $O(n)$ $\mathbb{Z}_q$-operations, and then by doing a fast evaluation of the polynomial using the PP scheme, which costs $\mathsf{poly}(\log q, n)$ due to the efficiency property of PP. Hence, $\mathsf{T}_{\hat{\Pi}.\mathcal{P}} = \mathsf{T}_{\mathbb{F}}(d,n) + \mathsf{T}_{\mathbb{i}}(d,n) \cdot \mathsf{poly}(\log q, n)$ which depends only polynomially on n if so do $\mathsf{T}_{\mathbb{F}}(d,n), \mathsf{T}_{\mathbb{i}}(d,n)$ and d, as assumed in the Theorem's statement. The same polynomial dependence on n holds for all the other efficiency measures.

Security of $\hat{\Pi}$. By construction, if Π has $\delta(\bar{d}, n)$-knowledge-soundness, then $\hat{\Pi}$ has $\delta(d, n)$-knowledge-soundness. By the property (5) of Π in the assumption of the theorem, Π has knowledge-soundness $\delta(\bar{d}, n) = \mathsf{negl}(\lambda)$ whenever $\bar{d} = \mathsf{poly}(\lambda)$. In $\hat{\Pi}$, the choice of d such that $(d+1)^m = 2^n = \mathsf{poly}(\lambda)$ ensures $d = \mathsf{poly}(\lambda)$, and thus $\delta(d, n) = \mathsf{negl}(\lambda)$.

5.3 Application to Table-Efficient Lookup Arguments

As a novel application of our techniques in this section, we can instantiate Theorem 6 using the GeneralizedLasso PIOP in [STW24, Appendix F.2] which supports lookup relations over arbitrary tables. In a nutshell, this is a PIOP in which the indexer produces a multilinear polynomial $\tilde{t} \in \mathbb{F}[X_1, \ldots, X_n]$ such that each $\tilde{t}(\mathbf{b})$ encodes a distinct value of the lookup table, and then the prover can send a multilinear polynomial $\tilde{a} \in \mathbb{F}[X_1, \ldots, X_\mu]$ and convince the verifier that $\forall \mathbf{j} \in \{0,1\}^\mu : \tilde{a}(\mathbf{j}) \in \{\tilde{t}(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n\}$.

A crucial building block of the GeneralizedLasso PIOP is a PIOP for sparse-dense sumcheck to prove that $y = \sum_{\mathbf{b} \in \{0,1\}^n} \tilde{u}(\mathbf{b}) \cdot \tilde{t}(\mathbf{b})$, where $\tilde{t}$ is the same multilinear polynomial produced by the indexer, and $\tilde{u}$ is a multilinear n-variate polynomial (implicitly defined by other polynomials previously sent by the prover) that is sparse, namely the number of nonzero values in $\{\tilde{u}(\mathbf{b}) : \mathbf{b} \in \{0,1\}^n\}$ is $2^\mu \ll 2^n$. This PIOP for sparse-dense sumcheck is the only component of the protocol that uses the large polynomial $\tilde{t}$.

This PIOP is simply a PIOP for the sumcheck protocol over the polynomial $g(\mathbf{X}) = \tilde{u}(\mathbf{X}) \cdot \tilde{t}(\mathbf{X})$ and its main distinction is that [STW24, Theorem 6] shows how to implement its prover in $(2^\mu \cdot n \cdot \mathsf{EvalTime}(\tilde{t}))$ field operations. This result is useful in those use cases where the polynomial associated to the lookup table has a closed form that allows fast evaluation (e.g., $\mathsf{poly}(n)$) and thus whenever $2^\mu \ll 2^n$ the prover time loosely depends on the size of the table.

Next, we show a generalized description of the sparse-dense sumcheck PIOP [STW24] that makes it obvious that this protocol works with any polynomial t which is not necessarily multilinear. Let

$S_u = \{\mathbf{b} \in \{0,1\}^n : \tilde{u}(\mathbf{b}) \neq 0\}$ be the set of indices where $\tilde{u}(\mathbf{b})$ is nonzero and assume that the prover stores $\tilde{u}$ in evaluation form with the pair $(S_u, \{\tilde{u}(\mathbf{b}) : \mathbf{b} \in S_u\})$. The sparse-dense sumcheck PIOP [STW24] is defined as follows:

SparseSumcheckProd PIOP $\langle \mathcal{P}^{\bar{t}(\cdot)}(y; S_u, \{\tilde{u}(\mathbf{b}) : \mathbf{b} \in S_u\}), \mathcal{V}^{\bar{t}(\cdot), \tilde{u}(\cdot)}(y) \rangle$

for $j = 1, \ldots, n$ **do** // Below, assume $E[\varepsilon] = 1$ and that all $g_{j,k}$ are initialized to 0.

Prover $\mathcal{P}^{\bar{t}(\cdot)}(y; S_u, \{\tilde{u}(\mathbf{b}) : \mathbf{b} \in S_u\})$:

for b in S_u **do**

// $E[b_1, \ldots, b_{j-1}] = \tilde{eq}(r_1, \ldots, r_{j-1}; b_1, \ldots, b_{j-1})$

$E[b_1, \ldots, b_{j-1}] \leftarrow E[b_1, \ldots, b_{j-2}] \cdot \tilde{eq}(r_{j-1}; b_{j-1})$

for $k = 0, 1, \ldots \bar{d} + 1$ **do**

// Incrementing $g_{j,k}$ with

// $\tilde{u}(\mathbf{b})\tilde{eq}(r_1, \ldots, r_{j-1}, k, b_{j+1}, \ldots, b_n; \mathbf{b})\bar{t}(r_1, \ldots, r_{j-1}, k, b_{j+1}, \ldots, b_n)$

$g_{j,k} \leftarrow g_{j,k} + \tilde{u}(\mathbf{b}) \cdot E[b_1, \ldots, b_{j-1}] \cdot \tilde{eq}(k; b_j) \cdot \bar{t}(r_1, \ldots, r_{j-1}, k, b_{j+1}, \ldots, b_n)$

// $g_{j,k} = \sum_{\mathbf{b} \in S_u} \tilde{u}(\mathbf{b})\tilde{eq}(r_1, \ldots, r_{j-1}, k, b_{j+1}, \ldots, b_n, \mathbf{b})\bar{t}(r_1, \ldots, r_{j-1}, k, b_{j+1}, \ldots, b_n)$

Send $(g_{j,0}, \ldots, g_{j,\bar{d}+1})$

Verifier: Send $r_j \leftarrow\!\!\$\ \mathbb{F}$

Verifier: $\mathcal{V}^{\bar{t}(\cdot), \tilde{u}(\cdot)}(y)$:

Query $\tilde{u}(r_1, \ldots, r_n), \bar{t}(r_1, \ldots, r_n)$ and accept if the following condition holds:

$$\left(\forall j \in [n] : g_{j-1}(r_{j-1}) \stackrel{?}{=} \sum_{b \in \{0,1\}} g_j(b)\right) \wedge \left(g_n(r_n) \stackrel{?}{=} \tilde{u}(r_1, \ldots, r_n) \cdot \tilde{t}(r_1, \ldots, r_n)\right)$$

where $g_0(r_0) = y$ and $g_j(X) \leftarrow \mathsf{Interpolate}(\{0, \ldots, \bar{d} + 1\}, \{g_{j,0}, \ldots, g_{j,\bar{d}+1}\})$

Fig. 1. PIOP SparseSumcheckProd for $y = \sum_{\mathbf{b} \in \{0,1\}^n} \tilde{u}(\mathbf{b}) \cdot \bar{t}(\mathbf{b})$.

Let us now see how this PIOP satisfies the 5 requirements of Theorem 6, and thus it can be compiled into an index-efficient PIOP.

- The first requirement, that the indexer produces a polynomial $\bar{t}$ follows by construction.
- If we replace the polynomial $\bar{t}$ by a polynomial t of (different) degree $d > 1$ (which agrees with $\bar{t}$ on the boolean hypercube), the correctness is preserved because the sumcheck claim $y = \sum_{\mathbf{b} \in \{0,1\}^n} \tilde{u}(\mathbf{b}) \cdot \bar{t}(\mathbf{b})$ remains identical.
- By our description above, it is straightforward to see that the prover can be implemented using evaluation calls to the $\bar{t}(\cdot)$ polynomial.
- Let $\bar{d} = \mathsf{deg}(\bar{t})$. Then by inspection of the algorithm above, the prover performs $\mathsf{T}_{\mathbb{F}} = O(2^\mu \cdot n \cdot \bar{d})$ field operations, plus $\mathsf{T}_{\mathbb{i}} = O(2^\mu \cdot n \cdot \bar{d})$ evaluation calls to $\bar{t}(\cdot)$. Furthermore, by the efficiency profile of the sumcheck protocol executed on a polynomial of degree $\leq \bar{d} + 1$, we have that: the verifier makes $\mathsf{q}_{\mathcal{I}}(\bar{d}, n) = 1$ query to the indexer's polynomial $\bar{t}$ and $\mathsf{q}_{\mathcal{P}}(\bar{d}, n) = 1$ query to the polynomial $\tilde{u}$, and the prover sends $O(\bar{d} \cdot n)$ field elements in n rounds.
- Π achieves $\delta(\bar{d}, n)$-knowledge-soundness with $\delta(\bar{d}, n) = \frac{n(\bar{d}+1)}{|\mathbb{F}|}$ and $\bar{d} = \mathsf{deg}(\bar{t})$.

5.4 Application to Index-Efficient SNARKs for Circuits

In this section, we show how to use techniques similar to those of Section 5.2 to construct an index-efficient PIOP for linear checks, namely to prove that $\mathbf{v} = \mathbf{M} \cdot \mathbf{z}$ in time $N \cdot \mathsf{polylog}(N)$ when the

$N \times N$ matrix $\mathbf{M}$ is fixed and preprocessed and $\mathbf{v}, \mathbf{z}$ are two vectors sent by the prover. This result is new because in existing PIOPs the prover complexity is at best $O(\max(||\mathbf{M}||, N))$, hence linear in the number of nonzero entries of the matrix $||\mathbf{M}||$, which is N^2) in the worst case.

Our new index-efficient PIOP for linear checks can be used to construct an index-efficient PIOP for (universal) rank-1 constraint systems (R1CS), which can be in turn used to build an index-efficient PIOP for circuit satisfiability. In R1CS, the universal relation is indexed by matrix tuples $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and the prover needs to convince the verifier that $\exists \mathbf{z} : (\mathbf{A} \cdot \mathbf{z}) \circ (\mathbf{B} \cdot \mathbf{z}) = (\mathbf{C} \cdot \mathbf{z})$. Several works show how to construct a PIOP for R1CS by combining a PIOP for linear checks (or "lincheck"), to prove $\exists \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{z} : \mathbf{a} = \mathbf{A} \cdot \mathbf{z}, \mathbf{b} = \mathbf{B} \cdot \mathbf{z}, \mathbf{c} = \mathbf{C} \cdot \mathbf{z}$, with a PIOP for Hadamard products, to prove that $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$. Therefore, to build an index-efficient PIOP for R1CS we can simply replace the lincheck PIOP in other constructions, e.g., [Set20], with our index-efficient one. This way, after preprocessing the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, the prover's running time does not depend on their density and thus it is index-efficient whenever the number of nonzero entries is $\omega(N\mathsf{polylog}(N))$. This is for example the case for circuits with many more addition gates than multiplication gates e.g., $O(N^2)$ addition gates and $O(N)$ multiplication gates.

For PIOPs for these specific indexed relations—lincheck, R1CS, and circuit satisfiability—we use a notion of index efficiency more restrictive than Definition 16. We additionally require the overall prover's running time $\mathsf{T}_{\mathcal{P}}$ to be $o(|\mathbb{i}|)$. This is because the generic requirement $\mathsf{T}_{\mathcal{P}} = \mathsf{poly}(\lambda, \log|\mathbb{i}|, |\mathbb{x}|, |\mathbb{w}|)$ might allow $\mathsf{T}_{\mathcal{P}} = |\mathbb{w}|^2$, which is too permissive in these cases.

Our Index-Efficient Lincheck PIOP. We construct an index-efficient PIOP for linear checks, where the index is an $N \times N$ matrix $\mathbf{M}$, the statement are two vectors $\mathbf{z}, \mathbf{v}$ of size $N = 2^n$, and the prover convinces the verifier that $\mathbf{v} = \mathbf{M} \cdot \mathbf{z}$ in time $N\mathsf{polylog}(N)$ (after a preprocessing of $\mathbf{M}$).

We present the PIOP assuming that the verifier has oracle access to the multilinear extensions $\tilde{z}, \tilde{v} \in \mathbb{F}[X_1, \ldots, X_n]$ of $\mathbf{z}, \mathbf{v} \in \mathbb{F}^N$. Our PIOP is inspired to the matrix multiplication interactive proof of Thaler [Tha13] (in the special case when one matrix, $\mathbf{M}$, is fixed and the others have only one column). In such a protocol letting $\tilde{M}$ be the $2n$-variate multilinear polynomial $\tilde{M}$ interpolating $\mathbf{M}$, one proves that $\forall \mathbf{h} \in \{0,1\}^n : \tilde{v}(\mathbf{h}) = \sum_{\mathbf{b} \in \{0,1\}^n} \tilde{M}(\mathbf{h}, \mathbf{b}) \cdot \tilde{z}(\mathbf{b})$.

To obtain index-efficiency, we use a technique similar to that of our generic transformation (Construction 2). Namely, we replace the multilinear $\tilde{M}$ with an m-variate polynomial $\hat{M}$ of degree $\leq d$ such that $(d+1)^m = 2^n$ and for all $\mathbf{h}, \mathbf{b} \in \{0,1\}^n$ it holds $\hat{M}(\mathsf{G}(\mathbf{h}), \mathsf{G}(\mathbf{b})) = \tilde{M}(\mathbf{h}, \mathbf{b})$. The indexer builds this polynomial $\hat{M}$ and uses the polynomial preprocessing scheme PP to produce a data structure that enables computing the evaluations of $\hat{M}$ efficiently. In the online phase, the goal is then to prove that $\forall \mathbf{h} \in \{0,1\}^n : \tilde{v}(\mathbf{h}) = \sum_{\mathbf{b} \in \{0,1\}^n} \hat{M}(\mathsf{G}(\mathbf{h}), \mathsf{G}(\mathbf{b})) \cdot \tilde{z}(\mathbf{b})$. We aggregate these 2^n statements using Lagrange polynomials in the $\mathbb{H}^m$ basis, namely we check the polynomial identity

$$\sum_{\mathbf{h} \in \{0,1\}^n} \tilde{v}(\mathbf{h}) \cdot L(\mathsf{G}(\mathbf{h}); \mathbf{Y}) = \sum\nolimits_{\mathbf{b} \in \{0,1\}^n} \hat{M}(\mathbf{Y}, \mathsf{G}(\mathbf{b})) \cdot \tilde{z}(\mathbf{b})$$

by evaluating it on a random point $\mathbf{Y} = \mathbf{s}$ and by using two sumchecks to test that $y = \sum_{\mathbf{h} \in \{0,1\}^n} \tilde{v}(\mathbf{h}) \cdot L(\mathsf{G}(\mathbf{h}); \mathbf{s})$ and $y = \sum_{\mathbf{b} \in \{0,1\}^n} \hat{M}(\mathbf{s}, \mathsf{G}(\mathbf{b})) \cdot \tilde{z}(\mathbf{b})$. In each sumcheck, the prover uses the algorithm of Fig. 1 by which the prover can be implemented with access to an evaluation oracle for one of the two polynomials.[23] In the first sumcheck, the polynomial to evaluate in each call is $L(\mathsf{G}(\mathbf{X}); \mathbf{s})$ that can be computed in time $O(md)$. In the second sumcheck, the polynomial to evaluate in each call is $\hat{M}(\mathbf{s}, \mathsf{G}(\mathbf{X}))$ whose evaluations can be computed using the preprocessing scheme.

[23] For this application, we do not need the property that one of the polynomials is sparse; we only rely on the fact that $\mathcal{P}$ works with oracle access to $\bar{t}(\cdot)$.

Below we present the construction in more detail.

Construction 3 (Index-efficient PIOP for linear checks) *Let $\hat{\Pi}$ be the PIOP for sumcheck on the product of two polynomials $\tilde{u}, \bar{t}$ of Fig. 1 and let PP be a polynomial preprocessing scheme (Definition 12). We build an index-efficient PIOP for linear checks as follows:*

Indexer $\mathcal{I}(\mathbf{M})$: *Let $\tilde{M}$ be the multilinear extension of the matrix $\mathbf{M}$.*

- *Choose integers m, d such that $(d+1)^m = 2^n$ and for which there exists an efficient PP scheme, and generate $\mathsf{PP.Params}(\mathbb{Z}_q, 2m, d)$.*
- *Build the $2m$-variate polynomial $\hat{M}(Y_1, \ldots, Y_m, Z_1, \ldots, Z_m) = \sum_{\mathbf{h},\mathbf{b} \in \{0,1\}^n} \tilde{M}(\mathbf{h}, \mathbf{b}) \cdot L(Y_1, \ldots, Y_m; \mathsf{G}(\mathbf{h})) \cdot L(Z_1, \ldots, Z_m; \mathsf{G}(\mathbf{b}))$ of individual degree $\leq d$ such that $(d+1)^m = 2^n$.*
- *Compute $\mathsf{hk}_{\mathbb{i}} = \mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(\mathsf{pp}, \hat{M})$.*

Output $\hat{M}$ and $\mathsf{hk}_{\mathbb{i}} = \mathsf{DS}$.

