LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs

Matteo Campanelli
matteo.campanelli@imdea.org
IMDEA Software Institute, Madrid, Spain

Dario Fiore
dario.fiore@imdea.org
IMDEA Software Institute, Madrid, Spain

Anaïs Querol
anais.querol@imdea.org
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain

ABSTRACT

We study the problem of building SNARKs modularly by linking small specialized "proof gadgets" SNARKs in a lightweight manner. Our motivation is both theoretical and practical. On the theoretical side, modular SNARK designs would be flexible and reusable. In practice, specialized SNARKs have the potential to be more efficient than general-purpose schemes, on which most existing works have focused. If a computation naturally presents different "components" (e.g. one arithmetic circuit and one boolean circuit), a general-purpose scheme would homogenize them to a single representation with a subsequent cost in performance. Through a modular approach one could instead exploit the nuances of a computation and choose the best gadget for each component.

Our contribution is LegoSNARK, a "toolbox" (or framework) for commit-and-prove zkSNARKs (CP-SNARKs) that includes:
1) General composition tools: build new CP-SNARKs from proof gadgets for basic relations *simply*.
2) A "lifting" tool: add commit-and-prove capabilities to a broad class of existing zkSNARKs *efficiently*. This makes them interoperable (linkable) within the same computation. For example, one QAP-based scheme can be used prove one component; another GKR-based scheme can be used to prove another.
3) A collection of succinct proof gadgets for a variety of relations.

Additionally, through our framework and gadgets, we are able to obtain new succinct proof systems. Notably:
– LegoGro16, a commit-and-prove version of Groth16 zkSNARK, that operates over data committed with a classical Pedersen vector commitment, and that achieves a 5000× speedup in proving time.
– LegoUAC, a pairing-based SNARK for arithmetic circuits that has a universal, circuit-independent, CRS, and proving time linear in the number of circuit gates (vs. the recent scheme of Groth et al. (CRYPTO'18) with quadratic CRS and quasilinear proving time).

1 INTRODUCTION

Zero-knowledge proofs (ZKPs), introduced by Goldwasser, Micali and Rackoff [38], let a prover convince a verifier of a statement without revealing more information than its validity. This power of ZKPs—simultaneously providing *integrity* (the prover cannot cheat) and *privacy* (the verifier does not learn any of the prover's secrets)—has found countless applications, including multiparty computation [36], signature schemes [62], public-key encryption [56], and, more recently, blockchain systems [6, 10].

Some zero-knowledge proofs systems—called *succinct* or simply *zkSNARKs*, zero-knowledge Succinct Non-interactive Argument of Knowledge—have short and efficiently verifiable proofs [15, 34, 55]. Succinctness is desirable in general but is especially critical in applications where verifiers would not invest significant computational resources (e.g. if they are unwilling to do it for reasons of scalability and cost, or if they are computationally weak).

Motivation. The last years have seen remarkable progress in the construction of zkSNARKs. Different lines of work (cf. Section 1.2 for a detailed review) have built a variety of schemes that are highly expressive, supporting general computations in the class NP. The general-purpose nature of these schemes makes them very attractive to practitioners. At the same time, this high expressivity comes at a cost in terms of performance. To achieve generality, these constructions abstract specific features of computation by assuming one *single unifying representation* (e.g., boolean or arithmetic circuits, state-machine transitions, RAM computations), and this abstraction is often a source of overhead, for two main reasons.

First, *computation tends to be heterogeneous*, often consisting of several subroutines of different nature. For example it may combine both arithmetic and boolean subroutines, and the single representation assumed by a SNARK may not be the best one for both.

Second, *general-purpose zk-SNARKs may miss opportunities for significant optimizations* by not exploiting the nuances of a computation. In contrast, specialized solutions can gain efficiency by exploiting specific structural properties, but become useless if the computation at hand does not fit these properties. For example, recently proposed variants of the GKR protocol [27, 68] show that it can be highly optimized for parallel computations; at the same time this protocol fails to be succinct if a computation is sequential. As a result, using such a protocol for a computation that includes both a sequential and a parallel subroutine is not an option.

A Modular Approach for zk-SNARKs. In this paper we study an alternative approach for building zkSNARKs that proceeds in a modular "bottom-up" fashion. Whereas most current works build general-purpose schemes using one representation that must be shared by the different computation's subroutines, we instead consider designing a "global" SNARK for a computation C through a (lightweight) linking of "smaller" specialized SNARKs (that we also call *"proof gadgets"*) for the different subroutines composing C. The advantage of this approach is that it inherits all the benefits of modularity, such as reducing complexity (only need to focus on specific, simpler problems), increased flexibility and costs reduction (proof gadgets can be reused, replaced, etc.).

Modularity from Commit-and-Prove SNARKs. To realize this modular approach we rely on the well known *commit-and-prove* (CP) methodology [24, 46]. With a CP scheme one can prove statements of the form "$c_{\mathsf{ck}}(x)$ contains x such that $R(x, w)$" where $c_{\mathsf{ck}}(x)$ is a commitment. To see how the CP capability can be used for modular composition consider the following example of sequential composition in which one wants to prove that $\exists w : z = h(x; w)$, where $h(x; w) := g(f(x; w); w)$. Such a proof can be built by combining two CP systems Π_f and Π_g for its two building blocks, i.e., respectively f and g: the prover creates a commitment $c_{\mathsf{ck}}(y)$ of y, and then uses Π_f (resp. Π_g) to prove that "$c_{\mathsf{ck}}(y)$ contains $y = f(x; w)$ (resp. contains y such that $z = g(y; w)$)".

Challenges of the CP modular composition. The composition idea sketched above implicitly assumes that Π_f and Π_g work on the same commitment $c_{\mathsf{ck}}(y)$. Namely, *in order to be composed, different CP schemes must be compatible with the same commitment scheme* (and commitment key). Essentially we need a sort of *universal commitment scheme* that is as decoupled as possible from the specific argument systems that will operate on it.

We argue that achieving such universality with state-of-the-art zkSNARKs entails major challenges:

(a) Most of the popular zkSNARKs, e.g., [41, 58], are not explicitly commit-and-prove. This limitation can be overcome using a (somewhat folklore) approach in which the SNARK Π additionally proves the correct opening of the commitment, i.e., $R(x, w) \wedge$ "$c_{\mathsf{ck}}(x)$ opens to x". This approach has two main drawbacks: *(i)* Π must be expressive enough to express the commitment verification in its language, but in our vision Π is a SNARK for a specialized task and may not have this capability; *(ii)* even if Π were expressive enough (e.g., supports arbitrary circuits), encoding commitment verification incurs significant overheads.[1]

(b) Some existing SNARKs have commit-and-prove capabilities [28, 40, 51, 65]. Yet, each of these schemes uses its own specific commitment scheme. In some cases [28] the commitment keys are *relation-dependent*, which means commitments cannot be generated before fixing one or multiple relations.[2] In the other cases, despite being relation-independent, commitment keys have a very specific structure that may not fit other proof systems. In summary, a main limitation of existing commit-and-prove SNARKs is their incompatibility, between them and with other potential candidates to be developed.

1.1 Our Results

LegoSNARK Framework. We present LegoSNARK, a framework for commit-and-prove zkSNARKs (CP-SNARKs) that includes:

- *Definitions* that formalize CP-SNARKs and their variants.
- *Composition recipes* that show how to use different CP-SNARKs in a generic and secure way for handling conjunction, disjunction and sequential composition of relations. This composition result enables the use of modularity in designing CP-SNARKs for complex relations out of schemes for simpler relations.
- *A generic construction* to efficiently turn a broad class of zkSNARKs into CP-SNARKs that can be composed together. This class includes several existing schemes such as ones based on quadratic arithmetic programs [28, 41, 58], or zk-vSQL [70, 71]. For this transformation we only need a "minimal" CP-SNARK, $\mathsf{CP}_{\mathsf{link}}$, for proving that two commitments (under different schemes) open to the same value.

[1]For example, we experimentally found that, when handling a Pedersen commitment to a vector of length 2048 with [41], the proving overhead is 428 secs (7 minutes).

[2]This could be mitigated by using universal circuits, paying a (multiplicative) logarithmic overhead in parameters size and prover complexity.

LegoSNARK Gadgets. We populate our framework by constructing new CP-SNARKs for several basic relations, such as:

- $\mathsf{CP}_{\mathsf{link}}$ for proving that two *different* Pedersen-like commitments open to the same vector.[3] Plugging $\mathsf{CP}_{\mathsf{link}}$ in our generic construction solves the challenges (a) and (b) mentioned above and gives us interoperable versions of several existing schemes.
- $\mathsf{CP}_{\mathsf{lin}}$ for proving that a linear relation $\mathbf{F} \cdot \vec{u} = \vec{x}$ holds for a committed vector $\vec{u}$, a public matrix $\mathbf{F}$ and public vector $\vec{x}$.
- $\mathsf{CP}_{\mathsf{had}}$ for proving a self-permutation, i.e. that a vector $\vec{u}_0$ is the Hadamard product of $\vec{u}_1$ and $\vec{u}_2$, when all the three vectors are committed.
- $\mathsf{CP}_{\mathsf{sfprm}}$ for proving that $y_i = y_{\phi(i)}$ for a public permutation ϕ and a committed vector $\vec{y}$.

All the aforementioned schemes have succinct proofs and work for Pedersen-like commitments in bilinear groups. This means that by using our generic construction with $\mathsf{CP}_{\mathsf{link}}$ they can be turned to support the same commitment and then be composed.

LegoSNARK Applications and Evaluation. Using our initial set of specialized proof gadgets, our next step is to combine them in order to build new succinct proof systems for different use cases, mentioned below. Our results offer various improvements over the state of the art. We have also implemented some of our solutions to test their concrete performance.

1) Efficient Commit-Ahead-of-Time. Through our generic construction instantiated with $\mathsf{CP}_{\mathsf{link}}$ we also obtained commit-and-prove versions of popular efficient zkSNARKs, such as Groth's [41], that can prove statements about data committed using the Pedersen scheme for vectors [59], in which bases are random group elements that can be generated without trusted setup. Such commit-and-prove schemes are useful in applications where one needs to commit *before* the SNARK keys for a relation are created, e.g., to post commitments on a blockchain so that one can later prove statements about the committed data. By applying our solution to [41] we obtain a scheme that is 5000× faster than a scheme where the commitment is encoded in the circuit.

2) CP-SNARKs for Parallel Computation. Consider the problem of proving (in zero-knowledge) correctness of a computation that consists of the same subcircuit executed in parallel. The recent Hyrax system [69] is suitably designed for and shows good performances on this type of circuit. It requires, however, an additional verification cost whenever the repeated subcircuits share (non-deterministic) inputs, which is common. The verifier thus pays an additional factor linear in the total width of the circuit. Using our LegoSNARK framework we show how to build a new CP-SNARK based on Hyrax that avoids this problem. The idea is that parallel computation on joint inputs can be expressed as the combination of a fully parallel computation (after inputs were appropriately duplicated) and a permutation check to ensure that inputs have been duplicated correctly. We build this by combining our $\mathsf{CP}_{\mathsf{lin}}$ gadget with a version of Hyrax modified to work with the polynomial commitment of zk-vSQL [71].

3) CP-SNARKs for Arithmetic Circuits. We give two main constructions of CP-SNARKs for arithmetic circuit (AC) satisfiability.

[3]By "Pedersen-like" we mean schemes where the verification algorithm is the same as in Pedersen scheme [59] for vectors (but the bases can have a different distribution).

Scheme	Uni	KG time	Prove time	Ver. time	$\lvert \mathbf{crs} \rvert$	$\lvert \pi \rvert$
[41, 58]	–	$n+m$	$m+n\log n$	$\lvert x \rvert$	$n+m$	$O(1)$
LegoAC1	–	$n+m$	$n\log n$	$\lvert x \rvert$	n	$O(1)$
LegoAC2	–	$n+m$	n	$\lvert x \rvert+\log n$	n	$\log n$
[42]	✓	n^2	$m+n\log n$	$\lvert x \rvert$	n^2	$O(1)$
LegoUAC	✓	N^*	N	$\lvert x \rvert+\log N$	N^*	$\log N$

Table 1: Comparing pairing-based zkSNARKs for arithmetic circuits with m wires and N gates, of which n are multiplication gates, n_a (resp. n_c) are addition (resp. multiplication-by-constant) gates, and $N^* = \max(n, n_a, n_c)$. Numbers in the table are in $O(\cdot)$ notation.

Table 1 summarizes a theoretical comparison with other schemes in the literature (selected among the ones with similar succinctness).

Our first scheme, LegoAC, relies on an encoding of AC based on Hadamard products and linear constraints from [18] and can be built from $\mathsf{CP_{lin}}$ and $\mathsf{CP_{had}}$ gadgets. We evaluate two instantiations:

- LegoAC1—from our $\mathsf{CP_{lin}}$ and a $\mathsf{CP_{had}}$ from [51]—is secure in the generic group model (GGM), enjoys constant-size proofs, and has a $\log n$ factor in proving time (similar to [41, 58]);
- LegoAC2—from our $\mathsf{CP_{lin}}$ and $\mathsf{CP_{had}}$ gadgets—is secure in the GGM and random oracle model, it has $\log n$-size proofs but only *linear* proving time.

The second CP-SNARK, LegoUAC, builds on an encoding of AC based on Hadamard products, additions and permutation from [19, 39] and can be built from our $\mathsf{CP_{had}}$ and $\mathsf{CP_{sfprm}}$ gadgets.[4] The main novelty of LegoUAC is to admit a *universal, circuit-independent* CRS, in the "specialization" model of [42] where the universal CRS can be specialized to a circuit C with a deterministic algorithm. LegoUAC's CRS has $O(N)$ size where N is an upper bound on the number of gates of the circuits; in contrast, the CRS has quadratic size in the recent scheme in [42]. Our LegoUAC also improves on the approach applying an efficient system, say [41], on a universal circuit [43, 64], which would incur at least a logarithmic multiplicative factor in circuit size.

1.2 Related Work

The idea of combining two different NIZKs to improve efficiency when handling heterogeneous computations has been considered by Chase et al. [26] and more recently by Agrawal et al. [5]. In [5], they propose combining the Pinocchio scheme [58] with Sigma-protocol-based NIZKs and show an efficient construction for computations that combines algebraic relations in a cryptographic group and arbitrary computation. Their scheme benefits from composing these different proof systems and obtains interesting performance results. The solution in [5] is tailored to two specific proof systems and their combination methodology does not always preserve succinctness. In contrast, the techniques we study are *general*, apply to a variety of existing proof systems and preserve succinctness (they compose succinct schemes into succinct schemes).

Succinct ZK Proofs. In the past years several research lines have built a variety of zk-SNARKs for general NP statements. Here we provide an overview of each line, especially focusing on their differences in performance.

A major research line is the one based on the seminal paper of Gennaro et al. [33] who proposed a pairing-based SNARK based on the NP-complete language of quadratic span/arithmetic programs. This approach improves on previous approaches by Ishai et al. [44], Groth [40] and Lipmaa [50], and is the basis of several works such as [8, 11, 13, 22, 28, 31, 41, 42, 49, 58, 67]. The zkSNARKs in this family enjoy constant-size proofs and fast verification, the latter depending only linearly on the statement size; on the downside, they feature large overheads at proving time, costly (although amortizable) preprocessing and security properties based on non-standard non-falsifiable assumptions.

[4]Additions are handled for free if the commitment is linearly homomorphic.

A second research line builds on the MPC-in-the-head approach of Ishai et al. [45] to construct a ZK argument from an MPC protocol. The first scheme that refined and experimented this approach is ZKBoo [35], then improved in [25]; a more recent work in this line is Ligero [7]. These schemes do not need trusted setup and show excellent proving performances on Boolean circuits, since they rely only on symmetric-key cryptographic primitives. On the downside their proofs are not fully succinct, being linear in the circuit size $|C|$ in [35], and $\tilde{O}(\sqrt{|C|})$ in [7].

The works [69–71] stem from the interactive proof techniques for low-depth circuits pioneered in Goldwasser et al. [37] and later refined in [27, 63, 66]. The resulting succinct ZK arguments are made non-interactive in the random oracle model. These schemes offer good proving performance and use asymptotically fewer cryptographic operations than those from the MPC-in-the-head family; they can be instantiated without [69] (or with a circuit-independent [71]) trusted setup. On the other hand their proof size and verification time depend on the structure of the circuit at hand, notably on the depth and in some cases on the width.

Building on the work of Groth [39], two recent proposals [18, 23] give ZK arguments for arithmetic circuit satisfiability that can be instantiated without trusted setup. The first scheme of Bootle et al. [18] has proofs of size $O(\sqrt{M})$ where M is the number of multiplication gates in the circuit, while their second scheme (improved in [23]) has proofs of size $O(\log M)$ but has a linear time verifier.

Compared to the results from the latter three research lines we described, our instantiations have the disadvantage of needing a trusted setup[5], although in some cases this is universal and thus reusable. In terms of performances, however, our results are more succinct, both in terms of proof size and verifier time.

A recent line of work [9] builds on the seminal works of Kilian [47] and Micali [54], and generalizations of PCPs (IOPs) [12, 60] in order to construct systems (dubbed zkSTARKs) that are general-purpose (capturing very general computations that can be expressed as state-machine transitions), do not require trusted setup and offer good timings for prover and verifier. On the downside, the memory costs for the prover are still high and their security relies on a non-standard conjecture about Reed-Solomon codes.

2 PRELIMINARIES

We use $\lambda \in \mathbb{N}$ to denote the security parameter, and 1^λ to denote its unary representation. Throughout the paper we assume that all the algorithms of the cryptographic schemes take as input 1^λ, and thus we omit it from the list of inputs. For a distribution D, we denote by $x \leftarrow D$ the fact that x is being sampled according to D. We remind

[5]We stress that only our concrete instantiations require a trusted setup—our general composition framework does not.

the reader that an ensemble $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of probability distributions over a family of domains $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda \in \mathbb{N}}$ are statistically indistinguishable (denoted by $\mathcal{D} \approx_s \mathcal{D}'$) if $\frac{1}{2}\sum_x |D_\lambda(x) - D'_\lambda(x)| < \mathsf{negl}(\lambda)$. If $\mathcal{A} = \{\mathcal{A}_\lambda\}$ is a (possibly non-uniform) family of circuits and $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ is an ensemble, then we denote by $\mathcal{A}(\mathcal{D})$ the ensemble of the outputs of $\mathcal{A}_\lambda(x)$ when $x \leftarrow D_\lambda$. We say two ensembles $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}' = \{D'_\lambda\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable (denoted by $\mathcal{D} \approx_c \mathcal{D}'$) if for every non-uniform polynomial time distinguisher $\mathcal{A}$ we have $\mathcal{A}(\mathcal{D}) \approx_s \mathcal{A}(\mathcal{D}')$.

We denote by $[n]$ the set of integers $\{1, \dots, n\}$ and by $[: n]$ the set $\{0, 1, \dots, n-1\}$. We use $(u_j)_{j \in [\ell]}$ to denote the tuple of elements $(u_1, \dots, u_\ell)$.

Relations. Let $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of polynomial-time decidable relations R on pairs (x, w) where $x \in \mathcal{D}_x$ is called the *statement* or *input*, and $w \in \mathcal{D}_w$ the *witness*. We write $R(x, w) = 1$ to denote that R holds on (x, w), else we write $R(x, w) = 0$. When discussing schemes that prove statements on committed values we assume that $\mathcal{D}_w$ can be split in two subdomains $\mathcal{D}_u \times \mathcal{D}_\omega$. Finally we sometimes use an even finer grained specification of $\mathcal{D}_u$ assuming we can split it over ℓ arbitrary domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity ℓ. In our security definitions we assume relations to be generated by a *relation generator* $\mathcal{RG}(1^\lambda)$ that, on input the security parameter 1^λ, outputs R together with some side information, an auxiliary input aux_R, that is given to the adversary. We define $\mathcal{RG}_\lambda$ as the set of all relations that can be returned by $\mathcal{RG}(1^\lambda)$.

2.1 Zero-Knowledge SNARKs

We recall the definition of (pre-processing) zkSNARKs [14, 15].

Definition 2.1 (SNARK). A succinct non-interactive argument of knowledge for $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of three algorithms $\Pi =$ (KeyGen, Prove, VerProof) that work as follows and satisfy the notions of *completeness*, *succinctness* and *knowledge soundness* defined below[6]. If Π also satisfies *zero-knowledge* we call it a zkSNARK.

- $\mathsf{KeyGen}(R \in \mathcal{R}_\lambda) \to \mathsf{crs} = (\mathsf{ek}, \mathsf{vk})$: outputs a common reference string consisting of an evaluation and a verification key.
- $\mathsf{Prove}(\mathsf{ek}, x, w) \to \pi$: returns proof that $R(x, w)$ holds.
- $\mathsf{VerProof}(\mathsf{vk}, x, \pi) \to b \in \{0 = \mathit{reject}, 1 = \mathit{accept}\}$

Completeness. For any pair (x, w) satisfying the relation, the verifier always accepts the corresponding proof.

Succinctness. Π is said *succinct* if the running time of VerProof is $\mathsf{poly}(\lambda)(\lambda + |x| + \log|w|)$ and the proof size is $\mathsf{poly}(\lambda)(\lambda + \log|w|)$.

Knowledge Soundness. Π is $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$ if there exist benign relation generator $\mathcal{RG}$ and auxiliary input generator $\mathcal{Z}$ such that for every non-uniform adversary $\mathcal{A}$ there is a non-uniform extractor $\mathcal{E}$ winning the following game with negligible probability:

```
$\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \to b$
$(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$ ; $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R)$
$\mathsf{aux}_Z \leftarrow \mathcal{Z}(R, \mathsf{aux}_R, \mathsf{crs})$ ; $(x, \pi) \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$
$w \leftarrow \mathcal{E}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ ; $b = \mathsf{VerProof}(\mathsf{vk}, x, \pi) \wedge \neg R(x, w)$
```

[6]Formal definitions in Appendix A.2

Composable Zero-Knowledge. Π is $\mathsf{CZK}(\mathcal{RG})$ if there exists a pair of simulators $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ such that any adversary cannot tell if it receives honestly generated $\mathsf{crs} \leftarrow \mathsf{KeyGen}(R)$ and proofs $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w)$, or a simulated $(\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R)$ and simulated proofs $\pi \leftarrow \mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td_k}, x)$.

zkSNARKs with specializable universal CRS. In [42] Groth et al. introduce a variant of the SNARK notion where Π works for universal relations, the keys output by KeyGen work for the whole family $\mathcal{R}_\lambda$ and can be used to prove and verify statements about any $R \in \mathcal{R}_\lambda$. This is convenient because the cost of setup can be amortized. In a nutshell, this notion formalizes the idea that key generation for R can be seen as the sequential combination of two steps: a first probabilistic algorithm that generates a CRS for the universal relation, and a second *deterministic* algorithm, Derive, that specializes this universal CRS into one for a specific R.

Commitments. We recall non-interactive commitment schemes.

Definition 2.2. A commitment scheme is a tuple of algorithms Com = (Setup, Commit, VerCommit) that work as follows and satisfy the notions of *correctness*, *binding* and *hiding*. Formal definitions of these properties are recalled in Appendix A.1.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ takes the security parameter and outputs a commitment key ck. This key includes descriptions of the input space $\mathcal{D}$, commitment space C and opening space O.
- $\mathsf{Commit}(\mathsf{ck}, u) \to (c, o)$ takes the commitment key ck and a value $u \in \mathcal{D}$, and outputs a commitment c and an opening o.
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \to b$ takes as input a commitment c, a value u and an opening o, and accepts ($b = 1$) or rejects ($b = 0$).

3 BUILDING THE LEGOSNARK FRAMEWORK

3.1 Commit and Prove SNARKs

In a nutshell, a *commit-and-prove SNARK* (CP-SNARK) is a SNARK that can prove knowledge of (x, w) such that $R(x, w)$ holds with respect to a witness $w = (u, \omega)$ such that u opens a commitment c_u. Our formal definitions below essentially add some syntactic sugar to this idea in order to explicitly handle relations in which the input domain $\mathcal{D}_u$ is more fine grained and splits over ℓ subdomains. For a reason that will shortly become clear, we call these subdomains *commitment slots*. This splitting is in most of the cases natural (e.g., if u is a binary string, one can think of $u := (u_1, \dots, u_\ell)$ for suitable substrings), and it is crucial to exploit the compositional power of CP-SNARKs, as we show in Section 3.2. We assume that the description of the splitting is part of R's description.

Definition 3.1 (CP-SNARKs). Let $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of relations R over $\mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_\omega$ such that $\mathcal{D}_u$ splits over ℓ arbitrary domains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity parameter $\ell \geq 1$. Let Com = (Setup, Commit, VerCommit) be a commitment scheme (as per Definition 2.2) whose input space $\mathcal{D}$ is such that $\mathcal{D}_i \subset \mathcal{D}$ for all $i \in [\ell]$. A commit and prove zkSNARK for Com and $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ is a zkSNARK for a family of relations $\{\mathcal{R}^{\mathsf{Com}}_\lambda\}_{\lambda \in \mathbb{N}}$ such that:

- every $\mathbf{R} \in \mathcal{R}^{\mathsf{Com}}$ is represented by a pair (ck, R) where $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$ and $R \in \mathcal{R}_\lambda$;
- $\mathbf{R}$ is over pairs $(\mathbf{x}, \mathbf{w})$ where the statement is $\mathbf{x} := (x, (c_j)_{j \in [\ell]}) \in \mathcal{D}_x \times C^\ell$, the witness is $\mathbf{w} := ((u_j)_{j \in [\ell]}, (o_j)_{j \in [\ell]}, \omega) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times O^\ell \times \mathcal{D}_\omega$, and the relation $\mathbf{R}$ holds iff

$$\bigwedge_{j\in[\ell]} \mathsf{VerCommit}(\mathsf{ck}, c_j, u_j, o_j) = 1 \wedge R(x, (u_j)_{j\in[\ell]}, \omega) = 1$$

Furthermore, when we say that CP is knowledge-sound for a relation generator $\mathcal{RG}$ and auxiliary input generator $\mathcal{Z}$ (denoted $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$, for short) we mean it is a knowledge-sound SNARK for the relation generator $\mathcal{RG}_{\mathsf{Com}}(1^\lambda)$ that runs $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and $(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$, and returns $((\mathsf{ck}, R), \mathsf{aux}_R)$.

We denote a CP-SNARK as a tuple of algorithms CP = (KeyGen, Prove, VerProof). For ease of exposition, in our constructions we adopt a more explicit syntax for CP's algorithms as defined below.

- $\mathsf{KeyGen}(\mathsf{ck}, R) \to \mathsf{crs} := (\mathsf{ek}, \mathsf{vk})$
- $\mathsf{Prove}(\mathsf{ek}, x, (c_j)_{j\in[\ell]}, (u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega) \to \pi$
- $\mathsf{VerProof}(\mathsf{vk}, x, (c_j)_{j\in[\ell]}, \pi) \to b \in \{0, 1\}$

Remark 1 (Comparing with existing definitions). *The authors of the Geppetto scheme [28] define a notion of commit-and-prove SNARKs. Here we highlight the main differences between their definition and ours. First, our commitment key can be generated without fixing a priori a relation (or a set of them , e.g., a multi-QAP). Second, in their model one needs to commit to data using a commitment key corresponding to a specific portion of the input (in their lingo a "bank"), whereas in our model one can just commit to a vector of data, and only at proving time one assigns that data to a specific input slot. Third, in out notion the commitment scheme does not need to be trapdoor. Ours is closer to Lipmaa's [51] (with the exception that again we do not need commitments to be trapdoor), and is in fact a specialization of the SNARK notion when considering specific families of relations that include verifying openings of commitments.*

3.2 Composition Properties of CP-SNARKs

In this section, we formally show how the commit and prove capability can be used to combine different CP-SNARK systems.

Conjunction of relations with shared inputs. Let $\{\mathcal{R}^{(0)}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{R}^{(1)}_\lambda\}_{\lambda\in\mathbb{N}}$ be two families of relations such that, for every $\lambda \in \mathbb{N}$ the input domains $\mathcal{D}^{(0)}_u$ and $\mathcal{D}^{(1)}_u$ of relations $R_0 \in \mathcal{R}^{(0)}_\lambda$ and $R_1 \in \mathcal{R}^{(1)}_\lambda$ respectively can split as follows: $\mathcal{D}^{(0)}_u := \mathcal{D}_0 \times \mathcal{D}_2$ and $\mathcal{D}^{(1)}_u := \mathcal{D}_1 \times \mathcal{D}'_2$ with $\mathcal{D}_2 = \mathcal{D}'_2$.[7] In other words we require these relations to share a commitment slot that we indeed call the *shared slot*. Given the above relation families, we define $\{\mathcal{R}^\wedge_\lambda\}_{\lambda\in\mathbb{N}}$ as the family of relations where for every $\lambda \in \mathbb{N}, \mathcal{R}^\wedge_\lambda = \{R^\wedge_{R_0,R_1} : R_0 \in \mathcal{R}^{(0)}_\lambda, R_1 \in \mathcal{R}^{(1)}_\lambda\}$ and $R^\wedge_{R_0,R_1}(x_0, x_1, u_0, u_1, u_2, (w_0, w_1))$ is defined as the conjunction $R_0(x_0, u_0, u_2, w_0) \wedge R_1(x_1, u_1, u_2, w_1)$.

Let Com be a commitment scheme, for $b \in \{0, 1\}$ let CP_b be a CP-SNARK for Com and $\{\mathcal{R}^{(b)}_\lambda\}_{\lambda\in\mathbb{N}}$. From these, we show how to generically build a CP-SNARK $\mathsf{CP}^\wedge$ for Com and $\{\mathcal{R}^\wedge_\lambda\}_{\lambda\in\mathbb{N}}$. The idea is simple: from the definition of $R^\wedge_{R_0,R_1}$ it is enough to use CP_b to prove and verify the two statements $(x_b, u_b, u_2, w_b)_{b\in\{0,1\}}$ using the same commitment to u_2. Our transformation also guarantees that if both CP_0 and CP_1 are CP-SNARKs with specializable universal CRS, then so is the resulting $\mathsf{CP}^\wedge$. This composition result is formalized in the following theorem. Its proof and a full description of the scheme $\mathsf{CP}^\wedge$ are in Appendix B. The main idea behind soundness is that, in order for an adversary to break $\mathsf{CP}^\wedge$, it must break either one of the two underlying schemes, $\mathsf{CP}_0, \mathsf{CP}_1$, or the binding of the commitment scheme.

[7]Note such a splitting is rather general, as $\mathcal{D}_2$ and $\mathcal{D}'_2$, or $\mathcal{D}_0$, or $\mathcal{D}_1$ may be empty.

Theorem 3.2. *If* Com *is a computationally binding commitment and, for $b = 0, 1$, CP_b is a zkCP-SNARK for* Com *and relation family $\{\mathcal{R}^{(b)}_\lambda\}_{\lambda\in\mathbb{N}}$, then there is a zkCP-SNARK* $\mathsf{CP}^\wedge$ *for* Com *and $\{\mathcal{R}^\wedge_\lambda\}_{\lambda\in\mathbb{N}}$.*

Functions composition. A CP-SNARK for conjunction of relations can be easily used for proving correctness of *composed functions*, e.g., proving that $\exists(y, w) : z = f(x, y, w)$, where $f(x, y, w) := h(g(x, w), y)$. Indeed, let $R_h(x', y, z) = 1$ iff $\exists(x', y) : h(x', y) = z$, and $R_g(x, x') = 1$ iff $\exists(x', w) : g(x, w) = x'$, then $\exists\,(y, w) : z = f(x, y, w)$ can be expressed as $R_h(x', y, z) \wedge R_g(x, x')$.

Disjunction of relations with shared inputs. We can reduce the case of OR composition to the conjunction construction above. For this we assume relations are defined over elements of a ring. For a relation $R(u)$ denote by $\hat{R}(u, t)$ the relation such that $\hat{R}(u, 0) = 1$ iff $R(u) = 1$ and $\hat{R}(u, t) = 1$ iff $R(u) = 0$ when $t \neq 0$. We can now express the disjunction of $R_0(u_0), R_1(u_1)$ as $R^\vee_{R_0,R_1}(u_0, u_1, t_0, t_1) := \hat{R}_0(u_0, t_0) \wedge \hat{R}_1(u_1, t_1) \wedge t_0 t_1 = 0$. For this approach to work we need the proof systems for the two relations R_0, R_1 to support their modified version $\hat{R}_0, \hat{R}_1$, which is the case for proof systems supporting general arithmetic or boolean circuits. Finally, we need a simple efficient proof system for the relation $R_{\mathsf{mul}}(t_0, t_1) = 1$ iff $t_0 t_1 = 0$, where both t_0 and t_1 are committed in two different slots.

Composing more than two relations. We can apply Theorem 3.2 several times to build CP-SNARKs for the composition of different relations (or even the same relation with itself). Succinctness degrades linearly with the number of relations involved into a statement and is preserved if one composes a constant or logarithmic number of relations. This is arguably the case when we deal with heterogeneous computations.

3.3 Commit-Carrying SNARKs

In this section we define a variant of SNARKs that lies in between standard SNARKs and CP-SNARKs. We call these schemes *SNARKs with commit-carrying proofs* (or commit-carrying SNARKs, cc-SNARKs for short). In a nutshell, a cc-SNARK is like a SNARK in which the proof contains a commitment to the portion u of the witness. Formalizing this idea requires to make explicit the commitment scheme associated to the SNARK, as well as the commitment key that is part of the common reference string. In the next section we discuss how many of the existing SNARK constructions satisfy this property. Later, in Section 3.5 we show that cc-SNARKs can be lifted to become full fledged, composable, CP-SNARKs. Taking these two results together allows us to compose several existing SNARKs. We define commit-carrying SNARKs as follows:

Definition 3.3 (cc-SNARK). A *commit-carrying zkSNARKs* for $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of algorithms $\mathsf{cc}\Pi = (\mathsf{KeyGen}(R) \to (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}), \mathsf{Prove}(\mathsf{ek}, x, w := (u, \omega)) \to (c, \pi; o), \mathsf{VerProof}(\mathsf{vk}, x, c, \pi) \to \{0, 1\}, \mathsf{VerCommit}(\mathsf{ck}, c, u, o) \to \{0, 1\})$ that satisfy *completeness*, *succinctness*, *zero knowledge* and *knowledge soundness*. In addition, the commitment underlying $\mathsf{cc}\Pi$ must satisfy the usual *binding*. Formal definitions can be found in Appendix A.4. Below we give the knowledge-soundness experiment that is the main difference with that of SNARKs. Essentially the difference is that in cc-SNARKs we assume the extractor outputs the opening of the commitment returned along with the proof.

$\mathsf{Game}^{\mathsf{ccKSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \to b \in \{0,1\}$

$(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$; $\mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R)$

$\mathsf{aux}_Z \leftarrow \mathcal{Z}(R, \mathsf{aux}_R, \mathsf{crs})$; $(x, c, \pi) \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$

$(u, o, \omega) \leftarrow \mathcal{E}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$

$b \leftarrow \mathsf{VerProof}(\mathsf{vk}, x, c, \pi) \wedge \neg(\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \wedge R(x, u, \omega))$

cc-SNARKs can be seen as a less versatile version of CP-SNARKs (clearly, a CP-SNARK implies a cc-SNARK). In a cc-SNARK the commitment key depends on the relation taken by KeyGen, and a commitment is freshly created by the Prove algorithm and is tied to a single proof; in a CP-SNARK the commitment key is independent of relations and commitments can also be created independently and shared across different proofs. In the literature, there are examples of schemes that lie in between our notions of CP-SNARK and cc-SNARK; this is the case for commit and prove SNARKs in which the commitment key is relation-dependent, e.g., [28, 65].

In our paper we also define a variant of cc-SNARKs with a weaker notion of binding in which for the underlying commitment it can be easy to find some collisions, yet it is hard to find two openings such that one one falsifies and the other one satisfies the relation. In fact, existing QAP-based schemes [13, 41, 58] are not fully binding but can satisfy our weak binding. In Appendix K.5 we prove that [41] is a *weak* cc-SNARK. Worth noting that our generic compiler can also turn weak cc-SNARKs into CP-SNARKs.

3.4 Existing CP-SNARKs and cc-SNARKs

In this section, we provide a summary of existing schemes that can be explained, with no or little modification, under our CP-SNARK and cc-SNARK notions.

Existing CP-SNARKs. The following list is a summary. Details supporting the following claims appear in Appendix K.

- Adaptive Pinocchio [65] is a CP-SNARK for relations $R_Q(x, (u_j)_{j\in[\ell]}, \omega)$ where R_Q is a quadratic arithmetic program (QAP), and the commitment scheme is the extended Pedersen commitment of Groth [40] in which the ith basis of the commitment key is g^{x^i} for a random x.
- The scheme in [51][Section 4] is a CP-SNARK for Hadamard product relations $R_{\mathsf{had}}(\vec{a}, \vec{b}, \vec{c})$ over $\mathbb{Z}_q^{3m}$, i.e. R_{had} holds iff $\forall i \in [m] : a_i \cdot b_i = c_i$. In this case the commitment scheme is a variant of the extended Pedersen scheme where the ith basis of the commitment key is $g^{\ell_i(x)}$ for a random x and ℓ_i being the i-th Lagrange basis polynomial.
- zk-vSQL [71] is a CP-SNARK for relations $R((u_j)_{j\in[\ell]})$ where R is an arithmetic circuit, and the commitment is a polynomial commitment that, we observe (cf. Appendix K), can also be explained as a variant of extended Pedersen.

Existing cc-SNARKs. Geppetto [28] is a commit-and-prove SNARK for QAP relations $R_Q(x, u, \omega)$, with a *relation-dependent* commitment key. This scheme immediately yields a cc-SNARK where VerCommit is also a variant of extended Pedersen.

A New Efficient cc-SNARK for QAPs. We show that the SNARK of [41] can be modified to obtain a cc-SNARK for QAP relations $R_Q(u, \omega)$, where the witness portion committed in a fully binding way can be chosen (see Appendix K.5). Compared to the other cc-SNARKs for QAPs mentioned above, this scheme offers nearly optimal efficiency (essentially due to the fact that we start from [41] whereas [28, 65] build on [58]).

3.5 Bootstrapping our Framework

A key requirement to apply the composition results of the LegoSNARK framework is to start from CP-SNARKs that share the same commitment scheme. In practice this is not always the case (see for example the discussion in the previous section). In this section we propose a solution to this issue by giving a generic compiler for turning a cc-SNARK $\mathsf{cc\Pi}$ for a family of relations $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ into a CP-SNARK CP that supports the same relations and works for a given, global, commitment scheme Com. Incidentally, since a CP-SNARK CP for commitment Com′ is also a cc-SNARK, our compiler can also turn CP into a CP-SNARK for another commitment Com.

As noted in the introduction one could solve this problem via a folklore approach that affects efficiency and requires $\mathsf{cc\Pi}$ to be expressive enough (which may not be true if $\mathsf{cc\Pi}$ supports only specialized tasks, as per our vision). Our compiler uses an alternative approach. The intuition is the following. Consider a relation $R(u, w)$ and denote by $c(u)$ a relation-independent commitment to u. A cc-SNARK $\mathsf{cc\Pi}_R$ could prove $R(u, w)$ by producing $(c_R(u), \pi_R)$; this proof however lacks of any link with the commitment $c(u)$. To solve this, we use a specialized CP-SNARK $\mathsf{CP}_{\mathsf{link}}$ for statements like "$c(u)$ and $c_R(u)$ open to the same value", which can be seen as a commit-and-prove relation about the opening of c_R with respect to a global commitment c. This minimal tool turns any cc-SNARK into a full fledged CP-SNARK. Despite the funny fact that we require a CP-SNARK to create CP-SNARKs, $\mathsf{CP}_{\mathsf{link}}$ can be a simple scheme (as confirmed by our construction in Section 4.1) so it allows us to create efficient instantiations of CP-SNARKs.

Our cc-SNARK-lifting compiler. Let $\mathsf{cc\Pi}$ be a cc-SNARK for a family of relations $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ where, for every λ, $R \in \mathcal{R}_\lambda$ is over tuples in $\mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_\omega$, and $\mathcal{D}_u$ splits over ℓ subdomains $(\mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell)$ for some arity parameter ℓ. Consider the commitment verification algorithm $\mathsf{cc\Pi}.\mathsf{VerCommit}$. For any $\lambda \in \mathbb{N}$ and any $\mathsf{ck}' \in \{\mathsf{cc\Pi}.\mathsf{KeyGen}(R)\}_{R\in\mathcal{R}_\lambda}$, we define the relation R° that has input space $\mathcal{D}^\circ_x = \mathcal{C}'$, and witness space $\mathcal{D}^\circ_\omega = \mathcal{D}^\circ_u \times \mathcal{D}^\circ_\omega$ such that $\mathcal{D}^\circ_u = \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell$ and $\mathcal{D}^\circ_\omega := \mathcal{O}'$, where $\mathcal{C}'$ and $\mathcal{O}'$ are the commitment and opening space of the commitment of $\mathsf{cc\Pi}$. For compactness we represent R° with $(\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$. Then, R° is defined as follows:

$$R^\circ(x^\circ, (u^\circ_j)_{j\in[\ell]}, \omega^\circ) := \mathsf{cc\Pi}.\mathsf{VerCommit}(\mathsf{ck}', x^\circ, (u^\circ_j)_{j\in[\ell]}, \omega^\circ)$$

We remark that, above, $x^\circ \in \mathcal{C}'$ is a commitment for $\mathsf{cc\Pi}.\mathsf{VerCommit}$ and $\omega^\circ \in \mathcal{O}'$ is its opening.

Let $\mathsf{CP}_{\mathsf{link}}$ be a CP-SNARK for Com and a family of relations $\{\mathcal{R}^\circ_\lambda\}_{\lambda\in\mathbb{N}}$ such that for every $\lambda \in \mathbb{N}$ the relation R° defined above is in $\mathcal{R}^\circ_\lambda$. In Figure 1 we describe a CP-SNARK CP for $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ that works by using $\mathsf{cc\Pi}$ and $\mathsf{CP}_{\mathsf{link}}$.

We state the following result about the security of our compiler. A formal statement and a proof appear in Appendix C.

Theorem 3.4. *If* $\mathsf{cc\Pi}$ *is a zk-cc-SNARK (or a weak cc-SNARK) for* $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ *and* $\mathsf{CP}_{\mathsf{link}}$ *is a zk-CP-SNARK for* $\{\mathcal{R}^\circ_\lambda\}_{\lambda\in\mathbb{N}}$*, then the scheme* CP *in Figure 1 is a zk-CP-SNARK for* $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$.

```
$\mathsf{CP.KeyGen}(\mathsf{ck}, R) \to (\mathsf{ek} := (\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ), \mathsf{vk} := (\mathsf{vk}', \mathsf{vk}^\circ))$
    $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}') \leftarrow \mathsf{cc\Pi.KeyGen}(R)$; Build $R^\circ$ from $(\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$
    $(\mathsf{ek}^\circ, \mathsf{vk}^\circ) \leftarrow \mathsf{CP_{link}.KeyGen}(\mathsf{ck}, R^\circ)$
$\mathsf{CP.Prove}(\mathsf{ek}, x, (c_j, u_j, o_j)_{j\in[\ell]}, \omega) \to \pi := (c', \pi^\circ, \pi')$
    $(c', \pi', o') \leftarrow \mathsf{cc\Pi.Prove}(\mathsf{ek}', x, (u_j)_{j\in[\ell]}; \omega)$; $(x^\circ, \omega^\circ) := (c', o')$
    $\pi^\circ \leftarrow \mathsf{CP_{link}.Prove}(\mathsf{ek}^\circ, x^\circ, (c_j)_{j\in[\ell]}, (u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega^\circ)$
$\mathsf{CP.VerProof}(\mathsf{vk}, x, (c_j)_{j\in[\ell]}, \pi) \to \{0, 1\}$
    $\mathsf{CP_{link}.VerProof}(\mathsf{vk}^\circ, c', (c_j)_{j\in[\ell]}, \pi^\circ) \wedge \mathsf{cc\Pi.VerProof}(\mathsf{vk}', x, c', \pi')$
```

Figure 1: Generic Construction of CP **from** $\mathsf{CP_{link}}$ **and** $\mathsf{cc\Pi}$**.**

4 CP-SNARKS FOR PEDERSEN-LIKE COMMITMENTS

In this section we propose two CP-SNARKs that work for any commitment scheme whose verification algorithm is the same as the extended Pedersen commitment (essentially a multiexponentiation). For vectors committed in this way, we show two schemes. Our first scheme (given in Section 4.1) allows to prove that another commitment, with the same verification algorithm but different key, opens to the same vector. This is essentially an efficient realization of the $\mathsf{CP_{link}}$ CP-SNARK needed in our compiler of Section 3.5, and that works for cc-SNARKs whose underlying commitment verification has the same structure as Pedersen. Our second scheme (given in Section 4.2) instead allows one to prove correctness of a linear function of the committed vector (i.e., that $\vec{x} = \mathbf{F}\vec{u}$).

In what follows we start by recalling facts and notation about bilinear groups and Pedersen commitment.

Preliminaries on Bilinear Groups. A *bilinear group generator* $\mathcal{BG}(1^\lambda)$ outputs $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are additive groups of prime order q, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map. In this paper, we consider Type-3 groups where it is assumed there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use bracket notation of [29], i.e., for $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_q$, we write $[x]_s$ to denote $a \cdot g_s \in \mathbb{G}_s$, where g_s is a fixed generator of $\mathbb{G}_s$. From an element $[a]_s \in \mathbb{G}_s$ and a scalar b it is possible to efficiently compute $[ab] \in \mathbb{G}_s$. Also, given elements $[a]_1 \in \mathbb{G}_1$ and $[b]_2 \in \mathbb{G}_2$, one can efficiently compute $[a \cdot b]_T$ by using the pairing $e([a]_1, [b]_2)$, that we compactly denote with $[a]_1 \cdot [b]_2$. Vectors and matrices are denoted in boldface. We use the bracket notation also for matrix operations, i.e., $[\vec{A}]_1 \cdot [\vec{B}]_2 = [\vec{A} \cdot \vec{B}]_T$. For a vector $\vec{a}$ and for $i < j$ we denote by $\vec{a}_{[i,j]}$ its portion $(a_i, \dots a_j)$.

Pedersen Vector Commitment. Let us recall the extended Pedersen commitment scheme for vectors of size n. Here we consider an instantiation on a group $\mathbb{G}_1$.

$\mathsf{Ped.Setup}(1^\lambda) \to \mathsf{ck} := [\vec{h}]_1 \leftarrow \mathbb{G}_1^{n+1}$ from distribution $\mathcal{D}$

$\mathsf{Ped.Commit}([\vec{h}]_1, \vec{w}) \to (c, o) := ((o, \vec{w}^\top) \cdot [\vec{h}]_1, o)$ where $o \leftarrow\!\!\$\, \mathbb{Z}_q$

$\mathsf{Ped.VerCommit}([\vec{h}]_1, c, \vec{w}, o) \to c \overset{?}{=} (o, \vec{w}^\top) \cdot [\vec{h}]_1$

Above $\mathcal{D}$ is a probability distribution over the group elements that allows to argue that the scheme is perfectly hiding and computationally binding. For example, $\mathcal{D}$ may be the uniform distribution, in which case we obtain the classical scheme that is binding under the discrete logarithm assumption, or $\mathcal{D}$ may output powers of random values, e.g., $h_i = s^i$ for an $s \leftarrow\!\!\$\, \mathbb{Z}_q$, that has also been proven computationally binding under a suitable assumption.

In our constructions we only require the commitment scheme to have the same verification algorithm as $\mathsf{Ped.VerCommit}$.

Tool: SNARK for Linear Subspaces. In our CP-SNARK constructions we make use of a SNARK for the linear subspace relation $R_{\mathbf{M}}([\vec{x}]_1, \vec{w}) = [\vec{x}]_1 \overset{?}{=} [\mathbf{M}]_1 \cdot \vec{w} \in \mathbb{G}_1^l$ where $[\mathbf{M}] \in \mathbb{G}_1^{l\times t}, \vec{w} \in \mathbb{Z}_q^t$. Namely, given a fixed public matrix $[\mathbf{M}]_1$ and a public vector $[\vec{x}]_1$, one can prove knowledge of a vector $\vec{w}$ such that $[\vec{x}]_1 = [\mathbf{M}]_1 \cdot \vec{w}$. We denote a SNARK for this family of relations with $\mathsf{ss\Pi}$. A candidate scheme for $\mathsf{ss\Pi}$ is the Kiltz-Wee QA-NIZK scheme Π'_{as} [48] that works in bilinear groups. As described in [48], the security of this scheme requires that $l > t$, which is not satisfied in our setting where matrices have always more columns than rows. This means that, when $\mathbf{M}$ has full rank, $R_{\mathbf{M}}$ is satisfied for any $[\vec{x}]_1$. In fact, what we need is an argument of knowledge for this language. For this, by extending a recent result [30], we show the knowledge soundness of Π'_{as} [48], without the $l > t$ restriction, under the discrete logarithm assumption, in the algebraic group model [32]. We recall the scheme and its security statement in Appendix F. For knowledge soundness, the matrix $[\mathbf{M}]_1$ must be generated using a witness sampleable distribution $\mathcal{D}_{\mathsf{mtx}}$, i.e., there must exist a polynomial time algorithm that samples $\mathbf{M}$ in $\mathbb{Z}_q$ such that $[\mathbf{M}]_1$ has the same distribution as the one sampled with $\mathcal{D}_{\mathsf{mtx}}$. We note that this is satisfied by our use cases where $\mathbf{M}$ includes bases of Pedersen-like commitment schemes.

4.1 CP-SNARK for Pedersen Verification

Let Com be a commitment scheme such that $\mathsf{Com.VerCommit} = \mathsf{Ped.VerCommit}$. We build a CP-SNARK $\mathsf{CP_{link}}$ for Com and for the following class of relations R°. Fixed a security parameter λ (and the group setting for λ as well), R° is over $(\mathcal{D}_x \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{D}_\omega)$, where $\mathcal{D}_x = \mathbb{G}_1$, $\mathcal{D}_\omega = \mathbb{Z}_q$ and $\mathcal{D}_j = \mathbb{Z}_q^{n_j}$ for some n_j such that $\sum_j n_j = m$. R° is parametrized by a commitment key $[\vec{f}]_1 \in \mathbb{G}_1^{m+1}$, and is defined as: $R^\circ(c', (\vec{u}_j)_{j\in[\ell]}, o') = 1 \iff c' \overset{?}{=} (o', \vec{u}_1^\top, \dots, \vec{u}_\ell^\top) \cdot [\vec{f}]_1$. Before describing the construction in full detail, let us present the main ideas.

Let $\mathsf{ck} = [\vec{h}]_1 \in \mathbb{G}_1^{n+1}$ be the key of the global commitment Com. In our $\mathsf{CP_{link}}$ the public inputs of the prover are ℓ commitments $(c_j)_{j\in[\ell]}$ and another commitment c'; the witness is a set of openings $((\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]})$ for commitments $(c_j)_{j\in[\ell]}$, and an opening o' for c'. In particular, the prover must prove that

$$R^\circ_{\mathsf{Ped}}(c', (c_j)_{j\in[\ell]}, (\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, o') = 1 \iff \bigwedge_{j\in[\ell]} c_j = (o_j, \vec{u}_j^\top) \cdot [\vec{h}_{[0,n_j]}]_1 \;\wedge\; c' = (o', \vec{u}_1^\top, \dots, \vec{u}_\ell^\top) \cdot [\vec{f}]_1$$

The description of our scheme $\mathsf{CP_{link}}$ follows:

$\mathsf{CP_{link}.KeyGen}(\mathsf{ck}, R^\circ)$**:** parse $\mathsf{ck} = [\vec{h}]_1 \in \mathbb{G}_1^{n+1}$, and let $R^\circ : \mathbb{G}_1 \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathbb{Z}_q$ be the relation defined above with $\mathsf{ck}' = [\vec{f}]_1 \in \mathbb{G}_1^{m+1}$. Use $[\vec{h}]_1$, $[\vec{f}]_1$ and R° to build a matrix $\mathbf{M}$ as in equation (1). Compute $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{ss\Pi.KeyGen}([\mathbf{M}]_1)$ and return $(\mathsf{ek}, \mathsf{vk})$.

$\mathsf{CP_{link}.Prove}(\mathsf{ek}, c', (c_j)_{j\in[\ell]}, (\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, o')$**:** define $[\vec{x}]_1$ and $\vec{w}$ as in as in equation (1), compute $\pi \leftarrow \mathsf{ss\Pi.Prove}(\mathsf{ek}, [\vec{x}]_1, \vec{w})$ and return π.

$\mathsf{CP}_{\mathsf{link}}.\mathsf{VerProof}(\mathsf{vk}, c', (c_j)_{j\in[\ell]}, \pi)$**:** set $[\vec{x}]_1$ as in (1) and return $\mathsf{ss\Pi}.\mathsf{VerProof}(\mathsf{vk}, [\vec{x}]_1, \pi)$.

The key idea of the construction is that this relation can be expressed as a linear subspace relation $R_{\mathbf{M}}([\vec{x}]_1, \vec{w})$ where $\mathbf{M}, \vec{x}, \vec{w}$ can be defined as follows from the inputs of R°_{Ped}, with $l = \ell + 1$ and $t = m + \ell + 1$:

$$\underbrace{\begin{bmatrix} c_1 \\ \vdots \\ c_\ell \\ c' \end{bmatrix}_1}_{[\vec{x}]_1} = \underbrace{\begin{bmatrix} h_0 & 0 & \dots & 0 & 0 & \vec{h}_{[1,n_1]} & 0 & \dots & 0 \\ 0 & h_0 & \dots & 0 & 0 & 0 & \vec{h}_{[1,n_2]} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_0 & 0 & 0 & 0 & \dots & \vec{h}_{[1,n_\ell]} \\ 0 & 0 & \dots & 0 & f_0 & \vec{f}_{[1,n_1]} & \vec{f}_{[n_1+1,n_2]} & \cdots & \vec{f}_{[n_{\ell-1}+1,n_\ell]} \end{bmatrix}_1}_{[\mathbf{M}]_1} \underbrace{\begin{pmatrix} o_1 \\ \vdots \\ o_\ell \\ o' \\ \vec{u}_1 \\ \vdots \\ \vec{u}_\ell \end{pmatrix}}_{\vec{w}} \quad (1)$$

In Appendix D we state and prove the security of $\mathsf{CP}_{\mathsf{link}}$ based on that of $\mathsf{ss\Pi}$. In Appendix D.2 we show how to extend $\mathsf{CP}_{\mathsf{link}}$ to handle a more general class of relations R°_{pre} that essentially checks that a set of vectors $(\vec{u}_j)_{j\in[\ell]}$ is a *prefix*, of known length, of a vector $\vec{u}'$ committed in c'.