Online phase $\langle \mathcal{P}^{\hat{M}(\cdot)}(\mathsf{DS}, \mathbf{v}, \mathbf{z}), \mathcal{V}^{\hat{M}(\cdot), \tilde{v}(\cdot), \tilde{z}(\cdot)} \rangle$: *let $\tilde{v}, \tilde{z}$ be the multilinear extensions of $\mathbf{v}, \mathbf{z}$.*

> *// Goal: prove* $\forall \mathbf{h} \in \{0,1\}^n : \tilde{v}(\mathbf{h}) = \sum_{\mathbf{b} \in \{0,1\}^n} \hat{M}(\mathsf{G}(\mathbf{h}), \mathsf{G}(\mathbf{b})) \cdot \tilde{z}(\mathbf{b})$
>
> ***Verifier*** $\mathcal{V}$: *Send* $\mathbf{s} \leftarrow\$ \mathbb{F}^m$
>
> ***Prover*** $\mathcal{P}$: *send* $y = \sum_{\mathbf{h} \in \{0,1\}^n} \tilde{v}(\mathbf{h}) \cdot L(\mathsf{G}(\mathbf{h}); \mathbf{s}) = \sum_{\mathbf{b} \in \{0,1\}^n} \hat{M}(\mathbf{s}, \mathsf{G}(\mathbf{b})) \cdot \tilde{z}(\mathbf{b})$
>
> *Let* $\phi(\mathbf{X}) = L(\mathsf{G}(\mathbf{X}); \mathbf{s}), \mu(\mathbf{X}) = \hat{M}(\mathbf{s}, \mathsf{G}(\mathbf{X}))$
>
> *$\mathcal{P}$ and $\mathcal{V}$ run $\langle \hat{\Pi}.\mathcal{P}^{\phi(\cdot)}(y, \tilde{v}), \hat{\Pi}.\mathcal{V}^{\tilde{v}, \phi}(y) \rangle$ on statement $y = \sum_{\mathbf{h} \in \{0,1\}^n} \tilde{v}(\mathbf{h}) \cdot \phi(\mathbf{h})$, where:*
>
> *$\mathcal{P}$ runs $\hat{\Pi}.\mathcal{P}^{\phi(\cdot)}(y, \tilde{v})$ and replies to each query $\phi(\mathbf{x})$ by computing $L(\mathsf{G}(\mathbf{x}); \mathbf{s})$;*
>
> *$\mathcal{V}$ runs $\hat{\Pi}.\mathcal{V}^{\tilde{v}, \phi}(y)$ and replies to the query $\tilde{v}(\mathbf{t})$ using its own oracle $\tilde{v}$ and to the query $\phi(\mathbf{t})$ by computing $L(\mathsf{G}(\mathbf{t}); \mathbf{s})$.*
>
> *$\mathcal{P}$ and $\mathcal{V}$ run $\langle \hat{\Pi}.\mathcal{P}^{\mu(\cdot)}(y, \tilde{z}), \hat{\Pi}.\mathcal{V}^{\tilde{z}, \mu}(y) \rangle$ on the statement $y = \sum_{\mathbf{b} \in \{0,1\}^n} \tilde{z}(\mathbf{h}) \cdot \mu(\mathbf{b})$, where:*
>
> *$\mathcal{P}$ runs $\hat{\Pi}.\mathcal{P}^{\mu(\cdot)}(y, \tilde{z})$ and replies to each query $y = \mu(\mathbf{x})$ by computing $\hat{\mathbf{x}} = \mathsf{G}(\mathbf{x})$, $(i_j)_{j \in [k]} \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, \hat{\mathbf{x}})$, and $y = \hat{M}(\mathbf{s}, \hat{\mathbf{x}}) \leftarrow \mathsf{PP.Reconstruct}(\mathsf{pp}, \mathsf{DS}_{i_1}, \ldots, \mathsf{DS}_{i_k})$;*
>
> *$\mathcal{V}$ runs $\hat{\Pi}.\mathcal{V}^{\tilde{v}, \mu}(y)$ and replies to the query $\tilde{v}(\mathbf{r})$ using its own oracle $\tilde{v}$ and to the query $\mu(\mathbf{r})$ by computing $\hat{\mathbf{r}} = \mathsf{G}(\mathbf{r})$ querying $\hat{M}(\mathbf{s}, \hat{\mathbf{r}})$.*

Theorem 7. *Let PP be a polynomial preprocessing scheme for m-variate polynomials of degree $\leq d$ such that $(d+1)^m = 2^n$ and d is at most $\mathsf{poly}(n)$. Let $\hat{\Pi}$ be the PIOP of Fig. 1 for sumcheck on the product of two polynomials over a finite field $\mathbb{F} = \mathbb{Z}_q$ for a prime $q \approx 2^\lambda$. Then the PIOP of Construction 3 is an index-efficient PIOP for the indexed relation $\{(\mathbb{i}, \mathbb{x}) = (\mathbf{M}, (\mathbf{z}, \mathbf{v})) : \mathbf{M} \cdot \mathbf{z} = \mathbf{v}\}$.*

Proof. Below we prove correctness, efficiency and soundness.

Correctness. First, notice that $\hat{M}$ is a $2m$-variate polynomial of individual degree $\leq d$ such that

$$\forall \mathbf{h}, \mathbf{b} \in \{0,1\}^n : \tilde{M}(\mathbf{h}, \mathbf{b}) = \hat{M}(\mathsf{G}(\mathbf{h}), \mathsf{G}(\mathbf{b}))$$

which follows by construction of $\hat{M}$ and by the fact that $\forall \mathbf{b}, \mathbf{b}' \in \{0,1\}^n$, $L(\mathsf{G}(\mathbf{b}), \mathsf{G}(\mathbf{b}')) = 1$ if $\mathbf{b}' = \mathbf{b}$ and is 0 otherwise.

Second, since the image of G over $\{0,1\}^n$ is the entire $\mathbb{H}^m$, if $\tilde{v}(\mathbf{h}) = \sum_{\mathbf{b}\in\{0,1\}^n} \hat{M}(\mathsf{G}(\mathbf{h}),\mathsf{G}(\mathbf{b}))\cdot \tilde{z}(\mathbf{b})$ holds $\forall \mathbf{h}\in\{0,1\}^n$, then

$$\begin{aligned}\sum_{\mathbf{h}\in\{0,1\}^n} \tilde{v}(\mathbf{h})\cdot L(\mathsf{G}(\mathbf{h});\mathbf{Y}) &= \sum_{\mathbf{b},\mathbf{h}\in\{0,1\}^n} \hat{M}(\mathsf{G}(\mathbf{h}),\mathsf{G}(\mathbf{b}))\cdot \tilde{z}(\mathbf{b})\cdot L(\mathsf{G}(\mathbf{h});\mathbf{Y})\\ &= \sum_{\mathbf{b}\in\{0,1\}^n} \hat{M}(\mathbf{Y},\mathsf{G}(\mathbf{b}))\cdot \tilde{z}(\mathbf{b})\end{aligned}$$

Thus, for any $\mathbf{s}\in\mathbb{F}^m$ it holds $y = \sum_{\mathbf{h}\in\{0,1\}^n} \tilde{v}(\mathbf{h})\cdot L(\mathsf{G}(\mathbf{h});\mathbf{s}) = \sum_{\mathbf{b}\in\{0,1\}^n}\hat{M}(\mathbf{s},\mathsf{G}(\mathbf{b}))\cdot\tilde{z}(\mathbf{b})$. Therefore, the correctness of our PIOP follows by the correctness properties of the sumcheck PIOP $\hat{\Pi}$ and of the scheme PP.

Index-Efficiency. In the first sumcheck, $\mathcal{P}$ executes $\hat{\Pi}.\mathcal{P}$ that makes $O(2^n\cdot n\cdot d)$ field operations and $O(2^n\cdot n\cdot d)$ evaluation calls to $\phi(\cdot)$. $\mathcal{P}$ can compute each $\phi(\mathbf{x}) = L(\mathsf{G}(\mathbf{x});\mathbf{s})$ with $O(md)$ field operations. The second sumcheck proceeds analogously. Notably, each of the $O(2^n\cdot n\cdot d)$ evaluation calls to $\mu(\cdot)$ is computed by $\mathcal{P}$ using the PP scheme, which costs $\mathsf{poly}(\log q, n)$ due to the efficiency property of PP. Hence, $\mathsf{T}_{\mathcal{P}} = O(2^n\cdot n\cdot d)\cdot\mathsf{poly}(\lambda,n)$ which is $N\mathsf{polylog}(N) = o(N^2)$.

Soundness. Assume that $\mathbf{v}\neq\mathbf{M}\cdot\mathbf{z}$, then there exists $\mathbf{h}^*\in\{0,1\}^n$ such that $\tilde{v}(\mathbf{h}^*)\neq\sum_{\mathbf{b}\in\{0,1\}^n}\hat{M}(\mathsf{G}(\mathbf{h}^*),\mathsf{G}(\mathbf{b}))\cdot\tilde{z}(\mathbf{b})$ and thus the

$$p(\mathbf{Y}) = \sum_{\mathbf{h}\in\{0,1\}^n}\tilde{v}(\mathbf{h})\cdot L(\mathsf{G}(\mathbf{h});\mathbf{Y}) - \sum_{\mathbf{b}\in\{0,1\}^n}\hat{M}(\mathbf{Y},\mathsf{G}(\mathbf{b}))\cdot\tilde{z}(\mathbf{b})\neq 0$$

Let E be the event $\mathsf{E}_1\wedge\mathsf{E}_2$ where

$$\mathsf{E}_1\equiv\left(y=\sum_{\mathbf{h}\in\{0,1\}^n}\tilde{v}(\mathbf{h})\cdot L(\mathsf{G}(\mathbf{h});\mathbf{s})\right),\mathsf{E}_2\equiv\left(y=\sum_{\mathbf{b}\in\{0,1\}^n}\hat{M}(\mathbf{s},\mathsf{G}(\mathbf{b}))\cdot\tilde{z}(\mathbf{b})\right)$$

and Ver_1 (resp. Ver_2) be the event that $\hat{\Pi}.\mathcal{V}^{\tilde{v},\phi}(y)$ (res. $\hat{\Pi}.\mathcal{V}^{\tilde{z},\mu}(y)$) accepts in the first (resp. second) sumcheck subprotocol.

To prove soundness, we have to bound the probability that $\mathcal{V}$ accepts, namely $\Pr[\mathsf{Ver}_1\wedge\mathsf{Ver}_2\mid p(\mathbf{Y})\neq 0]$ that, by partitioning over E and $\bar{\mathsf{E}}$, is

$$=\Pr[\mathsf{Ver}_1\wedge\mathsf{Ver}_2\wedge\mathsf{E}\mid p(\mathbf{Y})\neq 0]+\Pr[\mathsf{Ver}_1\wedge\mathsf{Ver}_2\wedge\bar{\mathsf{E}}\mid p(\mathbf{Y})\neq 0]$$

By the definition of the events and by a union bound on $\bar{\mathsf{E}}_1\vee\bar{\mathsf{E}}_2$, the probability can be upper bounded by

$$\leq\Pr[p(\mathbf{s})=0\mid p(\mathbf{Y})\neq 0]+\Pr[\mathsf{Ver}_1\mid\bar{\mathsf{E}}_1]+\Pr[\mathsf{Ver}_2\mid\bar{\mathsf{E}}_2]$$

By the soundness of the sumcheck protocol, both $\Pr[\mathsf{Ver}_1\mid\bar{\mathsf{E}}_1]$ and $\Pr[\mathsf{Ver}_1\mid\bar{\mathsf{E}}_1]$ are $\leq\frac{(d+1)m}{|\mathbb{F}|}$, while $\Pr[p(\mathbf{s})=0\mid p(\mathbf{Y})\neq 0]\leq\frac{dm}{|\mathbb{F}|}$ holds by Schwartz-Zippell. This concludes the proof that the PIOP for linear checks has negligible soundness.

6 Witness-Succinct UC SNARKs with Nearly Optimal Proving Time

Here, we show how to generically obtain UC-SNARKs with approximately-linear proving time from PCs with fast opening (for *univariate* polynomials). For a discussion on the importance of obtaining SNARKs in the UC model and its challenges, we refer the reader to the introduction.

Our starting point is [GKO+23], which proposes a general and elegant compiler which maps any simulation-extractable zkSNARK Π into an UC-secure one. In order to obtain straight-line extractability they adopt a Fischlin-style proof-of-work [Fis05]: the witness is extracted by observing the prover's queries to the random oracle (queries that are "forced on" the prover by the proof-of-work aspect of technique). A key step of their approach involves the prover sending a polynomial commitment, allegedly opening to (an encoding of) the witness.

Efficiency-preserving transformations. One of the advantages of the compiler in [GKO+23] is that it preserves the essential verification and communication complexity of the original zkSNARK Π Unfortunately, for proving time, the same compiler substantially changes the prover's efficiency profile: even if the original proving time of Π is $O(n)$ (where n is the size of the witness), the resulting UC zkSNARK will run in time at least $O(n^2)$.

The overhead above stems from the requirement in [GKO+23] that the prover queries the random oracle multiple times during the proof-of-work stage. Recall that, as mentioned above, the prover holds a commitment cm_f to a univariate polynomial f dependent on the witness. Each of the prover's query q to the random oracle is supposed to include, among other elements, a tuple (z, y, π) where π is an evaluation proof for cm_f showing that $f(z) = y$. The prover performs $\approx n$ of these invocations throughout the protocol. If at this step, we are applying a *standard* polynomial commitments, then each evaluation proof requires approximately $O(\mathsf{deg}(f))$ steps.[24] Since it holds that $\mathsf{deg}(f) \geq n$, a traditional polynomial commitment will lead to a total quadratic proving time.

A first (flawed) approach. It seems that using our PC with sublinear opening time in the compiler in [GKO+23], would remove this quadratic complexity. At commitment time, the prover will spend $O_\lambda(n^{1+\epsilon})$ time (see preliminaries) to generate the polynomial commitment and will then store the appropriate auxiliary information; for each of the $\approx n$ queries to the RO the prover computes an evaluation opening and each can be performed in $O(\mathsf{polylog}(n))$ time with our construction in Section 4. Since the total proving time is $\approx n \cdot \mathsf{polylog}(n) \in O_\lambda(n^{1+\epsilon})$, we obtain as a result an approximately-linear time prover (as long as the original zkSNARK prover runs in quasilinear-time). As an implication, we have the first compiler for general SNARKs with such property. Notice that, if the zkSNARK has polylogarithmic proof size and verification time, these are also preserved.

Unfortunately, the above approach is flawed because our PC is not directly applicable to the the compiler in [GKO+23] which requires the PC to satisfy some "special" properties which are *not* all satisfied by our PC.

Solving the additional challenges. The first special property is a soft form of non-malleability: it informally states that an efficient adversary should not be able to produce two different evaluation proofs for the same input point z for an honestly generated polynomial commitment ("*unique proofs*"). Our construction $\mathsf{PC}_{\mathsf{VC}}$ does already satisfy this property (see Theorem 8).

The second required property is a form of hiding. In order to state it, we first recall the methodology followed in the compiler in [GKO+23], which can be thought of as an *encode&commit* approach

[24] Note that the evaluation points are required to be sampled from a space of super-polynomial size; therefore, precomputing the proofs is not possible.

where the prover computes $\mathsf{cm}_f \leftarrow \mathsf{PC.Com}(f_{\mathbf{w}})$ where $f_{\mathbf{w}} \leftarrow \mathsf{PES.Enc}(\mathbf{w})$. That is, it first encodes the witness (a vector) into a polynomial—using a "polynomial encoding scheme" PES—to which it then commits using a polynomial commitment PC. The encoding algorithm is randomized and its role is to *mask* the witness, so that the latter is still hidden to the verifier even after seeing several—approximately λ—polynomial evaluation proofs. If this is the case we say that the polynomial commitment PC is "*evaluation-hiding*" with respect to the encoding scheme PES. The reason this notion relies on an additional object, the encoding scheme, is that it allows it to coexist with the requirement over unique proofs. In fact, the latter can be satisfied, at least seemingly, only by a PC with *deterministic* commitment and proving algorithms. This rules out some intuitive-but-wrong approaches; for example, applying a ZK proof systems to achieve this hiding property is not an option.

Our construction in Section 4 does not directly satisfy evaluation-hiding for any of the polynomial encoding schemes in literature we are aware of (we give additional details in Appendix F). The reason is that the underlying approach of Section 4 might inherently have too much leakage.[25] As a consequence, we need to develop a method that can be applied to the compiler from [GKO+23] despite the limitations of our constructions.

Our approach: "bootstrapping" evaluation-hiding. Recall that our main goal is to be able to produce evaluation proofs obtaining, at the same time, three requirements: (i) sublinear opening time; (ii) evaluation-hiding w.r.t. *some* encoding scheme; (iii) unique proofs. We will temporarily ignore the last property and focus on the tension between the first two. Our solution will turn out to be a *hybrid* approach that, given:

- a polynomial commitment subPC satisfying (i);
- a polynomial commitment hidPC that is evaluation hiding w.r.t. an encoding scheme hidPES (i.e., that together satisfy (ii));

obtains a polynomial commitment scheme (and a companion encoding scheme) allowing us to satisfy both (i) and (ii) at the same time. Our approach is relying on subPC to do the heavy lifting in terms of opening proofs, but at the same time it is relying completely on $(\mathsf{hidPC}, \mathsf{hidPES})$ for evaluation-hiding *without having to pay in the total size of the witness* (and thus losing property (i)).

In a nutshell, what we do is this. For encoding, we first sample a (symmetric) encryption key sk and encrypt the witness obtaining a ciphertext $\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{w})$ (both sk and $\mathbf{ct}$, vectors of scalars); we then build two polynomials $f(X) := \sum_i X^{i-1}\mathsf{ct}_i$ and $g(X) := \mathsf{hidPES.Enc}(\mathsf{sk})$. In order to commit to this encoding, we generate $\mathsf{cm}_f \leftarrow \mathsf{subPC.Com}(f)$ and $\mathsf{cm}_g \leftarrow \mathsf{hidPC.Com}(g)$. At opening time, we produce two proofs, one for each commitment. (Note: in exposition above, for simplicity, we "produce in output" *two* commitments and *two* encodings, whereas the framework in [GKO+23] expects just one of each. This is easy to fix—as we show in our formal treatment in Fig. 2—by what is essentially a concatenation of the coefficients and the commitments). In the rest of this section we refer to the resulting "hybrid" polynomial commitment and encoding scheme respectively as $\mathsf{PC}^{\mathrm{hyb}}$ and $\mathsf{PES}^{\mathrm{hyb}}$.

As for security, we are able to achieve property (i) because the (hiding) scheme with "slow" openings is operating on a very small polynomial (of degree roughly λ, the size of the encoding of the

[25] Intuitively, this is because at the core of our construction, each opening provides *several* evaluations of various "sub-polynomials" related to the committed one. These correspond to a CRT-like encoding of the original polynomial and may be providing too much leakage, one none of the existing encoding schemes provides immediate tools to handle. See also Appendix F

secret key); we are able to achieve property (ii) because the ciphertexts reveal nothing about the witness (by semantic security) and the openings from cm_g reveal nothing about the secret key thanks to its evaluation-hiding property. It is straightforward to see that if both polynomial commitments have unique proofs so does the resulting one. We see our approach as a technique of independent interest since it allows to transform any polynomial commitment scheme into one satisfying evaluation hiding by "piggybacking" on an some other *already* evaluation-hiding scheme, using it only over a limited portion of the polynomial (e.g., linear in λ). This can be useful, as in our setting, where we want to leverage efficiency properties of a polynomial commitment scheme that is otherwise "too leaky". The fact that our transformation also preserves unique proofs makes it essentially a bootstrapping technique for the [GKO+23] compiler (see also Appendix F).

6.1 Putting it all together: instantiations and main result

In Figure 2 we formally describe the "hybrid" polynomial commitment $\mathsf{PC}^{\mathrm{hyb}}$ and the encoding scheme $\mathsf{PES}^{\mathrm{hyb}}$. It is based on the two building blocks hidPC and subPC as described above. For the final $\mathsf{PC}^{\mathrm{hyb}}$ to be secure, we require: The following two lemmas—which we prove in Appendix E.2—describe the requirements on hidPC and subPC to obtained the desired properties from $\mathsf{PES}^{\mathrm{hyb}}$. The notions of unique proofs and evaluation hiding are provided in the appendix (Definition 31 and Definition 33), but the reader can find their informal description in the technical overview.

Lemma 1. *If* hidPC *and* subPC *are polynomial commitments satisfying completeness, (weak) evaluation binding and the unique proof property (Definition 31 in the appendix) then* PC^{hyb} *(previous paragraphs; Fig. 2) satisfies the unique proof property.*

Lemma 2. *If* hidPC *satisfies evaluation hiding (Definition 33) with respect to* hidPES *then* PC^{hyb} *in Fig. 2 satisfies evaluation hiding with respect to* PES^{hyb}*.*

Instantiating subPC. We now show that our PC Construction 1 satisfies the unique proof property whenever the underlying vector commitment also has unique proofs (this notion is a straightforward counterpart of the one for polynomial commitments and is in Definition 35 in the appendix). Recall that unique proofs property is one of the necessary properties of hidPC and subPC as stated in Lemma 5. We refer the reader to Appendix E.3 for the formal proof.

Lemma 3. *If the underlying VC satisfies position binding*[26] *(Definition 25) and unique proofs (Definition 35) then the* $\mathsf{PC}_{\mathsf{VC}}$ *construction in Section 4 is a polynomial commitment with unique proofs (Definition 31).*

The lemma above, together with the fact that Merkle Trees have computationally unique proofs (this follows directly from collision-resistance) gives us a candidate instantiation for our hybrid construction.

Theorem 8. *If there exist CRHF then there exists a polynomial commitment* subPC *satisfying evaluation binding and unique proofs; for a polynomial of degree d, it features $O_\lambda(1)$-sized commitments, $O_\lambda(d^{1+\epsilon})$ commitment time, $O_\lambda(\mathsf{polylog}(d))$ opening time and evaluation proofs/verification of size/time $O_\lambda(\mathsf{polylog}(d))$.*

$\mathsf{PES}^{\mathrm{hyb}}.\mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell)$:

$\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\lambda)$; $\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{w})$
Let $f(X) := \mathsf{polyFromCoeffs}(\mathbf{ct})$
$g \leftarrow \mathsf{hidPES.Enc}(\mathsf{sk}, \kappa, \ell)$
Return $h(X) := f(X) + \mathcal{Z}_k(X) \cdot g(X)$ where $k := \deg(f)$.

$\mathsf{PES}^{\mathrm{hyb}}.\mathsf{Dec}(1^\lambda, h, n, \ell)$:

Parse h as $h(X) := f(X) + \mathcal{Z}_k(X) \cdot g(X)$
Let $\mathbf{ct}$ be the coefficients of f
$\mathsf{sk} \leftarrow \mathsf{hidPES.Dec}(g, \kappa, \ell)$
Return $\tilde{\mathbf{w}} \leftarrow \mathsf{SKE.Dec}(\mathsf{sk}, \mathbf{ct})$

$\mathsf{PC}^{\mathrm{hyb}}.\mathsf{Setup}(1^\lambda, d)$:

// k is known (and upper bounded by $\mathsf{poly}(\lambda)$)
$\mathsf{ck} \leftarrow \mathsf{hidPC.Setup}(1^\lambda, k)$
$\mathsf{ck}' \leftarrow \mathsf{subPC.Setup}(1^\lambda, d-k)$
Return $(\mathsf{ck}, \mathsf{ck}')$

$\mathsf{PC}^{\mathrm{hyb}}.\mathsf{Com}(\mathsf{pp}, h(X) \in \mathbb{F}_{\leq d}[X])$:

Parse h as $h(X) := f(X) + \mathcal{Z}_k(X) \cdot g(X)$
$(\mathsf{cm}_g, \mathsf{aux}_g) \leftarrow \mathsf{hidPC.Com}(\mathsf{ck}, g)$
$(\mathsf{cm}_f, \mathsf{aux}_f) \leftarrow \mathsf{subPC.Com}(\mathsf{ck}', f)$
Return $(\mathsf{cm}_f, \mathsf{cm}_g), (\mathsf{aux}_f, \mathsf{aux}_g)$

$\mathsf{PC}^{\mathrm{hyb}}.\mathsf{Open}(\mathsf{pp}, (\mathsf{aux}_f, \mathsf{aux}_g), t, y)$:

Parse h as $h(X) := f(X) + \mathcal{Z}_k(X) \cdot g(X)$ using $(\mathsf{aux}_f, \mathsf{aux}_g)$
Let $y_f := f(t)$, $y_g := g(t)$
$\pi_g \leftarrow \mathsf{hidPC.Open}(\mathsf{ck}, \mathsf{aux}_g, t, y_g)$
$\pi_f \leftarrow \mathsf{subPC.Open}(\mathsf{ck}', \mathsf{aux}_f, t, y_f)$
Return $\pi := (\pi_f, \pi_g, y_g)$

$\mathsf{PC}^{\mathrm{hyb}}.\mathsf{Ver}(\mathsf{pp}, (\mathsf{cm}_f, \mathsf{cm}_g), (\pi_f, \pi_g, y_g), t, y)$:

Compute $y_f \leftarrow y - \mathcal{Z}_k(t) \cdot y_g$
Accept iff $\mathsf{hidPC.Ver}(\mathsf{ck}, \mathsf{cm}_g, \pi_g, t, y_g)$ and
$\mathsf{subPC.Ver}(\mathsf{ck}', \mathsf{cm}_f, \pi_f, t, y_f)$ both return 1.

Fig. 2. Our "hybrid" encoding $\mathsf{PES}^{\mathrm{hyb}}$ and related PC scheme $\mathsf{PC}^{\mathrm{hyb}}$ for the compiler in Appendix D.5. A PES consists of a pair $(\mathsf{Enc}, \mathsf{Dec})$: $\mathsf{Enc}(1^\lambda, w, n, \ell)$ encodes a vector $w \in \mathbb{F}^n$ into a polynomial by appending ℓ random masking coefficients, and Dec recovers w. Above $\mathcal{Z}_k$ is the polynomial vanishing in some known domain of size k. The key sk is parsed as a vector of κ scalars, where κ is a public element depending on the encryption scheme and the size of its key. We denote by $\mathsf{polyFromCoeffs}(\mathbf{v})$ the polynomial $p(X) := \sum_{i=0}^{n-1} v_{i+1} X^i$ whose coefficients are the elements of $\mathbf{v}$.

Instantiating hidPC and hidPES. We will use Theorem 8 in order to instantiate subPC in Fig. 2. All we are left with is to instantiate hidPC and hidPES, for which we will use some of the building blocks from the work in [GKO+23]. We will use KZG [KZG10] as a PC. It features constant-size commitments, proofs and verification; it is secure under the standard Strong Diffie-Hellman (SDH) assumption [BB04].

KZG was shown in [GKO+23] to be evaluation hiding with respect an encoding $\mathsf{PES}_{\mathrm{coeff}}$, which we pick as our choice for hidPES. This simple encoding essentially, transforms the input secret vector (in our case case the secret key sk) by parsing it as the vector of coefficients of a polynomial and concatenating $\approx \lambda$ random freshly sampled coefficients. Its resulting output polynomial is therefore of size $O(\mathsf{poly}(\lambda))$. Full formal treatment of these building blocks is in Appendix D.

Remark 2 (Efficiency of PC^{hyb} and PES^{hyb}). The resulting efficiency of our hybrid construction (in all metrics) stems from the sum of the efficiency profiles of the its building blocks. Recall that at encoding time we perform an encryption of the witness and we produce an encoding (through hidPES) of sk. The polynomial commitment algorithms invoke commitment/opening/verification of each of the two underlying schemes. For our specific instantiations, from the above and Theorem 8, we can conclude that: *i)* encryption dominates the costs in our hybrid encoding scheme, and thus the latter runs in linear time, i.e. $O_\lambda(|\mathbf{w}|)$; the costs in the resulting polynomial commitment are dominated by those of subPC.

In the following theorem we use the fact that the SDH assumption implies DLOG and the existence of CRHF (see Remark 3 in Appendix D). Its proof is immediate from the observations and results in this section.

[26] Although it is not necessary for our specific Merkle Tree instantiation, this hypothesis could be weakened by just assuming weak position binding (where the commitments is generated honestly).

Theorem 9. *Under the SDH assumption ([BB04]) there exist a polynomial commitment* PC^{hyb} *with the properties and efficiency in Theorem 8 and an encoding scheme* PES^{hyb} *with linear-time encoding/decoding such that* PC^{hyb} *is evaluation hiding with respect to* PES^{hyb}*.*

Our Main Result on Succinct UC-NIZKs. Before stating our own result, we first recall one regarding the efficiency of the compiler in [GKO+23]. The latter, on input a PC (with some extra properties) and a simulation-extractable NIZK, outputs a compiled protocol Π_{GKOPTT} implementing a UC-NIZK (we refer the reader to Appendix D.5 for a full description).

Theorem 10 (Implicit in [GKO+23, BCC+24]). *Consider the following primitives:*

- NIZK*: a simulation-extractable NIZK (Definition 29), which, for a witness of size* n*, has: proof size* $O_\lambda(\ell_{\mathsf{NIZK}}(n))$*, quasi-linear proving time (i.e.,* $\mathsf{T}_{\mathsf{NIZK.Prove}} = \widetilde{O}_\lambda(n)$*),*[27]*, and verification time* $O_\lambda(\ell_{\mathsf{NIZK}}(n))$*.*
- PC*: a polynomial commitment having weak evaluation binding (Definition 20), unique proofs (Definition 31), commitments and evaluation proofs both of size* $O_\lambda(\ell_{\mathsf{PC}}(d))$*, commitment and proving time* $O_\lambda(\mathsf{T}_{\mathsf{PC.Com}}(d))$ *and* $O_\lambda(\mathsf{T}_{\mathsf{PC.Open}}(d))$ *respectively, and verification time* $O_\lambda(\ell_{\mathsf{PC}}(d))$*.*
- PES*: an encoding scheme (Definition 32) s.t.* PC *is evaluation hiding (Definition 33) with respect to* PES *and with linear encoding time.*

Then, there exists a UC-NIZK (Definition 30) with proof size $O_\lambda\big(\ell_{\mathsf{NIZK}}(n) + \ell_{\mathsf{PC}}(n)\big)$*, proving time* $\widetilde{O}_\lambda\big(\mathsf{T}_{\mathsf{PC.Com}}(n) + n\mathsf{T}_{\mathsf{PC.Open}}(n)\big)$*, and verification time* $O_\lambda(\ell_{\mathsf{NIZK}}(n) + \ell_{\mathsf{PC}}(n))$*.*

Our main technical result of this section is a general theorem showing one can instantiate the compiler in [GKO+23] from any (reasonably efficient) simulation-extractable NIZK and obtain a UC-secure NIZK with approximately-linear time and polylog proof size and verification. The theorem follows by plugging in Theorem 10 our "hybrid" $\mathsf{PC}^{\mathrm{hyb}}$ and $\mathsf{PC}^{\mathrm{hyb}}$ instantiated as described in this section.

Theorem 11. *Under the Strong Diffie-Hellman assumption ([BB04]), if there exists a simulation-extractable NIZK scheme* NIZK *with quasilinear proving-time and proof size and verification time* $O_\lambda(\mathsf{polylog}(n))$ *(for a witness of size* n*), then there exists a UC-NIZK (Definition 30 in Appendix D) with* $O_\lambda(\mathsf{polylog}(n))$ *proof size and verification time and approximately-linear proving time.*

When instantiated with the pairing-based simulation-extractable scheme from Groth and Maller [GM17]—whose security relies on the usual flavor of knowledge-of-exponent assumptions (see Appendix D.4)—we obtain the following result.

Corollary 3. *Under the SDH assumption ([BB04]), the Computational Polynomial Assumption and the XPKE Assumption ([GM17]) there exists a UC-NIZK with* $O_\lambda(\mathsf{polylog}(n))$ *proof size and verification and approximately-linear proving time.*

Acknowledgments. The authors thank Dario Catalano and Emanuele Giunta for collaborating during an early stage of this project, and the anonymous CRYPTO 2025 reviewers for hinting the possibility of using preprocessing to construct index-efficient SNARKs for R1CS. The authors also

[27] Assuming quasi-linearity merely allows us to simplify notation; the core intuition about the resulting asymptotics does not rely on any specific profile of the proving time (we make similar simplifying assumptions, w.l.o.g., for verification time).

thank Nitin Singh for reporting a bug on the correctness of a previous version of our index-efficient PIOP for linear checks. Matteo Campanelli would like to thank Justin Thaler and Mathias Hall-Andersen for their thoughtful comments around polynomial commitments based on the techniques in Kedlaya-Umans.

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects ESPADA (PID2022-142290OB-I00), and CEX2024-001471-M funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU / PRTR. This work was also supported by the Estonian Research Council grant PRG2531.

References

ABC+24. Giuseppe Ateniese, Foteini Baldimtsi, Matteo Campanelli, Danilo Francati, and Ioanna Karantaidou. Advancing scalability in decentralized storage: a novel approach to proof-of-replication via polynomial evaluation. In *Annual International Cryptology Conference*, pages 3–39. Springer, 2024.

AGRS23. Behzad Abdolmaleki, Noemi Glaeser, Sebastian Ramacher, and Daniel Slamanig. Universally composable NIZKs: Circuit-succinct, non-malleable and CRS-updatable. Cryptology ePrint Archive, Report 2023/097, 2023.

ARS20. Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1987–2005, Virtual Event, USA, November 9–13, 2020. ACM Press.

BB04. Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part II*, volume 14439 of *Lecture Notes in Computer Science*, pages 77–110, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

BCC+24. Christian Badertscher, Matteo Campanelli, Michele Ciampi, Luigi Russo, and Luisa Siniscalchi. Universally composable SNARKs with transparent setup without programmable random oracle. Cryptology ePrint Archive, Paper 2024/1549, 2024.

BCFL23. David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Chainable functional commitments for unbounded-depth circuits. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part III*, volume 14371 of *Lecture Notes in Computer Science*, pages 363–393, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, Beijing, China, October 31 – November 3, 2016. Springer Berlin Heidelberg, Germany.

BFR+13. Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 341–357, New York, NY, USA, 2013. Association for Computing Machinery.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

BS21. Karim Baghery and Mahdi Sedaghat. Tiramisu: Black-box simulation extractable NIZKs in the updatable CRS model. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21: 20th International Conference on Cryptology and Network Security*, volume 13099 of *Lecture Notes in Computer Science*, pages 531–551, Vienna, Austria, December 13–15, 2021. Springer, Cham, Switzerland.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

CDG+18. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 280–312, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

CDPW07. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer Berlin Heidelberg, Germany.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72, Nara, Japan, February 26 – March 1, 2013. Springer Berlin Heidelberg, Germany.

CF24. Alessandro Chiesa and Giacomo Fenzi. zkSNARKs in the ROM with unconditional UC-security. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part I*, volume 15364 of *Lecture Notes in Computer Science*, pages 67–89, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

CFF+24. Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 14602 of *Lecture Notes in Computer Science*, pages 337–369, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland.

CFG+20. Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 3–35, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

CFG25. Matteo Campanelli, Dario Fiore, and Rosario Gennaro. Natively compatible super-efficient lookup arguments and how to apply them. *Journal of Cryptology*, 38(1):14, 2025.

CFT22. Dario Catalano, Dario Fiore, and Ida Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 159–188, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

CJS14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 597–608, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

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

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79, Kolkata, India, December 9–12, 2012. Springer Berlin Heidelberg, Germany.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 698–728, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

GKO+23. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 315–346, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

GOP+22. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-Shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 397–426, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

GOS12. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

KU08. Kiran S. Kedlaya and Christopher Umans. Fast modular composition in any characteristic. In *49th Annual Symposium on Foundations of Computer Science*, pages 146–155, Philadelphia, PA, USA, October 25–28, 2008. IEEE Computer Society Press.

KU11. Kiran S Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.

KZM+15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015.

LM19. Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 530–560, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

LMW22. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. Cryptology ePrint Archive, Report 2022/1703, 2022.

LMW23. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 595–608, Orlando, FL, USA, June 20–23, 2023. ACM Press.

Mer88. Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer Berlin Heidelberg, Germany.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume

12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

STW24. Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

TAB+20. Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 45–64, Amalfi, Italy, September 14–16, 2020. Springer, Cham, Switzerland.

Tha13. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

ZBK+22. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 3121–3134, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

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

A Further Preliminaries

A.1 Definitions of Polynomial Commitments

Definition 17 (Polynomial Commitment [KZG10]). *A polynomial commitment scheme* PC *for multivariate polynomials over a ring R is defined by the following algorithms:*

$\mathsf{Setup}(1^\lambda, m, d) \to \mathsf{ck}$**:** *On input the security parameter λ, the number of variables m, and a bound d for the degree of each variable, the setup algorithm outputs a public commitment key* ck*, which implicitly defines the message space of the commitment scheme $\mathcal{P}_{m,d} = \{f \in R[X_1, \ldots, X_m] \mid \mathsf{deg}_i(f) < d, \forall i \in [m]\}$.*

$\mathsf{Com}(\mathsf{ck}, f; r) \to (\mathsf{cm}, \mathsf{aux})$**:** *On input the commitment key* ck*, a polynomial $f \in \mathcal{P}_{m,d}$, and (possibly) a randomness r, the commit algorithm outputs a commitment* cm *and an auxiliary information* aux*.*

$\mathsf{Open}(\mathsf{ck}, x, \mathsf{aux}) \to \pi_x$**:** *On input the commitment key* ck*, a point $x \in R^m$, and auxiliary information* aux*, the open algorithm outputs a proof π_x.*

$\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) \to b$**:** *On input the commitment key* ck*, the commitment* cm*, $x \in R^m, y \in R$, and a proof π_x, the verification algorithm outputs $b \in \{0, 1\}$.*

Definition 18 (Correctness of PC). *A polynomial commitment scheme* PC *as defined above is correct if $\forall \lambda, m, d \in \mathbb{N}$, for any polynomial $f \in \mathcal{P}_{m,d}$, for any point $x \in R^m$, and for any randomness $r \in \{0,1\}^*$, the following holds:*

$$\Pr\left[\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, f(x), \pi_x) = 1 \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, m, d) \\ (\mathsf{cm}, \mathsf{aux}) \leftarrow \mathsf{Com}(\mathsf{ck}, f; r) \\ \pi_x \leftarrow \mathsf{Open}(\mathsf{ck}, x, \mathsf{aux}) \end{array}\right] = 1.$$

Definition 19 (Binding of PC). *A polynomial commitment scheme* PC *satisfies commitment binding if for any* PPT *adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{Com}(\mathsf{ck}, f; r) = \mathsf{Com}(\mathsf{ck}, f'; r') \\ \wedge\, f \neq f' \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, m, d) \\ (f, r, f', r') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right] \in \mathsf{negl}(\lambda).$$

Definition 20 (Weak Evaluation Binding of PC). *A polynomial commitment scheme* PC *satisfies weak evaluation binding if for any* PPT *adversary* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(\mathsf{ck},\mathsf{cm},x,y,\pi_x)=1\\ \wedge\, y\neq f(x)\end{array}\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,m,d)\\ (f,r,x,y,\pi_x)\leftarrow\mathcal{A}(\mathsf{ck})\\ (\mathsf{cm},\mathsf{aux})\leftarrow\mathsf{Com}(\mathsf{ck},f;r)\end{array}\right]\in\mathsf{negl}(\lambda).$$

Definition 21 (Strong Evaluation Binding of PC). *A polynomial commitment scheme* PC *satisfies strong evaluation binding if for any* PPT *adversary* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\begin{array}{l}\forall i\in[N],\ \mathsf{Ver}(\mathsf{ck},\mathsf{cm},x_i,y_i,\pi_i)=1\\ \wedge\,\nexists f\in\mathcal{P}_{m,d},\ \forall i\in[N],\ f(x_i)=y_i\end{array}\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,m,d)\\ (\mathsf{cm},\{x_i,y_i,\pi_i\}_{i\in[N]})\leftarrow\mathcal{A}(\mathsf{ck})\end{array}\right]\in\mathsf{negl}(\lambda).^{28}$$

Definition 22 (Non-black-box Knowledge Soundness of PC). *A polynomial commitment scheme* PC *satisfies non-black-box knowledge soundness if for any* PPT *adversary* $\mathcal{A}$ *there exists a polynomial-time extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that:*

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(\mathsf{ck},\mathsf{cm},x,y,\pi_x)=1\ \wedge\\ ((\mathsf{cm},\cdot)\neq\mathsf{Com}(\mathsf{ck},f;r)\vee y\neq f(x))\end{array}\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,m,d)\\ (\mathsf{cm},x,y,\pi_x)\leftarrow\mathcal{A}(\mathsf{ck})\\ (f,r)\leftarrow\mathsf{Ext}_{\mathcal{A}}(\mathsf{ck})\end{array}\right]\in\mathsf{negl}(\lambda).$$

Sometimes we will refer to black-box knowledge soundness which is essentially identical to the above definition except that the extractor Ext is universal.