4.2 CP-SNARK for linear properties

In this section we show a CP-SNARK for the relation R^{lin} that checks linear properties of (committed) vectors: for a fixed public matrix $\mathbf{F} \in \mathbb{Z}_q^{N\times m}$, relation $R^{\mathsf{lin}}_{\mathbf{F}}$ over public input $\vec{x} \in \mathbb{Z}_q^N$ and witness $\vec{u} \in \mathbb{Z}_q^m$, with $\vec{u} := (\vec{u}_j)_{j\in[\ell]}$ and $\vec{u}_j \in \mathbb{Z}_q^{n_j}$, holds iff $\vec{x} \overset{?}{=} \mathbf{F}\cdot\vec{u}$.

Our scheme, called $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$, considers each $\vec{u}_j$ to be committed using a commitment scheme Com such that Com.VerCommit = Ped.VerCommit, and whose key is $\mathsf{ck} = [\vec{h}]_1 \in \mathbb{G}_1^{m^*+1}$, with $m^* \geq m$.[8] The idea is to express such a commit-and-prove relation with the linear subspace relation $R_{\mathbf{M}}([\vec{x}']_1, \vec{w}')$ that holds iff $[\vec{x}']_1 = [\mathbf{M}]_1 \cdot \vec{w}'$, where $[\vec{x}']_1 \in \mathbb{G}_1^l$, $[\mathbf{M}]_1 \in \mathbb{G}_1^{l\times t}$ and $\vec{w}' \in \mathbb{Z}_q^t$ can be built from the inputs of $R^{\mathsf{lin}}_{\mathbf{F}}$ as follows (for $l = \ell + N$ and $t = m + \ell$):

$$\underbrace{\begin{bmatrix} c_1 \\ \vdots \\ c_\ell \\ \vec{x} \end{bmatrix}_1}_{[\vec{x}']_1} = \underbrace{\left[\begin{array}{cccc|cccc} h_0 & 0 & \dots & 0 & \vec{h}_{[1,n_1]} & 0 & \dots & 0 \\ 0 & h_0 & \dots & 0 & 0 & \vec{h}_{[1,n_2]} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_0 & 0 & 0 & \dots & \vec{h}_{[1,n_\ell]} \\ \hline \multicolumn{4}{c|}{\vec{0}} & \multicolumn{4}{c}{\mathbf{F}} \end{array}\right]_1}_{[\mathbf{M}]_1} \underbrace{\begin{pmatrix} o_1 \\ \vdots \\ o_\ell \\ \vec{u}_1 \\ \vdots \\ \vec{u}_\ell \end{pmatrix}}_{\vec{w}'} \quad (2)$$

The scheme $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ is quite similar to $\mathsf{CP}_{\mathsf{link}}$ and essentially consists into invoking $\mathsf{ss\Pi}$ to prove that the above subspace relation holds. The full description of $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ appears in Appendix E.

Efficiency. When using $\mathsf{ss\Pi}$ from [48], the prover cost is one multiexponentiation of length $m+\ell$ while the verifier needs $\ell+|\vec{x}|+1$ pairings. If $\vec{x}$ is some fixed value, as in our applications, $|\vec{x}|$ of these pairings either disappear (if $\vec{x} = \vec{0}$) or can be precomputed. Furthermore, it is possible to see that the cost of KeyGen is $O(\ell \cdot t + n_F$ where n_F is the number of nonzero entries of $\mathbf{F}$. Essentially this cost depends on the sparsity of the matrix; this is crucial in our applications where for example $\mathbf{F}$ includes the $\mathbf{W}$ matrices representing the linear constraints of a circuit [18].

[8]While in our description we use the same commitment key for every $\vec{u}_j$, our scheme easily extends to the case where different commitment keys are used.

5 EFFICIENT CP-SNARKS FOR POLYNOMIAL COMMITMENTS

This section shows a collection of zero-knowledge CP-SNARKs for a variety of relations over vectors committed using a specific polynomial commitment scheme from [71], that we call PolyCom.

5.1 Preliminaries and Building Blocks

Polynomial Commitments. We abstract away the VPD primitive of [71] distinguishing between the commitment scheme for multivariate polynomials (that we call PolyCom) and the proof system for committed evaluation at a public point (that we call $\mathsf{CP}_{\mathsf{poly}}$). PolyCom is a *linearly homomorphic extractable trapdoor polynomial commitment* consisting of a tuple of algorithms:[9]

$\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: outputs a commitment key for a class $\mathcal{F} \subset \mathbb{F}[\vec{X}]$.
$\mathsf{ComPoly}(\mathsf{ck}, f) \to (c_f, o_f)$: commits a polynomial $f \in \mathcal{F}$.
$\mathsf{ComVal}(\mathsf{ck}, y) \to (c_y, o_y)$: commits a value $y \in \mathbb{F}$.
$\mathsf{CheckCom}(\mathsf{ck}, c) \to b$: takes a commitment and accepts it or not.
$\mathsf{VerCommit}(\mathsf{ck}, c_f, f, o_f) \to b$: accepts or rejects openings.
$\mathsf{HomEval}(\mathsf{ck}, g : \mathbb{F}^\ell \to \mathbb{F}, (c_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}) \to (c', o')$: computes commitment and opening corresponding to the homomorphic evaluation of a linear function on commitments-openings (c_j, o_j).

Also, denote by $\mathsf{ComPoly}^*$ a version of ComPoly that uses fixed randomness (e.g., 0), and similarly ComVal.

CP-SNARKs for PolyCom. We use the following ZK-CP-SNARKs for PolyCom (whose ek is just ck, and vk := cvk, a subset of ck enough to run CheckCom, ComVal and HomEval):

- $\mathsf{CP}_{\mathsf{eq}}$ for the relation $R_{\mathsf{eq}}(u_1, u_2) := u_1 \overset{?}{=} u_2 : \forall u_j \in \mathbb{F}$
- $\mathsf{CP}_{\mathsf{prd}}$ for the relation $R_{\mathsf{prd}}(u_1, u_2, u_3) := u_3 \overset{?}{=} u_1 \cdot u_2 : \forall u_j \in \mathbb{F}$
- $\mathsf{CP}_{\mathsf{poly}}$ for the relation $R_{\mathsf{poly}}(\vec{x} \in \mathbb{F}^\mu, f \in \mathcal{F}, y \in \mathbb{F}) := y \overset{?}{=} f(\vec{x})$

The first two can be obtained using classical Sigma protocols; $\mathsf{CP}_{\mathsf{poly}}$ is extracted from [71] and shown in Appendix G. We recall that $\mathsf{CP}_{\mathsf{poly}}$ proving time is $O(m)$ (with m the number of nonzero monomials of f), whereas verification time and proof size is $O(\mu)$.

Multilinear Extensions. Given a function $f : \{0,1\}^\mu \to \mathbb{F}$, its unique multilinear extension (MLE) is the (unique) multilinear polynomial $\tilde{f} : \mathbb{F}^\mu \to \mathbb{F}$ such that $f(\vec{b}) = \tilde{f}(\vec{b})$ for all $\vec{b} \in \{0,1\}^\mu$. Such multilinear extension is defined as the following polynomial

$$\tilde{f}(X_1, \dots, X_\mu) = \sum_{\vec{b}\in\{0,1\}^\mu} \chi_{\vec{b}}(X_1, \dots, X_\mu) \cdot f(\vec{b})$$

where $\chi_{\vec{b}}(X_1, \dots, X_\mu) = \prod_{j=1}^{\mu} \chi_{b_j}(X_j)$, $\chi_1(X) = X$ and $\chi_0(X) = 1 - X$. For a vector $\vec{u} \in \mathbb{F}^m$ (for some $m = 2^\mu$), its unique MLE is the MLE $\tilde{u}$ of the function $u : \{0,1\}^\mu \to \mathbb{F}$ such that, for every $0 \leq i \leq m-1$ with $i = \sum_{j=0}^{\mu-1} i_j 2^j$, $u(i_0, \dots, i_{\mu-1}) = u_{i+1}$. Note that by using MLEs one can commit to a vector $\vec{u}$ using PolyCom by committing to its MLE $\tilde{u}$.

[9]See Appendix H for formal definitions related to PolyCom

Let $eq : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}$ be the equality predicate ($eq(a,b) = 1$ iff $a = b$) and let $\tilde{eq}$ be its MLE (which has a closed-form representation that allows evaluation in time $O(\mu)$ [63]). We recall the following lemma from [61] (as restated in [63]):

Lemma 5.1 ([61, Lemma 3.2.1]). *For any polynomial $h : \mathbb{F}^\mu \to \mathbb{F}$ extending $p : \{0,1\}^\mu \to \mathbb{F}$ (i.e., such that $\forall \vec{b} \in \{0,1\}^\mu : h(\vec{b}) = p(\vec{b})$), it holds that $\tilde{p}(\vec{X}) = \sum_{\vec{b}\in\{0,1\}^\mu} \tilde{eq}(\vec{X},\vec{b}) \cdot h(\vec{b})$*

5.2 A CP-SNARK for Sum-Check

The sum-check protocol [52] is an interactive proof that allows a prover to convince a verifier of the validity of a statement of the form $t = \sum_{\vec{b}\in\{0,1\}^\mu} g(\vec{b})$ where $g : \mathbb{F}^\mu \to \mathbb{F}$. The protocol consists of μ rounds, it is public coin, and the running time of the verifier in it is $O(\sum_{i=1}^{\mu} deg_i(g))$ plus the cost of evaluating g once (on a random point).

Here we propose a zero-knowledge variant of the sum-check protocol where both the polynomial g and the target value t are committed and one proves knowledge of these values such that $t = \sum_{\vec{b}\in\{0,1\}^\mu} g(\vec{b})$. Precisely, we work with polynomials g defined as the product of $p+1$ polynomials of the form $g(\vec{S}) = \prod_{i=0}^{p} g_i(\vec{S})$, such that all the g_i's, except g_0, are committed. Namely, we show a CP-SNARK $\mathsf{CP}_{\mathsf{sc}}$ for commitment scheme PolyCom and the relation $R_{\mathsf{sc}}(\vec{x},\vec{u})$, with $\vec{x} \in \mathcal{F}$ and $\vec{u} \in \mathbb{F} \times \mathcal{F}^p$, that is formally defined as:

$$R_{\mathsf{sc}}(g_0, (t, (g_j)_{j\in[p]})) = 1 \iff g(\vec{S}) = \prod_{i=0}^{p} g_i(\vec{S}) \wedge t = \sum_{\vec{b}\in\{0,1\}^\mu} g(\vec{b})$$

Our scheme, dubbed $\mathsf{CP}_{\mathsf{sc}}$, is built as a generalization of the protocol recently proposed in [69, 71] that works for a relation that is the same as the above one except that only t is committed while g is public to the verifier. For the reader familiar with the zero-knowledge sum-check protocol in [71, Construction 2], what we do here is to modify their protocol using the following ideas: whereas in [71] the verifier has access to g and computes a commitment to $g(\vec{s})$ for a random point $\vec{s}$ on its own, in our case the verifier has access to a commitment c_g of g and we let the prover create a commitment to $g(\vec{s})$ and use $\mathsf{CP}_{\mathsf{poly}}$ to prove its correctness with respect to c_g. More precisely, the verifier does not have a commitment to g but rather commitments to the factors of g. Hence our prover proceeds by additionally creating commitments to each $g_i(\vec{s})$, it proves their correct evaluations and then uses $\mathsf{CP}_{\mathsf{prd}}$ to prove that $g(\vec{s}) = \prod_{i=0}^{p} g_i(\vec{s})$ with respect to these commitments. Making these changes results in a protocol that is the same as that in [71] except for the last round from the prover to the verifier. Indeed we can prove the security of our protocol by making a reduction to the one of [71]. We state the following result. For lack of space the full description of the protocol and the proof are in Appendix H.1.

Theorem 5.2. *Assume* PolyCom *is an extractable linearly homomorphic commitment,* $\mathsf{CP}_{\mathsf{poly}}$ *and* $\mathsf{CP}_{\mathsf{prd}}$ *are zkSNARKs for relations* R_{poly} *and* R_{prd} *respectively, and Construction 2 in [71] is a ZK interactive argument for sum-check. Then there is a ZK interactive argument for relation* R_{sc}*. Furthermore, by applying the Fiat-Shamir heuristic we get a zkSNARK in the random oracle model, that we call* $\mathsf{CP}_{\mathsf{sc}}$.

Efficiency. In $\mathsf{CP}_{\mathsf{sc}}$, the verifier needs time $O(\mu)$ plus the time to compute $g_0(\vec{s})$. The prover's costs include the running time in the sum-check protocol and the creation of the $\mathsf{CP}_{\mathsf{poly}}$ proofs. If the g_i are multilinear, $\mathsf{CP}_{\mathsf{poly}}$.Prove's time is $O(2^\mu)$. Also, from [63], if the polynomials g_i allow for evaluation in $O(\mu)$ time or are MLE of vectors, the prover's cost in sum-check can be reduced to $O(2^\mu)$.

5.3 A CP-SNARK for Hadamard Products

In this section we propose a CP-SNARK for PolyCom for the relation R_{had} over $(\mathbb{F}^m)^3$ such that

$$R_{\mathsf{had}}(\vec{u}_0, \vec{u}_1, \vec{u}_2) = 1 \iff \forall i \in [m] : u_{0,i} = u_{1,i} \cdot u_{2,i}$$

Let $m = 2^\mu$ and let $\tilde{u}_j : \mathbb{F}^\mu \to \mathbb{F}$ be the MLE of $\vec{u}_j$. Clearly, the relation holds iff for all $\vec{b} \in \{0,1\}^\mu$ we have $\tilde{u}_0(\vec{b}) = \tilde{u}_1(\vec{b}) \cdot \tilde{u}_2(\vec{b})$. If the relation holds, observe that the polynomial $\tilde{u}_1(\vec{X}) \cdot \tilde{u}_2(\vec{X})$ is an extension of the vector $\vec{u}_0$, but not a multilinear one. From Lemma 5.1 the following equality holds

$$\tilde{u}_0(\vec{X}) = \sum_{\vec{b}\in\{0,1\}^\mu} \tilde{eq}(\vec{X},\vec{b}) \cdot \tilde{u}_1(\vec{b}) \cdot \tilde{u}_2(\vec{b})$$

Without considering zero-knowledge, the main idea of our protocol is that, to check the above equality, the verifier starts by picking a random point $\vec{r} \leftarrow_\$ \mathbb{F}^\mu$, and then the prover uses $\mathsf{CP}_{\mathsf{sc}}$ to show that $t = \tilde{u}_0(\vec{r}) = \sum_{\vec{b}\in\{0,1\}^\mu} g(\vec{b})$, where $g(\vec{S}) = \tilde{eq}(\vec{r},\vec{S}) \cdot \tilde{u}_1(\vec{S}) \cdot \tilde{u}_2(\vec{S})$. Notice indeed that g can be written as the product of three polynomials $g(\vec{S}) := \prod_0^2 g_i(\vec{S})$, of which the first one is public: $g_1(\vec{S}) = \tilde{u}_1(\vec{S})$, $g_2(\vec{S}) = \tilde{u}_2(\vec{S})$ and $g_0(\vec{S}) := \tilde{eq}(\vec{r},\vec{S})$. Finally, the prover also needs to convince the verifier that $t = \tilde{u}_0(\vec{r})$, which is done using a CP-SNARK $\mathsf{CP}_{\mathsf{poly}}$ for proving correctness of polynomial evaluations.

Therefore we build a CP-SNARK $\mathsf{CP}_{\mathsf{had}}$ for R_{had} and PolyCom by using CP-SNARKs $\mathsf{CP}_{\mathsf{poly}}, \mathsf{CP}_{\mathsf{sc}}$ for PolyCom as building blocks. Furthermore, we describe the scheme as a non-interactive one by letting $\vec{r} \leftarrow H((c_j)_{j\in[:3]})$ using the random oracle model for H.

The full description of the scheme is given below.

```
$\mathsf{KeyGen}(\mathsf{ck}) \to (\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_s, \mathsf{ek}_p, H), \mathsf{vk} := (\mathsf{cvk}, \mathsf{vk}_s, \mathsf{vk}_p, H))$
    $(\mathsf{ek}_s, \mathsf{vk}_s) \leftarrow \mathsf{CP}_{\mathsf{sc}}.\mathsf{KeyGen}(\mathsf{ck})$ ; $(\mathsf{ek}_p, \mathsf{vk}_p) \leftarrow \mathsf{CP}_{\mathsf{poly}}.\mathsf{KeyGen}(\mathsf{ck})$
$\mathsf{Prove}(\mathsf{ek}, (c_j)_{j\in[:3]}, (\vec{u}_j)_{j\in[:3]}, (o_j)_{j\in[:3]}) \to \pi := (c_t, \pi_0, \pi_{\mathsf{sc}})$
    $\vec{r} \leftarrow H((c_j)_{j\in[:3]})$ ; $t \leftarrow \tilde{u}_0(\vec{r})$ ; $(c_t, o_t) \leftarrow \mathsf{ComVal}(\mathsf{ck}, t)$
    $\pi_0 \leftarrow \mathsf{CP}_{\mathsf{poly}}.\mathsf{Prove}(\mathsf{ek}_p, \vec{r}, (c_0, c_t), (\tilde{u}_0, t), (o_0, o_t))$
    $\pi_{\mathsf{sc}} \leftarrow \mathsf{CP}_{\mathsf{sc}}.\mathsf{Prove}(\mathsf{ek}_s, \tilde{eq}(\vec{r},\vec{S}), (c_t, c_1, c_2), (t, o_t, \tilde{u}_1, o_1, \tilde{u}_2, o_2))$
$\mathsf{VerProof}(\mathsf{vk}, (c_j)_{j\in[:3]}, \pi) \to b$
    $\vec{r} \leftarrow H((c_j)_{j\in[:3]})$ ; $b \leftarrow \mathsf{CP}_{\mathsf{poly}}.\mathsf{VerProof}(\mathsf{vk}_p, \vec{r}, c_0, c_t, \pi_0)$
    $b \leftarrow b \wedge \mathsf{CP}_{\mathsf{sc}}.\mathsf{VerProof}(\mathsf{vk}_s, \tilde{eq}(\vec{r},\vec{S}), (c_t, c_1, c_2), \pi_{\mathsf{sc}})$
```

Efficiency. Computing π_0 takes time $O(m)$, and the same holds for π_{sc}. The latter follows by observing that the factors of $g(\vec{S})$ satisfy the good efficiency conditions for $\mathsf{CP}_{\mathsf{sc}}$, i.e., $\tilde{eq}(\vec{r},\vec{s})$ can be computed in $O(\mu)$ time and $\tilde{u}_1, \tilde{u}_2$ are MLE of vectors of length $m = 2^\mu$. For similar reasons, the verifier's time is $O(\mu)$.

We state the following result; its proof is in Appendix H.2.

Theorem 5.3. *In the random oracle model, assuming that* PolyCom *is an extractable trapdoor commitment,* $\mathsf{CP}_{\mathsf{poly}}, \mathsf{CP}_{\mathsf{sc}}$ *are zero-knowledge CP-SNARKs for* PolyCom *and relations* R_{poly} *and* R_{sc} *respectively, then the scheme* $\mathsf{CP}_{\mathsf{had}}$ *described above is a zero-knowledge CP-SNARK for* PolyCom *and relation* R_{had}.

5.4 A CP-SNARK for Self Permutation

Let $\phi : [m] \to [m]$ be a permutation. In this section we propose a CP-SNARK for PolyCom for the relation $R^{\mathsf{sfprm}}_{\phi}$ such that $R^{\mathsf{sfprm}}_{\phi}(\vec{y} := (\vec{x}, (\vec{u}_j)_{j\in[\ell]})) = 1 \iff \forall i \in [m] : y_i = y_{\phi(i)}$.

Our scheme uses a probabilistic trick to prove a permutation of vectors due to [19, 39]. For this we need of a CP-SNARK for proving that $z = \prod_{i=1}^{n} y_i$ with respect to a commitment to point z and vector $\vec{y}$. We call such a relation *internal product* R_{ipd}.

In what follows we present the main ideas to build a CP-SNARK for R^{sfprm} from one for R_{ipd}. Next, we discuss how a CP-SNARK for R_{ipd} can be instantiated.

Recall that the goal is to prove that, for a permutation $\phi : [m] \to [m]$ a committed vector $\vec{y}$ satisfies $y_i = y_{\phi(i)}, \forall i \in [m]$. Consider the following vectors in $\mathbb{F}^m$, $\vec{1}, \vec{v} = (1, \dots, m)$, and $\vec{\phi} = (\phi(1), \dots, \phi(m))$, and assume that the prover committed to $\vec{y}$. Let the verifier choose two random values $r, s \leftarrow_{\$} \mathbb{F}$ and define the vectors $\vec{y}' := \vec{y} + r \cdot \vec{v} - s \cdot \vec{1}$ and $\vec{y}'' := \vec{y} + r \cdot \vec{\phi} - s \cdot \vec{1}$.

If $\vec{y}$ is a permutation of itself according to ϕ, then $(\vec{y} + r \cdot \vec{\phi})$ is a permutation of $(\vec{y} + r \cdot \vec{v})$ according to ϕ; however, if $\vec{y}$ is *not* a self-permutation according to ϕ then with overwhelming probability over the choice of r some of the entries of $\vec{y} + r \cdot \vec{\phi}$ will not be in the vector $\vec{y} + r \cdot \vec{v}$. In our scheme the idea is to let the prover show that $\prod_i y'_i = z = \prod_i y''_i$ using $\mathsf{CP_{ipd}}$ on $(z, \vec{y}')$ and $(z, \vec{y}'')$. However, if some entries of $\vec{y} + r \cdot$ are not in $\phi \neq \vec{y} + r \cdot \vec{v}$, $\prod_i (y_i + r \cdot i - s) = \prod_i (y_i + r \cdot \phi(i) - s)$ holds with negligible probability over the choice of s by the Schwartz-Zippel lemma, thus a prover can be succesful only by cheating with $\mathsf{CP_{ipd}}$.

Our scheme $\mathsf{CP_{sfprm}}$ formalizes the above ideas with some additional details to deal with the fact that $\vec{y}$ is the concatenation of ℓ vectors $(\vec{u}_j)_{j\in[\ell]}$; also it defines the values r, s as output of a random oracle modeled hash function $H((c_{\phi,j})_{j\in[0,\ell]}, \vec{x}, (c_j)_{j\in[\ell]})$. For lack of space a full description of $\mathsf{CP_{sfprm}}$ is deferred to Appendix H.3.

Theorem 5.4. *In the random oracle model, assuming that* PolyCom *is an extractable and linearly-homomorphic trapdoor commitment,* $\mathsf{CP_{ipd}}, \mathsf{CP_{prd}}$ *are zero-knowledge CP-SNARKs for* PolyCom *and relations* R_{ipd} *and* R_{prd} *respectively, then* $\mathsf{CP_{sfprm}}$ *in Figure 13 (Appendix H.3) is a zero-knowledge CP-SNARK for* PolyCom *and relation* R^{sfprm}.

Instantiating $\mathsf{CP_{ipd}}$. R_{ipd} can be expressed with an arithmetic circuit that is a tree of multiplications over $n = 2^{\nu}$ inputs. Thaler [63] showed that for this specially regular circuit the CMT protocol can be adapted so that the prover runs in time $O(n)$. To build a CP-SNARK for R_{ipd}, we thus modify the zk-vSQL protocol [71] so as to work over Thaler's protocol instead of CMT. The changes are quite minimal and mainly regard the equation that links the adjacent layers of the tree. We show this protocol in Appendix I.

From the efficiency observations about $\mathsf{CP_{ipd}}$ given above, we get that $\mathsf{CP_{sfprm}}$.Prove and $\mathsf{CP_{sfprm}}$.VerProof run in time $O(m)$ and $O(\log m)$ respectively.

6 LEGOSNARK APPLICATIONS AND EVALUATION

In this section we first show how to use the modular commit-and-prove approach to obtain new CP-SNARKs for computation expressible by arithmetic circuits (ACs) and then we discuss the resulting instantiations. Our treatment here will not be completely formal; a full formalization is in Appendix J.

CP-SNARKs for Arithmetic Circuits. Bootle et al. [18, 23] show that satisfiability of an arithmetic circuit C holds if the assignment of the inputs-output wires of multiplication gates satify an Hadamard product relation and a set of linear constraints, i.e., if there are three vectors $\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O \in \mathbb{F}^N$ such that

$$\vec{u}^M_L \circ \vec{u}^M_R = \vec{u}^M_O \ \wedge \ \mathbf{W}_L \cdot \vec{u}^M_L + \mathbf{W}_R \cdot \vec{u}^M_R + \mathbf{W}_O \cdot \vec{u}^M_O = \vec{c} \qquad (3)$$

where $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{F}^{Q\times N}, \vec{c} \in \mathbb{F}^Q$ can be extracted from C's description. By defining $\mathbf{F} := [\mathbf{W}_L | \mathbf{W}_R | \mathbf{W}_O]$, we can encode (3) as $R_{\mathsf{had}}(\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O) \wedge R^{\mathsf{lin}}_{\mathbf{F}}(\vec{c}, (\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O))$. Hence, by applying our Theorem 3.2, we obtain a CP-SNARK for AC, dubbed LegoAC.

Instantiations. We evaluate two instantiations of LegoAC:

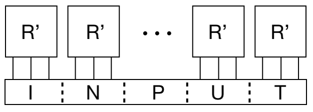
- LegoAC1: from our $\mathsf{CP^{Ped}_{lin}}$ (Section 4.2) and Lipmaa's CP-SNARK for Hadamard products [51]. LegoAC1 is a CP-SNARK for the commitment scheme of [51], and its security holds in the generic group model (due to GGM security of $\mathsf{CP^{Ped}_{lin}}$).
- LegoAC2: from our $\mathsf{CP^{Ped}_{lin}}$ (Section 4.2) and our $\mathsf{CP_{had}}$ (Section 5.3). This is a CP-SNARK for PolyCom, and its security holds in the GGM and random oracle model (the latter due to $\mathsf{CP_{had}}$).

If needed, both schemes can be lifted to work with a standard Pedersen commitment using $\mathsf{CP_{link}}$. Their complexity, summarized in Table 1, results from the combined efficiency of the building blocks plus the observation that the matrices $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O$ are sparse and with a number of nonzero entries linear in the number of circuit wires.

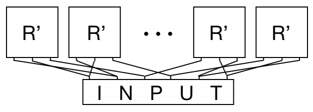
CP-SNARKs for Arithmetic Circuits with Universal CRS. Both LegoAC1 and LegoAC2 have a circuit-specific CRS due to the circuit-specific CRS of $\mathsf{CP^{Ped}_{lin}}$.[10] Aiming for a CP-SNARK for AC satisfiability with a linear-size CRS, we consider a different encoding of AC due to [19, 39]. Let C be an arithmetic circuit C with N_A addition and N_M multiplication gates. Each gate has a left input, a right input and an output wire; each output wire can also be input to another gate. This means that C can be described by integers N_A, N_M, and the wiring information saying that the output wire of addition/multiplication i is the left/right input of addition/multiplication gate j. Having this in mind, satisfiability of C holds if there is a satisfying assignment to all the gates, i.e., $\vec{u}^A_L + \vec{u}^A_R = \vec{u}^A_O$ and $\vec{u}^M_L \circ \vec{u}^M_R = \vec{u}^M_O$, and if these assignments are consistent with C's wiring. This consistency is essentially a check that specific entries of the vector $(\vec{u}^A_L, \vec{u}^A_R, \vec{u}^A_O, \vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O)$ must be equal, which can be formalized with $R^{\mathsf{sfprm}}_{\phi}$ for an appropriate permutation ϕ.

Using our composition theorem we obtain a CP-SNARK for AC satisfiability, dubbed LegoUAC, through the combination of R_{had} and R^{sfprm}. We propose to instantiate it with our $\mathsf{CP_{had}}$ and $\mathsf{CP_{sfprm}}$.[11] Both these schemes admit a universal CRS that can be deterministically specialized (due to specializing $\mathsf{CP_{sfprm}}$'s CRS to the circuit-dependent permutation ϕ). The asymptotic efficiency of LegoUAC is shown in Table 1 and results from that of our $\mathsf{CP_{had}}$ and $\mathsf{CP_{sfprm}}$.

[10]Using our $\mathsf{CP_{lin}}$ for PolyCom (Appendix H.4) would give us an instantiation with a universal CRS, but unfortunately one of size $Q \cdot N$, that is quadratic in circuit size.
[11]Additions come for free by using a linearly homomorphic commitment.



(a) In R^{par}, R' inputs are disjoint



(b) In R^{parjnt}, R' inputs are joint

Figure 2: Inputs structures for parallel relations.

Parallel Checks on Joint Inputs. Consider a relation $R^{\mathsf{parjnt}}(u) := \bigwedge_{j=1}^{N} R'(u'_j)$ consisting of N parallel checks of the same relation R' on (partially) shared inputs. This relation has several use cases, e.g., proving knowledge of all the leaves of a Merkle tree of height N with respect to a public root, which can be seen as the parallel check of $2^N - 1$ hash verification relations (i.e., $R_H(x_1, x_2, y) := H(x_1, x_2) \overset{?}{=} y$) that share some of the inputs.

One way to deal with R^{parjnt} is by defining the arithmetic circuit that computes it (cf. Fig. 2b). The Hyrax system is particularly designed for parallel circuits [69]; they deal with non-parallel input by introducing a (non-parallel) redistribution layer (RDL) layer that *redistributes* the input and feeds it to the identical sub-circuits at the next level. Unfortunately an effect of using an RDL is that the verifier must pay an additional cost linear in the total width of the circuit. In several cases (including Merkle tree verification described earlier) this can result in a bottleneck in verification performances dominating its asymptotics.

We solve this issue by modeling R^{parjnt} as the simple conjunction of two relations: R^{par} for *fully parallel* checks of R' on *disjoint* inputs (cf. Fig. 2), and another relation that checks the consistency of the shared inputs across the parallel executions (that we can express with R^{sfprm} or R^{lin}). This way, we can build a CP-SNARK for R^{parjnt} (called LegoPar) through our composition result applied to our $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ and a CP-SNARK for R^{par} relations. For the latter, we use an adaptation of Hyrax using the polynomial commitment PolyCom of zk-vSQL. We call HyrPoly-Par this scheme invoked on circuits without RDL (i.e., it supports R^{par}), and HyrPoly-RDL the same scheme for circuits with an RDL (i.e., it supports R^{parjnt}).

We compare the efficiency of LegoPar and HyrPoly-RDL on R^{parjnt} relations. Let d and G be depth and width of the arithmetic circuit evaluating R'. Proving time and proof size have the same complexity in both solutions; verifier time is $O(d(G + \log(NG)))$ in LegoPar and $O(d(G + \log(NG)) + |u| + NG)$ in HyrPoly-RDL. We note that due to the use of $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$, the CRS of LegoPar becomes specific to the input wiring of R^{parjnt}, whereas in HyrPoly-RDL the CRS is just the commitment key. On the other hand, this one-time preprocessing allows the verifier to later check any number of proofs in shorter time.[12]

7 EXPERIMENTAL EVALUATION

We designed and implemented LegoSNARK,[13] a library for commit-and-prove SNARKs that includes a design framework for composable CP-SNARKs, and the implementation of a collection of proof gadgets: our $\mathsf{CP}_{\mathsf{link}}$ and $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$, the Hadamard product CP-SNARK of [51], and the $\mathsf{CP}_{\mathsf{poly}}$ from [71][14]. LegoSNARK is written in C++; for polynomial operations and bilinear pairings we use the libraries underlying libsnark [3]. We executed our experiments on a virtual machine running Debian GNU/Linux with 8 Xeon Gold 6154 cores and 30 GB of RAM. We ran all tests single threaded.

In our experiments we tested the performance of some of our instantiations and compared to different baseline systems.

7.1 Commit-and-Prove SNARKs

We consider a generic application of proving commit-and-prove relations where commitments are created using the Pedersen scheme for vectors, i.e., proving $\exists(u, o, \omega)R(u, \omega) \wedge \mathsf{VerCommit}(\mathsf{ck}, c, u, o)$. As baseline system, we use the Groth16 zkSNARK in libsnark on the libsnark gadget circuit for multi-scalar additions over a SNARK-friendly elliptic curve (to model the Pedersen computation). We call this CPGro16. We compare CPGro16 to a CP-SNARK, LegoGro16, obtained by applying our cc-SNARK-lifting compiler with our $\mathsf{CP}_{\mathsf{link}}$ scheme to the cc-variant of [41] described in Appendix K.5. In our experiments we measured the overhead of dealing with the commitment in both schemes (the costs related to R would be the same in both cases) at the increase of the committed vector's dimension (from 8 to 2048).[15] On the largest instance ($n = 2048$), our LegoGro16 proving time is $5,000\times$ (0.08 vs. 428 s) faster than CPGro16, at the price of a verification that is $1.2\times$ slower (4.1 vs 3.4 ms), and a slightly larger proof (191 vs. 127 Bytes). LegoGro16's CRS is also $7000\times$ shorter (130KB vs. 950MB). Details of our experiments can be found in Appendix L.

7.2 Parallel Checks on Joint Inputs

We compare performances of our LegoPar system with a baseline system, i.e. HyrPoly-RDL (see Appendix H.5). Recall that LegoPar consists of our $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ and HyrPoly-Par. To evaluate HyrPoly-Par and HyrPoly-RDL we executed separately the part concerning PolyCom and $\mathsf{CP}_{\mathsf{poly}}$, and the one that includes the ZKGir^{++} core. To benchmark the latter, we used the original Python code (appropriately modified for the commitment part) from the Hyrax project [1] (executed using the JIT-compiling interpreter PyPy [4]).[16]

We benchmarked LegoPar and HyrPoly-RDL on a highly parallel computation, that is proving knowledge of an assignment to the leaves of a Merkle tree [53] (cf. Section 6 to see how it can be expressed using R^{parjnt}). We used SHA256 for the hash and a varying number of leaves (from 2 to 2^9). For this computation we generated two circuits using the Hyrax tool: one fully parallel to be fed to HyrPoly-Par and one with the RDL for HyrPoly-RDL. Recall that in LegoPar the RDL is checked using $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$. We finally note that the two largest input sizes in our evaluation required extending the available RAM from 30 to 75GB for both HyrPoly-RDL and HyrPoly-Par.

Results. Figure 3 compares the costs (proving and verification time) in the two schemes for repeated computation. Overall LegoPar is faster than HyrPoly-RDL, both in proving and verification time. On our largest input, proving in LegoPar is $1.25\times$ faster; verifying is more than $2.5\times$ faster. Verification is expected to become faster due to the asymptotic difference in the verification time.

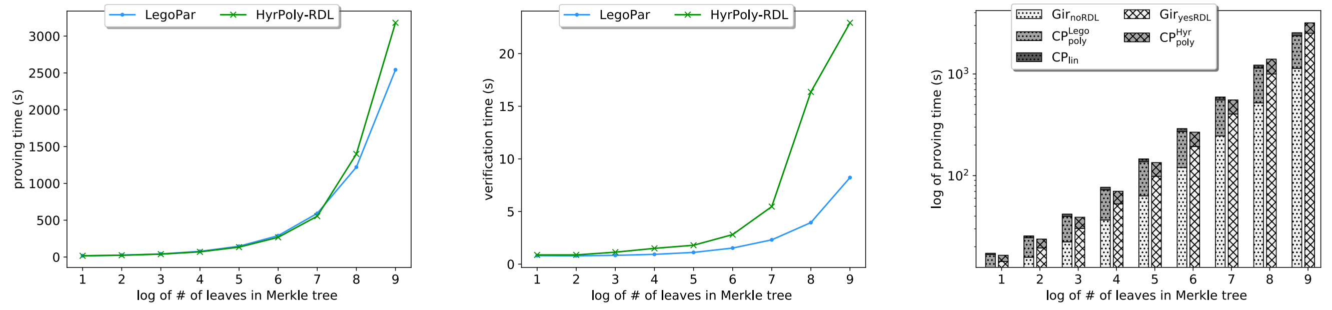
[12]We do not see a way to run a similar preprocessing in HyrPoly. We evaluated the possibility to commit, in preprocessing, to the MLE of the RDL wiring so that the prover would compute this on behalf of the verifier and prove its correct evaluation using $\mathsf{CP}_{\mathsf{poly}}$. This idea however would require a commitment key quadratic in the circuit width, which is prohibitively large.

[13]The library will soon be made open source.

[14]For this we adapted to our library the code provided by the authors of [71].

[15]At $n = 4096$ CPGro16 ran out of memory.

[16]Full integration of this component into our library is future work.



(a) Proving time comparison for LegoPar **and** HyrPoly-RDL**.**

(b) Verification time comparison for LegoPar **and** HyrPoly-RDL**.**

(c) $\mathcal{P}$ time (component-wise) for LegoPar **(left bars) and** HyrPoly-RDL **(right).**

Figure 3: Performance comparison of systems for parallel relations. Lower on the y axis is better (in (c), axis y is log-scale). We remind the reader that LegoPar = HyrPoly-Par + $\mathsf{CP}_{\mathsf{lin}}$ = ($\mathsf{Gir}_{\mathsf{noRDL}}$ + $\mathsf{CP}_{\mathsf{poly}}^{\mathsf{Lego}}$) + $\mathsf{CP}_{\mathsf{lin}}$ **and** HyrPoly-RDL = $\mathsf{Gir}_{\mathsf{yesRDL}}$ + $\mathsf{CP}_{\mathsf{poly}}^{\mathsf{Hyr}}$**.**

- *Proving time*: On larger inputs LegoPar has a faster (up to 1.25×) proving time (Figs. 3a). In both schemes most of the computation is due to ZKGir^{++}: approximately 50% for LegoPar and 75% for HyrPoly-RDL. The higher time of ZKGir^{++} in HyrPoly-RDL is explained by the additional round for the RDL. On the other hand, LegoPar spends twice as much time for the proving step of $\mathsf{CP}_{\mathsf{poly}}$. This is because it evaluates a polynomial with twice as many terms, in turn requiring roughly twice the number of exponentiations. (This is due to the RDL output u_2, on which LegoPar operates, being twice as long as the RDL input u_1 (also the "bottom-layer" input), on which $\mathsf{CP}_{\mathsf{poly}}$ runs in HyrPoly-RDL).
- *Verification time*: On larger inputs LegoPar has a shorter (up to 2.5×) verification time (Fig. 3b). This speedup is due to increase with larger inputs, as the verifier in HyrPoly-RDL has to perform an additional verification step for the RDL in ZKGir^{++} (requiring a number of field operations roughly linear in the width of the circuit). On the other hand LegoPar performs the same step through a constant number of pairings (two) in $\mathsf{CP}_{\mathsf{veq}}$. In both schemes, the running time is dominated by ZKGir^{++}, requiring more than 99.5% of the total verification time[17].

Discussion. Partly, the different performances we observed are due to specific features of the circuit chosen for benchmarks (in our case, Merkle tree verification). In a circuit for parallel computation, at least two features, both related to the RDL, can have impact: *(i)* how "large" the output u_2 of the RDL is with respect to its input u_1; *(ii)* how "complex" the RDL is. A higher ratio between $|u_2|$ and $|u_1|$ will determine the difference in running time for the $\mathsf{CP}_{\mathsf{poly}}$.Prove component. As mentioned, in our circuit of choice the ratio was 2.

7.3 LegoAC1 for Arithmetic Circuits

We tested our LegoAC1 scheme (see Section 6) for arithmetic circuits and compared it to Groth16 as a baseline system. We considered two benchmark applications:
(a) proving knowledge of a *SHA256 pre-image* on 512-bit inputs; for this we used the existing circuit gadgets implemented in libsnark (for Groth16), and in Bulletproofs [2] (for LegoAC1).
(b) *matrix factoring*, i.e., proving knowledge of two $n \times n$ matrices A, B whose product is a public matrix C; for this we designed suitable constraints systems, considering 32-bit integers entries and a varying $n = 16, 32, 64, 128$.

In Appendix L we provide tables with detailed numbers of these experiments. Overall, our experiments show that LegoAC1 performs slightly worse than Groth16. For example, for SHA256 proving time is 1.2× slower (0.7 vs. 0.9 s) and verification is 2× slower (0.9 vs. 1.8 ms). Proof size is constant: 350B in LegoAC1 and 127B in Groth16. In a way this result is not surprising: Groth16 is an extremely optimized and well explored scheme, whereas for LegoAC1 we believe that more optimizations could be explored (in a similar way as Groth16 optimized Pinocchio). More remarkably, LegoAC1 *has a built-in commit-and-prove capability* that is not present in Groth16 and that can be useful in several applications. For example, in our matrix factoring application LegoAC1 works with commitments to the three matrices that could be reused. As an example of the power of this feature, we could prove a statement like "$B = A^{2^k}$ for a committed matrix A" by doing k proofs, one for each squaring step (i.e., to show that $B_i = B_{i-1}^2$); this can be done by reusing the same CRS for one matrix factoring relation. In contrast, proving $B = A^{2^k}$ directly with Groth16 would require a very large CRS and a memory intensive prover that would not scale for large k and n.

8 CONCLUSIONS

We have described LegoSNARK, a framework for commit-and-prove zkSNARKs that comprises definitions, a general composition result, and a "lifting" construction. The LegoSNARK tools are useful as they enable designing zkSNARKs in a modular way (due to the framework of definitions and the composition theorem) and they allow to efficiently add commit-and-prove capabilities to a variety of existing schemes thus made interoperable. Furthermore we have proposed efficient proof gadgets for specialized relations and shown how to combine them into succinct proof systems for more complex relations. We have described instantiations of these new proof systems and evaluated them against prior work. The results show they have competitive performances. Specifically they show slightly worse (but still acceptable) performances in some applications (general arithmetic circuits) and significant improvements in others (commit-ahead-of-time systems, parallel computations).

[17]This is also why we do not show a detailed bar plot for each component as done for proving time.

A limitation of our current instantiations is that they rely on pairing-based systems requiring a trusted setup. Interestingly in some cases a trusted setup is only needed to generate the commitment key of PolyCom. We expect this be doable with a large-scale MPC ceremony similar to the powers-of-tau round 1 of [20]. It will be future work to explore this direction. Nonetheless we note that this limitation is not inherent. The basic results of the framework (i.e., Section 3) would also apply to schemes without trusted setup and hence are general enough to allow for future instantiations without trust assumptions.

Finally, another future work direction is investigating new and more efficient proof gadgets CP-SNARKs for specialized relations and test them in specific applications.

ACKNOWLEDGEMENTS

The project that gave rise to these results received the support of a grant by Protocol Labs and a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/ES18/11670018.

REFERENCES

[1] Hyrax. https://github.com/hyraxZK. (Hyrax).

[2] libsecp256k1. https://github.com/apoelstra/secp256k1-mw/tree/bulletproofs. (libsecp256k1).

[3] libsnark. https://github.com/scipr-lab/libsnark. (libsnark).

[4] PyPy. https://pypy.org. (PyPy).

[5] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. 2018. Non-Interactive Zero-Knowledge Proofs for Composite Statements. In *CRYPTO 2018, Part III (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10993. Springer, Heidelberg, 643–673. https://doi.org/10.1007/978-3-319-96878-0_22

[6] Kurt M. Alonso and Jordi Herrera Joancomartí. 2018. Monero - Privacy in the Blockchain. Cryptology ePrint Archive, Report 2018/535. (2018). https://eprint.iacr.org/2018/535.

[7] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 17*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2087–2104. https://doi.org/10.1145/3133956.3134104

[8] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. 2015. ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 271–286. https://doi.org/10.1109/SP.2015.24

[9] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046. (2018). https://eprint.iacr.org/2018/046.

[10] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 459–474. https://doi.org/10.1109/SP.2014.36

[11] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, 90–108. https://doi.org/10.1007/978-3-642-40084-1_6

[12] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *TCC 2016-B, Part II (LNCS)*, Martin Hirt and Adam D. Smith (Eds.), Vol. 9986. Springer, Heidelberg, 31–60. https://doi.org/10.1007/978-3-662-53644-5_2

[13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *USENIX Security*. 781–796.

[14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. 2017. The Hunting of the SNARK. *Journal of Cryptology* 30, 4 (Oct. 2017), 989–1066.

[15] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 326–349. https://doi.org/10.1145/2090236.2090263

[16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. 2014. On the existence of extractable one-way functions. In *46th ACM STOC*, David B. Shmoys (Ed.). ACM Press, 505–514. https://doi.org/10.1145/2591796.2591859

[17] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. 2013. Succinct Non-interactive Arguments via Linear Interactive Proofs. In *TCC 2013 (LNCS)*, Amit Sahai (Ed.), Vol. 7785. Springer, Heidelberg, 315–333. https://doi.org/10.1007/978-3-642-36594-2_18

[18] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *EUROCRYPT 2016, Part II (LNCS)*, Marc Fischlin and Jean-Sébastien Coron (Eds.), Vol. 9666. Springer, Heidelberg, 327–357. https://doi.org/10.1007/978-3-662-49896-5_12

[19] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. 2017. Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability. In *ASIACRYPT 2017, Part III (LNCS)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.), Vol. 10626. Springer, Heidelberg, 336–365. https://doi.org/10.1007/978-3-319-70700-6_12

[20] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Report 2017/1050. (2017). https://eprint.iacr.org/2017/1050.

[21] Elette Boyle and Rafael Pass. 2015. Limits of Extractability Assumptions with Distributional Auxiliary Input. In *ASIACRYPT 2015, Part II (LNCS)*, Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9453. Springer, Heidelberg, 236–261. https://doi.org/10.1007/978-3-662-48800-3_10

[22] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. 2013. Verifying computations with state. In *Proc. of the ACM SOSP*.

[23] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2017. *Bulletproofs: Efficient range proofs for confidential transactions*. Technical Report. Cryptology ePrint Archive, Report 2017/1066, 2017. https://eprint. iacr. org/2017/1066.

[24] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. 2002. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*. ACM Press, 494–503. https://doi.org/10.1145/509907.509980

[25] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. 2017. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In *ACM CCS 17*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 1825–1842. https://doi.org/10.1145/3133956.3133997

[26] Melissa Chase, Chaya Ganesh, and Payman Mohassel. 2016. Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials. In *CRYPTO 2016, Part III (LNCS)*, Matthew Robshaw and Jonathan Katz (Eds.), Vol. 9816. Springer, Heidelberg, 499–530. https://doi.org/10.1007/978-3-662-53015-3_18

[27] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 90–112. https://doi.org/10.1145/2090236.2090245

[28] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile Verifiable Computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 253–270. https://doi.org/10.1109/SP.2015.23

[29] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. 2013. An Algebraic Framework for Diffie-Hellman Assumptions. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, 129–147. https://doi.org/10.1007/978-3-642-40084-1_8

[30] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. 2017. An Efficient Pairing-Based Shuffle Argument. In *ASIACRYPT 2017, Part II (LNCS)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.), Vol. 10625. Springer, Heidelberg, 97–127. https://doi.org/10.1007/978-3-319-70697-9_4

[31] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. 2016. Hash First, Argue Later: Adaptive Verifiable Computations on Outsourced Data. In *ACM CCS 16*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1304–1316. https://doi.org/10.1145/2976749.2978368

[32] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The Algebraic Group Model and its Applications. In *CRYPTO 2018, Part II (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10992. Springer, Heidelberg, 33–62. https://doi.org/10.1007/978-3-319-96881-0_2

[33] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *EUROCRYPT 2013 (LNCS)*, Thomas Johansson and Phong Q. Nguyen (Eds.), Vol. 7881. Springer, Heidelberg, 626–645. https://doi.org/10.1007/978-3-642-38348-9_37

[34] Craig Gentry and Daniel Wichs. 2011. Separating succinct non-interactive arguments from all falsifiable assumptions. In *43rd ACM STOC*, Lance Fortnow and Salil P. Vadhan (Eds.). ACM Press, 99–108. https://doi.org/10.1145/1993636.1993651

[35] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1069–1083.

[36] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420
[37] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *40th ACM STOC*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, 113–122. https://doi.org/10.1145/1374376.1374396
[38] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18, 1 (1989), 186–208.
[39] Jens Groth. 2009. Linear Algebra with Sub-linear Zero-Knowledge Arguments. In *CRYPTO 2009 (LNCS)*, Shai Halevi (Ed.), Vol. 5677. Springer, Heidelberg, 192–208. https://doi.org/10.1007/978-3-642-03356-8_12
[40] Jens Groth. 2010. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In *ASIACRYPT 2010 (LNCS)*, Masayuki Abe (Ed.), Vol. 6477. Springer, Heidelberg, 321–340. https://doi.org/10.1007/978-3-642-17373-8_19
[41] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT 2016, Part II (LNCS)*, Marc Fischlin and Jean-Sébastien Coron (Eds.), Vol. 9666. Springer, Heidelberg, 305–326. https://doi.org/10.1007/978-3-662-49896-5_11
[42] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. 2018. Updatable and Universal Common Reference Strings with Applications to zk-SNARKs. In *CRYPTO 2018, Part III (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10993. Springer, Heidelberg, 698–728. https://doi.org/10.1007/978-3-319-96878-0_24
[43] Daniel Günther, Ágnes Kiss, and Thomas Schneider. 2017. More Efficient Universal Circuit Constructions. In *ASIACRYPT 2017, Part II (LNCS)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.), Vol. 10625. Springer, Heidelberg, 443–470. https://doi.org/10.1007/978-3-319-70697-9_16
[44] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2007. Efficient Arguments Without Short PCPs. In *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity (CCC '07)*. IEEE Computer Society, Washington, DC, USA, 278–291.
[45] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2007. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, David S. Johnson and Uriel Feige (Eds.). ACM Press, 21–30. https://doi.org/10.1145/1250790.1250794
[46] J. Kilian. 1989. Uses of Randomness in Algorithms and Protocols. PhD Thesis. Massachusetts Institute of Technology. (1989).
[47] Joe Kilian. 1992. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In *24th ACM STOC*. ACM Press, 723–732. https://doi.org/10.1145/129712.129782
[48] Eike Kiltz and Hoeteck Wee. 2015. Quasi-Adaptive NIZK for Linear Subspaces Revisited. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 101–128. https://doi.org/10.1007/978-3-662-46803-6_4
[49] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. 2014. TRUESET: Faster Verifiable Set Computations. In *USENIX Security*. 765–780.
[50] Helger Lipmaa. 2012. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In *TCC 2012 (LNCS)*, Ronald Cramer (Ed.), Vol. 7194. Springer, Heidelberg, 169–189. https://doi.org/10.1007/978-3-642-28914-9_10
[51] Helger Lipmaa. 2016. Prover-Efficient Commit-and-Prove Zero-Knowledge SNARKs. In *AFRICACRYPT 16 (LNCS)*, David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi (Eds.), Vol. 9646. Springer, Heidelberg, 185–206. https://doi.org/10.1007/978-3-319-31517-1_10
[52] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (Oct. 1992), 859–868.
[53] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO'87 (LNCS)*, Carl Pomerance (Ed.), Vol. 293. Springer, Heidelberg, 369–378. https://doi.org/10.1007/3-540-48184-2_32
[54] Silvio Micali. 1994. CS Proofs (Extended Abstracts). In *35th FOCS*. IEEE Computer Society Press, 436–453. https://doi.org/10.1109/SFCS.1994.365746
[55] Silvio Micali. 2000. Computationally Sound Proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298. https://doi.org/10.1137/S0097539795284959
[56] Moni Naor and Moti Yung. 1990. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *22nd ACM STOC*. ACM Press, 427–437. https://doi.org/10.1145/100216.100273
[57] Bryan Parno. 2015. A Note on the Unsoundness of vnTinyRAM's SNARK. Cryptology ePrint Archive, Report 2015/437. (2015). http://eprint.iacr.org/2015/437.
[58] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 238–252. https://doi.org/10.1109/SP.2013.47
[59] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91 (LNCS)*, Joan Feigenbaum (Ed.), Vol. 576. Springer, Heidelberg, 129–140. https://doi.org/10.1007/3-540-46766-1_9
[60] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2016. Constant-round interactive proofs for delegating computation. In *48th ACM STOC*, Daniel Wichs and Yishay Mansour (Eds.). ACM Press, 49–62. https://doi.org/10.1145/2897518.2897652
[61] Guy Rothblum. 2009. Delegating computation reliably: paradigms and constructions. (2009). PhD thesis.
[62] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991), 161–174.
[63] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO 2013, Part II (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, Heidelberg, 71–89. https://doi.org/10.1007/978-3-642-40084-1_5
[64] Leslie G. Valiant. 1976. Universal Circuits (Preliminary Report). In *STOC*. ACM, 196–203.
[65] Meilof Veeningen. 2017. Pinocchio-Based Adaptive zk-SNARKs and Secure/Correct Adaptive Function Evaluation. In *AFRICACRYPT 17 (LNCS)*, Marc Joye and Abderrahmane Nitaj (Eds.), Vol. 10239. Springer, Heidelberg, 21–39.
[66] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. 2017. Full Accounting for Verifiable Outsourcing. In *ACM CCS 17*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2071–2086. https://doi.org/10.1145/3133956.3133984
[67] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS 2015*. The Internet Society.
[68] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. 2017. Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2017/1132. (2017). https://eprint.iacr.org/2017/1132.
[69] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 926–943. https://doi.org/10.1109/SP.2018.00060
[70] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 863–880. https://doi.org/10.1109/SP.2017.43
[71] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. A Zero-Knowledge Version of vSQL. Cryptology ePrint Archive, Report 2017/1146. (2017). https://eprint.iacr.org/2017/1146.

A FORMAL DEFINITIONS

This section gives formal definitions regarding commitment schemes, SNARKs, SNARKs with universal CRS and cc-SNARKs.

A.1 Properties of Commitment Schemes

A commitment scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ must satisfy the following properties:

Correctness. For all $\lambda \in \mathbb{N}$ and any input $u \in \mathcal{D}$ we have:

$$\Pr\left[\begin{array}{l}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)\\ (c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, u)\end{array} : \mathsf{VerCommit}(\mathsf{ck}, c, u, o) = 1\right] = 1$$

Binding. For every polynomial-time adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{l}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)\\ (c, u, o, u', o') \leftarrow \mathcal{A}(\mathsf{ck})\end{array} : \begin{array}{c}\mathsf{VerCommit}(\mathsf{ck}, c, u', o')\\ \wedge\ \mathsf{VerCommit}(\mathsf{ck}, c, u, o)\\ \wedge\ u \neq u'\end{array}\right] = \mathsf{negl}$$

Hiding. For $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and every values $u, u' \in \mathcal{D}$, the following two distributions are statistically close: $\mathsf{Commit}(\mathsf{ck}, u) \approx \mathsf{Commit}(\mathsf{ck}, u')$.

A.2 Properties of SNARKs

We first give more insight on the properties of a SNARK, which is a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{VerProof})$ satisfying the notions of *completeness*, *succinctness* and *knowledge soundness*. If Π also satisfies *zero knowledge* then we call it a zkSNARK.

Completeness. For any $\lambda \in \mathbb{N}$, $R \in \mathcal{R}_\lambda$ and (x, w) such that $R(x, w)$, it holds $\Pr[(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R), \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w) : \mathsf{VerProof}(\mathsf{vk}, x, \pi) = 1] = 1$.

Succinctness. Π is said *succinct* if the running time of VerProof is $\mathrm{poly}(\lambda)\,(\lambda + |x| + \log|w|)$ and the proof size is $\mathrm{poly}(\lambda)\,(\lambda + \log|w|)$.