A.2 Definitions of Vector Commitments

Definition 23 (Vector Commitments [CF13]). *A vector commitment scheme* VC *for a message space* $\mathcal{M}$ *is defined by the following algorithms:*

$\mathsf{Setup}(1^\lambda,n)\to\mathsf{ck}$**:** *On input the security parameter* λ *and a bound* n *for the length of the committed vectors, the setup algorithm outputs a public commitment key* ck*.*

$\mathsf{Com}(\mathsf{ck},\mathbf{v};r)\to(\mathsf{cm},\mathsf{aux})$**:** *On input the commitment key* ck*, a vector* $\mathbf{v}\in\mathcal{M}^n$*, and (possibly) a randomness* r*, the commit algorithm outputs a commitment* cm *and an auxiliary information* aux*.*

$\mathsf{Open}(\mathsf{ck},i,\mathsf{aux})\to\pi_i$**:** *On input the commitment key* ck*, an index* $i\in[n]$*, and auxiliary information* aux*, the open algorithm outputs a proof* π_i*.*

$\mathsf{Ver}(\mathsf{ck},\mathsf{cm},i,y,\pi_i)\to b$**:** *On input the commitment key* ck*, the commitment* cm*, index* $i\in[n]$*, value* $y\in\mathcal{M}$*, and a proof* π_i*, the verification algorithm outputs a bit* $b\in\{0,1\}$*.*

Definition 24 (Correctness of VC). *A vector commitment scheme* VC *as defined above is correct if* $\forall\lambda,n\in\mathbb{N}$*, for any vector* $\mathbf{v}\in\mathcal{M}^n$*, for any randomness* $r\in\{0,1\}^*$*, and* $\forall i\in[n]$*, the following holds:*

$$\Pr\left[\mathsf{Ver}(\mathsf{ck},\mathsf{cm},i,v_i,\pi_i)=1\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,n)\\ (\mathsf{cm},\mathsf{aux})\leftarrow\mathsf{Com}(\mathsf{ck},\mathbf{v};r)\\ \pi_i\leftarrow\mathsf{Open}(\mathsf{ck},i,\mathsf{aux})\end{array}\right]=1.$$

[28] The condition $\nexists f\in\mathcal{P}_{m,d},\ \forall i\in[N],\ f(x_i)=y_i$ is equivalent to saying that there is no solution $\mathbf{z}$ to the linear system $\mathbf{M}\cdot\mathbf{z}=\mathbf{y}$, where $\mathbf{M}$ is the $(N\times d^m)$-size matrix whose row i contains all the monomials evaluated in the x_i's, $\mathbf{y}$ is the N-size vector consisting of the outputs y_i, and $\mathbf{z}$ is the unknown for the d^m-size vector of coefficients of the polynomial f.

Definition 25 (Position Binding). *A vector commitment scheme* VC *satisfies position binding if for any* PPT *adversary* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\begin{array}{c}\mathsf{Ver}(\mathsf{ck},\mathsf{cm},i,y,\pi_i)=1\\ \wedge\ \mathsf{Ver}(\mathsf{ck},\mathsf{cm},i,y',\pi_i')=1\\ \wedge\ y\neq y'\end{array}\middle|\begin{array}{l}\mathsf{ck}\leftarrow\mathsf{Setup}(1^\lambda,n)\\ (\mathsf{cm},i,y,\pi_i,y',\pi_i')\leftarrow\mathcal{A}(\mathsf{ck})\end{array}\right]\in\mathsf{negl}(\lambda).$$

Instantiations of VC with fast openings. We recall a few instantiations of VCs with fast opening:

- Merkle trees [Mer88] are position binding based on collision resistant hash functions, achieve $O(\lambda\log n)$ succinctness, and have fast opening with $\mathsf{T}_{\mathsf{Com}}(\lambda,n)=\mathsf{poly}(\lambda)\cdot n$ and $\mathsf{T}_{\mathsf{Open}}(\lambda,n)=O(\lambda\log n)$. The fast opening follows by storing all the nodes of the tree which can be accessed in constant time in the RAM model.
- VCs with quasilinear algorithms to compute openings for all positions. There are a variety of schemes where succinctness and $\mathsf{T}_{\mathsf{Open}}(\lambda,n)$ are a fixed $\mathsf{poly}(\lambda)$, and where $\mathsf{T}_{\mathsf{Com}}(\lambda,n)=\mathsf{poly}(\lambda)\cdot n\log n$ (e.g., for the KZG-based VC from pairings [KZG10]) or $\mathsf{T}_{\mathsf{Com}}(\lambda,n)=\mathsf{poly}(\lambda)\cdot n\log^2 n$ (e.g., for schemes based on hidden-order groups like [CF13, BBF19, CFG+20]). In fact, Campanelli et al. [CFG+20] show that fast opening follows from any VC with incremental (dis)aggregation.

B Lack of Strong Evaluation Binding of $\mathsf{PC}_{\mathsf{VC}}$

Theorem 4 states the evaluation binding property of $\mathsf{PC}_{\mathsf{VC}}$, which is the standard security notion of polynomial commitments. One may wonder if our construction can satisfy one of the stronger properties such as strong evaluation binding (Definition 21) or knowledge soundness (Definition 22). In the following theorem we show that, when the PP scheme of Construction 1 is instantiated with the KU-based scheme [KU08, KU11], then the resulting $\mathsf{PC}_{\mathsf{VC}}$ is not strongly evaluation binding.

Theorem 12. *If the PP scheme of Construction 1 is instantiated with the PP scheme from Theorem 2, then the resulting* $\mathsf{PC}_{\mathsf{VC}}$ *does not satisfy strong evaluation binding.*

Proof. We first describe the general strategy to construct attacks, and then show a concrete instantiation.

The core observation at the base of the proof is that the space of the data structures is significantly larger than the space of the polynomials we preprocess, whereas, to answer a specific point query, we read only a small subset (k values) of the data structure DS. To win the strong evaluation binding experiment, it is sufficient to satisfy the following two conditions:

1. provide d^m valid proofs $\{\pi_{x_i}\}_{i\in[d^m]}$ and evaluations $\{y_i\}_{i\in[d^m]}$ w.r.t. a set of points $\{x_i\}_{i\in[d^m]}$ such that the evaluations uniquely determine a polynomial $f\in\mathcal{P}_{m,d}$ with $f(x_i)=y_i$ for all $i\in[d^m]$. For simplicity we will set $f=0$, i.e. the set of points $\{x_i\}_{i\in[d^m]}$ should be such that no nonzero polynomial from $\mathcal{P}_{m,d}$ can evaluate to 0 in all points. Accordingly, we will set $y_i=0$ for all $i\in[d^m]$.
2. provide an additional verifying proof $\tilde{\pi}_x$ for a new point $\tilde{x}$ whose claimed evaluation $\tilde{y}$ is different from 0.

A sufficient condition to satisfy condition 1 above is that the set of points $\{x_i\}_{i\in[d^m]}$ is of the form S^m, with $S \subseteq \mathbb{Z}_q$ being of size d (indeed in these conditions we have a Lagrange basis of $\mathcal{P}_{m,d}$ with respect to these set of points as shown in Section 5.2). When using the $\mathsf{PC}_{\mathsf{VC}}$ of Theorem 12, condition 2 can be achieved by selecting a point $\tilde{x}$ such that running $\mathsf{PP.Lookup}(\mathsf{pp}, \tilde{x})$ produces indices $\tilde{i}_1, \ldots, \tilde{i}_k$ some of which are never queried by $\mathsf{PP.Lookup}(\mathsf{pp}, x_i)$ for all $x_i \in S^m$. Without loss of generality let $\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_\ell$ be those previously unqueried indices. Given $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(pp, 0)$ we need to be able to change some of $v_{\tilde{i}_1}, \ldots, v_{\tilde{i}_\ell}$ in DS (without modifying the remaining $v_{\tilde{i}_{\ell+1}}, \ldots, v_{\tilde{i}_k}$) in such a way that $\mathsf{PP.Reconstruct}(\mathsf{pp}, \tilde{x}, v_{\tilde{i}_1}, \ldots, v_{\tilde{i}_k})$ outputs a value y different from 0.

Under the conditions above we have the following attack:

- Run $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(pp, 0)$. Modify the values $v_{\tilde{i}_1}, \ldots, v_{\tilde{i}_\ell}$ in DS as above and commit to the resulting modified DS with $\mathsf{PC}_{\mathsf{VC}}.\mathsf{Com}$.
- Compute the proofs π_{x_i} for all x_i and $\tilde{\pi}_x$ for $\tilde{x}$ as in $\mathsf{PP.Open}$.

Clearly the resulting openings will be correct, the proofs π_{x_i} will open to $f = 0$ and the proof $\tilde{x}$ will open to a value y different from 0. Because 0 is the only polynomial consistent with the evaluations on all x_i being 0, there cannot be any polynomial f in $\mathcal{P}_{m,d}$ simultaneously consistent with $f(x_i) = 0$ for $i \in [d^m]$ and $f(\tilde{x}) = y$, breaking strong binding.

We show an explicit choice of $\{x_i\}_{i\in[d^m]}$ and $\tilde{x}$ for large enough q. For this, we recall that we are using the 2-level recursion version of the Kedlaya-Umans preprocessing algorithm presented in [LMW22, Theorem A.4]). The data structure output by $\mathsf{PP.PreProcess}(pp, f)$ consists of subtables $\mathcal{T}_{p,p'}$ indexed by pairs (p, p') running over a certain subset of pairs of primes where in particular it always holds $p' < p$. The primes p run over the set $\mathcal{P}$ of all primes less than some bound that guarantees that $\prod_{p\in\mathcal{P}} p > M$ where M is the maximum possible value of any $f(x)$ (for all $f \in \mathcal{P}_{m,d}$ and $x \in \mathbb{Z}_q^m$) when both f and x are lifted to the integers (i.e. computed without reduction mod q)[29]. Concretely, $M = d^m q^{m(d-1)+1}$. While in [KU08, KU11, LMW22] the set $\mathcal{P}$ is taken to be all primes up to $16 \log M$ so that it works uniformly for any choice of d, q, m, here we are going to assume that we use instead the minimal set such that $\prod_{p\in\mathcal{P}} p > M$ (i.e., if $p_{\max}$ denotes the largest prime in the set, then $\prod_{p\in\mathcal{P}\setminus\{p_{\max}\}} < M$) since this is enough to apply the algorithm and simplifies the exposition. Of course, in particular we have $p_{\max} \leq 16 \log M$.

Each sub-table $\mathcal{T}_{p,p'}$ of the data structure contains the evaluations of certain polynomial $f_{p,p'}$ on all $a \in \mathbb{Z}_{p'}^m$; the polynomial $f_{p,p'}$ is obtained by first lifting f to $\mathbb{Z}[X_1, \ldots, X_m]$, then reducing modulo p and then repeating this operation on the resulting polynomial and p'. Note that, in particular for $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(pp, 0)$, all entries at all tables $\mathcal{T}_{p,p'}$ are 0. When evaluating f on any $x \in \mathbb{Z}_q^m$, only one index is queried for each $\mathcal{T}_{p,p'}$, namely the one containing $f_{p,p'}(a)$ such that a is obtained from x by first lifting to $\mathbb{Z}^m$, then reducing modulo p and repeating the operation again for p' (we abbreviate this by writing $a = x \bmod p \bmod p'$).

Let us partition $\mathcal{P}$ as $\mathcal{P} = \mathcal{P}_{\text{small}} \cup \mathcal{P}_{\text{large}}$ such that: i) all primes in $\mathcal{P}_{\text{small}}$ are smaller than those in $\mathcal{P}_{\text{large}}$; ii) $(\prod_{p\in\mathcal{P}_{\text{small}}} p) > p_{\max}$; and iii) $\mathcal{P}_{\text{small}}$ is minimal satisfying conditions i) and ii). Note $\mathcal{P}_{\text{large}}$ is non-empty: indeed $\mathcal{P}_{\text{small}} \subseteq \mathcal{P} \setminus \{p_{\max}\}$ because the product of all primes in $\mathcal{P} \setminus \{p_{\max}\}$ is always larger than the next prime $p_{\max}$, as otherwise we would reach a contradiction when considering the primality of $(\prod_{p\in\mathcal{P}\setminus\{p_{\max}\}} p) - 1$. [30] We call $P = \prod_{p\in\mathcal{P}_{\text{small}}} p$ and $Q = \prod_{p\in\mathcal{P}_{\text{large}}} p$. From the above we have $P > p_{\max}$ but also $Q = PQ/P < PQ/p_{\max} = \prod_{p\in\mathcal{P}\setminus\{p_{\max}\}} < M$.

[29] This is because the algorithm reconstruct $f(x)$ over the integers, using the Chinese Remainder Theorem
[30] Here we are assuming $\mathcal{P} \setminus \{p_{\max}\}$ contains at least 2 primes

Our attack now applies when $q > d(p_{\max})^2$. This holds when q is large enough with respect to d and m, because we know $d(p_{\max})^2 \leq 16^2 d(\log M)^2$, but $\log M = O(\log q)$. Under these conditions, we have that $q > dP$. Indeed, if $P > q/d > (p_{\max})^2$ then we would contradict our choice that our partition is such that $\mathcal{P}_{\text{small}}$ is minimal: we can remove the largest prime from $\mathcal{P}_{\text{small}}$ and we would still have $P > p_{\max}$.

From $q > dP$, we conclude there are at least d values s in $\mathbb{Z}_q$ such that when lifted to $\mathbb{Z}$, we have $s \bmod P = 1$. Let S be the set of these d values, let $\{x_i\}_{i\in[d^m]} = S^m$ (which satisfies condition 1 as explained above) and $\tilde{x} = (0, 0, \ldots, 0)$. Finally we will choose the evaluation $\tilde{y}$ to be $Q \bmod q$, which is different than 0 because Q is a product of primes smaller than q.

By construction, for every x_i, we have the following: at every $\mathcal{T}_{p,p'}$ with $p \in \mathcal{P}_{\text{small}}$, $\mathsf{PP.Lookup}(\mathsf{pp}, x_i)$ queries the index containing $f_{p,p'}(1, 1, \ldots, 1)$ because all the coordinates of every x_i are 1 modulo $P = \prod_{p\in\mathcal{P}_{\text{small}}} p$.

On the other hand, $\mathsf{PP.Lookup}(\mathsf{pp}, \tilde{x})$ queries the index of every table $\mathcal{T}_{p,p'}$ containing $f_{p,p'}(0, 0, \ldots, 0)$. In particular the indices queried by $\mathsf{PP.Lookup}(\mathsf{pp}, \tilde{x})$ at every table $\mathcal{T}_{p,p'}$ with p in $\mathcal{P}_{\text{small}}$ have not been queried by any $\mathsf{PP.Lookup}(\mathsf{pp}, x_i)$.

We can therefore change the entries v_i containing $f_{p,p'}(0, 0, \ldots, 0)$ at tables $\mathcal{T}_{p,p'}$ for every prime p in $\mathcal{P}_{\text{small}}$ to be instead $v'_i = Q \bmod p \bmod p'$. On the other hand, for p in $\mathcal{P}_{\text{large}}$ we have $Q = 0 \bmod p$, so the unmodified 0 values in tables $\mathcal{T}_{p,p'}$ are already consistent with the claimed evaluation $f(\tilde{x}) = Q \bmod q$.

In summary, after doing these changes and calling $v'_{i_1}, \ldots, v'_{i_k}$ the modified values at the queried indices for $\tilde{x}$ we have $\mathsf{PP.Reconstruct}(pp, \tilde{x}, v'_{i_1}, \ldots, v'_{i_k}) = Q \bmod q$ by correctness of Kedlaya-Umans (where we are also using that $Q < M$), while $\mathsf{PP.Reconstruct}$ outputs 0 when applied to x_i and the corresponding unmodified entry values, which are all 0. Therefore the attack above will work.

C Knowledge Sound (SNARK-based) Polynomial Commitment with Fast Opening

The following construction make use of a SNARK for the (non-indexed) relation defined in Equation (1). Despite our preliminaries focus on indexed relations, the syntax and properties of non-indexed relations can be easily derived from Definition 6.

Construction 4 *Let* PP *be a PP scheme (Definition 10) and* VC *be a vector commitment scheme (Definition 23) for a message space* $\{0,1\}^*$. *Also, let* SNARK *be a SNARK for the relation* $\mathcal{R}_{\mathsf{VC},\mathsf{PP}}$ *(Equation (1)) parametrized by a* VC *and* PP *scheme (e.g., as the ones of Construction 1). We build a PC scheme* $\mathsf{PC}_{\mathsf{SNARK}}$ *for multivariate polynomials over a ring* R *as follows:*

$\mathsf{Setup}(1^\lambda, m, d) \to \mathsf{ck}$**:** *On input the security parameter* λ*, the number of variables* m*, and a bound* d *for the degree of each variable, the setup algorithm computes* $\mathsf{pp} \leftarrow \mathsf{PP.Params}(R, m, d)$, $\mathsf{ck}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, S)$*, and* $\mathsf{srs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda, \mathcal{R}_{\mathsf{VC},\mathsf{PP}})$*, where* S *is the upper bound on the size of the data structure included in* pp *(see Definition 10). Finally, it outputs* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs})$.

$\mathsf{Com}(\mathsf{ck}, f; r) \to (\mathsf{cm}, \mathsf{aux})$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs})$*, a polynomial* $f \in \mathcal{P}_{m,d}$*, and a randomness* r*, the commit algorithm computes* $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(\mathsf{pp}, f)$ *and* $(\mathsf{cm}_{\mathsf{DS}}, \mathsf{aux}_{\mathsf{DS}}) \leftarrow \mathsf{VC.Com}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{DS}; r)$ *(recall that* DS *is a vector, i.e.,* $\mathsf{DS} = (v_1, \ldots, v_S)$*). Finally, it outputs* $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$ *and* $\mathsf{aux} = (\mathsf{aux}_{\mathsf{DS}}, \mathsf{DS}, \pi_{\mathsf{DS}})$ *where* $\pi_{\mathsf{DS}} \leftarrow \mathsf{SNARK.Prove}(\mathsf{srs}, (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}), (f, r))$.