Knowledge Soundness. Π has knowledge soundness for $\mathcal{RG}$ and auxiliary input distribution $\mathcal{Z}$, denoted $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$ for brevity, if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}$. We say that Π is knowledge sound if there exists benign $\mathcal{RG}$ and $\mathcal{Z}$ such that Π is $\mathsf{KSND}(\mathcal{RG}, \mathcal{Z})$.

```
$\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \rightarrow b$
$(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$ ; $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R)$
$\mathsf{aux}_\mathcal{Z} \leftarrow \mathcal{Z}(R, \mathsf{aux}_R, \mathsf{crs})$ ; $(x, \pi) \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_\mathcal{Z})$
$w \leftarrow \mathcal{E}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_\mathcal{Z})$ ; $b = \mathsf{VerProof}(\mathsf{vk}, x, \pi) \wedge \neg R(x, w)$
```

Composable Zero-Knowledge. A scheme Π satisfies composable zero-knowledge for a relation generator $\mathcal{RG}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ such that both following conditions hold for all adversaries $\mathcal{A}$:

Keys Indistinguishability

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ \mathsf{crs} \leftarrow \mathsf{KeyGen}(R)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\end{array} : \right] \approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\end{array} : \right]$$

Proof Indistinguishability For all (x, w) such that $R(x, w) = 1$,

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R);\\ \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi) = 1\end{array} : \right] \approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(R);\\ \pi \leftarrow \mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td_k}, x)\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi) = 1\end{array} : \right]$$

Remark 2. *In the notion of knowledge soundness defined above we consider two kinds of auxiliary inputs,* aux_R *generated together with the relation by* $\mathcal{RG}$*, and* $\mathsf{aux}_\mathcal{Z}$ *that is generated from some distribution* $\mathcal{Z}$ *that may depend on the common reference string that in turns depends on R. An example of this appears in our proof of Theorem C.1. Notice that although our notion is implied by a notion where auxiliary inputs can be arbitrary, our aim is a precise formalization of auxiliary inputs; this is useful to justify why certain auxiliary inputs should be considered benign, as required to avoid known impossibility results [16, 21]. Finally, we also note that our notion is also implied by SNARKs that admit black-box extractors (as may be the case for those relying on random oracles [55]).*

A.3 zkSNARKs with Universal CRS

In the SNARK notion presented in the previous section the common reference string generated by KeyGen is tied to a specific relation $R \in \mathcal{R}_\lambda$. Now we define a variant of this notion introduced in [42] where the CRS only depends on the family $\mathcal{R}_\lambda$, working for any R in that family. More formally, let $\mathcal{R}_\lambda$ be a family of relations. The universal relation R^* for $\mathcal{R}_\lambda$ defines a language with instances (R, x) such that $R^*(R, x, w)$ holds iff $R \in \mathcal{R}_\lambda$ and $R(x, w)$ holds.

A Π = (KeyGen, Prove, VerProof) is said a *zkSNARK with specializable universal common reference string* [42] if there exist algorithms Derive, Prove*, VerProof* such that:

- $\mathsf{Derive}(\mathsf{crs}, R) \rightarrow \mathsf{crs}_R$ is a *deterministic* algorithm that takes as input a crs := (ek, vk) produced by $\mathsf{KeyGen}(R^*)$ and a relation $R \in \mathcal{R}_\lambda$, and outputs a specialized common reference string $\mathsf{crs}_R := (\mathsf{ek}_R, \mathsf{vk}_R)$.
- $\mathsf{Prove}(\mathsf{ek}, (R, x), w) \rightarrow \pi$ runs $(\mathsf{ek}_R, \mathsf{vk}_R) \leftarrow \mathsf{Derive}(\mathsf{crs}, R)$ and returns $\pi \leftarrow \mathsf{Prove}^*(\mathsf{ek}_R, x, w)$.
- $\mathsf{VerProof}(\mathsf{vk}, (R, x), \pi) \rightarrow b$ runs $(\mathsf{ek}_R, \mathsf{vk}_R) \leftarrow \mathsf{Derive}(\mathsf{crs}, R)$ and returns $b \leftarrow \mathsf{VerProof}^*(\mathsf{vk}_R, x, \pi)$.

A.4 Properties of cc-SNARKs

This section gives formal definitions of the properties of a commit-carrying SNARK. Recall a cc-SNARK is a tuple ccΠ of algorithms working as follows:

- $\mathsf{KeyGen}(R) \rightarrow (\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$: the key generation takes as input the security parameter λ and a relation $R \in \mathcal{R}_\lambda$, and outputs a common reference string that includes a commitment key, an evaluaton key and verification key.
- $\mathsf{Prove}(\mathsf{ek}, x, w) \rightarrow (c, \pi; o)$: the proving algorithm takes as input an evaluation key, a statement x and a witness $w := (u, \omega)$ such that the relation $R(x, u, \omega)$ holds, and it outputs a proof π, a commitment c and opening o such that VerCommit(ck,c,u,o) = 1.
- $\mathsf{VerProof}(\mathsf{vk}, x, c, \pi) \rightarrow b$: the verification algorithm takes a verification key, a statement x, a commitment c, and either accepts ($b = 1$) or rejects ($b = 0$) the proof π.
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \rightarrow b$: the commitment verification algorithm takes as input a commitment key, a commitment c, a message u and an opening o and accepts ($b = 1$) or rejects ($b = 0$).

Completeness. For any $\lambda \in \mathbb{N}, R \in \mathcal{R}_\lambda$ and (x, w) such that $R(x, w)$, it holds

$$\Pr\left[\begin{array}{l}(\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R);\\ (c, \pi; o) \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w)\end{array} : \mathsf{VerProof}(\mathsf{vk}, x, c, \pi)\right] = 1$$

Succinctness. ccΠ is said *succinct* if the running time of VerProof is $\mathrm{poly}(\lambda)\,(\lambda + |x| + \log|w|)$ and the size of the proof is $\mathrm{poly}(\lambda) \cdot (\lambda + \log|w|)$.

Knowledge Soundness. Let $\mathcal{RG}$ be a relation generator such that $\mathcal{RG}_\lambda \subseteq \mathcal{R}_\lambda$. cc$\Pi$ satisfies knowledge soundness for $\mathcal{RG}$ and auxiliary input distribution $\mathcal{Z}$, or $\mathsf{ccKSND}(\mathcal{RG}, \mathcal{Z})$, if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that $\Pr[\mathsf{Game}^{\mathsf{ccKSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}$. We say that cc$\Pi$ is knowledge sound if there exist benign $\mathcal{RG}$ and $\mathcal{Z}$ such that ccΠ is $\mathsf{ccKSND}(\mathcal{RG}, \mathcal{Z})$.

```
$\mathsf{Game}^{\mathsf{ccKSND}}_{\mathcal{RG},\mathcal{Z},\mathcal{A},\mathcal{E}} \rightarrow b \in \{0, 1\}$
$(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)$ ; $\mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R)$
$\mathsf{aux}_\mathcal{Z} \leftarrow \mathcal{Z}(R, \mathsf{aux}_R, \mathsf{crs})$ ; $(x, c, \pi) \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_\mathcal{Z})$
$(u, o, \omega) \leftarrow \mathcal{E}(R, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_\mathcal{Z})$
$b \leftarrow \mathsf{VerProof}(\mathsf{vk}, x, c, \pi) \wedge \neg(\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \wedge R(x, u, \omega))$
```

Composable Zero-Knowledge. A scheme ccΠ has composable zero-knowledge for a relation generator $\mathcal{RG}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S} = (\mathcal{S}_{\text{kg}}, \mathcal{S}_{\text{prv}})$ such that both following conditions hold for all adversaries $\mathcal{A}$:

Keys Indistinguishability.

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ \mathsf{crs} \leftarrow \mathsf{KeyGen}(R)\end{array} : \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\text{kg}}(R)\end{array} : \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R) = 1\right]$$

Proof Indistinguishability. For all (x, w),

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\text{kg}}(R);\\ (c, \pi; o) \leftarrow \mathsf{Prove}(\mathsf{ek}, x, w)\end{array} : \begin{array}{r}\mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, c, \pi) = 1 \land\\ R(x, w) = 1\end{array}\right]$$
$$\approx \Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\text{kg}}(R);\\ (c, \pi) \leftarrow \mathcal{S}_{\text{prv}}(\mathsf{crs}, \mathsf{td_k}, x)\end{array} : \begin{array}{r}\mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, c, \pi) = 1 \land\\ R(x, w) = 1\end{array}\right]$$

Binding. For every polynomial-time adversary $\mathcal{A}$ the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ \mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(R)\\ (c, u, o, u', o') \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R)\end{array} : \begin{array}{r}\mathsf{VerCommit}(\mathsf{ck}, c, u', o')\\ \land \mathsf{VerCommit}(\mathsf{ck}, c, u, o)\\ \land u \neq u'\end{array}\right]$$

Remark 3. *While our definitions consider the case where the proof contains a commitment to a portion u of the witness $w = (u, \omega)$, notice that this partition of the witness is arbitrary and thus this notion also captures those constructions where the commitment is to the entire witness if one thinks of a void ω.*

cc-SNARKs with Weak Binding Let us now define a weaker variant of cc-SNARKs that differs from the one given in Definition 3.3 in that the underlying commitment scheme is not binding in the usual sense. Slightly more in detail we consider the case where the commitment refers to the whole witness (i.e., ω is an empty string) and it is actually possible to find collisions for a given commitment as long as these collisions are among valid witnesses, or more precisely we require to be computationally infeasible to find two different witnesses that validly open the commitment such that one falsifies the relation and the other one satisfies it.

Definition A.1 (cc-SNARKs with Weak Binding). We define cc-SNARKs with Weak Binding as in Definition 3.3 with two exceptions: we assume that the scheme is defined only for relations such that $\mathcal{D}_\omega = \emptyset$; we replace the binding property with the one below.

Weak Binding. For every polynomial-time adversary $\mathcal{A}$ the following probability is $\mathsf{negl}(\lambda)$

$$\Pr\left[\begin{array}{l}(R, \mathsf{aux}_R) \leftarrow \mathcal{RG}(1^\lambda)\\ \mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KG}(R)\\ (x, c, u, o, u', o', \pi)\\ \quad \leftarrow \mathcal{A}(R, \mathsf{crs}, \mathsf{aux}_R)\end{array} : \begin{array}{l}\mathsf{VerCommit}(\mathsf{ck}, c, u, o)\\ \land \mathsf{VerCommit}(\mathsf{ck}, c, u', o')\\ \land \mathsf{VerProof}(\mathsf{vk}, x, c, \pi)\\ \land u \neq u' \land \neg R(x, u) \land R(x, u')\end{array}\right]$$

B SECURITY PROOF OF CP-SNARK COMPOSITION

In this section we provide a proof of Theorem 3.2 for $\mathsf{CP}^\land$ whose construction is the following:

```
$\mathsf{CP}^\land.\mathsf{KeyGen}(\mathsf{ck}, R^\land_{R_0,R_1}) \to (\mathsf{ek}^*, \mathsf{vk}^*)$
  $\{(\mathsf{ek}_b, \mathsf{vk}_b) \leftarrow \mathsf{CP}_b.\mathsf{KeyGen}(\mathsf{ck}, R_b)\}_{b\in\{0,1\}}$
  $\mathsf{ek}^* := (\mathsf{ek}_b)_{b\in\{0,1\}}$ ; $\mathsf{vk}^* := (\mathsf{vk}_b)_{b\in\{0,1\}}$
$\mathsf{CP}^\land.\mathsf{Prove}(\mathsf{ek}^*, x_0, x_1, (c_j)_{j\in[:3]}, (u_j)_{j\in[:3]}, (o_j)_{j\in[:3]}, \omega_0, \omega_1)$ :
  $\{\pi_b \leftarrow \mathsf{CP}_b.\mathsf{Prove}(\mathsf{ek}_b, x_b, (c_b, c_2), (u_b, u_2), (o_b, o_2), \omega_b)\}_{b\in\{0,1\}}$
  return $\pi^* := (\pi_b)_{b\in\{0,1\}}$
$\mathsf{CP}^\land.\mathsf{VerProof}(\mathsf{vk}^*, x_0, x_1, (c_j)_{j\in[:3]}, \pi^*) \to \mathsf{b}_0 \land \mathsf{b}_1$
  $\{\mathsf{b}_b \leftarrow \mathsf{CP}_b.\mathsf{VerProof}(\mathsf{vk}_b, x_b, (c_b, c_2), \pi_b)\}_{b\in\{0,1\}}$
```

Figure 4: CP-SNARK construction for AND composition

We first define relation generators and auxiliary input generators for this construction.

```
$\mathsf{Aux}^{\mathcal{RG}}(1^\lambda)$ :
  $(R_0, \mathsf{aux}_R^{(0)}) \leftarrow \mathcal{RG}_0(1^\lambda)$
  $(R_1, \mathsf{aux}_R^{(1)}) \leftarrow \mathcal{RG}_1(1^\lambda)$
  return $(R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}}$

$\mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}})$ :
  $\mathsf{aux}_Z^{(0)} \leftarrow \mathcal{Z}_0(\mathsf{ck}, R_0, \mathsf{crs}_0, \mathsf{aux}_R^{(0)})$
  $\mathsf{aux}_Z^{(1)} \leftarrow \mathcal{Z}_1(\mathsf{ck}, R_1, \mathsf{crs}_1, \mathsf{aux}_R^{(1)})$
  return $(\mathsf{aux}_Z^{(b)})_{b\in\{0,1\}}$

$\mathcal{RG}^*(1^\lambda)$ :
  $(R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}}$
      $\leftarrow \mathsf{Aux}^{\mathcal{RG}}(1^\lambda)$
  return $(R^\land_{R_0,R_1},$
      $(\mathsf{aux}_R^{(b)})_{b\in\{0,1\}})$

$\mathcal{Z}^*((\mathsf{ck}, R^\land_{R_0,R_1}), (\mathsf{ek}^*, \mathsf{vk}^*), (\mathsf{aux}_R, \mathsf{aux}'_R))$ :
  $(\mathsf{aux}_Z^{(b)})_{b\in\{0,1\}}$
      $\leftarrow \mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}})$
  return $(\mathsf{aux}_Z^{(b)})_{b\in\{0,1\}}$

$\overline{\mathcal{RG}}_b(1^\lambda)$ :
  $(R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}}$
      $\leftarrow \mathsf{Aux}^{\mathcal{RG}}(1^\lambda)$
  return $(R_b, \overline{\mathsf{aux}}_R^{(b)}$
      $:= (R_{1-b}, (\mathsf{aux}_R^{(b)})_{b\in\{0,1\}}))$

$\overline{\mathcal{Z}}_b(\mathsf{ck}, R_b, \mathsf{crs}_b, \overline{\mathsf{aux}}_R^{(b)})$ :
  Parse $\overline{\mathsf{aux}}_R$ as $(R_{1-b}, (\mathsf{aux}_R^{(b)})_{b\in\{0,1\}})$
  $\mathsf{crs}_{1-b} \leftarrow \mathsf{CP}_{1-b}.\mathsf{KeyGen}(\mathsf{ck}, R_{1-b})$
  $(\mathsf{aux}_Z^{(b)})_{b\in\{0,1\}} \leftarrow \mathsf{Aux}^{\mathcal{Z}}(\mathsf{ck}, \ldots$
      $\ldots, (\mathsf{crs}_b, R_b, \mathsf{aux}_R^{(b)})_{b\in\{0,1\}})$
  $\overline{\mathsf{aux}}_Z^{(b)} := (\mathsf{crs}_{1-b}, (\mathsf{aux}_Z^{(b)})_{b\in\{0,1\}})$
  return $\overline{\mathsf{aux}}_Z^{(b)}$
```

Figure 5: Relation and Auxiliary Input Generators for AND Composition Construction

B.1 Proof of Knowledge Soundness

We state the following lemma.

Lemma B.1. *If* Com *is computationally binding, and if* CP_b *is* $\mathsf{KSND}(\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b)$ *(where* $\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b$ *are defined in terms of* $\mathcal{RG}_b, \mathcal{Z}_b$ *in Figure 5) for* $b \in \{0, 1\}$*, then the scheme* $\mathsf{CP}^\land$ *in Figure* **??** *is* $\mathsf{KSND}(\mathcal{RG}^*, \mathcal{Z}^*)$ *where* $\mathcal{RG}^*, \mathcal{Z}^*$ *are as defined in Figure 5.*

Proof Let $\mathcal{A}^*$ be an adversary against the soundness of $\mathsf{CP}^\land$ with respect to $\mathcal{RG}^*$ and $\mathcal{Z}^*$. Now for $b \in \{0, 1\}$ consider adversary

$\mathcal{A}_b$ (defined in Figure 6) against CP_b with respect to $\overline{\mathcal{RG}}_b$ and $\overline{\mathcal{Z}}_b$. By the fact that CP_b is $\mathsf{KSND}(\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b)$ there exists an extractor $\mathcal{E}_b$ such that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\overline{\mathcal{RG}}_b, \overline{\mathcal{Z}}_b, \mathcal{A}_b, \mathcal{E}_b} = 1]$ is negligible.

We define an extractor $\mathcal{E}^*$ for $\mathsf{CP}^{\wedge}$ in Figure 6, and we claim is such that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG}^*, \mathcal{Z}^*, \mathcal{A}^*, \mathcal{E}^*} = 1]$. First observe that with overwhelming probability the values u_2 and u_2' in $\mathcal{E}^*$ are equal, conditioned to the openings being all correct for their respective commitments (i.e., conditioned to VerCommit returning 1 on each of them). In fact, if it were otherwise, we could then break the binding of Com (as done in the proof of Theorem C.1).

We now define the following notations:

$$\{\mathsf{GdCom}(c_b, u_b, o_b) := \mathsf{Com.VerCommit}(\mathsf{ck}, c_b, u_b, o_b) = 1\}_{b \in \{0,1\}}$$
$$\mathsf{GdCom}(c_2, u_2, o_2) := \mathsf{Com.VerCommit}(\mathsf{ck}, c_2, u_2, o_2) = 1$$
$$\mathsf{GdCom}(c_2, u_2', o_2') := \mathsf{Com.VerCommit}(\mathsf{ck}, c_2, u_2', o_2') = 1$$

For $b \in \{0, 1\}$, by the soundness properties of CP_b and the definition of $\mathcal{E}_b, \mathcal{E}^*$ we have that p_b, as defined below, is negligible.

$$p_b := \Pr[b^{(b)}_{\mathsf{ok}} \wedge (\neg \mathsf{GdCom}(c_b, u_b, o_b) \vee \neg \mathsf{GdCom}(c_2, u_2, o_2) \vee R_b(x_b, u_b, u_2, \omega_b) = 0)]$$

where all the symbols above are as defined in the construction of $\mathcal{E}^*$. Now we can observe that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathcal{RG}^*, \mathcal{Z}^*, \mathcal{A}^*, \mathcal{E}^*} = 1] = \dots$

$$\begin{aligned}
&= \Pr[b^{(0)}_{\mathsf{ok}} \wedge b^{(1)}_{\mathsf{ok}} \wedge (\neg \mathsf{GdCom}(c_0, u_0, o_0) \\
&\qquad \vee \neg \mathsf{GdCom}(c_1, u_1, o_1) \vee \neg \mathsf{GdCom}(c_2, u_2, o_2) \\
&\qquad \vee R_0(x_0, u_0, u_2; \omega_0) = 0 \vee R_1(x_1, u_1, u_2; \omega_1) = 0)] \\
&\leq \Pr[b^{(0)}_{\mathsf{ok}} \wedge (\neg \mathsf{GdCom}(c_0, u_0, o_0) \\
&\qquad \vee \neg \mathsf{GdCom}(c_2, u_2, o_2) \vee R_0(u_0, u_2, \omega_0) = 0)] + \\
&\quad \Pr[b^{(1)}_{\mathsf{ok}} \wedge (\neg \mathsf{GdCom}(c_1, u_1, o_1) \\
&\qquad \vee \neg \mathsf{GdCom}(c_2, u_2', o_2') \vee R_1(u_1, u_2', \omega_1) = 0)] + \mathsf{negl}(\lambda) \\
&\leq p_0 + p_1 + \mathsf{negl}(\lambda) \leq \mathsf{negl}(\lambda)
\end{aligned}$$

where in the last two inequalities we used our earlier observations on the openings of u_2 and u_2' and p_0 and p_1 being negligible respectively. □

```
$\mathcal{A}_b(\mathsf{ck}, (\mathsf{crs}_b, R_b), \overline{\mathsf{aux}}^{(b)}_R, \overline{\mathsf{aux}}^{(b)}_Z)$ :
    Parse $\overline{\mathsf{aux}}^{(b)}_R$ as $(R_{1-b}, (\mathsf{aux}^{(b)}_R)_{b \in \{0,1\}})$
    Parse $\overline{\mathsf{aux}}^{(b)}_Z$ as $(\mathsf{crs}_{1-b}, (\mathsf{aux}^{(b)}_Z)_{b \in \{0,1\}})$
    $(x_0, x_1, (c_j)_{j \in [:3]}, \pi^* := (\pi_b)_{b \in \{0,1\}})$
        $\leftarrow \mathcal{A}^*(\mathsf{ck}, (\mathsf{crs}_0, \mathsf{crs}_1, R^{\wedge}_{R_0,R_1}), (\mathsf{aux}^{(b)}_R)_{b \in \{0,1\}}, (\mathsf{aux}^{(b)}_Z)_{b \in \{0,1\}})$
    return $(x_b, c_b, c_2, \pi_b)$
$\mathcal{E}^*(\mathsf{ck}, ((\mathsf{crs}_b)_{b \in \{0,1\}}, R^{\wedge}_{R_0,R_1}), \mathsf{aux}^{(b)}_R, \mathsf{aux}^{(b)}_Z)$ :
    $\overline{\mathsf{aux}}^{(b)}_R := (R_{1-b}, (\mathsf{aux}^{(b)}_R)_{b \in \{0,1\}})$ for $b \in \{0, 1\}$
    $\overline{\mathsf{aux}}^{(b)}_Z := (\mathsf{crs}_{1-b}, (\mathsf{aux}^{(b)}_Z)_{b \in \{0,1\}})$ for $b \in \{0, 1\}$
    $((x_0, u_0, u_2), (o_0, o_2), \omega_0) \leftarrow \mathcal{E}_0(\mathsf{ck}, (\mathsf{crs}_0, R_0), \overline{\mathsf{aux}}^{(0)}_R, \overline{\mathsf{aux}}^{(0)}_Z)$
    $((x_1, u_1, u_2'), (o_1, o_2'), \omega_1) \leftarrow \mathcal{E}_1(\mathsf{ck}, (\mathsf{crs}_1, R_1), \overline{\mathsf{aux}}^{(1)}_R, \overline{\mathsf{aux}}^{(1)}_Z)$
    return $((x_b)_{b \in \{0,1\}}, (u_j)_{j \in [:3]}, (o_j)_{j \in [:3]}, (\omega_b)_{b \in \{0,1\}})$
```

Figure 6: Adversary and Extractor for Proof of Lemma B.1

B.2 Proof of Zero-Knowledge

We state the following lemma.

Lemma B.2. *If CP_b is zero-knowledge for* Com *and $\overline{\mathcal{RG}}_b$ for $b \in \{0, 1\}$, then the scheme $\mathsf{CP}^{\wedge}$ in Figure* **??** *is a zero-knowledge CP-SNARK for* Com *and $\mathcal{RG}^*$ (where relation generators are defined in Figure 5).*

Proof We construct the following two simulators for $\mathcal{RG}^*$ from simulators for $\mathsf{CP}_0, \mathsf{CP}_1$. Then ZK follows through a standard hybrid argument.

```
$\mathcal{S}^*_{\mathsf{kg}}(\mathsf{ck}, R^{\wedge}_{R_0,R_1})$
for $b \in \{0, 1\}$ :
    $(\mathsf{crs}_b, \mathsf{td}^{(b)}_{\mathsf{k}}) \leftarrow \mathcal{S}^{(b)}_{\mathsf{kg}}(\mathsf{ck}, R_b)$
$\mathsf{crs}^* := (\mathsf{crs}_b)_{b \in \{0,1\}}$
$\mathsf{td}^*_{\mathsf{k}} := (\mathsf{td}^{(b)}_{\mathsf{k}})_{b \in \{0,1\}}$
return $(\mathsf{crs}^*, \mathsf{td}^*_{\mathsf{k}})$
```

```
$\mathcal{S}^*_{\mathsf{prv}}((\mathsf{crs}_b)_{b \in \{0,1\}}, (\mathsf{td}^{(b)}_{\mathsf{k}})_{b \in \{0,1\}},$
    $(x_b)_{b \in \{0,1\}}, (c_j)_{j \in [:3]})$
for $b \in \{0, 1\}$ :
    $\pi_b \leftarrow \mathcal{S}^{(b)}_{\mathsf{prv}}(\mathsf{crs}_b, \mathsf{td}^{(b)}_{\mathsf{k}}, x_b, \dots$
        $\dots, (c_b, c_2))$
return $(\pi_b)_{b \in \{0,1\}}$
```

□

C PROOFS FOR THE GENERAL COMPILER

Theorem C.1. *Let $\mathsf{CP}.\mathcal{RG}$ be a relation generator such that $\mathsf{CP}.\mathcal{RG}_\lambda \subseteq \mathcal{R}_\lambda$, and let $\mathsf{CP}.\mathcal{Z}$ be an auxiliary input distribution. Then the scheme* CP *in Figure 1 is $\mathsf{KSND}(\mathsf{CP}.\mathcal{RG}, \mathsf{CP}.\mathcal{Z})$ and composable zero-knowledge for $\mathsf{CP}.\mathcal{RG}$ whenever: (i) $\mathsf{cc\Pi}$ is $\mathsf{ccKSND}(\mathsf{cc\Pi}.\mathcal{RG}, \mathsf{cc\Pi}.\mathcal{Z})$ and composable zero-knwledge for $\mathsf{cc\Pi}.\mathcal{RG}$, (ii) $\mathsf{CP}_{\mathsf{link}}$ is $\mathsf{KSND}(\mathsf{CP}_{\mathsf{link}}.\mathcal{RG}, \mathsf{CP}_{\mathsf{link}}.\mathcal{Z})$ and composable zero-knowledge for $\mathsf{CP}_{\mathsf{link}}.\mathcal{RG}$, where the relation generators and auxiliary input distributions $\mathsf{cc\Pi}.\mathcal{RG}, \mathsf{cc\Pi}.\mathcal{Z}, \mathsf{CP}_{\mathsf{link}}.\mathcal{RG}, \mathsf{CP}_{\mathsf{link}}.\mathcal{Z}$ are the ones in Figure 7. This result also holds when $\mathsf{cc\Pi}$ is a cc-SNARK with weak binding (Definition A.1).*

```
$\mathsf{CP}_{\mathsf{link}}.\mathcal{RG}(1^\lambda)$ :
    $(R, \mathsf{aux}_R) \leftarrow \mathsf{CP}.\mathcal{RG}(1^\lambda)$
    $\mathsf{crs}' \leftarrow \mathsf{cc\Pi}.\mathsf{KeyGen}(R)$
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    $R^\circ := (\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$
    $\mathsf{aux}^\circ_R := (\mathsf{ek}', \mathsf{vk}', R, \mathsf{aux}_R)$
return $(R^\circ, \mathsf{aux}^\circ_R)$

$\mathsf{CP}_{\mathsf{link}}.\mathcal{Z}((\mathsf{ck}, R^\circ), \mathsf{aux}^\circ_R, \mathsf{crs}^\circ)$ :
    Parse $\mathsf{aux}^\circ_R$ as $(\mathsf{ek}', \mathsf{vk}', R, \mathsf{aux}_R)$
    Get $\mathsf{ck}'$ from $R^\circ$
    Parse $\mathsf{crs}^\circ$ as $(\mathsf{ek}^\circ, \mathsf{vk}^\circ)$
    $\mathsf{ek} := (\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ); \mathsf{vk} := (\mathsf{vk}', \mathsf{vk}^\circ)$
    $\mathsf{aux}_Z \leftarrow \mathsf{CP}.\mathcal{Z}((\mathsf{ck}, R), \mathsf{aux}_R, (\mathsf{ek}, \mathsf{vk}))$
return $\mathsf{aux}^\circ_Z := \mathsf{aux}_Z$

$\mathsf{cc\Pi}.\mathcal{RG}(1^\lambda)$ :
    $\mathsf{ck} \leftarrow \mathsf{CP}.\mathsf{Setup}(1^\lambda)$
    $(R, \mathsf{aux}_R) \leftarrow \mathsf{CP}.\mathcal{RG}(1^\lambda)$
    $\mathsf{aux}'_R := (\mathsf{ck}, \mathsf{aux}_R)$
return $(R, \mathsf{aux}'_R)$

$\mathsf{cc\Pi}.\mathcal{Z}(R, \mathsf{aux}'_R, \mathsf{crs}')$ :
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    Parse $\mathsf{aux}'_R$ as $(\mathsf{ck}, \mathsf{aux}_R)$
    Build $R^\circ$ from $(\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$
    $(\mathsf{ek}^\circ, \mathsf{vk}^\circ) \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{KeyGen}(\mathsf{ck}, R^\circ)$
    $\mathsf{ek} := (\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ); \mathsf{vk} := (\mathsf{vk}', \mathsf{vk}^\circ)$
    $\mathsf{aux}_Z \leftarrow \mathsf{CP}.\mathcal{Z}((\mathsf{ck}, R), \mathsf{aux}_R, (\mathsf{ek}, \mathsf{vk}))$
return $\mathsf{aux}'_Z := (\mathsf{ek}^\circ, \mathsf{vk}^\circ, \mathsf{aux}_Z)$
```

Figure 7: Relation and Auxiliary Input Generators for Theorem C.1

C.1 Proof of Knowledge Soundness

Proof First, recall that proving the knowledge soundness of a CP-SNARK scheme CP for relation generator $\mathsf{CP}.\mathcal{RG}$ means proving the knowledge soundness of CP as a SNARK for the corresponding relation generator $\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}}$ that, we recall, honestly generates the commitment key $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and generates (R, aux_R) using $\mathsf{CP}.\mathcal{RG}$ and outputs $((\mathsf{ck}, R), \mathsf{aux}_R)$.