$\mathsf{Open}(\mathsf{ck}, x, \mathsf{aux}) \to \pi_x$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs})$*, a point* $x \in R^m$*, and auxiliary information* $\mathsf{aux} = (\mathsf{aux}_{\mathsf{DS}}, \mathsf{DS}, \pi_{\mathsf{DS}})$ *where* $\mathsf{DS} = (v_1, \ldots, v_S)$*, the open algorithm retrieves the indexes* $(i_1, ..., i_k) \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, x)$ *and compute an opening proof* $\pi_j \leftarrow \mathsf{VC.Open}(\mathsf{ck}_{\mathsf{VC}}, i_j, \mathsf{aux}_{\mathsf{DS}})$ *for every* $j \in [k]$*. Finally, it outputs* $\pi_x = (\pi_{\mathsf{DS}}, v_j, \pi_j)_{j\in[k]}$.

$\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) \to b$**:** *On input the commitment key* $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs})$*, the commitment* $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$*, a point* $x \in R^m$*, an evaluation* $y \in R$*, and a proof* $\pi_x = (\pi_{\mathsf{DS}}, v_j, \pi_j)_{j\in[k]}$*, the verification algorithm outputs a decision bit* $b \in \{0, 1\}$ *equals to* $\bigwedge_{j\in[k]} \mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v_j, \pi_j) \stackrel{?}{=} 1 \ \wedge \ y \stackrel{?}{=} \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v_1, \ldots, v_k) \ \wedge \ \mathsf{SNARK.Ver}(\mathsf{srs}, (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}), \pi) \stackrel{?}{=} 1$.

Below, we state the properties and security of our $\mathsf{PC}_{\mathsf{SNARK}}$ of Construction 4. Observe that the theorem below states that $\mathsf{PC}_{\mathsf{SNARK}}$ is knowledge sound which, in turn, implies strong evaluation binding (Theorem 1).

Theorem 13. *Let* PP*,* VC*,* SNARK*, and* $\mathsf{PC}_{\mathsf{SNARK}}$ *as defined in Construction 4.*

1. *If* PP*,* VC*, and* SNARK *satisfy correctness (Definitions 7, 11 and 24), then* $\mathsf{PC}_{\mathsf{SNARK}}$ *satisfies correctness (Definition 18).*
2. *If* VC *satisfies position binding (Definition 25) and* SNARK *satisfies non-black-box (resp. black-box) knowledge soundness (Definition 9), then* $\mathsf{PC}_{\mathsf{SNARK}}$ *satisfies non-black-box (resp. black-box) knowledge soundness (Definition 22).*
3. *If* PP *satisfies efficiency (Definition 12),* VC *and* SNARK *satisfy succinctness (Definitions 4 and 8), then* $\mathsf{PC}_{\mathsf{SNARK}}$ *satisfies succinctness (Definition 2).*
4. *If* PP *satisfies efficiency (Definition 12),* VC *satisfies fast openings (Definition 5), and* SNARK *satisfies succinctness (Definition 8), then* $\mathsf{PC}_{\mathsf{SNARK}}$ *satisfies fast openings (Definition 13).*

Proof.

Correctness. The correctness of $\mathsf{PC}_{\mathsf{SNARK}}$ follows directly from the correctness of PP, VC, and SNARK.

Non-black-box (resp. black-box) knowledge soundness. The following proof is for the non-black-box case. Still, assuming that SNARK satisfies black-box knowledge soundness, the proof can be extended to the black-box knowledge soundness case of $\mathsf{PC}_{\mathsf{SNARK}}$.

We begin by showing that an adversary $\mathcal{A}$ for the (non-black-box) knowledge soundness experiment of $\mathsf{PC}_{\mathsf{SNARK}}$ can be transformed into an adversary $\mathcal{B}$ for the (non-black-box) knowledge soundness experiment of SNARK. We define $\mathcal{B}$ as follows:

- Receive as input srs.
- Sample $\mathsf{pp} \leftarrow \mathsf{PP.Setup}(R, m, d)$ and $\mathsf{ck}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, S)$.
- Execute $\mathcal{A}$ on input $\mathsf{ck} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs})$ and receive $(\mathsf{cm}, x, y, \pi_x)$ where $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$ and $\pi_x = (\pi_{\mathsf{DS}}, v_j, \pi_j)$.
- Output $((\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{srs}), \pi_{\mathsf{DS}})$.

It is easy to see that the output of $\mathcal{B}$ satisfies $\mathsf{SNARK.Ver}(\mathsf{srs}, (\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{cm_{DS}}), \pi) = 1$ whenever the output of $\mathcal{A}$ satisfies $\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1$. This is because $\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1$ implies:

$$\wedge_{j\in[k]} \mathsf{VC.Ver}(\mathsf{ck_{VC}}, \mathsf{cm_{DS}}, i_j, v_j, \pi_j) = 1, \tag{2}$$

$$\mathsf{PP.Reconstruct}(\mathsf{pp}, x, v_1, \ldots, v_k) = y, \text{ and} \tag{3}$$

$$\mathsf{SNARK.Ver}(\mathsf{srs}, (\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{cm_{DS}}), \pi) = 1. \tag{4}$$

By leveraging the (non-black-box) knowledge soundness of SNARK, we have that there exists an extractor $\mathsf{Ext}_{\mathcal{B}}$ satisfying the following:

$$\Pr\left[\begin{array}{l|l} \mathsf{SNARK.Ver}(\mathsf{srs}, (\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{srs}), \pi_{\mathsf{DS}}) = 1 & \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda), \\ \wedge\ ((\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{srs}), (f, r)) \notin \mathcal{R}_{\mathsf{PP},\mathsf{VC}} & ((\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{srs}), \pi_{\mathsf{DS}}) \leftarrow \mathcal{B}(\mathsf{srs}), \\ & (f, r) \leftarrow \mathsf{Ext}_{\mathcal{B}}(\mathsf{srs}) \end{array}\right] \in \mathsf{negl}(\lambda).$$

To conclude the proof, it is sufficient to show that $\mathsf{Ext}_{\mathcal{B}}$ can be used to construct an extractor $\mathsf{Ext}_{\mathcal{A}}$ for the adversary $\mathcal{A}$. We now demonstrate that it is sufficient to set $\mathsf{Ext}_{\mathcal{A}}(\mathsf{ck}) = \mathsf{Ext}_{\mathcal{A}}(\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{srs}) = \mathsf{Ext}_{\mathcal{B}}(\mathsf{srs})$.

First, given that $\mathcal{A}$ satisfies $\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1$, it follows that Equation (4) holds. Furthermore, assuming Equation (4) holds, we obtain $\mathsf{Ext}_{\mathcal{A}}(\mathsf{ck}) = \mathsf{Ext}_{\mathcal{B}}(\mathsf{srs}) \rightarrow (f, r) \in \mathcal{R}_{\mathsf{VC},\mathsf{PP}}$, which implies that $(\mathsf{cm_{DS}}, \cdot) = \mathsf{VC.Com}(\mathsf{ck_{VC}}, \mathsf{DS}; r)$, where $\mathsf{DS} \leftarrow \mathsf{PP.PreProcess}(\mathsf{pp}, f)$. That is, $\mathsf{cm_{DS}}$ and DS are honestly computed using f and randomness r. This implies that:

$$(\mathsf{cm}, \cdot) = \mathsf{Com}(\mathsf{ck}, f; r). \tag{5}$$

Second, given that $\mathcal{A}$ satisfies $\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1$ and considering Equation (3), Equation (2), along with the position-binding property of VC, we obtain that each v_j (for $j \in [k]$) contained in the proof π_x corresponds to the correct block of the honestly computed data structure DS, located at the correct position i_j, where $(i_1, \ldots, i_k) \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, x)$.

Consequently, by the correctness of PP, we obtain:

$$y = f(x) = \mathsf{PP.Reconstruct}(\mathsf{pp}, v_{i_1}, \ldots, v_{i_k}). \tag{6}$$

By combining Equations (5) and (6), we conclude that whenever $\mathcal{A}$ satisfies $\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1$, the extractor $\mathsf{Ext}_{\mathcal{A}}(\mathsf{cm}) = \mathsf{Ext}_{\mathcal{B}}(\mathsf{srs})$ is able to extract, with overwhelming probability, a polynomial f and randomness r such that $(\mathsf{cm}, \cdot) = \mathsf{Com}(\mathsf{ck}, f; r) \wedge y = f(x)$. Thus, $\mathsf{PC_{SNARK}}$ satisfies (non-black-box) knowledge soundness.

Succinctness. For both univariate and multivariate polynomials, the proof is analogous to that of Theorem 4 (Appendix E.1), except for the following aspects:

- The opening proof of $\mathsf{PC_{SNARK}}$ additionally includes a SNARK proof π_{DS}. Since SNARK satisfies succinctness, we have $|\pi_{\mathsf{DS}}| \in \mathsf{poly}(\lambda, \log |\mathbb{w}|)$, where $\mathbb{x} = (\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{cm_{DS}})$ and $|\mathbb{w}| = |f| + |r|$. Hence, the opening proofs of $\mathsf{PC_{SNARK}}$ remain bounded by $\mathsf{poly}(\lambda, \log |R|, d, m)$, as in the proof of Theorem 4.
- During verification, Ver of $\mathsf{PC_{SNARK}}$ executes $\mathsf{SNARK.Ver}$. Due to the succinctness of SNARK, we conclude that the running time of Ver remains $\mathsf{poly}(\lambda, \log |R|, d, m)$ since $\mathsf{SNARK.Ver}$ runs in time $\mathsf{poly}(\lambda, |\mathbb{x}|, \log |\mathbb{w}|) = \mathsf{poly}(\lambda, \log |R|, d, m)$, where $\mathbb{x} = (\mathsf{pp}, \mathsf{ck_{VC}}, \mathsf{cm_{DS}})$ and $\mathbb{w} = (f, r)$.

Fast Openings. For both univariate and multivariate polynomials, the proof is analogous to that of Theorem 4 (Appendix E.1), except that the running time of Com in $\mathsf{PC}_{\mathsf{SNARK}}$ additionally includes the running time of $\mathsf{SNARK.Prove}$. The latter is polynomial in the statement $\mathbb{x} = (\mathsf{pp}, \mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}})$ and witness $\mathbb{w} = (f, r)$. Hence, the running time of Com remains bounded by $\mathsf{poly}(\lambda, \log |R|, d^m)$, as in the proof of Theorem 4.

□

Engineering the (SNARK) relation through algebraic approaches. The aforementioned approach requires a SNARK supporting $\mathcal{R}_{\mathsf{VC},\mathsf{PP}}$, which, in turn, relies on the entire computation of $\mathsf{PP.PreProcess}(\mathsf{pp}, f)$. By opening the box, we observe that the PP scheme of Kedlaya and Umans [KU08, KU11] has an algebraic structure. For example, the data structure DS (output by $\mathsf{PP.PreProcess}$) consists of multiple small tables, each containing all evaluations of a multivariate polynomial f_i (derived by manipulating the original one) modulo a small prime p_i. Due to the algebraic nature of this PP scheme, we envision the possibility of engineering and simplifying the SNARK relation to achieve a faster PC. However, this direction is outside the scope of this paper, and we leave it as a possible direction for future work.

D Supporting Material for UC SNARKs

D.1 UC Framework

In this work, we use the *Universal Composability* (UC) framework [Can01] for security proofs. UC follows the simulation-based paradigm where the security of a protocol is defined with respect to an ideal world where a trusted party, the functionality $\mathcal{F}$, does the all of the computation. Informally, a protocol securely realizes $\mathcal{F}$ in the real world if for any real world adversary there exist an *equivalent* ideal world adversary (the simulator), "equivalent" meaning that any outside observer (the environment) cannot distinguish between the real protocol execution and the ideal execution. UC's composition theorem ensures that one can safely compose protocols that have been proven UC-secure.

Global Random Oracle. More precisely, we use the generalized UC (GUC) framework [CDPW07] which allows to model global functionalities that are shared between different protocol instances. We consider a hybrid-model where parties have access to a (non-programmable) global random oracle $\mathcal{G}_{\mathsf{RO}}$ as introduced in [CJS14]. We follow the simplified description from [CDG+18]. The $\mathcal{G}_{\mathsf{RO}}$ functionality can be queried by any party and the ideal adversary with two commands: QUERY and OBSERVE. The environment can query $\mathcal{G}_{\mathsf{RO}}$ by spawning additional dummy parties outside the context of the current protocol execution. $\mathcal{G}_{\mathsf{RO}}$ answers all new QUERY command by lazy sampling from the domain and stores them locally in a list $\mathcal{Q}$. A repeated query requires a simple lookup in $\mathcal{Q}$. Some QUERY queries are marked "illegitimate" and can be observed via OBSERVE command. Next we explain which query counts as an illegitimate one. Each party is associated with its party identifier pid and a session identifier sid. When a party queries $\mathcal{G}_{\mathsf{RO}}$ with the command (QUERY, x), the query is parsed as (s, x') where s denotes the session identifier associated with the party. A query is marked as illegitimate if the sid field of the query differs from the sid associated to the party making the query. In other words, these are the queries made outside the context of the current session execution. We formally define the functionality $\mathcal{G}_{\mathsf{RO}}$ in Fig. 3.

$\mathcal{G}_{\mathsf{RO}}$ is parametrized by the output length $\ell(\lambda)$.

- **Query** Upon receiving a query (QUERY, x), from some party $\mathcal{P} = (\mathsf{pid}, \mathsf{sid})$ or from the adversary Sim do:
  - Look up v if there is a pair (x, v) for some $v \in \{0,1\}^{\ell(\lambda)}$ in the (initially empty) list $\mathcal{Q}$ of past queries. Else, choose uniformly $v \in \{0,1\}^{\ell(\lambda)}$ and store the pair (x, v) in $\mathcal{Q}$.
  - Parse x as (s, x'). If $\mathsf{sid} \neq s$ then add (s, x', v) to the (initially empty) list of illegitimate queries for SID s, that is denoted by $\mathcal{Q}_{|s}$.
  - Return v to $\mathcal{P}$.
- **Observe** Upon receiving a request $(\text{OBSERVE}, \mathsf{sid})$ from the adversary Sim, return the list $\mathcal{Q}_{|\mathsf{sid}}$ of illegitimate queries for SID sid to the adversary.

Fig. 3. Functionality for Global Random Oracle $\mathcal{G}_{\mathsf{RO}}$ [CDG+18]

Definition 26 (UC Security in the Global ROM [CDPW07, CJS14]). *Let $\mathcal{F}, \mathcal{F}'$ be m-party functionalities and Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ in the $\mathcal{G}_{\mathsf{RO}}, \mathcal{F}'$-hybrid model if for any hybrid-model* PPT *adversary $\mathcal{A}$, there exists an ideal process* PPT *adversary* Sim *such that for every* PPT *environment $\mathcal{Z}$, it holds that:*

$$\{\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z} \approx \{\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}',\Pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}$$

where REAL *is the outputs of the honest parties and the adversary $\mathcal{A}$ after a real execution of protocol Π with input $\mathbf{x} = (x_1, \ldots, x_m)$ for parties $P_1, \ldots, P_m$ where each $x_i \in \{0,1\}^*$, $z \in \{0,1\}^*$ is the auxiliary input for $\mathcal{A}$ and λ is the security parameter.* IDEAL *is the analogous distribution in an ideal execution with a trusted party that computes $\mathcal{F}$ for the parties and hands the output to the designated players.*

D.2 (Succinct) Non Interactive Zero-Knowledge Proofs

Although we already defined SNARK for indexed relations in Section 2.3, in this section we provide an alternative syntax and formalism in order to stay as faithful as possible to the original treatment in [GKO+23]; we will also not be needing indexed relations.

A *non-interactive proof system* for relation $\mathcal{R}$, denoted by $\Pi_{\mathcal{R}}$, consists a tuple of algorithms $(\mathsf{Setup}, \mathcal{O}_{\mathsf{Setup}}, \mathcal{P}, \mathcal{V})$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: Takes as input the security parameter λ and outputs public parameters pp. Once Setup is invoked we assume that all of the following algorithms take pp as an implicit input.
- $\mathsf{out} \leftarrow \mathcal{O}_{\mathsf{Setup}}(\mathsf{in})$: A stateful setup oracle that takes an input string in and outputs out.
- $\pi \leftarrow \mathcal{P}^{\mathcal{O}_{\mathsf{Setup}}}(x, w)$: Takes as input a statement x and witness w, and outputs a proof π if $(x, w) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}^{\mathcal{O}_{\mathsf{Setup}}}(x, \pi)$: Takes as input a statement x and proof π, and outputs a bit b, indicating "accept" or "reject".

We introduce the setup oracle $\mathcal{O}_{\mathsf{Setup}}$ to the notation of NIZKs to capture the two typical setup assumptions in an abstract manner. That is, if a proof system is instantiated in the CRS model, then $\mathcal{O}_{\mathsf{Setup}}$ internally generates crs upon receiving a query with any input for the first time, and

keeps outputting the same crs regardless of the input. When instantiating the RO model, $\mathcal{O}_{\mathsf{Setup}}$ is initialized with an empty query-response table and proceeds as follows. On receiving $\mathsf{in} \in \{0,1\}^*$, if in has never been queried, sample uniform $\mathsf{out} \in \{0,1\}^{\ell(\lambda)}$, store $(\mathsf{in}, \mathsf{out})$ in the table, and return out. Otherwise, look up the table to find out associated with in, and return out.

We define three basic security properties for $\Pi_{\mathcal{R}}$ in the stand-alone setting.

Definition 27 (Completeness). *$\Pi_{\mathcal{R}}$ satisfies* completeness *if for every $(x,w) \in \mathcal{R}$, it holds that*

$$\Pr\left[b = 1 \ : \ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathcal{P}^{\mathcal{O}_{\mathsf{Setup}}}(x,w); b \leftarrow \mathcal{V}^{\mathcal{O}_{\mathsf{Setup}}}(x,\pi)\right] = 1.$$

We define zero-knowledge by following the syntax of [FKMV12, GOP+22]. A zero-knowledge simulator $\mathcal{S}$ is defined as a stateful algorithm with initial state $\mathsf{st} = \mathsf{pp}$ that operates in two modes. The first mode, $(\mathsf{out}, \mathsf{st}') \leftarrow \mathcal{S}(1, \mathsf{st}, \mathsf{in})$ takes care of handling calls to the oracle $\mathcal{O}_{\mathsf{Setup}}$ on input in. The second mode, $(\pi, \mathsf{st}') \leftarrow \mathcal{S}(2, \mathsf{st}, x)$ simulates a proof for the input statement x. For convenience we define three "wrapper" oracles. These oracles are stateful and share the internal state st, which initially contains an empty string.

- $\mathcal{S}_1(\mathsf{in})$ to denote the oracle that returns the first output of $\mathcal{S}(1, \mathsf{st}, \mathsf{in})$;
- $\mathcal{S}_2(x,w)$ that returns the first output of $\mathcal{S}(2, \mathsf{st}, x)$ if $(x,w) \in \mathcal{R}$ and $\perp$ otherwise;
- $\mathcal{S}_2'(x)$ that returns the first output of $\mathcal{S}(2, \mathsf{st}, x)$.