Our proof proceeds in the following steps.

First, assume there exists an adversary $\mathsf{CP}.\mathcal{A}$ against scheme CP that runs in the experiment $\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP}.\mathcal{Z}}$ and outputs a tuple $(x, (c_j)_{j\in[\ell]}, \pi)$ such that $\mathsf{CP}.\mathsf{VerProof}(\mathsf{vk}, x, (c_j)_{j\in[\ell]}, \pi) = 1$. Then, from such $\mathsf{CP}.\mathcal{A}$ we can build:

(1) an adversary $\mathsf{cc\Pi}.\mathcal{A}$ against $\mathsf{cc\Pi}$ that runs in the experiment $\mathsf{Game}^{\mathsf{ccKSND}}_{\mathsf{cc\Pi}.\mathcal{RG},\mathsf{cc\Pi}.\mathcal{Z}}$ (with the relation and auxiliary input generators $\mathsf{cc\Pi}.\mathcal{RG}, \mathsf{cc\Pi}.\mathcal{Z}$ defined in Fig. 7), and outputs (x, c', π');

(2) an adversary $\mathsf{CP_{link}}.\mathcal{A}$ against $\mathsf{CP_{link}}$ that runs in the experiment $\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP_{link}}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP_{link}}.\mathcal{Z}}$ (with the relation and auxiliary input generators $\mathsf{CP_{link}}.\mathcal{RG}_{\mathsf{Com}}, \mathsf{CP_{link}}.\mathcal{Z}$ defined in Fig. 7), and that outputs $(c', (c_j)_{j\in[\ell]}, \pi^\circ)$;

The two adversaries $\mathsf{cc\Pi}.\mathcal{A}, \mathsf{CP_{link}}.\mathcal{A}$ are defined below. By looking at the way their inputs are sampled in their respective games $\mathsf{Game}^{\mathsf{ccKSND}}_{\mathsf{cc\Pi}.\mathcal{RG},\mathsf{cc\Pi}.\mathcal{Z}}$ and $\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP_{link}}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP_{link}}.\mathcal{Z}}$, and how the relation and auxiliary input generators are defined, the input received by $\mathsf{CP}.\mathcal{A}$ in both simulations (the one by $\mathsf{cc\Pi}.\mathcal{A}$ and the one by $\mathsf{CP_{link}}.\mathcal{A}$) is distributed identically as the input $\mathsf{CP}.\mathcal{A}$ would receive in $\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP}.\mathcal{Z}}$.

```
$\mathsf{cc\Pi}.\mathcal{A}(R, \mathsf{crs}', \mathsf{aux}'_R, \mathsf{aux}'_Z)$:
  Parse $\mathsf{aux}'_R$ as $(\mathsf{ck}, \mathsf{aux}_R)$
  Parse $\mathsf{aux}'_Z$ as $(\mathsf{ek}^\circ, \mathsf{vk}^\circ, \mathsf{aux}_Z)$
  Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
  $\mathsf{ek} := (\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ)$
  $\mathsf{vk} := (\mathsf{vk}', \mathsf{vk}^\circ)$
  $(x, (c_j)_{j\in[\ell]}, \pi) \leftarrow \mathsf{CP}.\mathcal{A}(\ldots$
    $(\mathsf{ck}, R), (\mathsf{ek}, \mathsf{vk}), \mathsf{aux}_R, \mathsf{aux}_Z)$
  Parse $\pi$ as $(c', \pi^\circ, \pi')$
  return $(x, c', \pi')$
```

```
$\mathsf{CP_{link}}.\mathcal{A}((\mathsf{ck}, R^\circ), \mathsf{crs}^\circ, \mathsf{aux}^\circ_R, \mathsf{aux}^\circ_Z)$:
  Parse $\mathsf{aux}^\circ_R$ as $(\mathsf{ek}', \mathsf{vk}', R, \mathsf{aux}_R)$
  Parse $\mathsf{crs}^\circ$ as $(\mathsf{ek}^\circ, \mathsf{vk}^\circ)$
  Parse $\mathsf{aux}^\circ_Z$ as $\mathsf{aux}_Z$
  Parse $R^\circ$ as $(\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$
  $\mathsf{ek} := (\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ)$ ; $\mathsf{vk} := (\mathsf{vk}', \mathsf{vk}^\circ)$
  $(x, (c_j)_{j\in[\ell]}, \pi) \leftarrow \mathsf{CP}.\mathcal{A}(\ldots$
    $(\mathsf{ck}, R), (\mathsf{ek}, \mathsf{vk}), \mathsf{aux}_R, \mathsf{aux}_Z)$
  Parse $\pi$ as $(c', \pi^\circ, \pi')$
  return $(c', (c_j)_{j\in[\ell]}, \pi^\circ)$
```

Second, observe that:

- If $\mathsf{cc\Pi}$ is $\mathsf{ccKSND}(\mathsf{cc\Pi}.\mathcal{RG}, \mathsf{cc\Pi}.\mathcal{Z})$ then for every $\mathsf{cc\Pi}.\mathcal{A}$ there exists an extractor $\mathsf{cc\Pi}.\mathcal{E}$ that returns $((u'_j)_{j\in[\ell]}, o', w')$ such that $\Pr[\mathsf{Game}^{\mathsf{ccKSND}}_{\mathsf{cc\Pi}.\mathcal{RG},\mathsf{cc\Pi}.\mathcal{Z},\mathsf{cc\Pi}.\mathcal{A},\mathsf{cc\Pi}.\mathcal{E}} = 1]$ is negligible.
- If $\mathsf{CP_{link}}$ is $\mathsf{KSND}(\mathsf{CP_{link}}.\mathcal{RG}_{\mathsf{Com}}, \mathsf{CP_{link}}.\mathcal{Z})$ then for every $\mathsf{CP_{link}}.\mathcal{A}$ there exists extractor $\mathsf{CP_{link}}.\mathcal{E}$ that returns $\big((u^\circ_j)_{j\in[\ell]}, (o^\circ_j)_{j\in[\ell]}, \omega^\circ\big)$ such that the following probability is negligible $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP_{link}}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP_{link}}.\mathcal{Z},\mathsf{CP_{link}}.\mathcal{A},\mathsf{CP_{link}}.\mathcal{E}} = 1]$.

Hence, let $\mathsf{cc\Pi}.\mathcal{E}$ and $\mathsf{CP_{link}}.\mathcal{E}$ be the extractors corresponding to our adversaries $\mathsf{cc\Pi}.\mathcal{A}$ and $\mathsf{CP_{link}}.\mathcal{A}$ respectively. From the existence of the two extractors $\mathsf{cc\Pi}.\mathcal{E}$ and $\mathsf{CP_{link}}.\mathcal{E}$ we construct extractor $\mathsf{CP}.\mathcal{E}$ as below.

```
$\mathsf{CP}.\mathcal{E}((\mathsf{ck}, R), (\mathsf{ek}, \mathsf{vk}), \mathsf{aux}_R, \mathsf{aux}_Z)$ :
  Parse ek as $(\mathsf{ck}', \mathsf{ek}', \mathsf{ek}^\circ)$, vk as $(\mathsf{vk}', \mathsf{vk}^\circ)$
  $\mathsf{crs}' := (\mathsf{ck}', \mathsf{ek}'\ \mathsf{vk}')$; $\mathsf{aux}'_R := (\mathsf{ck}, \mathsf{aux}_R)$; $\mathsf{aux}'_Z := (\mathsf{ek}^\circ, \mathsf{vk}^\circ, \mathsf{aux}_Z)$
  $\big((u'_j)_{j\in[\ell]}, o', \omega'\big) \leftarrow \mathsf{cc\Pi}.\mathcal{E}(R, \mathsf{crs}', \mathsf{aux}'_R, \mathsf{aux}'_Z)$
  $R^\circ := (\mathsf{ck}', \mathcal{D}^\circ_x, \mathcal{D}^\circ_u, \mathcal{D}^\circ_\omega)$; $\mathsf{aux}^\circ_R := \big(\mathsf{ek}', \mathsf{vk}', R^*, \mathsf{aux}_R\big)$; $\mathsf{aux}^\circ_Z := \mathsf{aux}_Z$
  $\big((u^\circ_j)_{j\in[\ell]}, (o^\circ_j)_{j\in[\ell]}, \omega^\circ\big) \leftarrow \mathsf{CP_{link}}.\mathcal{E}((\mathsf{ck}, R^\circ), \mathsf{crs}^\circ, \mathsf{aux}^\circ_R, \mathsf{aux}^\circ_Z)$
return $\big((u^\circ_j)_{j\in[\ell]}, (o^\circ_j)_{j\in[\ell]}, \omega'\big)$
```

Combining the steps above, we have shown that for any CP adversary $\mathsf{CP}.\mathcal{A}$ there exists a corresponding extractor $\mathsf{CP}.\mathcal{E}$. We are left to prove that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP}.\mathcal{Z},\mathsf{CP}.\mathcal{A},\mathsf{CP}.\mathcal{E}} = 1] = \mathsf{negl}$. Recall that the output of $\mathsf{CP}.\mathcal{A}$ is of the form $\big(x, (c_j)_{j\in[\ell]}, \pi\big)$ with $\pi = (c', \pi^\circ, \pi')$, and for $\mathsf{CP}.\mathcal{E}$ is of the form $\big((u^\circ_j)_{j\in[\ell]}, (o^\circ_j)_{j\in[\ell]}, w'\big)$.

For convenience we use the following shorter notations about "good proofs" and "good commitments":

$$\begin{aligned}
&\mathsf{GdPf}(\pi') := \mathsf{cc\Pi}.\mathsf{VerProof}(\mathsf{vk}', x, c', \pi') = 1\\
&\mathsf{GdPf}(\pi^\circ) := \mathsf{CP_{link}}.\mathsf{VerProof}(\mathsf{vk}^\circ, c', (c_j)_{j\in[\ell]}, \pi^\circ) = 1\\
&\mathsf{GdCom}(c_j, u^\circ_j) := \mathsf{VerCommit}(\mathsf{ck}, c_j, u^\circ_j, o^\circ_j) = 1\\
&\mathsf{GdCom}'(c', u^\circ) := \mathsf{cc\Pi}.\mathsf{VerCommit}(\mathsf{ck}', c', (u^\circ_j)_{j\in[\ell]}, \omega^\circ) = 1\\
&\mathsf{GdCom}'(c', u') := \mathsf{cc\Pi}.\mathsf{VerCommit}(\mathsf{ck}', c', (u'_j)_{j\in[\ell]}, o') = 1\\
&R^\circ(x^\circ, u^\circ, \omega^\circ) := \mathsf{cc\Pi}.\mathsf{VerCommit}(\mathsf{ck}', x^\circ, (u^\circ_j)_{j\in[\ell]}, \omega^\circ)
\end{aligned}$$

Let us define the following events:

$$\begin{aligned}
\mathsf{bad} &:= \Big(\bigvee_{j\in[\ell]} \neg\mathsf{GdCom}(c_j, u^\circ_j) \vee \neg R(x, u^\circ, \omega')\Big)\\
\mathsf{bad}' &:= \big(\neg\mathsf{GdCom}'(c', u') \vee \neg R(x, u', \omega')\big);\\
\mathsf{bad}^\circ &:= \Big(\bigvee_{j\in[\ell]} \neg\mathsf{GdCom}(c_j, u^\circ_j) \vee \neg\mathsf{GdCom}(c', x^\circ) \vee \neg R^\circ(x^\circ, u^\circ, \omega^\circ)\Big)
\end{aligned}$$

By the knowledge soundness of $\mathsf{CP_{link}}$ and $\mathsf{cc\Pi}$ we have that $\Pr[\mathsf{GdPf}(\pi^\circ) \wedge \mathsf{bad}^\circ] = \mathsf{negl}(\lambda)$ and $\Pr[\mathsf{GdPf}(\pi') \wedge \mathsf{bad}'] = \mathsf{negl}(\lambda)$, and we abbreviate $\mathsf{n}_\lambda := \mathsf{negl}(\lambda)$ for convenience. Let us now first consider the case when cc-SNARK is binding and observe that:

$$\begin{aligned}
&\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP}.\mathcal{Z},\mathsf{CP}.\mathcal{A},\mathsf{CP}.\mathcal{E}} = 1]\\
&= \Pr[\mathsf{GdPf}(\pi') \wedge \mathsf{GdPf}(\pi^\circ) \wedge \mathsf{bad}] && (1)\\
&\le \Pr[\mathsf{GdPf}(\pi^\circ) \wedge \mathsf{bad}^\circ] + \Pr[\mathsf{GdPf}(\pi') \wedge \mathsf{bad} \wedge R^\circ(c', u^\circ, \omega^\circ) \bigwedge_{j\in[\ell]} \mathsf{GdCom}(c_j, u^\circ_j)] && (2)\\
&\le \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^\circ, \omega') \wedge R^\circ(c', u^\circ, \omega^\circ)] + \mathsf{n}_\lambda && (3)\\
&\le \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^\circ, \omega') \wedge R^\circ(c', u^\circ, \omega^\circ) \wedge (\neg R^\circ(c', u', o') \vee u' = u^\circ)] + && (4)\\
&\quad \Pr[R^\circ(c', u^\circ, \omega^\circ) \wedge R^\circ(c', u', o') \wedge u' \neq u^\circ] + \mathsf{n}_\lambda\\
&\le \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^\circ, \omega') \wedge R^\circ(c', u^\circ, \omega^\circ) \wedge (\neg R^\circ(c', u', o') \vee u' = u^\circ)] + \mathsf{n}_\lambda && (5)\\
&\le \Pr[\mathsf{GdPf}(\pi') \wedge \big((\neg R(x, u', \omega') \wedge R^\circ(c', u', \omega^\circ)) \vee\\
&\qquad (\neg R^\circ(c', u', o') \wedge \neg R(x, u^\circ, \omega') \wedge R^\circ(c', u^\circ, \omega^\circ))\big)] + \mathsf{n}_\lambda && (6)\\
&\le \Pr[\mathsf{GdPf}(\pi') \wedge \big(\neg R(x, u', \omega') \vee \neg R^\circ(c', u', o')\big)] + \mathsf{n}_\lambda && (7)\\
&\le \mathsf{negl}(\lambda) && (8)
\end{aligned}$$

Above, (1) follows by spelling out the winning condition of the experiment considering our construction of CP.VerCommit; (2) follows first partitioning over bad° and then by observing that $\neg\mathsf{bad}^\circ := R^\circ(c', u^\circ, \omega^\circ) \bigwedge_{j\in[\ell]} \mathsf{GdCom}(c_j, u^\circ_j)$; (3) follows by knowledge soundness of $\mathsf{CP_{link}}$; (4) follows after partitioning on the event

```
$\mathcal{S}_{\mathsf{kg}}(\mathsf{ck}, R)$
    $(\mathsf{crs}', \mathsf{td}'_{\mathsf{k}}) \leftarrow \mathcal{S}'_{\mathsf{kg}}(R)$
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    $(\mathsf{crs}^{\circ}, \mathsf{td}^{\circ}_{\mathsf{k}}) \leftarrow \mathcal{S}^{\circ}_{\mathsf{kg}}((\mathsf{ck}', \mathcal{D}^{\circ}_{u}, \mathcal{D}^{\circ}_{w}))$
    $\mathsf{crs} := (\mathsf{crs}^{\circ}, \mathsf{crs}'); \mathsf{td}_{\mathsf{k}} := (\mathsf{td}^{\circ}_{\mathsf{k}}, \mathsf{td}'_{\mathsf{k}})$
    return $(\mathsf{crs}, \mathsf{td}_{\mathsf{k}})$
```

```
$\mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td}_{\mathsf{k}}, x, (c_j)_{j\in[\ell]})$
    Parse crs as $(\mathsf{crs}^{\circ}, \mathsf{crs}')$
    Parse $\mathsf{td}_{\mathsf{k}}$ as $(\mathsf{td}^{\circ}_{\mathsf{k}}, \mathsf{td}'_{\mathsf{k}})$
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    $(c', \pi') \leftarrow \mathcal{S}'_{\mathsf{prv}}(\mathsf{crs}', \mathsf{td}'_{\mathsf{k}}, x)$
    $\pi^{\circ} \leftarrow \mathcal{S}^{\circ}_{\mathsf{prv}}(\mathsf{crs}^{\circ}, \mathsf{td}^{\circ}_{\mathsf{k}}, c', (c_j)_{j\in[\ell]})$
    $\pi := (c', \pi^{\circ}, \pi')$
    return $\pi$
```

Figure 8: Zero-knowledge simulators for our generic CP.

$R^{\circ}(c',u',o') \wedge u' \neq u^{\circ}$; (5) is by the binding property of the commitment of ccΠ;[18] (7) holds by using that $\Pr[((E_1 \wedge E'_1) \vee (E_2 \wedge E'_2))] \leq \Pr(E_1 \vee E_2)]$; finally, (8) follows by knowledge soundness of ccΠ.

The case of weak binding. Let us now consider the case in which ccΠ has only weak binding. In this case the commitment returned by ccΠ.Prove refers to the whole witness $w = u$, which in the previous proof means that the value ω' returned by ccΠ.$\mathcal{E}$ is empty.

To show that with this change the adversary and extractor still have negligible probability of making the knowledge soundness experiment output 1, we closely follow the analysis we already carried out by equations 1 through 8 above. We slightly deviate after (3) and obtain

$$\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP}.\mathcal{RG}_{\mathsf{Com}},\mathsf{CP}.\mathcal{Z},\mathsf{CP}.\mathcal{A},\mathsf{CP}.\mathcal{E}} = 1]$$

$$\vdots$$

$$\leq \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^{\circ}) \wedge R^{\circ}(c', u^{\circ}, \omega^{\circ})] + \mathsf{negl}(\lambda) \quad (3)$$

$$\leq \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^{\circ}) \wedge R^{\circ}(c', u^{\circ}, \omega^{\circ}) \wedge (\neg R^{\circ}(c', u', o') \vee u' = u^{\circ})] +$$
$$\Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^{\circ}) \wedge R^{\circ}(c', u^{\circ}, \omega^{\circ}) \wedge R^{\circ}(c', u', o') \wedge u' \neq u^{\circ}] + \mathsf{n}_{\lambda}$$

For the case $u' = u^{\circ}$ we proceed exactly as before. For the case $u' \neq u^{\circ}$, defining $\mathsf{comsOpen} := R^{\circ}(c', u^{\circ}, \omega^{\circ}) \wedge R^{\circ}(c', u', o')$, we have

$$\Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^{\circ}) \wedge \mathsf{comsOpen} \wedge u' \neq u^{\circ}]$$
$$\leq \Pr[\mathsf{GdPf}(\pi') \wedge \neg R(x, u^{\circ}) \wedge \mathsf{comsOpen} \wedge u' \neq u^{\circ} \wedge R(u', w')] + \mathsf{n}_{\lambda}$$
$$\leq \mathsf{negl}(\lambda)$$

where the two inequalities follow respectively from the knowledge soundness and weak binding of ccΠ. □

C.2 Proof of Zero-Knowledge

Proof Let $\mathcal{A}$ be an adversary. Since the scheme $\mathsf{CP}_{\mathsf{link}}$ is zero-knowledge there exists a simulator $\mathcal{S}^{\circ} = (\mathcal{S}^{\circ}_{\mathsf{kg}}, \mathcal{S}^{\circ}_{\mathsf{prv}})$ such that keys and proof indistinguishability hold for $\mathcal{A}$ as in Definition A.2. Similarly, since the scheme ccΠ is zero-knowledge[19] there exists a simulator $\mathcal{S}' = (\mathcal{S}'_{\mathsf{kg}}, \mathcal{S}'_{\mathsf{prv}})$ such that keys and proof indistinguishability hold for $\mathcal{A}$ as in Definition A.4. In Figure 8 we show simulators $\mathcal{S} = (\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ for the CP scheme of Figure 1, and below we argue that keys and proof indistinguishability hold for such simulators.

[18] We can do this through an adversary that would first run $\mathcal{A}$ and $\mathcal{E}$ and then return $(c', (u^{\circ}, \omega^{\circ}), (u', o'))$.

[19] We notice that for this proof we only need the zero-knowledge of ccΠ, and it does not matter if ccΠ has binding or weak binding.

```
$\mathcal{HS}_{\mathsf{kg}}(\mathsf{ck}, R)$
    $\mathsf{crs}' \leftarrow \mathsf{cc\Pi}.\mathsf{KeyGen}(R)$
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    $(\mathsf{crs}^{\circ}, \mathsf{td}^{\circ}_{\mathsf{k}}) \leftarrow$
        $\mathcal{S}^{\circ}_{\mathsf{kg}}(\mathsf{ck}', \mathcal{D}^{\circ}_{x}, \mathcal{D}^{\circ}_{u}, \mathcal{D}^{\circ}_{\omega})$
    $\mathsf{crs} := (\mathsf{crs}^{\circ}, \mathsf{crs}'); \mathsf{td}_{\mathsf{k}} := \mathsf{td}^{\circ}_{\mathsf{k}}$
return $(\mathsf{crs}, \mathsf{td}_{\mathsf{k}})$
```

```
$\mathcal{HS}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td}_{\mathsf{k}}, \mathbf{x}, \mathbf{w})$
    Parse $\mathbf{x}$ as $(x, (c_j)_{j\in[\ell]})$
    Parse crs as $(\mathsf{crs}^{\circ}, \mathsf{crs}')$
    Parse $\mathbf{w}$ as $((u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)$
    Parse $\mathsf{crs}'$ as $(\mathsf{ck}', \mathsf{ek}', \mathsf{vk}')$
    Parse $\mathsf{td}_{\mathsf{k}}$ as $(\mathsf{td}^{\circ}_{\mathsf{k}}, \mathsf{td}'_{\mathsf{k}})$
    $(c', \pi', o') \leftarrow$
        $\mathsf{cc\Pi}.\mathsf{Prove}(\mathsf{ek}', x, (u_j)_{j\in[\ell]}, \omega)$
    $\pi^{\circ} \leftarrow \mathcal{S}^{\circ}_{\mathsf{prv}}(\mathsf{crs}^{\circ}, \mathsf{td}^{\circ}_{\mathsf{k}}, c', (c_j)_{j\in[\ell]})$
return $(c', \pi^{\circ}, \pi')$
```

Figure 9: Hybrids for proof of ZK of Theorem C.1 (differences with original simulators in blue).

Proof indistinguishability: fixed arbitrary $\mathcal{A}$, x, $(c_j)_{j\in[\ell]}$, $(o_j)_{j\in[\ell]}$, $(u_j)_{j\in[\ell]}$, ω, we define three hybrids (Figure ??): $\mathcal{H}_0$, $\mathcal{H}_1$ and $\mathcal{H}_{\mathsf{sim}}$, and claim that $\mathcal{H}_0 \approx \mathcal{H}_1 \approx \mathcal{H}_{\mathsf{sim}}$, which, by definition of the hybrids, implies proof indistinguishability. We skip the proof of the claim as it follows from a standard hybrid argument.

Below we use the same notation as in Definition 3.1: define $\mathbf{x} := (x, (c_j)_{j\in[\ell]})$, $\mathbf{w} := ((u_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, \omega)$; the relation $\mathbf{R}$ over pairs $(\mathbf{x}, \mathbf{w})$ both tests commitment openings and the underlying relation R. $\mathcal{H}_0$ is defined as the probability that an adversary outputs 1 when a proof is computed through CP.Prove. This is the same as in Definition A.2 for the case in which $\mathcal{A}$ takes in input an actual proof:

$$\mathcal{H}_0 := \Pr\left[\begin{array}{l}(\mathbf{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^{\lambda});\\ (\mathsf{crs}, \mathsf{td}_{\mathsf{k}}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathbf{R});\\ \pi \leftarrow \mathsf{CP}.\mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w})\end{array} : \begin{array}{r}\mathbf{R}(\mathbf{x}, \mathbf{w}) = 1 \wedge\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi) = 1\end{array}\right]$$

In $\mathcal{H}_1$ we replace the sub-proof π° for $\mathsf{CP}_{\mathsf{link}}$ with its respective simulated version (see Figure 9 for a definition of $\mathcal{HS}_{\mathsf{prv}}$):

$$\mathcal{H}_1 := \Pr\left[\begin{array}{l}(\mathbf{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^{\lambda});\\ (\mathsf{crs}, \mathsf{td}_{\mathsf{k}}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathbf{R});\\ \pi \leftarrow \mathcal{HS}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td}_{\mathsf{k}}, \mathbf{x}, \mathbf{w})\end{array} : \begin{array}{r}\mathbf{R}(\mathbf{x}, \mathbf{w}) = 1 \wedge\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi) = 1\end{array}\right]$$

We define $\mathcal{H}_{\mathsf{sim}}$ as the simulated proof output as in the standard zero-knowledge experiment (Definition A.2). We point out that the only change from $\mathcal{H}_{\ell}$ consists in replacing the actual proof for ccΠ with its simulated version:

$$\mathcal{H}_{\mathsf{sim}} := \Pr\left[\begin{array}{l}(\mathbf{R}, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^{\lambda});\\ (\mathsf{crs}, \mathsf{td}_{\mathsf{k}}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathbf{R});\\ \pi \leftarrow \mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td}_{\mathsf{k}}, \mathbf{x})\end{array} : \begin{array}{r}\mathbf{R}(\mathbf{x}, \mathbf{w}) = 1 \wedge\\ \mathcal{A}(\mathsf{crs}, \mathsf{aux}_R, \pi) = 1\end{array}\right]$$

Figure 10: Hybrids for proof indistinguishability of CP.