Definition 28 ((Unbounded) Zero-Knowledge). *Consider a non-interactive proof system $\Pi_{\mathcal{R}} = (\mathsf{Setup}, \mathcal{O}_{\mathsf{Setup}}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$. $\Pi_{\mathcal{R}}$ is unbounded* non-interactive zero-knowledge *(*NIZK*), if there exists a* PPT *simulator $\mathcal{S}$ with wrapper oracles $\mathcal{S}_1$ and $\mathcal{S}_2$ such that for all* PPT *adversaries $\mathcal{A}$ it holds that*

$$\left|\Pr\left[b = 1 \ : \ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Setup}}, \mathcal{P}}(\mathsf{pp}) \end{array}\right] - \Pr\left[b = 1 \ : \ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ b \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}_2}(\mathsf{pp}) \end{array}\right]\right| < \mathsf{negl}(\lambda).$$

Next, we define simulation extractability, which essentially guarantees that proofs are *non-malleable*. We stress that the present definition is weaker than what is necessary for realizing UC security, because the extractor algorithm here is *non-black-box*, i.e., it requires looking into the code of the adversary. The definition is an abstracted version of [GM17] and the schemes satisfying their definition clearly meet the version below by instantiating $\mathcal{S}$ with trapdoor'd CRS generator in mode 1 and ZK simulator in mode 2. (Following [GM17], we also assume the relation $\mathcal{R}$, which may potentially contain auxiliary inputs, is *benign*, i.e. it is distributed in such a way that the SNARKs for $\mathcal{R}$ can be simulation extractable)

Definition 29 ((Non-black-box) Simulation Extractability). *Consider a non-interactive proof system $\Pi_{\mathcal{R}} = (\mathsf{Setup}, \mathcal{O}_{\mathsf{Setup}}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ with a* NIZK *simulator $\mathcal{S}$. Let $(\mathcal{S}_1, \mathcal{S}_2')$ be wrapper oracles for $\mathcal{S}$ as defined above. $\Pi_{\mathcal{R}}$ is non-black-box* simulation-extractable *(SIM-EXT) with respect to $\mathcal{S}$, if for any* PPT *adversary $\mathcal{A}$, there exists a* PPT *extractor $\mathcal{E}_{\mathcal{A}}$ such that*

$$\Pr\left[\begin{array}{r} (x,\pi) \notin \mathcal{Q} \wedge (x,w) \notin \mathcal{R} \\ \wedge\, b = 1 \end{array} \ : \ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_1, \mathcal{S}_2'}(\mathsf{pp}); \\ b \leftarrow \mathcal{V}^{\mathcal{S}_1}(x,\pi); w \leftarrow \mathcal{E}_{\mathcal{A}}(x, \pi, \mathsf{state}_{\mathcal{A}}, \mathsf{st}) \end{array}\right] < \mathsf{negl}(\lambda)$$

where st is the final state of the simulator $\mathcal{S}$, $\mathsf{state}_{\mathcal{A}}$ is a string containing all inputs and outputs of $\mathcal{A}$, including random coins, and $\mathcal{Q}$ is a set of statement-proof pairs (x,π) with x being a statement

$\mathcal{F}_{\mathsf{Setup}}$ is parametrized by a security parameter λ and a degree bound $D > 0$ and runs with parties $P_1, \ldots, P_N$ and an ideal process adversary Sim.

- **Parameters** Upon receiving input $(\textsc{genparams}, \mathsf{sid})$ from a party P_i, if no pp has been stored, run $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, initialize oracle $\mathcal{O}_{\mathsf{Setup}}$ with pp, and store pp. Send $(\textsc{params}, \mathsf{sid}, \mathsf{pp})$ to P_i.
- **Commitment Key** Upon receiving input $(\textsc{genkey}, \mathsf{sid})$ from a party P_i, if no ck has been stored, run $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, D)$ and store ck. Send $(\textsc{comkey}, \mathsf{sid}, \mathsf{ck})$ to P_i.
- **Setup** Upon receiving input $(\textsc{setup}, \mathsf{sid}, \mathsf{in})$ from a party P_i, ignore if $\mathcal{O}_{\mathsf{Setup}}$ has not been initialized with pp. Otherwise run $\mathsf{out} \leftarrow \mathcal{O}_{\mathsf{Setup}}(\mathsf{in})$ using the current state of $\mathcal{O}_{\mathsf{Setup}}$ and send $(\textsc{setup}, \mathsf{sid}, \mathsf{out})$ to P_i.

Fig. 4. N-party functionality for setup $\mathcal{F}_{\mathsf{Setup}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parametrized by polynomial-time-decidable relation $\mathcal{R} \in \{0,1\}^* \times \{0,1\}^*$, and runs with parties $P_1, \ldots, P_N$ and an ideal process adversary Sim. It stores proof table $\mathcal{Q}$ which is initially empty.

- **Proof** Upon receiving input $(\textsc{prove}, \mathsf{sid}, \mathsf{ssid}, x, w)$ from a party P_i, ignore if $(x, w) \notin \mathcal{R}$. Otherwise, send $(\textsc{prove}, \mathsf{sid}, x)$ to Sim. Upon receiving $(\textsc{proof}, \pi)$ from Sim, store (x, π) in $\mathcal{Q}$ and send $(\textsc{proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$ to P_i.
- **Verification** Upon receiving input $(\textsc{verify}, \mathsf{sid}, \mathsf{ssid}, x, \pi)$ from a party P_i, if (x, π) is not stored in $\mathcal{Q}$, then send $(\textsc{verify}, \mathsf{sid}, x, \pi)$ to Sim. Upon receiving $(\textsc{witness}, w)$ from Sim, if $(x, w) \in \mathcal{R}$, store (x, π) in $\mathcal{Q}$. Finally, return $(\textsc{verification}, \mathsf{sid}, \mathsf{ssid}, (x, \pi) \in_? \mathcal{Q})$ to P_i.

Fig. 5. N-party functionality for non-interactive zero-knowledge $\mathcal{F}_{\mathsf{NIZK}}$

queried by $\mathcal{A}$ to the proof simulation wrapper oracle $\mathcal{S}'_2$, and π being the corresponding simulated proof, respectively.

The ideal functionality $\mathcal{F}_{\mathsf{Setup}}$ that provides the setup and oracle for non-interactive proof system $\Pi_{\mathcal{R}} = (\mathsf{Setup}, \mathcal{O}_{\mathsf{Setup}}, \mathcal{P}, \mathcal{V})$ is described in Fig. 4.

Our final goal is to compile $\Pi_{\mathcal{R}}$ with the above basic security properties into a UC-secure NIZK protocol $\Pi_{\mathcal{R}}$. The ideal functionality for Non-interactive Zero-Knowledge $\mathcal{F}_{\mathsf{NIZK}}$ is defined in Fig. 5. The functionality is taken from [GOS12] with a minor difference being that $\mathcal{F}_{\mathsf{NIZK}}$ explicitly informs Sim of the associated session ID.

Definition 30. *Let $\mathcal{R}$ be an NP-relation. We say that a protocol Π is a UC NIZK if it UC-realizes $\mathcal{F}_{\mathsf{NIZK}}$ (Fig. 5) in the $(\mathcal{G}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Setup}})$-hybrid model for relation $\mathcal{R}$.*

D.3 Special Properties of Polynomial Commitments and Polynomial Encoding Schemes [GKO+23, BCC+24]

In this section we follow [GKO+23, BCC+24] and define a series of properties and notions that are ancillary to polynomial commitments. These include: unique proofs property, polynomial encoding scheme, evaluation hiding, non-extrapolation (which we will present only to show it is implied by evaluation hiding and thus justify why we do not include it among our hypothesis in Theorem 10). Notice that all the polynomials assumed in this section are *univariate*.

Definition 31 (Unique Proof (Polynomial Commitments)). *We say a polynomial commitment scheme has unique proofs if, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\ b = 1 \\ \wedge\ b' = 1 \end{array} : \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{Setup}(\lambda, d); \\ (c, z, y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}); \\ b \leftarrow \mathsf{Ver}(\mathsf{ck}, c, z, y, \pi); \\ b' \leftarrow \mathsf{Ver}(\mathsf{ck}, c, z, y, \pi') \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We adopt the definition of polynomial encoding scheme from [GKO+23] (using the variant formalization presented in [BCC+24]). At its essence, a polynomial encoding scheme takes a vector of field elements and outputs a corresponding (appropriately masked) polynomial.

Definition 32 (Polynomial Encoding Scheme). *A* polynomial encoding scheme*, denoted by* $= \mathsf{PES}$*, is a tuple of algorithms* $(\mathsf{Enc}, \mathsf{Dec})$

- $f \leftarrow \mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho})$*: Takes as inputs a security parameter,* $\mathbf{w} \in \mathbb{F}^n$*, dimension of the vector* $n > 0$*, evaluation bound* $\ell > 0$*, and randomness* $\boldsymbol{\rho} \in \mathbb{F}^\ell$*, and outputs a polynomial* $f \in \mathbb{F}_{<d}[X]$ *where* d *is a function of* n *and* ℓ*.*
- $\mathbf{w}' \leftarrow \mathsf{Dec}(1^\lambda, f, n, \ell)$*: Takes as inputs a security parameter,* $f \in \mathbb{F}_{<n+\ell}[X]$*,* $n > 0$*, and* $\ell > 0$*, and deterministically outputs* $\mathbf{w}' \in \mathbb{F}^n$*.*

We say $= \mathsf{PES}$ *is* correct *if* $\mathbf{w} = \mathsf{Dec}(1^\lambda, \mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho}), n, \ell)$ *for any* $n > 0$*,* $\ell > 0$*,* $\mathbf{w} \in \mathbb{F}^n$*, and* $\boldsymbol{\rho} \in \mathbb{F}^\ell$*. We define the stretch factor* $\mathsf{stretch}(\lambda, n, \ell)$ *of the PES as the difference between the size of the encoding and the original size of the vector* $\mathbf{w}$*, i.e.,* $\mathsf{stretch}(\lambda, n, \ell)$ *will always be equal to* $\mathsf{deg}(f) + 1 - n$*. We say that a polynomial encoding scheme is "*$n \to_\lambda d$*" if, on input a security parameter* λ *and a string of size* n*, it outputs a polynomial of degree* d*.*

We only consider polynomial encoding schemes where the size of the field domain is exponential in the security parameter, i.e. $|\mathbb{F}| \in O(2^\lambda)$.

Definition 33 (ϕ-Evaluation Hiding). *Let* $PCS = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *be a polynomial commitment scheme in the random oracle model* $\mathcal{O}_{\mathsf{Setup}}$ *and* $PES = (\mathsf{Enc}, \mathsf{Dec})$ *be a polynomial encoding scheme. We say* PCS *is* ϕ-evaluation hiding with respect to $= \mathsf{PES}$ *if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, for all* $\lambda, n, r \in \mathbb{N}$

$$\Pr\left[b = b' \ : \begin{array}{r} \ell := \phi(\lambda, n, r); d := n + \mathsf{stretch}(\lambda, n, \ell); \\ \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, d); \\ \mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}_1(\mathsf{ck}); \mathbf{z} \leftarrow\!\$\ \mathbb{F}^r \\ \boldsymbol{\rho}_w \leftarrow\!\$\ \mathbb{F}^\ell; b \leftarrow\!\$\ \{0,1\}; \\ f \leftarrow \mathsf{Enc}(1^\lambda, b \cdot \mathbf{w}, n, \ell; \boldsymbol{\rho}_w); \\ c \leftarrow \mathsf{Com}(\mathsf{ck}, f); \\ \mathbf{y} := f(\mathbf{z}); \\ \boldsymbol{\pi} \leftarrow \mathsf{Open}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f); \\ b' \leftarrow \mathcal{A}_2(c, \mathbf{y}, \boldsymbol{\pi}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{A}_1, \mathcal{A}_2$ *share the internal states,* $\mathbf{y} := f(\mathbf{z})$ *denotes setting* $y_i := f(z_i)$ *for all* $i \in [|\mathbf{z}|]$, $\boldsymbol{\pi} \leftarrow \mathsf{Open}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f)$ *denotes setting* $\pi_i \leftarrow \mathsf{Open}(\mathsf{ck}, c, z_i, y_i, f)$ *for all* $i \in [|\mathbf{z}|]$. *If PCS is* ϕ*-evaluation hiding with respect to PES for some* $\phi \in O_\lambda(n + r)$ *then we simply say that "PCS is evalation hiding with respect to PES".*

Definition 34 (ϕ-Non-Extrapolation). *Let* $PCS = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ *be a polynomial commitment scheme in the random oracle model* $\mathcal{O}_{\mathsf{Setup}}$ *and* $PES = (\mathsf{Enc}, \mathsf{Dec})$ *be a polynomial encoding scheme. We say* PCS *supports* ϕ-non-extrapolation with respect to $= \mathsf{PES}$ *if for all* PPT *adversaries* $\mathcal{A}$*, for all* $\lambda, n, r \in \mathbb{N}$

$$\Pr\left[\begin{array}{r} \ell := \phi(\lambda, n, r); d := n + \mathsf{stretch}(\lambda, n, \ell); \\ \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, d); \\ \mathbf{z} \leftarrow\!\$\, \mathbb{F}^r; \boldsymbol{\rho}_w \leftarrow\!\$\, \mathbb{F}^\ell; \\ f \leftarrow \mathsf{Enc}(1^\lambda, 0^n, n, \ell; \boldsymbol{\rho}_w); \\ c \leftarrow \mathsf{Com}(\mathsf{ck}, f); \\ \mathbf{y} := f(\mathbf{z}); \\ \boldsymbol{\pi} \leftarrow \mathsf{Open}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, f); z^* \leftarrow\!\$\, \mathbb{F} \\ (y^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, c, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}, z^*); \\ v \leftarrow \mathsf{Ver}(\mathsf{ck}, c, z^*, y^*, \pi^*) \end{array} : v = 1 \ \wedge \ z^* \notin \mathbf{z}\right] \in \mathsf{negl}(\lambda)$$

In its original presentation in [GKO+23], non-extrapolation property is described as something to be "proven separately". However, it is in fact implied by ϕ-evaluation hiding in all scenarios of interest (as also hinted in [BCC+24]). An implication of this is that one can remove non-extrapolation from the requirements of the compiler in [GKO+23] (and therefore in Theorem 10). We formalize this intuition in the following theorem (whose proof is essentially the same as in [BCC+24] where it is used to prove a more specific statement).

Theorem 14. *A polynomial commitment satisfying weak evaluation binding and* ϕ*-evaluation hiding with respect to a polynomial encoding scheme also satisfies* ϕ*-non-extrapolation with respect to the same encoding scheme.*

Proof. Consider the following hybrids:

- Hyb_0: this is the same as the game in Definition 34 where an all-zero vector of length n is encoded as a polynomial and we provide the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with up to r evaluation points and corresponding evaluation proofs.
- Hyb_1: we now change part of the challenger's code. Instead of encoding an all-zero vector, we proceed by sampling a set of random evaluations and then using (in part) the evaluation points required by the adversary to interpolate the polynomial. More in detail:
  - we first sample d random evaluations $y_i \leftarrow\!\$\, \mathbb{F}$.
  - Let $\mathbf{z}$ be the sampled evaluation points and let $\mathbf{z}'$ a vector of unique points in $\mathbf{z}$. Let $r' := |\mathbf{z}'|$ and let $n' := d - r'$.
  - Sample n' points $\mathbf{z}''$ from $\mathbb{F}^{n'}$.
  - Interpolate f so that $f(z'_i) = y_i$ for $i \in [r']$ and $f(z''_j) = y_{j+r'}$ for $j \in [n']$

- Compute commitments and evaluation proofs as before.

By applying ϕ-evaluation hiding we can conclude that the two hybrids are indistinguishable and therefore the polynomial f looks random to $\mathcal{A}$ after requesting r evaluations. Let us now consider (y^*, π^*), the output of $\mathcal{A}_2$ for $z^* \leftarrow\!\!\$\ \mathbb{F}$. By the previous observation, the probability that $\Pr[y^* = f(z^*)]$ is negligible. If $y^* \neq f(z^*)$ and $\mathcal{A}$ wins it is then possible to break evaluation binding since we can produce two valid evaluation proofs for two distinct points for the same committed polynomial. This concludes the proof.

□

D.4 Additional Relevant Cryptographic Preliminaries

Unique Proofs for Vector Commitments. We leverage the following property to prove that our construction of polynomial commitments with sublinear opening $\mathsf{PC}_{\mathsf{VC}}$ has unique proofs.

Definition 35 (Unique Proof (Vector Commitments)). *We say a vector commitment scheme has unique proofs if, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\ b = 1 \\ \wedge\ b' = 1 \end{array} : \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (\mathsf{cm}, i, y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}); \\ b \leftarrow \mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, i, y, \pi); \\ b' \leftarrow \mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, i, y, \pi') \end{array}\right] \in \mathsf{negl}(\lambda).$$

Strong Diffie-Hellman Assumption.

Definition 36 (SDH Assumption [BB04]). *The strong Diffie-Hellman assumption (SDH) holds with respect to a bilinear group generator* BGen *if for all* PPT *adversaries* $\mathcal{A}$ *and degree bound* $D > 0$*, we have*

$$\Pr\left[t = g^{\frac{1}{\alpha+c}} \mid \mathcal{G} \leftarrow \mathsf{BGen}(1^\lambda); \alpha \leftarrow\!\!\$\ \mathbb{F}; \sigma := (\{g^{\alpha^i}\}_{i=0}^{D-1}, h^\alpha); (t, c) \leftarrow \mathcal{A}(\mathcal{G}, \sigma)\right] \in \mathsf{negl}(\lambda)$$

Remark 3. In the statement of Theorem 9 we use some of the implications of SDH in terms of cryptographic assumptions and primitives, namely: SDH $\implies$ DLOG $\implies$ $\exists$CRHF (via Pedersen hashing) and DLOG $\implies$ $\exists$OWF $\implies$ $\exists$SKE.

The KZG Polynomial Commitment Scheme [KZG10].

Construction 5 (KZG PC scheme [KZG10]) *The KZG PC scheme of [KZG10] is defined as follows:*

$\mathsf{Setup}(1^\lambda, D)$**:** *Generate the parameters of a bilinear group* $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g, h, e)$ *where* $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = q$ *is prime,* $\langle g \rangle = \mathbb{G}_1$*,* $\langle h \rangle = \mathbb{G}_2$*, and* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is an efficiently computable, non-degenerate biinear map. The group order* q *also determines* $\mathbb{F} := \mathbb{F}_q$ *and a set of supported polynomials* $\mathbb{F}_{<D}[X]$*. Sample* $\alpha \in \mathbb{F}$ *uniformly, and compute* $\sigma = (g, g^\alpha, \ldots, g^{\alpha^{D-1}}, h, h^\alpha)$*. Output* $\mathsf{ck} = (\mathcal{G}, \sigma)$*.*

$\mathsf{Com}(\mathsf{ck}, f, d)$**:** *On input* ck*, a polynomial* $f \in \mathbb{F}_{<d}[X]$*, and a degree bound* $d \leq D$*, compute a shifted polynomial* $\hat{f} = X^{D-d} \cdot f$*, and generate a commitment as* $\mathbf{c} = (g^{f(\alpha)}, g^{\hat{f}(\alpha)})$ *and output* $\mathbf{c}$*.*

$\mathsf{Open}(\mathsf{ck}, c, d, z, f(z), f)$**:** *Compute* $\omega(X) = (f(X) - f(z))/(X - z)$ *and* $\hat{\omega}(X) = (\hat{f}(X) - \hat{f}(z))/(X - z)$, $\hat{f}$ *computed as above. Output* $\boldsymbol{\pi} = (g^{\omega(\alpha)}, g^{\hat{\omega}(\alpha)}, \hat{f}(z))$.