Keys indistinguishability: we proceed by a standard hybrid argument. Consider the hybrid simulator $\mathcal{HS}_{\mathsf{kg}}$ in Figure 9. By construction of $\mathcal{HS}_{\mathsf{kg}}$ and the keys indistinguishability for $\mathcal{S}'_{\mathsf{kg}}, \mathcal{S}^{\circ}_{\mathsf{kg}}$ we have that:

$$\Pr\left[\begin{array}{l}(\mathsf{ck}, R, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^\lambda);\\ \mathsf{crs} \leftarrow \mathsf{CP.KeyGen}(\mathsf{ck}, R) = 1\end{array} : \mathcal{A}(\mathsf{ck}, \mathsf{crs}, \mathsf{aux}_R) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(\mathsf{ck}, R, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{HS}_{\mathsf{kg}}(\mathsf{ck}, R)\end{array} : \mathcal{A}(\mathsf{ck}, \mathsf{crs}, \mathsf{aux}_R) = 1\right]$$
$$\approx \Pr\left[\begin{array}{l}(\mathsf{ck}, R, \mathsf{aux}_R) \leftarrow \mathcal{RG}_{\mathsf{Com}}(1^\lambda);\\ (\mathsf{crs}, \mathsf{td_k}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathsf{ck}, R)\end{array} : \mathcal{A}(\mathsf{ck}, \mathsf{crs}, \mathsf{aux}_R) = 1\right]$$

□

D SUPPLEMENTARY RESULTS ON $\mathsf{CP_{LINK}}$

This section contains the security proof and an extension of the $\mathsf{CP_{link}}$ scheme.

D.1 Proof of $\mathsf{CP_{link}}$ Security

In the following theorem we show that $\mathsf{CP_{link}}$ is knowledge-sound and zero-knowledge assuming so is $\mathsf{ss\Pi}$.

Theorem D.1. *Let $\mathsf{CP_{link}}.\mathcal{RG}$ be a relation generator and $\mathsf{CP_{link}}.\mathcal{Z}$ be an auxiliary input distribution. If $\mathsf{ss\Pi}$ is $\mathsf{KSND}(\mathsf{ss\Pi}.\mathcal{RG}, \mathsf{ss\Pi}.\mathcal{Z})$ where $\mathsf{ss\Pi}.\mathcal{RG}$ is a relation generator as in Figure 11 and $\mathsf{ss\Pi}.\mathcal{Z} = \mathsf{CP_{link}}.\mathcal{Z}$, then the CP-SNARK construction $\mathsf{CP_{link}}$ given above is $\mathsf{KSND}(\mathsf{CP_{link}}.\mathcal{RG}, \mathsf{CP_{link}}.\mathcal{Z})$. Furthermore, if $\mathsf{ss\Pi}$ is composable ZK for $\mathsf{ss\Pi}.\mathcal{RG}$, then $\mathsf{CP_{link}}$ is composable ZK for $\mathsf{CP_{link}}.\mathcal{RG}$.*

```
$\mathsf{ss\Pi}.\mathcal{RG}(1^\lambda) \to ([\mathbf{M}]_1, \mathsf{aux}_R^\circ)$
$[\vec{h}]_1 \leftarrow \mathsf{Ped.Setup}(1^\lambda)$ using distribution $\mathcal{D}$
$(R^\circ, \mathsf{aux}_R^\circ) \leftarrow \mathsf{CP_{link}}.\mathcal{RG}(1^\lambda)$ ; Define $[\mathbf{M}]_1$ from $[\vec{h}]_1$, $R^\circ$
```

Figure 11: Relation generator on which we base $\mathsf{ss\Pi}$ security.

Knowledge Soundness. Consider an arbitrary adversary $\mathcal{A}$ against $\mathsf{CP_{link}}$. From $\mathcal{A}$ we can construct an adversary $\mathcal{A}'$ against $\mathsf{ss\Pi}$ as follows.

```
$\mathcal{A}'([\mathbf{M}]_1, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ :
Extract $[\vec{f}]_1$, $[\vec{h}]_1$ from $[\mathbf{M}]_1$
$([\vec{x}]_1, \pi) \leftarrow$
    $\mathcal{A}(([\vec{h}]_1, R^\circ), \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$
Parse $[\vec{x}]_1$ as $((c_j)_{j\in[\ell]}, c')$
return $(c', (c_j)_{j\in[\ell]}, \pi)$
```

```
$\mathcal{E}(([\vec{h}]_1, R^\circ), \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$ :
Compute matrix $[\mathbf{M}]_1$
$\vec{w} \leftarrow$
    $\mathsf{ss\Pi}.\mathcal{E}([\mathbf{M}]_1, \mathsf{crs}, \mathsf{aux}_R, \mathsf{aux}_Z)$
Parse $\vec{w}$ as $((o_j)_{j\in[\ell]}, o', (\vec{u}_j)_{j\in[\ell]})$
return $((\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}, o')$
```

By knowledge soundness of $\mathsf{ss\Pi}$, for every such $\mathcal{A}'$ there is an extractor $\mathsf{ss\Pi}.\mathcal{E}$, that we can use to build the above extractor $\mathcal{E}$ for $\mathcal{A}$. In particular, the knowledge soundness of $\mathsf{ss\Pi}$ and the definition of $\mathbf{M}$ give us that $\mathcal{E}$'s output is such that:

$$\Pr\left[\begin{array}{l}\mathsf{ss\Pi.VerProof}(\mathsf{vk}, (c_j)_{j\in[\ell]}, c') = 1 \quad \wedge\\ \Big(\bigvee_{j\in[\ell]} \big(c_j \neq (o_j, \vec{u}_j^\top)\cdot[\vec{h}_{[0,n_j]}]_1\big) \quad \vee\\ \quad c' \neq (o', \vec{u}_1^\top, \ldots, \vec{u}_\ell^\top)\cdot[\vec{f}]_1\Big)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

Hence we can conclude that $\Pr[\mathsf{Game}^{\mathsf{KSND}}_{\mathsf{CP_{link}}.\mathcal{RG}, \mathsf{CP_{link}}.\mathcal{Z}, \mathcal{A}, \mathcal{E}} = 1] = \Pr[\mathsf{GdPf} \ \wedge \ (\mathsf{BadComm} \vee \mathsf{BadRel})] \leq \mathsf{negl}(\lambda)$ using: $\mathsf{GdPf} := \mathsf{CP_{link}.VerProof}(\mathsf{vk}, c', (c_j)_{j\in[\ell]}, \pi) = 1$, $\mathsf{BadComm} := \bigvee_{j\in[\ell]} c_j \neq (o_j, \vec{u}_j^\top)\cdot[\vec{h}_{[0,n_j]}]_1$, $\mathsf{BadRel} := c' \neq (o', \vec{u}_1^\top, \ldots, \vec{u}_\ell^\top)\cdot[\vec{f}]_1$.

Zero-Knowledge. From the zero-knowledge property of $\mathsf{ss\Pi}$ we know there exists a simulator $\mathsf{ss\Pi}.\mathcal{S} = (\mathsf{ss\Pi}.\mathcal{S}_{\mathsf{kg}}, \mathsf{ss\Pi}.\mathcal{S}_{\mathsf{prv}})$ such that keys and proof indistinguishability hold for an arbitrary $\mathcal{A}$ as in Definition A.2. We now define the following key simulator $\mathsf{CP_{link}}.\mathcal{S}_{\mathsf{kg}}$ such that $\mathsf{CP_{link}}.\mathcal{S}_{\mathsf{kg}}([\vec{h}]_1, R^\circ) := \mathsf{ss\Pi}.\mathcal{S}_{\mathsf{kg}}([\mathbf{M}]_1)$. Keys indistinguishability follows directly from the assumption on $\mathsf{ss\Pi}.\mathcal{S}_{\mathsf{kg}}$. Analogously, we obtain proof indistinguishability by defining a proof simulator $\mathsf{CP_{link}}.\mathcal{S}_{\mathsf{prv}}$ such that $\mathsf{CP_{link}}.\mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td_k}, c', (c_j)_{j\in[\ell]}) := \mathsf{ss\Pi}.\mathcal{S}_{\mathsf{prv}}(\mathsf{crs}, \mathsf{td_k}, [\vec{x}]_1)$, with $[\vec{x}]_1 = ((c_j)_{j\in[\ell]}, c')$.

D.2 An extension of $\mathsf{CP_{link}}$ for Prefixes of a Committed Vector

Fixed a security parameter λ (and the bilinear group setting for λ as well), R°_{pre} is a relation over $(\mathcal{D}_x \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_\ell \times \mathcal{D}_\omega)$, where $\mathcal{D}_x = \mathbb{G}_1$, $\mathcal{D}_\omega = \mathbb{Z}_q^{n_\omega+1}$ and $\mathcal{D}_j = \mathbb{Z}_q^{n_j}$ for some n_j such that $n_\omega + \sum_j n_j = m$. R°_{pre} is parametrized by a commitment key $[\vec{f}]_1 \in \mathbb{G}_1^{m+1}$, and is defined as:

$$R^\circ_{\mathsf{pre}}\big(c', (\vec{u}_j)_{j\in[\ell]}, (\vec{u}_{\ell+1}, o')\big) = 1 \iff c' \overset{?}{=} (o', \vec{u}_1^\top, \ldots, \vec{u}_{\ell+1}^\top)\cdot[\vec{f}]_1$$

Similarly to the case of R°, this relation can be expressed as a linear subspace relation, $R_{\mathbf{M}}([\vec{x}]_1, \vec{w})$, where $\mathbf{M}, \vec{x}, \vec{w}$ are as follows:

$$\overbrace{\begin{bmatrix} c_1 \\ \vdots \\ c_\ell \\ c' \end{bmatrix}}^{\vec{x}} = \overbrace{\begin{bmatrix} h_0 & 0 & \ldots & 0 & 0 & \vec{h}_{[1,n_1]} & 0 & \ldots & 0 & 0 \\ 0 & h_0 & \ldots & 0 & 0 & 0 & \vec{h}_{[1,n_2]} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & h_0 & 0 & 0 & 0 & \ldots & \vec{h}_{[1,n_\ell]} & 0 \\ 0 & 0 & \ldots & 0 & f_0 & \vec{f}_{[1,n_1]} & \vec{f}_{[n_1+1,n_2]} & \cdots & \vec{f}_{[n_{\ell-1}+1,n_\ell]} & \vec{f}_{[n_\ell+1,n_{\ell+1}]} \end{bmatrix}}^{\mathbf{M}} \overbrace{\begin{pmatrix} o_1 \\ \vdots \\ o_\ell \\ o' \\ \vec{u}_1 \\ \vdots \\ \vec{u}_{\ell+1} \end{pmatrix}}^{\vec{w}}$$

Given the above encoding, it is straightforward to extend our scheme $\mathsf{CP_{link}}$ to support the relation R°_{pre} instead of R°.

E DESCRIPTION OF $\mathsf{CP^{PED}_{LIN}}$

The description of our scheme $\mathsf{CP^{Ped}_{lin}}$ follows:

$\mathsf{CP^{Ped}_{lin}.KeyGen}(\mathsf{ck}, R^{\mathsf{lin}}_{\mathbf{F}})$: parse $\mathsf{ck} = [\vec{h}]_1 \in \mathbb{G}_1^{m+1}$. Use $[\vec{h}]_1$ and $R^{\mathsf{lin}}_{\mathbf{F}}$ to build a matrix $[\mathbf{M}]$ as in equation (2). Compute $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{ss\Pi.KeyGen}([\mathbf{M}]_1)$ and return $(\mathsf{ek}, \mathsf{vk})$.

$\mathsf{CP^{Ped}_{lin}.Prove}(\mathsf{ek}, \vec{x}, (c_j)_{j\in[\ell]}, (\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]})$: define $[\vec{x}']_1$ and $\vec{w}'$ as in equation (2), and return $\pi \leftarrow \mathsf{ss\Pi.Prove}(\mathsf{ek}, [\vec{x}']_1, \vec{w}')$.

$\mathsf{CP^{Ped}_{lin}.VerProof}(\mathsf{vk}, \vec{x}, (c_j)_{j\in[\ell]}, \pi)$: set $[\vec{x}']_1$ as in (2) and return $\mathsf{ss\Pi.VerProof}(\mathsf{vk}, [\vec{x}']_1, \pi)$.

We state the following theorem. We omit the proof, which is essentially the same as that of Theorem D.1.

Theorem E.1. *Let $\mathbf{F} \in \mathbb{Z}_q^{N\times m}$ be a matrix from a distribution $\mathcal{D}_{\mathsf{mtx}}$, and $\mathcal{Z}$ be an auxiliary input distribution. If $\mathsf{ss\Pi}$ is $\mathsf{KSND}(\mathsf{ss\Pi}.\mathcal{RG}, \mathcal{Z})$ where $\mathsf{ss\Pi}.\mathcal{RG}$ is a relation generator that samples ck and $\mathbf{F} \leftarrow \mathcal{D}_{\mathsf{mtx}}$, then the CP-SNARK construction $\mathsf{CP^{Ped}_{lin}}$ given* *above is* $\mathsf{KSND}(\mathcal{D}_{\mathsf{mtx}}, \mathcal{Z})$*. Furthermore, if* $\mathsf{ss}\Pi$ *is composable ZK for* $\mathsf{ss}\Pi.\mathcal{RG}$*, then* $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ *is composable ZK for* $\mathcal{D}_{\mathsf{mtx}}$.

F A ZKSNARK FOR LINEAR SUBSPACES

Here we recall the QA-NIZK scheme for linear subspaces Π'_{as} of Kiltz and Wee [48], in the MDDH setting where $k = 1$.

$\mathsf{ss}\Pi.\mathsf{KeyGen}([\mathbf{M}]_1 \in \mathbb{G}_1^{l \times t})$: $\vec{k} \leftarrow\$ \mathbb{Z}_q^l$, $a \leftarrow\$ \mathbb{Z}_q$; $\mathbf{P} := \mathbf{M}^\top \vec{k}$; $\mathrm{C} := a \cdot \vec{k}$; **return** $\mathsf{ek} := [\mathbf{P}]_1 \in \mathbb{G}_1^t$; $\mathsf{vk} := ([\mathrm{C}]_2, [a]_2) \in \mathbb{G}_2^l \times \mathbb{G}_2$.

$\mathsf{ss}\Pi.\mathsf{Prove}(\mathsf{ek}, [\vec{x}]_1, \vec{w})$: **return** $[\pi]_1 \leftarrow \vec{w}^\top [\mathbf{P}]_1 \in \mathbb{G}_1$;

$\mathsf{ss}\Pi.\mathsf{VerProof}(\mathsf{vk}, [\vec{x}]_1, [\pi]_1)$:] check that $[\vec{x}]_1^\top \cdot [\mathrm{C}]_2 = [\pi]_1 \cdot [a]_2$.

$\mathsf{ss}\Pi.\mathcal{S}_{\mathsf{kg}}(1^\lambda)$: run as $\mathsf{ss}\Pi.\mathsf{KeyGen}$ and output $\mathsf{td}_\mathsf{k} = \vec{k}$ and $(\mathsf{ek}, \mathsf{vk})$.

$\mathsf{ss}\Pi.\mathcal{S}_{\mathsf{prv}}(\mathsf{td}_\mathsf{k}, [\vec{x}]_1)$: **return** $[\pi]_1 \leftarrow \vec{k}^\top [\vec{x}]_1$.

In the following theorem we prove the knowledge soundness of the scheme given above. The proof holds under the discrete logarithm assumption in the algebraic group model of [32]; this can also be interpreted as a proof in the (bilinear) generic group model. We also note that a similar proof about the use of this scheme in a non-falsifiable setting [48] also appeared in [30].

Theorem F.1. *Assume that* $\mathcal{D}_{\mathsf{mtx}}$ *is a witness sampleable matrix distribution. Then, under the discrete logarithm assumption, in the algebraic group model, the QA-NIZK* Π'_{as} *in [48] (in the MDDH setting* $k = 1$*) is a knowledge-sound SNARK for linear subspace relations with matrices from* $\mathcal{D}_{\mathsf{mtx}}$.

Proof Consider an algebraic adversary $\mathcal{A}$ against the knowledge soundness of $\mathsf{ss}\Pi$. Its input consists of the matrix $[\mathbf{M}]_1$ and the associated auxiliary input aux, along with the common reference string $[\mathbf{P}]_1, [\mathrm{C}]_2, [a]_2$. Let $[\vec{z}]_1$ be a vector that contains $\mathbf{M}$ and the portion of aux that has elements from the group $\mathbb{G}_1$, and also assume $[\vec{z}]$ includes $[1]_1$. $\mathcal{A}$ returns a pair $([\vec{x}]_1, [\pi]_1)$ along with coefficients that "explain" these elements as linear combinations of its input in the group $\mathbb{G}_1$. Let these coefficients be:

$$\begin{aligned} [\vec{x}]_1 &:= \mathbf{X}_0 \, [\mathbf{P}]_1 + \mathbf{X}_1 \, [\vec{z}]_1 = \mathbf{X}_0 \, [\mathbf{M}^\top \vec{k}]_1 + \mathbf{X}_1 \, [\vec{z}]_1 \\ [\pi]_1 &:= \vec{\pi}_0^\top [\mathbf{P}]_1 + \vec{\pi}_1^\top [\vec{z}]_1 = \vec{\pi}_0^\top [\mathbf{M}^\top \vec{k}]_1 + \vec{\pi}_1^\top [\vec{z}]_1 \end{aligned}$$

We define the extractor $\mathcal{E}$ to be the algorithm that runs the algebraic $\mathcal{A}$ and returns $\vec{w} := \vec{\pi}_0$, i.e., the coefficients of $[\pi]_1$ corresponding to $\mathbf{P}$. Next, we have to show that the probability that the output of $(\mathcal{A}, \mathcal{E})$ satisfies verification while $\vec{x} \neq \mathbf{M}\vec{w}$ is negligible. In other words, assume that the output of $\mathcal{A}$ is such that:

$$[\vec{x}]_1^\top \cdot [a \cdot \vec{k}]_2 = [\pi]_1 \cdot [a]_2 \quad \text{and} \quad [\vec{x}]_1 \neq [\mathbf{M}]_1 \vec{\pi}_0$$

If $\mathcal{A}$ returns such a tuple with non-negligible probability, we show how to build an algorithm $\mathcal{B}$ that on input $([\vec{k}]_1, [\vec{k}]_2)$ outputs nonzero elements $\mathbf{A} \in \mathbb{Z}_q^{l \times l}$, $\vec{b} \in \mathbb{Z}_q^l$, $c \in \mathbb{Z}_q$ such that

$$\vec{k}^\top \mathbf{A} \, \vec{k} + \vec{k}^\top \vec{b} + c = 0$$

Such a $\mathcal{B}$ can in turn be reduced to an algorithm $\mathcal{B}'$ that solves discrete log, i.e., on input $([\alpha]_1, [\alpha]_2)$ return α.

Algorithm $\mathcal{B}([\vec{k}]_1, [\vec{k}]_2)$ proceeds as follows. First, it uses $\mathcal{D}_{\mathsf{mtx}}$ to sample $([\mathbf{M}]_1, \mathsf{aux})$ along with its $\mathbb{G}_1$ witness (i.e., a vector $\vec{z}$ of entries in $\mathbb{Z}_q$). Second, it samples $a \leftarrow\$ \mathbb{Z}_q$ and runs $\mathcal{A}([\vec{z}, \mathbf{P}]_1, [a, a \cdot \vec{k}]_2)$ (notice that $\mathcal{A}$'s input can be efficiently simulated). Third, once received the output of $\mathcal{A}$, $\mathcal{B}$ sets $\mathbf{A} := \mathbf{X}_0 \, \mathbf{M}^\top$, $\vec{b} := \mathbf{X}_1 \vec{z} - \mathbf{M} \, \vec{\pi}_0$ and $c = -\vec{\pi}_1^\top \vec{z}$. Notice that

$$\begin{aligned} \vec{k}^\top \mathbf{A} \vec{k} + \vec{k}^\top \vec{b} + c &= \vec{k}^\top \, \mathbf{X}_0 \, \mathbf{M}^\top \vec{k} + \vec{k}^\top \mathbf{X}_1 \vec{z} - \vec{k}^\top \mathbf{M} \, \vec{\pi}_0 - \vec{\pi}_1^\top \vec{z} \\ &= \vec{k}^\top \, \mathbf{X}_0 \, \mathbf{M}^\top \vec{k} + \vec{k}^\top \mathbf{X}_1 \vec{z} - \pi \\ &= \vec{k}^\top \vec{x} - \pi = 0 \end{aligned}$$

Also, one among $\mathbf{A}, \vec{b}$ and c must be nonzero. Indeed, if they are all zero then $\mathbf{X}_1 \vec{z} - \mathbf{M} \vec{\pi}_0 = 0$, that is $\vec{x} = \mathbf{M} \vec{\pi}_0$, which contradicts our assumption on $\mathcal{A}$'s output.

To finish the proof, we show how the above problem can be reduced to discrete log in asymmetric groups, i.e., $\mathcal{B}'$ on input $([\alpha]_1, [\alpha]_2)$ returns α. $\mathcal{B}'$ samples $\vec{r}, \vec{s} \in \mathbb{Z}_q^l$ and implicitly sets $\vec{k} := \alpha \cdot \vec{r} + \vec{s}$. It is easy to see that $([\vec{k}]_1, [\vec{k}]_2)$ can be efficiently simulated with a distribution identical to the one expected by $\mathcal{B}$. Next, given a solution $(\mathbf{A}, \vec{b}, c)$ such that $\vec{k}^\top \mathbf{A} \, \vec{k} + \vec{k}^\top \vec{b} + c = 0$ one can find $a', b', c' \in \mathbb{Z}_q$ such that:

$$\begin{aligned} 0 &= (\alpha \vec{r} + \vec{s})^\top \, \mathbf{A} \, (\alpha \vec{r} + \vec{s}) + (\alpha \vec{r} + \vec{s})^\top \vec{b} + c \\ &= \alpha^2 (\vec{r}^\top \mathbf{A} \vec{r}) + \alpha \cdot (\vec{r}^\top \mathbf{A} \vec{s} + \vec{s}^\top \mathbf{A} \vec{r} + \vec{r}^\top \vec{b}) + (\vec{s}^\top \mathbf{A} \vec{s} + \vec{s}^\top \vec{b} + c) \\ &= a' \alpha^2 + b' \alpha + c' \end{aligned}$$

In particular, with overwhelming probability (over the choice of $\vec{s}$ that is information theoretically hidden from $\mathcal{B}$'s view) $c' \neq 0$. From this solution $\mathcal{B}'$ can solve the system and extract α. ☐

G A CONSTRUCTION OF PolyCom AND CP$_{\mathsf{POLY}}$ FROM ZK-VSQL

We show a pairing-based construction of the commitment PolyCom and CP-SNARK $\mathsf{CP}_{\mathsf{poly}}$ that are "extracted" from the verifiable polynomial delegation scheme of Zhang et al. [71]. Basically, we separate the algorithms related to committing from the ones related to proving and verifying evaluations of committed polynomials. Except for that, the only noticeable difference is that in our case we can prove that c_y opens to $y = f(\vec{x})$ (with respect to c_f which opens to f) for a given c_y instead of one that is freshly generated at proving time. As we show below, this difference would matter only for zero-knowledge, for which we give a proof a slightly different than the one in [71].

$\mathsf{Setup}(1^\lambda)$**:** let $\mathcal{F}$ be μ-variate polynomials of degree d in each variable. Sample $\alpha, \beta, s_1, \ldots, s_{\mu+1} \leftarrow\$ \mathbb{Z}_q$ uniformly at random, compute $\mathbb{P} = [\prod_{i \in W} s_i, \alpha \prod_{i \in W} s_i]_1$, and output $\mathsf{ck} = (\mathbb{P}, [s_{\mu+1}, \alpha s_{\mu+1}, \beta s_{\mu+1}]_1, [\alpha, \beta, s_1, \ldots, s_{\mu+1}]_2)$.

$\mathsf{ComPoly}(\mathsf{ck}, f) \to (c_f, o_f)$**:** sample $o_f \leftarrow\$ \mathbb{Z}_q$, compute $c_{f,1} = [f(s_1, \ldots, s_\mu) + o_f s_{\mu+1}]_1$, $c_{f,2} = [\alpha(f(s_1, \ldots, s_\mu) + o_f s_{\mu+1})]_1$ and output $c_f = (c_{f,1}, c_{f,2})$.

$\mathsf{ComVal}(\mathsf{ck}, y) \to (c_y, o_y)$**:** sample $o_y \leftarrow\$ \mathbb{Z}_q$, compute $c_{y,1} = [y + o_y s_{\mu+1}]_1$, $c_{y,2} = [\beta(y + o_f s_{\mu+1})]_1$ and output $c_y = (c_{y,1}, c_{y,2})$.

$\mathsf{CheckCom}(\mathsf{ck}, c)$**:** we assume one knows the type for which c was created. If type = pol, output 1 iff $c_1 \cdot [\alpha]_2 = c_2 \cdot [1]_2$. If type = val, output 1 iff $c_1 \cdot [\beta]_2 = c_2 \cdot [1]_2$.

$\mathsf{VerCommit}(\mathsf{ck}, c, f, o) \to b$ **:** output $c_1 \stackrel{?}{=} [f(s_1, \ldots, s_\mu) + o s_{\mu+1}]_1$.

Theorem G.1 ([71]). *Under the* $(\mu + 1)d$*-Strong Diffie Hellmand and the* (d, μ)*-Extended Power Knowledge of Exponent assumptions* *(see [71]),* PolyCom *is an extractable trapdoor polynomial commitment.*

The proof of the theorem follows from Theorem 1 in [71]. The only property that is not proved there is the trapdoor property, which is however straightforward to see if one considers a simulator $\mathcal{S}_{\mathsf{ck}}$ that sets the values $\alpha, \beta, s_1, \ldots, s_{\mu+1}$ as trapdoor.

Next, we show a CP-SNARK for polynomial evaluation relations R_{poly}:

$\mathsf{CP_{poly}.KeyGen(ck)}$**:** set $\mathsf{ek} := \mathsf{ck}$ and $\mathsf{vk} := ([\alpha, \beta, s_1, \ldots, s_{\mu+1}]_2)$

$\mathsf{CP_{poly}.Prove}(\mathsf{ek}, \vec{x}, f, y, o_f, o_y)$**:** sample $o_1, \ldots, o_\mu \leftarrow\$ \mathbb{Z}_q$; find polynomials q_i such that $f(Z_1, \ldots, Z_\mu) + o_f Z_{\mu+1} - (y + o_y Z_{\mu+1}) = \sum_{i=1}^{\mu} (Z_i - x_i)(q_i(Z_i, \ldots, Z_\mu) + o_i Z_{\mu+1}) + X_{\mu+1}(o_f - o_y - \sum_{i=1}^{\mu} o_i(Z_i - x_i))$.
For $i = 1$ to μ, compute $c_i := (c_{i,1}, c_{i,2}) = [q_i(s_1, \ldots, s_\mu) + o_i s_{\mu+1}, \alpha(q_i(s_1, \ldots, s_\mu) + o_i s_{\mu+1})]_1$, $c_{\mu+1} := (c_{\mu+1,1}, c_{\mu+1,2}) = [o_f - o_y - \sum_{i=1}^{\mu} o_i(s_i - x_i), \alpha(o_f - o_y - \sum_{i=1}^{\mu} o_i(s_i - x_i))]_1$. Output $\pi := (c_1, \ldots, c_{\mu+1})$.