$\mathsf{Ver}(\mathsf{ck}, \mathbf{c}, d, z, y, \boldsymbol{\pi})$**:** *Parse* $\mathbf{c} = (c, \hat{c})$ *and* $\boldsymbol{\pi} = (\pi, \hat{\pi}, \hat{y})$. *Accept if and only if* $e(c/g^y, h) = e(\pi, h^\alpha/h^z)$, $e(\hat{c}/g^{\hat{y}}, h) = e(\hat{\pi}, g^\alpha/h^z)$, *and* $\hat{y} = z^{D-d} \cdot y$.

The security of $\mathsf{PC}_{\mathsf{KZG}}$ relies on the SDH assumption (Definition 36). We can prove it satisfies some of our properties of interests with respect to the simple following polynomial encoding scheme:

Definition 37 (Coefficient-based PES [GKO+23]). *The coefficient-based polynomial encoding scheme* PES_{coeff} *is defined as follows:*

- $\mathsf{Enc}(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho})$*: Return* $f(X) := \sum_{i=1}^{n} w_i X^{i-1} + \sum_{i=1}^{\ell} \rho_i X^{n+i-1}$;
- $\mathsf{Dec}(1^\lambda, f, n, \ell)$*: Return the first* n *coefficients of* f.

Lemma 4. *([GKO+23])* PC_{KZG} *satisfies the unique proofs (Definition 31) and evaluation binding property under the SDH assumption. It is also* ϕ*-evaluation hiding (Definition 33) with respect to the polynomial encoding scheme* PES_{coeff} *(Definition 37) where* $\phi(\cdot, \cdot, r) := r + 1$.

Secret-Key Encryption A symmetric encryption (SKE) scheme with message space $\mathcal{M}$ is composed of the following polynomial-time algorithms:

$\mathsf{KGen}(1^\lambda)$**:** The randomized key generator takes as input the security parameter 1^λ and outputs a symmetric key sk.

$\mathsf{Enc}(\mathsf{sk}, m)$**:** The randomized encryption algorithm takes as input a symmetric key sk and a message $m \in \mathcal{M}$, it outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$**:** The deterministic decryption algorithm takes as input a symmetric key sk and a ciphertext ct, it outputs a message m.

A SKE is correct if the honest ciphertext correctly decrypts.

Definition 38 (Correctness of SKE). *A SKE* SKE *with message space* $\mathcal{M}$ *is correct if* $\forall \lambda \in \mathbb{N}$, $\forall m \in \mathcal{M}$, *we have*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, m)) = m | \mathsf{sk} \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)] = 1.$$

Definition 39 (Semantic security of SKE). *We say that a SKE* SKE *with message space* $\mathcal{M}$ *is* semantically secure *if for every* $m_0, m_1 \in \mathcal{M}$, *every PPT adversaries* $\mathcal{A}$, *we have:*

$$\left| \Pr[\mathbf{G}^{\mathsf{SKEsec}}_{\mathsf{SKE},\mathcal{A},m_0,m_1}(\lambda) = 1] - \frac{1}{2} \right| \in \mathsf{negl}(\lambda),$$

where the experiment $\mathbf{G}^{\mathsf{SKEsec}}_{\mathsf{SKE},\mathcal{A},m_0,m_1}(\lambda)$ *is depicted in Figure 6.*

Definition 40 (Admissible SKE). *We say that a SKE* SKE *is admissible with respect to a polynomial commitment* PC *if the following holds: let* λ *be any security parameter and let* $\mathbb{F}$ *be the scalar field supported by* PC *when the commitment key is generated with security parameter* λ, *then on input the same security parameter* λ, SKE *has keys, plaintexts and ciphertexts that are vectors of elements in* $\mathbb{F}$.

$\mathbf{G}^{\mathsf{SKEsec}}_{\mathsf{SKE},\mathcal{A},m_0,m_1}(\lambda)$

$\mathsf{sk} \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$

$b \leftarrow\!\!\$\ \{0,1\}$

$\mathsf{ct} \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{sk}, m_b)$

$b' \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathsf{ct})$

If $b' = b$, **return** 1

return 0

Fig. 6. SKE semantic security.

Simulation-extractability of NIZKs in literature. In the theorems below n is the multiplicative complexity of the arithmetic circuit describing the relation. The verifier has an additive overhead linear in the size of the public input which below we keep implicit since irrelevant in the compiler from [GKO+23].

Theorem 15 ([GM17]). *Under the Computational Polynomial Assumption and the XPKE Assumption*[31] *(See Assumption 4.1 and 4.2 in [GM17]), there exists a simulation-extractable NIZK with $O_\lambda(1)$ proofs of size and verification time and proving time $\widetilde{O}_\lambda(n)$.*

D.5 The compiler Π_{GKOPTT} of [GKO+23]

In this section, we describe the compiler Π_{GKOPTT} of [GKO+23] for NP-relation $\mathcal{R}$ following the presentation in [BCC+24]. Specifically, the compiler makes use of the following tools:

- Π_{PCS}, a polynomial commitment scheme with weak evaluation binding, unique proofs (Definition 31), ϕ-evaluation hiding (Definition 33) with respect to a companion encoding scheme $\mathsf{PES} = (\mathsf{Enc}, \mathsf{Dec})$ (Definition 32).
- Let Π_{NIZK} be a simulation-extractable NIZK (Definition 29), for the relation $\mathcal{R}_{\mathsf{NIZK}} = \{((x, \mathsf{ck}, n, \ell), (w, \boldsymbol{\rho}_w)) : (x, w) \in \mathcal{R} \ \wedge\ \mathsf{cm} = \mathsf{Com}(\mathsf{ck}, \mathsf{Enc}(\mathbf{w}, n, \ell; \boldsymbol{\rho}_w))\}$ where $\mathbf{w}$ denotes the witness w parsed as a vector of field elements in $\mathbb{F}^n$.

The protocol Π_{GKOPTT} is parameterized by:

1. Security parameter λ
2. Finite field $\mathbb{F}$
3. Evaluation hiding factor $\phi : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$ and stretch function $\mathsf{stretch} : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$
4. Number of parallel repetitions $r = r(\lambda) > 0$
5. Proof-of-work parameter $b(\lambda) > 0$
6. Bound $T(\lambda) > 0$
7. Maximum degree bound $D > 0$ for Π_{PCS}

The protocol

– **Proof:** On input (PROVE, sid, x, w), ignore if $(x, w) \notin \mathcal{R}$; otherwise, P_i does:

[31] eXtended Power of Knowledge of Exponent assumption.

1. Send (QUERY, (sid, x, genparamsproof)) to $\mathcal{G}_{\mathrm{RO}}$ receiving back pp.
2. Send (QUERY, (sid, x, genparamspc)) to $\mathcal{G}_{\mathrm{RO}}$ receiving back ck.
3. Parse $w = \mathbf{w} \in \mathbb{F}^n$. Let $\ell := \phi(\lambda, n, r)$ and $d := \mathsf{stretch}(\lambda, n, \ell) + n$. If $d > D$, abort by outputting (PROOF, sid, $\perp$).
4. Generate a polynomial encoding of the witness vector: $f \leftarrow \mathsf{Enc}\left(1^\lambda, \mathbf{w}, n, \ell; \boldsymbol{\rho}_w\right)$, where $\boldsymbol{\rho}_w \leftarrow \mathbb{F}^\ell$.
5. Generate a commitment to the polynomial encoding: $(\mathsf{cm}, \mathsf{aux}) \leftarrow \mathsf{Com}\left(\mathsf{ck}, f\right)$
6. Run the prover $\mathcal{P}$ of Π_{NIZK} on input $x' = (\mathsf{pp}, (x, \mathsf{ck}, n, \ell))$ and $w' = (w, \boldsymbol{\rho}_w)$ to obtain a proof π'. Whenever $\mathcal{P}$ makes a call to $\mathcal{O}_{\mathsf{Setup}}$ with input in, send (QUERY, (sid, in, proof)) to $\mathcal{G}_{\mathrm{RO}}$ to receive a response out which is forwarded to $\mathcal{P}$.
7. Initialize empty sets $\mathbf{z}, \mathbf{y}$, and $\boldsymbol{\pi}_{\mathrm{PCS}}$.
8. For each iteration $i \in [r]$ do:
   (a) Initialize counter $\mathsf{ctr} := 0$ and set of used evaluation points $\mathcal{D}_i := \emptyset$.
   (b) If $\mathsf{ctr} = T$, abort by outputting (PROOF, sid, runout_eval).
   (c) Sample point: $z_i \leftarrow\!\!\$\ \mathbb{F} \backslash \mathcal{D}_i$. Update $\mathsf{ctr} := \mathsf{ctr} + 1$ and $\mathcal{D}_i := \mathcal{D}_i \cup \{z_i\}$.
   (d) Compute $y_i = f\left(z_i\right)$ and evaluation proof $\pi_i \leftarrow \mathsf{Open}\left(\mathsf{ck}, \mathsf{cm}, z_i, y_i, \mathsf{aux}\right)$
   (e) Send (QUERY, (sid, $(\mathcal{C}', \mathsf{cm}, z_i, y_i, \pi_i, i)$)) to $\mathcal{G}_{\mathrm{RO}}$. Upon receiving v from $\mathcal{G}_{\mathrm{RO}}$, if the first b bits of v are not 0^b, go to step 8b. Otherwise, store z_i, y_i, and π_i in $\mathbf{z}, \mathbf{y}$, and $\boldsymbol{\pi}_{\mathrm{PCS}}$, respectively.
9. Output (PROOF, sid, π), where $\pi := (\pi', \mathsf{cm}, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}_{\mathrm{PCS}})$.

– **Verification:** Upon receiving input (VERIFY, sid, $\mathcal{C}, \pi$), P_i does:

1. Send (QUERY, (sid, x, genparamsproof)) to $\mathcal{G}_{\mathrm{RO}}$ receiving back pp.
2. Send (QUERY, (sid, x, genparamspc)) to $\mathcal{G}_{\mathrm{RO}}$ receiving back ck.
3. Parse $\pi = (\pi', \mathsf{cm}, \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}_{\mathrm{PCS}})$. Derive the witness size n from the description of $\mathcal{C}$. Compute ℓ and d as Proof would and if $d > D$ abort by outputting (VERIFICATION, sid, 0).
4. Define the statement x' as the **Proof** step would.
5. Parse $\mathbf{z} = (z_i)_{i\in[r]}, \mathbf{y} = (y_i)_{i\in[r]}$, and $\pi_{\mathrm{PCS}} = (\pi_i)_{i\in[r]}$.
6. Output (VERIFICATION, sid,1) if all of the following checks pass, otherwise output (VERIFICATION, sid, 0):
   (a) $\Pi_{\mathcal{R}}.\mathcal{V}(\mathsf{pp}, x', \pi')$ outputs 1. (Calls to $\mathcal{O}_{\mathsf{Setup}}$ by $\mathcal{V}$ handled like above).
   (b) For all $i \in [r] : 1 = \mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, d, z_i, y_i, \pi_i)$
   (c) For all $i \in [r]$: send (QUERY, (sid, $(\mathcal{C}', \mathsf{cm}, z_i, y_i, \pi_i, i)$)) to $\mathcal{G}_{\mathrm{RO}}$, and the first b bits of the return value v_i are 0^b.

E Supporting Proofs

E.1 Proof of Theorem 4

Correctness. The correctness of $\mathsf{PC}_{\mathsf{VC}}$ follows from the correctness of PP and VC, respectively; this is straightforward by inspection.

Evaluation Binding. We proceed by contradiction. Assume an adversary $\mathcal{A}$ that breaks the evaluation binding of $\mathsf{PC}_{\mathsf{VC}}$ with non-negligible probability ϵ. We demonstrate how to construct an adversary $\mathcal{B}$ that breaks the position binding of VC with the same probability. $\mathcal{B}$ selects the public parameters $\mathsf{pp} \leftarrow \mathsf{PP.Params}(R, m, d)$ of the PP scheme and receives from the challenger a VC commitment key $\mathsf{ck}_{\mathsf{VC}}$ corresponding to the length S in pp. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{ck} = (\mathsf{ck}_{\mathsf{VC}}, \mathsf{pp})$, which, in turn, returns $(\mathsf{cm}, x, y, \pi_x, y', \pi'_x)$ to $\mathcal{B}$, where:

$$\mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y, \pi_x) = 1 \ \wedge\ \mathsf{Ver}(\mathsf{ck}, \mathsf{cm}, x, y', \pi'_x) = 1 \wedge\ y \neq y'.$$

Let $\mathsf{cm} = \mathsf{cm}_{\mathsf{DS}}$, $\pi_x = (v_1, \pi_1, \ldots, v_k, \pi_k)$, and $\pi'_x = (v'_1, \pi'_1, \ldots, v'_k, \pi'_k)$. Then, the above equation can be rewritten as follows:

$$\bigwedge_{j\in[k]} \mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v_j, \pi_j) = 1 \ \wedge \ y = \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v_1, \ldots, v_k)$$
$$\wedge$$
$$\bigwedge_{j\in[k]} \mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v'_j, \pi'_j) = 1 \ \wedge \ y' = \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v'_1, \ldots, v'_k)$$
$$\wedge$$
$$y \neq y'$$

where $(i_1, \ldots, i_k) \leftarrow \mathsf{PP.Lookup}(\mathsf{pp}, x)$.

Consider the following two mutually exclusive cases:

- $(v_1, ..., v_k) = (v'_1, ..., v'_k)$. Then, we obtain

$$y = \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v_1, \ldots, v_k) = \mathsf{PP.Reconstruct}(\mathsf{pp}, x, v'_1, \ldots, v'_k) = y'$$

  which contradicts $y \neq y'$ as defined above.
- $(v_1, ..., v_k) \neq (v'_1, ..., v'_k)$. Then, there exists some $j \in [k]$ such that $v_j \neq v'_j$ and $\mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v_j, \pi_j) = \mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{cm}_{\mathsf{DS}}, i_j, v'_j, \pi'_j) = 1$. In this case, $\mathcal{B}$ returns $(\mathsf{cm}_{\mathsf{DS}}, i_j, v_j, \pi_j, v'_j, \pi'_j)$, which contradicts the position binding property of VC.

This concludes the proof.

Succinctness. We prove the succinctness of $\mathsf{PC}_{\mathsf{VC}}$ for the multivariate case. The proof for univariate polynomials is analogous.

Regarding the commitment size, we observe that cm corresponds to $\mathsf{cm}_{\mathsf{DS}}$, i.e., the VC commitment over the vector DS (data structure) of size S. By combining the succinctness of VC and the efficiency of PP, we have:

$$S \in \mathsf{poly}(\log|R|, d^m) \text{ and } |\mathsf{cm}| = |\mathsf{cm}_{\mathsf{DS}}| \in \mathsf{poly}(\lambda \log|R|, \log(S)). \tag{7}$$

The above implies that the size of cm in PC is $\mathsf{poly}(\lambda \log|R|, \log(d^m))$, where d^m represents the size of the polynomial.

Regarding the opening proof size, we note that π_x consists of k opening proofs $\{\pi_j\}_{j\in[k]}$ and k blocks $\{v_j\}_{j\in[k]}$ of the data structure S, where k is the number of indices generated by $\mathsf{PP.Lookup}$. By combining the succinctness of VC and the efficiency of PP, we obtain:

$$k, |v_j| \in \mathsf{poly}(\log|R|, d, m) \text{ and } |\pi_j| \in \mathsf{poly}(\lambda, \log|R|, \log(S)), \tag{8}$$

where S is as defined in Equation (7), and the bounds on k and v_j are derived from the running time of both $\mathsf{PP.Lookup}$ and $\mathsf{PP.Reconstruct}$ (see Definition 12). By combining these observations, we conclude that $|\pi_x| = k(|\pi_j| + |v_j|) \in \mathsf{poly}(\lambda, \log|R|, d, m)$.

Finally, the running time of Ver in $\mathsf{PC}_{\mathsf{VC}}$ can be decomposed as follows:

- The running time of both $\mathsf{PP.Lookup}$ and $\mathsf{PP.Reconstruct}$, which is bounded by $\mathsf{poly}(\log|R|, d, m)$, as PP satisfies efficiency.

– k times the running time of $\mathsf{VC.Ver}$, where both are bounded by $\mathsf{poly}(\lambda, \log|R|, d, m)$. This follows from the fact that $k \in \mathsf{poly}(\log|R|, d, m)$, as defined in Equation (8), and that VC satisfies succinctness.

This concludes the proof.

Fast Openings. We prove the fast openings of $\mathsf{PC}_{\mathsf{VC}}$ for the multivariate case. The proof for univariate polynomials is analogous.

The running time of Com in $\mathsf{PC}_{\mathsf{VC}}$ can be decomposed as follows:

– The running time of $\mathsf{PP.PreProcess}$, which is bounded by $\mathsf{poly}(\log|R|, d^m)$ since PP is efficient.

– The running time of $\mathsf{VC.Com}$. Since VC satisfies fast openings, its running time is bounded by $\mathsf{poly}(\lambda, \log S) \cdot S$, where S is the size of the data structure generated by $\mathsf{PP.PreProcess}$. Due to the efficiency of PP, we get $S \in \mathsf{poly}(\log|R|, d^m)$.

By combining these observations, we conclude that Com runs in time $\mathsf{poly}(\lambda, \log|R|, d^m)$.

Finally, the running time of Open in $\mathsf{PC}_{\mathsf{VC}}$ can be decomposed as follows:

– The running time of $\mathsf{PP.Lookup}$, which is bounded by $\mathsf{poly}(\log|R|, d, m)$ since PP is efficient.

– k times the running time of $\mathsf{VC.Open}$. Since VC satisfies fast openings, the running time of $\mathsf{VC.Open}$ is bounded by $\mathsf{poly}(\lambda, \log S)$. Given that PP is efficient, we have $S \in \mathsf{poly}(\log|R|, d^m)$ and $k \in \mathsf{poly}(\log|R|, d, m)$, where the latter is bounded by the running time of $\mathsf{PP.Lookup}$.

These bounds imply that Open runs in time $\mathsf{poly}(\lambda, \log|R|, d, m)$.

E.2 Proof of Lemmas 5 and 6

Proof of Lemma 5.

Lemma 5. *If* hidPC *and* subPC *are polynomial commitments satisfying completeness, (weak) evaluation binding and the unique proof property (Definition 31 in the appendix) then* PC^{hyb} *(previous paragraphs; Fig. 2) satisfies the unique proof property.*

The proof is straightforward and we present only an informal outline. Recall that the unique proof property (informally) states that no efficient adversary can produce a polynomial commitment cm, a claimed evaluation output y and two different accepting proofs π and π' for cm and y. The proof follows by contradiction: if we assume that there exists such an adversary for $\mathsf{PC}^{\mathrm{hyb}}$ then we are able to break either the unique proof property or the evaluation binding for at least one of the two underlying schemes hidPC and subPC. More in detail: recall from Fig. 2 that a proof for $\mathsf{PC}^{\mathrm{hyb}}$ consists of a tuple (π_f, π_g, y_g). Now assume an adversary that successfully provides two different tuples $\pi := (\pi_f, \pi_g, y_g)$ and $\pi' := \left(\pi'_f, \pi'_g, y'_g\right)$. There are two possibilities: either we have that *(i)* $y_g = y'_g$, or *(ii)* $y_g \neq y'_g$. If *(i)* holds then we must have $\pi_\omega \neq \pi'_\omega$ (with $\omega \in \{f, g\}$) implying that we are able to produce two distinct proofs for the same point y_g (resp. $y_f := y - y_g$) for the polynomial commitment hidPC (resp. subPC). If *(ii)* then π_g is a valid proof claiming evaluation output y_g and π'_g is a valid proof claiming evaluation output $y'_g \neq y_g$, contradicting the assumption on hidPC satisfying evaluation binding.

Proof of Lemma 6.

Lemma 6. *If* hidPC *satisfies evaluation hiding (Definition 33) with respect to* hidPES *then* PC^{hyb} *in Fig. 2 satisfies evaluation hiding with respect to* PES^{hyb}.

Consider an adversary $\mathcal{A}_\phi^{\mathrm{hyb}} = (\mathcal{A}_{\phi,1}^{\mathrm{hyb}}, \mathcal{A}_{\phi,2}^{\mathrm{hyb}})$ against the ϕ-evaluation game for $\mathsf{PC}^{\mathrm{hyb}}$ and parameters $\lambda, n, r \in \mathbb{N}$ (all other parameters are determined by these; see caption of Fig. 8). We define a series of hybrids. The first hybrid $\mathcal{H}_0$ (Fig. 8) corresponds to the ϕ-hiding game where we encode the vector $\mathbf{w}$ provided by the adversary. We fully expand the encoding step of the polynomial encoding scheme $\mathsf{PES}^{\mathrm{hyb}}$ since this is where the changes will occur between hybrids. The last hybrid $\mathcal{H}_3$ (Fig. 8) corresponds to the same game as $\mathcal{H}_0$ but where we encode the vector of all zeros instead of what is provided by the adversary.

- $\mathcal{H}_0 \approx \mathcal{H}_1$: the difference between these two games is that in $\mathcal{H}_1$ has to do with how we define the polynomial g: instead of producing it as an encoding of sk, we instead encode a dummy string of all zeros. In order to show that $\mathcal{H}_0 \not\approx \mathcal{H}_1$ we can build an adversary $\mathcal{A}_\phi^{\mathrm{hid}}$ (shown in Appendix E.2) against the ϕ-hiding game. This adversary would emulate all the other parts of the execution (the ciphertexts, the commitment and the polynomial commitment proofs for f, etc.) and then use the output of $\mathcal{A}_\phi^{\mathrm{hyb}}$ to identify whether it has received an encoding of sk or $\mathbf{0}$.
- $\mathcal{H}_1 \approx \mathcal{H}_2$: the only difference between these two hybrids is what is actually encrypted in the output of $\mathsf{PES}^{\mathrm{hyb}}.\mathsf{Enc}$ ($\mathbf{w}$ or $\mathbf{0}$). We can rely on semantic security to claim that the difference in the advantage of the adversary is negligible. We construct a reduction against semantic security (Definition 39) so that a noticeable difference in output between the two hybrids corresponds to a noticeable advantage against the semantic security experiment leading to a contradiction. Our reduction to semantic security (Definition 39) is straightforward and at the high level works as follows:
  - the target messages are $(m_0 := \mathbf{w}, m_1 := \mathbf{0})$;
  - the challenger generates a secret key and provides the adversary with an encryption $\mathsf{ct}^\star$ of m_b where $b \leftarrow\!\$\ \{0,1\}$;
  - the adversary constructs g exactly as done in both $\mathcal{H}_1$ and $\mathcal{H}_2$ (as an encoding of $\mathbf{0}$), f as $\mathsf{polyFromCoeffs}(\mathsf{ct}^\star)$, it then proceeds following the rest of the pseudocode in $\mathcal{H}_1$ (and hence $\mathcal{H}_2$).
  - the adversary finally outputs b'.
- $\mathcal{H}_2 \approx \mathcal{H}_3$: here we restore the encoding of sk to generate g; we can argue exactly as we did to show $\mathcal{H}_0 \approx \mathcal{H}_1$.

Since we have shown that $\mathcal{H}_0 \approx \mathcal{H}_3$, we can immediately conclude that the advantage of any PPT adversary against ϕ-hiding would be negligible.

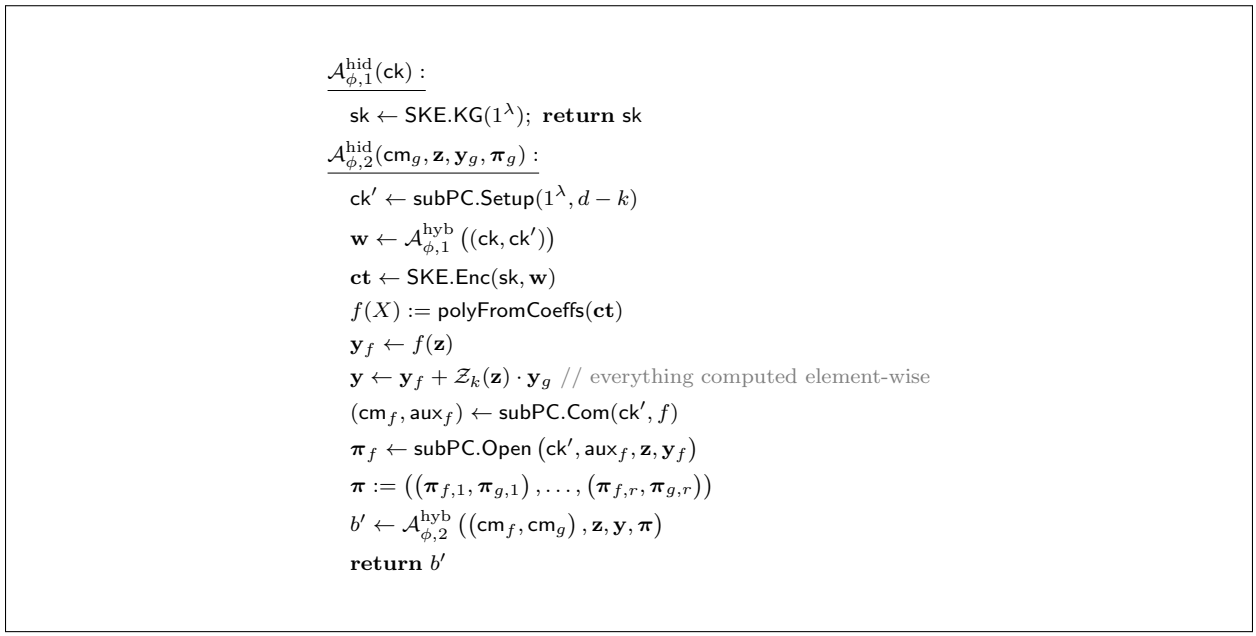

Fig. 7. Reduction to the evaluation hiding of hidPES. We assume $\mathcal{A}^{\mathrm{hid}}_{\phi}$ is stateful. Notation-wise, $F(\mathbf{v})$ denotes the result of applying F to each element of $\mathbf{v}$.

$\underline{\mathcal{H}_0:}$

$\mathsf{ck} \leftarrow \mathsf{hidPC.Setup}(1^\lambda, k)$

$\mathsf{ck}' \leftarrow \mathsf{subPC.Setup}(1^\lambda, d-k)$

$\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,1}\left((\mathsf{ck}, \mathsf{ck}')\right)$

$\mathbf{z} \leftarrow\!\$\ \mathbb{F}^r$

$\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\lambda)$

$\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{w})$

$f(X) := \mathsf{polyFromCoeffs}(\mathbf{ct})$

$\mathbf{y}_f \leftarrow f(\mathbf{z})$

$g \leftarrow \mathsf{hidPES.Enc}(\mathsf{sk}, \kappa, \ell)$

$\mathbf{y}_g \leftarrow g(\mathbf{z})$

$\mathbf{y} \leftarrow \mathbf{y}_f + \mathcal{Z}_k(\mathbf{z}) \cdot \mathbf{y}_g$

$(\mathsf{cm}_f, \mathsf{aux}_f) \leftarrow \mathsf{subPC.Com}(\mathsf{ck}', f)$

$(\mathsf{cm}_g, \mathsf{aux}_g) \leftarrow \mathsf{hidPC.Com}(\mathsf{ck}, g)$

$\boldsymbol{\pi}_f \leftarrow \mathsf{subPC.Open}\left(\mathsf{ck}', \mathsf{aux}_f, \mathbf{z}, \mathbf{y}_f\right)$

$\boldsymbol{\pi}_g \leftarrow \mathsf{hidPC.Open}\left(\mathsf{ck}, \mathsf{aux}_g, \mathbf{z}, \mathbf{y}_g\right)$

$\boldsymbol{\pi} := \left(\left(\boldsymbol{\pi}_{f,1}, \boldsymbol{\pi}_{g,1}\right), \ldots, \left(\boldsymbol{\pi}_{f,r}, \boldsymbol{\pi}_{g,r}\right)\right)$

$b' \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,2}\left(\left(\mathsf{cm}_f, \mathsf{cm}_g\right), \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}\right)$

return b'

$\underline{\mathcal{H}_1:}$

$\mathsf{ck} \leftarrow \mathsf{hidPC.Setup}(1^\lambda, k)$

$\mathsf{ck}' \leftarrow \mathsf{subPC.Setup}(1^\lambda, d-k)$

$\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,1}\left((\mathsf{ck}, \mathsf{ck}')\right)$

$\mathbf{z} \leftarrow\!\$\ \mathbb{F}^r$

$\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\lambda)$

$\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{w})$

$f(X) := \mathsf{polyFromCoeffs}(\mathbf{ct})$

$\mathbf{y}_f \leftarrow f(\mathbf{z})$

$g \leftarrow \mathsf{hidPES.Enc}(\mathbf{0}, \kappa, \ell)$

$\mathbf{y}_g \leftarrow g(\mathbf{z})$

$\mathbf{y} \leftarrow \mathbf{y}_f + \mathcal{Z}_k(\mathbf{z}) \cdot \mathbf{y}_g$

$(\mathsf{cm}_f, \mathsf{aux}_f) \leftarrow \mathsf{subPC.Com}(\mathsf{ck}', f)$

$(\mathsf{cm}_g, \mathsf{aux}_g) \leftarrow \mathsf{hidPC.Com}(\mathsf{ck}, g)$

$\boldsymbol{\pi}_f \leftarrow \mathsf{subPC.Open}\left(\mathsf{ck}', \mathsf{aux}_f, \mathbf{z}, \mathbf{y}_f\right)$

$\boldsymbol{\pi}_g \leftarrow \mathsf{hidPC.Open}\left(\mathsf{ck}, \mathsf{aux}_g, \mathbf{z}, \mathbf{y}_g\right)$

$\boldsymbol{\pi} := \left(\left(\boldsymbol{\pi}_{f,1}, \boldsymbol{\pi}_{g,1}\right), \ldots, \left(\boldsymbol{\pi}_{f,r}, \boldsymbol{\pi}_{g,r}\right)\right)$

$b' \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,2}\left(\left(\mathsf{cm}_f, \mathsf{cm}_g\right), \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}\right)$

return b'

$\underline{\mathcal{H}_2:}$

$\mathsf{ck} \leftarrow \mathsf{hidPC.Setup}(1^\lambda, k)$

$\mathsf{ck}' \leftarrow \mathsf{subPC.Setup}(1^\lambda, d-k)$

$\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,1}\left((\mathsf{ck}, \mathsf{ck}')\right)$

$\mathbf{z} \leftarrow\!\$\ \mathbb{F}^r$

$\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\lambda)$

$\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{0})$

$f(X) := \mathsf{polyFromCoeffs}(\mathbf{ct})$

$\mathbf{y}_f \leftarrow f(\mathbf{z})$

$g \leftarrow \mathsf{hidPES.Enc}(\mathbf{0}, \kappa, \ell)$

$\mathbf{y}_g \leftarrow g(\mathbf{z})$

$\mathbf{y} \leftarrow \mathbf{y}_f + \mathcal{Z}_k(\mathbf{z}) \cdot \mathbf{y}_g$

$(\mathsf{cm}_f, \mathsf{aux}_f) \leftarrow \mathsf{subPC.Com}(\mathsf{ck}', f)$

$(\mathsf{cm}_g, \mathsf{aux}_g) \leftarrow \mathsf{hidPC.Com}(\mathsf{ck}, g)$

$\boldsymbol{\pi}_f \leftarrow \mathsf{subPC.Open}\left(\mathsf{ck}', \mathsf{aux}_f, \mathbf{z}, \mathbf{y}_f\right)$

$\boldsymbol{\pi}_g \leftarrow \mathsf{hidPC.Open}\left(\mathsf{ck}, \mathsf{aux}_g, \mathbf{z}, \mathbf{y}_g\right)$

$\boldsymbol{\pi} := \left(\left(\boldsymbol{\pi}_{f,1}, \boldsymbol{\pi}_{g,1}\right), \ldots, \left(\boldsymbol{\pi}_{f,r}, \boldsymbol{\pi}_{g,r}\right)\right)$

$b' \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,2}\left(\left(\mathsf{cm}_f, \mathsf{cm}_g\right), \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}\right)$

return b'

$\underline{\mathcal{H}_3:}$

$\mathsf{ck} \leftarrow \mathsf{hidPC.Setup}(1^\lambda, k)$

$\mathsf{ck}' \leftarrow \mathsf{subPC.Setup}(1^\lambda, d-k)$

$\mathbb{F}^n \ni \mathbf{w} \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,1}\left((\mathsf{ck}, \mathsf{ck}')\right)$

$\mathbf{z} \leftarrow\!\$\ \mathbb{F}^r$

$\mathsf{sk} \leftarrow \mathsf{SKE.KG}(1^\lambda)$

$\mathbf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, \mathbf{0})$

$f(X) := \mathsf{polyFromCoeffs}(\mathbf{ct})$

$\mathbf{y}_f \leftarrow f(\mathbf{z})$

$g \leftarrow \mathsf{hidPES.Enc}(\mathsf{sk}, \kappa, \ell)$

$\mathbf{y}_g \leftarrow g(\mathbf{z})$

$\mathbf{y} \leftarrow \mathbf{y}_f + \mathcal{Z}_k(\mathbf{z}) \cdot \mathbf{y}_g$

$(\mathsf{cm}_f, \mathsf{aux}_f) \leftarrow \mathsf{subPC.Com}(\mathsf{ck}', f)$

$(\mathsf{cm}_g, \mathsf{aux}_g) \leftarrow \mathsf{hidPC.Com}(\mathsf{ck}, g)$

$\boldsymbol{\pi}_f \leftarrow \mathsf{subPC.Open}\left(\mathsf{ck}', \mathsf{aux}_f, \mathbf{z}, \mathbf{y}_f\right)$

$\boldsymbol{\pi}_g \leftarrow \mathsf{hidPC.Open}\left(\mathsf{ck}, \mathsf{aux}_g, \mathbf{z}, \mathbf{y}_g\right)$

$\boldsymbol{\pi} := \left(\left(\boldsymbol{\pi}_{f,1}, \boldsymbol{\pi}_{g,1}\right), \ldots, \left(\boldsymbol{\pi}_{f,r}, \boldsymbol{\pi}_{g,r}\right)\right)$

$b' \leftarrow \mathcal{A}^{\mathrm{hyb}}_{\phi,2}\left(\left(\mathsf{cm}_f, \mathsf{cm}_g\right), \mathbf{z}, \mathbf{y}, \boldsymbol{\pi}\right)$

return b'

Fig. 8. Hybrids in the proof of evaluation hiding (changes compared to the previous hybrid are hinted in magenta). Hybrids are parametrized by λ, n, r. Above, $\kappa = |\mathsf{sk}|$, $k := \kappa + \mathsf{stretch}(\lambda, \kappa, \ell)$ where $\ell := \phi(\lambda, \kappa, r)$ and $\mathsf{stretch}$ is the stretch function of hidPES; $d := k + |\mathsf{ct}_{\mathbf{w}}|$.

□

E.3 Proof of Lemma 3

Consider an adversary against the unique proof property for polynomial commitments (Definition 31) and let π, π' be the two proofs that it generates. Denote by $\mathbf{v}$ and $\mathbf{v}'$ the values in π and π' respectively as by the notation in the verification algorithm in Section 4. Then there are two cases: (A) $\mathbf{v} \neq \mathbf{v}'$; (B) $\mathbf{v} = \mathbf{v}'$. If case (A) occurs it is easy to see that one can break position binding. This is because the two verification executions will have the same set of indices returned by $\mathsf{PP.Lookup}(\mathsf{pp}, x)$ and therefore there will be an index i for which the adversary is successfully proving two distinct values $v_i \neq v_i'$. Case (B) can directly be reduced to the unique proof property of the underlying vector commitment.

F Why Our "Hybrid" Construction is Necessary in Light of Other Instantiations of the [GKO+23] Compiler

Instantiations in [GKO+23] In [GKO+23], Ganesh et al. discuss simple forms of polynomial encodings (some described in Appendix D) which allows the KZG polynomial commitment [KZG10] to satisfy evaluation-hiding. One of their approaches is based on adding *masking coefficients* to the polynomial whose vector coefficients is $\mathbf{w}$. That is, $\mathsf{Enc}(\mathbf{w})$ returns $f(X) := \sum_{i=1}^{n} w_i X^{i-1} + \sum_{i=1}^{\ell} \rho_i X^{n+i-1}$ where the ρ_i-s are freshly sampled scalars and ℓ is a parameter determining the level of "masking". They also propose a similar approach based on polynomial interpolations, that is, the encoding returns a polynomial obtained as follows: let $\rho_1, \ldots, \rho_\ell$ be some random scalars; encoding then returns the polynomial obtained through interpolation (over a fixed domain) of the vector $(\mathbf{w}||\rho_1, \ldots, \rho_\ell)$. Ganesh et al. show that the KZG polynomial commitment [KZG10] is evaluation-hiding with respect to these two encoding schemes.

The encoding schemes above cannot provide hiding for our construction in Section 4, at least directly and with the analysis from [GKO+23]. The reason is that those techniques, when adding ℓ random coefficients/evaluations, can provide security up to $\ell - 1$ evaluations seen by the adversary (the intuition behind the security of KZG when paired with them can roughly be reduced to this observation). However, our construction, at each opening provides *several* evaluations of polynomials directly related to $\mathbf{w}$; recall, in fact that at each opening, we provide evaluations of polynomials corresponding to some type of CRT encoding of the original polynomial (each of these sub-polynomials is related to the original polynomials evaluated modulo p, where p is a relatively small prime). As a consequence, at each evaluation we are intuitively leaking significantly more than what we leak with an approach like the one based on KZG.

It is at least conceivable that a more sophisticated analysis could show that this type of encodings could yield evaluation hiding if one used an appropriate number of extra coefficient/evaluations. However, this may reveal to be quite complicated (at the very least, it eluded some initial attempts on our side). We leave this as an interesting open problem.

The recent work in [BCC+24] also provides a PC and PES with similar properties. Their constructions cannot unfortunately be used in our setting: their PC verifier runs in linear time and, while their PES *might* be evaluation hiding when paired with our $\mathsf{PC}_{\mathsf{VC}}$, it would require an extremely complex analysis, similar to the one they need to do to show the compatibility of their PES with their Bulletproofs-based PC (in contrast, our hybrid approach is very simple to prove secure). For this reason we do not pursue this path.

Viceversa, our construction, *cannot* provide a simpler solution for their setting: since they require transparent constructions they cannot rely on KZG-like constructions as we do and thus need to bootstrap the [GKO+23] framework with new tools themselves.