$\mathsf{CP_{poly}.VerProof}(\mathsf{vk}, \vec{x}, c_f, c_y, \pi)$**:** parse $\pi := (c_1, \ldots, c_{\mu+1})$, output $\mathsf{CheckCom}(\mathsf{vk}, c_f) \wedge \mathsf{CheckCom}(\mathsf{vk}, c_y) \bigwedge_{i=1}^{\mu+1} \mathsf{CheckCom}(\mathsf{vk}, c_i)$ and $(c_{f,1} - c_{y,1}) \cdot [1]_2 = c_{\mu+1,1} \cdot [s_{\mu+1}]_2 \sum_{i=1}^{\mu} c_{i,1} \cdot [(s_i - x_i)]_2$.

Theorem G.2 ([71]). *Under the* $(\mu+1)d$*-Strong Diffie Hellmand and the* (d, μ)*-Extended Power Knowledge of Exponent assumptions (see [71]),* $\mathsf{CP_{poly}}$ *is a zero-knowledge CP-SNARK for* R_{poly}.

Correctness and knowledge soundness are immediate from Theorem 1 in [71]. The only difference is in the zero-knowledge property. For this, consider the following proof simulator algorithm, $\mathcal{S}_{\mathsf{prv}}(\mathsf{td}, \vec{x}, c_f, c_y)$: for $i = 1$ to μ, sample $c_{i,1} \leftarrow\$ \mathbb{G}_1$ and compute $c_{i,2} = \alpha \cdot c_{i,1}$. Next, compute $c_{\mu+1,1}$ such that $(c_{f,1} - c_{y,1}) \cdot [1]_2 = c_{\mu+1,1} \cdot [s_{\mu+1}]_2 + \sum_{i=1}^{\mu} c_{i,1} \cdot [(s_i - x_i)]_2$ holds and set $c_{\mu+1,2} \leftarrow \beta \cdot c_{\mu+1,1}$. It is straightforward to check that proofs created by $\mathcal{S}_{\mathsf{prv}}$ are identically distributed to the ones returned by $\mathsf{CP_{poly}.Prove}$.

H MORE ON CP-SNARKS FOR POLYCOM

In this section we present more CP-SNARKs for PolyCom, for which we first give formal definitions:

Polynomial Commitments. The specific commitment scheme we consider here is the polynomial commitment underlying the verifiable polynomial delegation (VPD) scheme of Zhang et al. [71]. In a nutshell, a VPD allows one to commit to multivariate polynomials and later prove their evaluations (also committed) at a public point. Here we show that their VPD scheme can be seen as a CP-SNARK for such polynomial commitment, for relations encoding polynomial evaluations. Namely, whereas in [71] VPD is presented as a single primitive, here we separate the commitment scheme from the argument system. With this simple change (together with a slightly stronger zero-knowledge notion) we can use our composition results to argue security when commitments are reused across different proofs.

Formally, we consider a commitment scheme whose message space $\mathcal{D}$ includes both values in a finite field $\mathbb{F}$ and a class $\mathcal{F}$ of polynomials with coefficients in $\mathbb{F}$, with μ variables and maximal degree d in each variable. We denote these partitions of $\mathcal{D} = \mathbb{F} \cup \mathcal{F}$ as $\mathcal{D}_{\mathsf{pol}} = \mathcal{F}$ and $\mathcal{D}_{\mathsf{val}} = \mathbb{F}$ and we use a flag type to differentiate between them so that $f \in \mathcal{F}$ when type = pol, and $f \in \mathbb{F}$ when type = val.[20] In addition to satisfying the notion of Definition 2.2 we assume the scheme to be knowledge extractable and to have a trapdoor generation. For convenience, we summarize its definition below.

Definition H.1 (Extractable Trapdoor Polynomial Commitments). An extractable trapdoor polynomial commitment scheme for a class of polynomials $\mathcal{F}$ is a tuple of algorithms PolyCom = (Setup, Commit, CheckCom, VerCommit) that work as follows.

$\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: takes the security parameter and outputs a commitment key ck.

$\mathsf{Commit}(\mathsf{ck}, f, \mathsf{type}) \to (c_f, o_f)$: takes the commitment key ck, a flag type $\in \{\mathsf{pol}, \mathsf{val}\}$ and an element $f \in \mathcal{D}_{\mathsf{type}}$, and outputs a commitment c_f and an opening o_f. We use $\mathsf{ComPoly}(\mathsf{ck}, \cdot)$ and $\mathsf{ComVal}(\mathsf{ck}, \cdot)$ as shorthands for $\mathsf{Commit}(\mathsf{ck}, \cdot, \mathsf{pol})$ and $\mathsf{Commit}(\mathsf{ck}, \cdot, \mathsf{val})$ respectively. We also assume that type is part of c_f, namely it is not hidden.

$\mathsf{CheckCom}(\mathsf{ck}, c) \to b$: takes as input a commitment c and accepts it as valid ($b = 1$) or not ($b = 0$).

$\mathsf{VerCommit}(\mathsf{ck}, c_f, f, o_f) \to b$: takes as input commitment c, element $f \in \mathcal{D}$ and opening o_f, and accepts ($b = 1$) or rejects ($b = 0$).

PolyCom must satisfy correctness, binding and (perfect) hiding as in Definition 2.2 (with the additional requirements that correctness also implies that CheckCom accepts, and binding holds for adversarial commitments that are accepted by CheckCom). In addition PolyCom must satisfy the trapdoor and extractability properties defined below.

Trapdoor. There exists three algorithms $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathcal{S}_{\mathsf{ck}}(1^\lambda)$, $(c, st) \leftarrow \mathsf{TdCom}(\mathsf{td}, \mathsf{type})$ and $o \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, c, f)$ such that: the distribution of the commitment key returned by $\mathcal{S}_{\mathsf{ck}}$ is perfectly/statistically close to the one of the key returned by Setup; for any type $\in (\mathsf{pol}, \mathsf{val})$, any $f \in \mathcal{D}_{\mathsf{type}}$, $(c, o) \approx (c', o')$ where $(c, o) \leftarrow \mathsf{Commit}(\mathsf{ck}, f, \mathsf{type})$, $(c', st) \leftarrow \mathsf{TdCom}(\mathsf{td}, \mathsf{type})$ and $o' \leftarrow \mathsf{TdOpen}(\mathsf{td}, st, c', f)$.

Extractability. PolyCom has knowledge extractability for auxiliary input distribution $\mathcal{Z}$ if for every (non-uniform) efficient adversary $\mathcal{A}$ there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that $\Pr[\mathsf{Game}^{\mathsf{extr}}_{\mathcal{Z},\mathcal{A},\mathcal{E}} = 1] = \mathsf{negl}$. We say that PolyCom is knowledge extractable if there is a benign $\mathcal{Z}$ such that the above holds.

```
$\mathsf{Game}^{\mathsf{extr}}_{\mathcal{Z},\mathcal{A},\mathcal{E}}$
  $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{aux}_Z \leftarrow \mathcal{Z}(1^\lambda)$
  $c \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{aux}_Z)$, $(f, o) \leftarrow \mathcal{E}(\mathsf{ck}, \mathsf{aux}_Z)$
  return 1 iff: $\mathsf{CheckCom}(\mathsf{ck}, c) = 1 \wedge \mathsf{VerCommit}(\mathsf{ck}, c, f, o) = 0$
```

Linearly Homomorphic Commitments. For the constructions presented in this section we assume that the commitments are linearly homomorphic. That is we assume existence of a deterministic algorithm $(c', o') \leftarrow \mathsf{HomEval}(\mathsf{ck}, g, (c_j)_{j \in [\ell]}, (o_j)_{j \in [\ell]})$ such that, for a linear function $g : \mathbb{F}^\ell \to \mathbb{F}$, if $\mathsf{VerCommit}(\mathsf{ck}, c_j, a_j, o_j) = 1$ then $\mathsf{VerCommit}(\mathsf{ck}, c', g((a_j)_{j \in [\ell]}), o') = 1$. In the paper we assume HomEval takes in the vector of ℓ coefficients of g.

[20]Note that the only ambiguity can occur when differentiating a degree-0 polynomial from a point.

Zero-knowledge CP-SNARKs for PolyCom. In the constructions of this section we use the following existing CP-SNARKs for the scheme PolyCom:

- $\mathsf{CP}_{\mathsf{eq}}$: a CP-SNARK for relation $R_{\mathsf{eq}}(u_1, u_2) := u_1 \overset{?}{=} u_2$, where $u_1, u_2 \in \mathbb{F}$.
- $\mathsf{CP}_{\mathsf{prd}}$: a CP-SNARK for relation $R_{\mathsf{prd}}(u_1, u_2, u_3) := u_3 \overset{?}{=} u_1 \cdot u_2$, where $u_1, u_2, u_3 \in \mathbb{F}$.
- $\mathsf{CP}_{\mathsf{poly}}$: a CP-SNARK for the relation R_{poly} over $\mathcal{D}_x \times \mathcal{D}_1 \times \mathcal{D}_2$ where $\mathcal{D}_x = \mathbb{F}^\mu, \mathcal{D}_1 = \mathcal{F}, \mathcal{D}_2 = \mathbb{F}$ and $R_{\mathsf{poly}}(\vec{x}, f, y) := y \overset{?}{=} f(\vec{x})$. For zero-knowledge, we assume that $\mathsf{CP}_{\mathsf{poly}}$ satisfies a notion where the commitment key is generated in trapdoor mode and the $\mathsf{CP}_{\mathsf{poly}}$ simulators $(\mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{prv}})$ get access to the commitment trapdoor produced by $\mathcal{S}_{\mathsf{ck}}$. Note that such notion is weaker than the one of Definition 3.1 but sufficient to argue that a scheme satisfying this notion is a cc-SNARK.

In Appendix G we show pairing-based constructions of PolyCom and $\mathsf{CP}_{\mathsf{poly}}$ extracted from the verifiable polynomial delegation scheme of Zhang et al. [71]. As observed by Zhang et al. constructions for $\mathsf{CP}_{\mathsf{eq}}$ and $\mathsf{CP}_{\mathsf{prd}}$ can be obtained using standard techniques from classical sigma-protocols. Finally, we observe that all these schemes share the same (deterministic) KeyGen algorithm that, on input the commitment key ck, simply partitions the elements of ck into ek = ck and vk = cvk, where cvk is a subset of the elements in ck that is sufficient to run algorithms CheckCom, ComVal and HomEval.

H.1 Our CP-SNARK for Sum-check

We give a full description of the interactive protocol in Figure 12.
Proof We show the security of our protocol by reducing it to the one of [71, Construction 2]. For this let us recall the following theorem from [71]:

Theorem H.2 ([71, Theorem 2]). *For any μ-variate total-degree-d polynomial $g : \mathbb{F}^\mu \to \mathbb{F}$ with m non-zero coefficients, assuming* Com *is an extractable linearly homomorphic commitment scheme, and* $\mathsf{CP}_{\mathsf{eq}}$ *is a zero-knowledge non-interactive argument for testing equality of commitments for* Com*, then there is an interactive argument for the relation*

$$\mathsf{VerCommit}(\mathsf{ck}, c_t, t, o_t) = 1 \wedge t = \sum_{\vec{b} \in \{0,1\}^\mu} g(\vec{b})$$

Moreover, we recall below the last two steps of Construction 2 in [71] (i.e., Construction 2 is the same as in our Figure 12 with the blue part replaced by the following steps):

```
1 : Common input: $c_t$, $g$;   $\mathcal{P}$'s input: $(t, o_t)$
2 : $\mathcal{P}$ : $(c^*_\mu, o^*_\mu) \leftarrow \mathsf{ComVal}(\mathsf{ck}, g(\vec{s}))$,
3 :       $\pi^* \leftarrow \mathsf{CP}_{\mathsf{eq}}.\mathsf{Prove}(\mathsf{ck}, (c^*_\mu, \mathsf{com}_\mu), g(\vec{s}), (o^*_\mu, \rho_\mu))$
4 : $\mathcal{P} \rightarrow \mathcal{V}$ : $c^*_\mu, o^*_\mu, \pi^*$
5 : $\mathcal{V}$ : $\mathsf{VerCommit}(\mathsf{cvk}, c^*_\mu, g(\vec{s}), o^*_\mu) \wedge \mathsf{CP}_{\mathsf{eq}}.\mathsf{VerProof}(\mathsf{vk}, (c^*_\mu, \mathsf{com}_\mu), \pi^*)$
```

For knowledge soundness, the idea of the proof is that for any adversary $\mathcal{A}$ against $\mathsf{CP}_{\mathsf{sc}}$ we can create an adversary $\mathcal{B}$ against Construction 2 in [71].

Similarly to [71], we begin by observing that the commitments c_1, c_2 as well as all the commitments com_{a_j}'s sent during the μ rounds are extractable. By extractability, for any successful $\mathcal{A}$ there exists an extractor $\mathcal{E}_{\mathcal{A}}$ that, on the same input of $\mathcal{A}$, outputs with all but negligible probability valid openings of all these commitments. Thus we define $\mathcal{B}$ as the adversary that executes $(\mathcal{A}, \mathcal{E}_{\mathcal{A}})$, obtains g_1, g_2, reconstructs the polynomial $g(\vec{S})$, and then keeps executing $\mathcal{A}$ until the end of the protocol, forwarding its messages to its challenger. This is done until the last step where $\mathcal{A}$ sends c'_1, c'_2, π^*. Notice that $\mathcal{B}$ also has the commitments com_{a_j} sent by $\mathcal{A}$ in step μ as well as their openings extracted through $\mathcal{E}_{\mathcal{A}}$. Thus, $\mathcal{B}$ can compute homomorphically the commitment com_μ and its opening.

With this knowledge, $\mathcal{B}$ executes the last two lines in Figure 12 (acting as the verifier): if all verifications pass and $\mathcal{B}$ has an opening of com_μ to $g(\vec{s})$, then it executes the lines 2–4 above and sends $(c^*_\mu, o^*_\mu, \pi^*)$ to its challenger.

If all verifications pass but $\mathcal{B}$ has an opening of com_μ to a value different from $g(\vec{s})$, then it must be the case that $\mathcal{A}$ cheated in one of the proofs π_1, π_2, π^*. By the knowledge soundness of $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{prd}}$ this however occurs only with negligible probability.

To show zero-knowledge, we build a simulator that can simulate the verifier's view without knowing the prover's input. Our simulator is the same as the one in [71] up to their step (d). For step (e), we let our simulator additionally create commitments (c'_1, c'_2) to dummy values and then run the ZK simulators of $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{prd}}$ to simulate proofs (π_1, π_2, π^*). By the proof in [71], the verifier's transcript except for the last message $(c'_1, c'_2, \pi_1, \pi_2, \pi^*)$ is indistinguishable from an honest one. The indistinguishability with respect to the last message follows immediately from the zero-knowledge $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{prd}}$. □

H.2 Proof of security of $\mathsf{CP}_{\mathsf{had}}$

Proof Let $\mathcal{A}_{\mathsf{had}}$ be the adversary against $\mathsf{CP}_{\mathsf{had}}$ that, on input $(\mathsf{ck}, \mathsf{ek}_s, \mathsf{ek}_p)$ and interacting with the random oracle H, returns a statement $(c_j)_{j\in[:3]}$ and a proof π that verifies correctly. For any such $\mathcal{A}_{\mathsf{had}}$ we can define a non-interactive adversary $\mathcal{A}^*_{\mathsf{had}}$ that additionally takes as input a sequence of random values $\vec{r}_i$, for $i = 1$ to Q, such that $\vec{r}_i$ is used to answer the i-th query of $\mathcal{A}_{\mathsf{had}}$ to the random oracle H. For any $\mathcal{A}_{\mathsf{had}}$ making Q queries to H there exists an index $i \in [0, Q]$ such that the commitments $(c_j)_{j\in[:3]}$ returned at the end of its execution were queried to H in the i-th query (letting $i = 0$ being the case in which they were not asked at all). From the above adversary $\mathcal{A}^*_{\mathsf{had}}$ we can define $\mathcal{A}_{com}$ as the non-uniform adversary that on input $(\mathsf{ck}, \mathsf{ek}_s, \mathsf{ek}_p, \vec{r}_1, \ldots, \vec{r}_{i-1})$ runs $\mathcal{A}_{\mathsf{had}}$ (in the same way as $\mathcal{A}^*_{\mathsf{had}}$ does) up to its i-th query $H((c_j)_{j\in[:3]})$ and returns $(c_j)_{j\in[:3]}$. By the extractability of the commitment, for $\mathcal{A}_{com}$ there exists an extractor Ext_{com} that on the same input of $\mathcal{A}_{com}$ outputs openings $(\tilde{u}_j)_{j\in[:3]}, (o_j)_{j\in[:3]}$. We define the extractor $\mathcal{E}_{\mathsf{had}}$ to be the one that runs Ext_{com} and returns its output. Notice that by the extractability of PolyCom it holds $\mathsf{VerCommit}(\mathsf{ck}, c_j, \tilde{u}_j, o_j)$ for $j = 0, 1, 2$ with all but negligible probability.

Next, we need to argue that this adversary-extractor pair $(\mathcal{A}_{\mathsf{had}}, \mathcal{E}_{\mathsf{had}})$ has negligible probability of winning in the knowledge soundness experiment. From $\mathcal{A}^*_{\mathsf{had}}$ we can define two adversaries $\mathcal{A}_p$ and $\mathcal{A}_{sc}$ against $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{sc}}$ respectively, and by using the knowledge soundness of the two CP-SNARKs we have that for each of these adversaries there is a corresponding extractor that gives us a value t such that $\tilde{u}_0(\vec{r}) = t$ and $t = \sum_{\vec{b}\in\{0,1\}^\mu} \tilde{eq}(\vec{r}, \vec{b}) \cdot \tilde{u}_1(\vec{b}) \cdot \tilde{u}_2(\vec{b})$

```
Protocol $\Pi_{sc}$:
$\text{Common input: } c_t, g_0, c_1, c_2; \quad \mathcal{P}\text{'s input: } (t, o_t, g_1, o_1, g_2, o_2)$
$\mathcal{P} : g(\vec{S}) := \prod_{i=0}^{2} g_i(\vec{S}),\ c_0 := c_t,\ t_0 := t,\ \rho_0 := o_t,\ \text{let } f(A_0, \ldots A_k) := A_0 + \sum_{j=0}^{k} A_j := (2, 1, \ldots, 1)$
for $i = 1 \ldots \mu$ :
  $\mathcal{P} : h_i(X) := \sum_{b_{i+1},\ldots,b_\mu \in \{0,1\}} g(s_1, \ldots, s_{i-1}, X, b_{i+1}, \ldots, b_\mu) := \sum_{j=0}^{k} a_j X^j$
  $\mathcal{P} : \text{compute } \{(\mathsf{com}_{a_j}, \rho_{a_j}) \leftarrow \mathsf{ComVal}(\mathsf{ck}, a_j)\}_{j=0}^{k}, (\mathsf{com}^*_{i-1}, \rho^*_{i-1}) \leftarrow \mathsf{HomEval}(\mathsf{ck}, f, \{\mathsf{com}_{a_j}\}_{j=0}^{k}), \{\rho_{a_j}\}_{j=0}^{k})$
      $\pi_{\mathsf{eq}} \leftarrow \mathsf{CP}_{\mathsf{eq}}.\mathsf{Prove}(\mathsf{ck}, \mathsf{com}_{i-1}, \mathsf{com}^*_{i-1}, t_{i-1}, g_i(0) + g_i(1), \rho_{i-1}, \rho^*_{i-1})$
  $\mathcal{P} \rightarrow \mathcal{V} : \{\mathsf{com}_{a_j}\}_{j=0}^{k}, \pi_{\mathsf{eq}}$
  $\mathcal{V} : \{\mathsf{CheckCom}(\mathsf{cvk}, \mathsf{com}_{a_j})\}_{j=0}^{k},\ \text{compute } (\mathsf{com}^*_{i-1}, \cdot) \leftarrow \mathsf{HomEval}(\mathsf{ck}, f, \{\mathsf{com}_{a_j}\}_{j=0}^{k}), \cdot)$
  $\mathcal{V} : \mathsf{CP}_{\mathsf{eq}}.\mathsf{VerProof}(\mathsf{cvk}, \mathsf{com}_{i-1}, \mathsf{com}^*_{i-1}, \pi_{\mathsf{eq}}), s_i \leftarrow_{\$} \mathbb{F}, (\mathsf{com}_i, \cdot) \leftarrow \mathsf{HomEval}(\mathsf{ck}, (1, s_i, \ldots, s_i^k), \{\mathsf{com}_{a_j}\}_{j=0}^{k}, \cdot)$
  $\mathcal{V} \rightarrow \mathcal{P} : s_i \in \mathbb{F}$
  $\mathcal{P} : t_i \leftarrow h_i(s_i), (\mathsf{com}_i, \rho_i) \leftarrow \mathsf{HomEval}(\mathsf{ck}, (1, s_i, \ldots, s_i^k), \{\mathsf{com}_{a_j}\}_{j=0}^{k}, \{\rho_{a_j}\}_{j=0}^{k})$
endfor
$\mathcal{P} : \{(c'_j, o'_j) \leftarrow \mathsf{ComVal}(\mathsf{ck}, g'_j := g_j(\vec{s})), \pi_j \leftarrow \mathsf{CP}_{\mathsf{poly}}.\mathsf{Prove}(\mathsf{ek}, \vec{s}, (c_j, c'_j), (g_j, g'_j), (o_j, o'_j))\}_{j=1,2}$
$\mathcal{P} : (c^*_1, o^*_1) \leftarrow \mathsf{HomEval}(\mathsf{ck}, g_0(\vec{s}), c'_1, o'_1), \pi^* \leftarrow \mathsf{CP}_{\mathsf{prd}}.\mathsf{Prove}(\mathsf{ck}, (c^*_1, c'_2, \mathsf{com}_\mu), (g_0(\vec{s}) \cdot g'_1, g'_2, g(\vec{s})), (o^*_1, o'_2, \rho_\mu))$
$\mathcal{P} \rightarrow \mathcal{V} : c'_1, c'_2, \pi_1, \pi_2, \pi^*$
$\mathcal{V} : \bigwedge_{j=1,2} \mathsf{CheckCom}(\mathsf{cvk}, c'_j) \wedge \mathsf{CP}_{\mathsf{poly}}.\mathsf{VerProof}(\mathsf{vk}, \vec{s}, c_j, c'_j, \pi_j)$
$\mathcal{V} : (c^*_1, \cdot) \leftarrow \mathsf{HomEval}(\mathsf{ck}, g_0(\vec{s}), c'_1, \cdot), \mathsf{CP}_{\mathsf{prd}}.\mathsf{VerProof}(\mathsf{vk}, (c^*_1, c'_2, \mathsf{com}_\mu), \pi^*)$
```

Figure 12: Our sum-check protocol over committed polynomial and result; in black are the steps identical to [71].

hold respectively with all but negligible probability. Furthermore, the binding of PolyCom implies that the values and openings for all the commitments $(c_j)_{j\in[:3]}, c_t$ obtained using these extractors are all the same with all but negligible probability (otherwise we could define a reduction against the binding of PolyCom).

Since $\mathsf{VerCommit}(\mathsf{ck}, c_j, \tilde{u}_j, o_j)$ for $j = 0, 1, 2$, the only way for the adversary to win is when the relation R_{had} is not satisfied. Since we have vectors in MLE form, the check of relation R_{had} can be equivalently written as $\forall \vec{b} \in \{0,1\}^\mu : \tilde{u}_0(\vec{b}) \stackrel{?}{=} \tilde{u}_1(\vec{b}) \cdot \tilde{u}_2(\vec{b})$. Let us define the polynomial $\tilde{u}^*_0(\vec{X}) = \sum_{\vec{b}\in\{0,1\}^\mu} \tilde{eq}(\vec{X}, \vec{b}) \cdot \tilde{u}_1(\vec{b}) \cdot \tilde{u}_2(\vec{b})$; essentially $\tilde{u}^*_0(\vec{X})$ is the MLE of the vector that should correctly verify the R_{had} relation. In particular, by lemma 5.1, $\tilde{u}^*_0(\vec{X})$ agrees with $\tilde{u}_1(\vec{X})\cdot\tilde{u}_2(\vec{X})$ on all boolean points. Thus, if the relation does not hold we must have $\tilde{u}^*_0(\vec{X}) \neq \tilde{u}_0(\vec{X})$. However, from above we have that $\tilde{u}_0(\vec{r}) = \tilde{u}^*_0(\vec{r})$ holds. Notice that from the construction of $\mathcal{E}_{\mathsf{had}}$, the polynomials $\tilde{u}_0(\vec{X}), \tilde{u}_1(\vec{X}), \tilde{u}_2(\vec{X})$ are independent from $\vec{r}$ (this is because the extractor $\mathcal{E}_{com}$ that returned this polynomial did not have $\vec{r} = \vec{r}_i$ among its inputs), and $\tilde{u}^*_0(\vec{X})$ is fully determined from $\tilde{u}_1(\vec{X}), \tilde{u}_2(\vec{X})$. Therefore, by the Schwartz-Zippel lemma, the event $\tilde{u}^*_0(\vec{X}) \neq \tilde{u}_0(\vec{X}) \wedge \tilde{u}_0(\vec{r}) = \tilde{u}^*_0(\vec{r})$ occurs with negligible probability over the random choice of $\vec{r}$.

The zero-knowledge of $\mathsf{CP}_{\mathsf{had}}$ relies on the hiding of PolyCom and the zero-knowledge of $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{sc}}$. Building simulators $\mathcal{S}_{\mathsf{kg}}$ and $\mathcal{S}_{\mathsf{prv}}$ for $\mathsf{CP}_{\mathsf{had}}$ from the corresponding simulators for $\mathsf{CP}_{\mathsf{poly}}$ and $\mathsf{CP}_{\mathsf{sc}}$ is fairly straightforward and is omitted here. □

H.3 Full description and proof of $\mathsf{CP}_{\mathsf{sfprm}}$

We provide in this section the description of $\mathsf{CP}_{\mathsf{sfprm}}$, a CP-SNARK for the self-permutation relation. As explained in Section 5.4, this construction makes use of a CP-SNARK $\mathsf{CP}_{\mathsf{ipd}}$, and in particular, in order to evaluate the efficiency of $\mathsf{CP}_{\mathsf{sfprm}}$, we consider an instantiation of $\mathsf{CP}_{\mathsf{ipd}}$ based on Thaler's protocol for trees of multiplications [63] (we refer the reader to Appendix I for more details on this protocol).

One detail to be noted here is that such $\mathsf{CP}_{\mathsf{ipd}}$ works with binary tree circuits, meaning that their input should be a power of two length. In our definition of the self-permutation relation $R^{\mathsf{sfprm}}_{\phi}(\vec{y} := (\vec{x}, (\vec{u}_j)_{j\in[l]}) \in \mathbb{F}^m)$ however we must work on $\ell + 1$ vectors such that, each has length $n_j = 2^{\mu_j}$ (this is immediate since we commit to MLEs of vectors) but their concatenation has length $\sum_{j=0}^{\ell} n_j = m$ which may not be a power-of-two. To solve this issue, we execute $\mathsf{CP}_{\mathsf{ipd}}$ on each block and then aggregate the $\ell + 1$ committed results using a simple zero-knowledge argument for proving a product relation over three commitments, i.e., $\mathsf{CP}_{\mathsf{prd}}$. This results in about $\ell + 1$ calls to $\mathsf{CP}_{\mathsf{ipd}}$ and $\mathsf{CP}_{\mathsf{prd}}$. Although this makes proofs grow with ℓ, we observe that in all our applications ℓ is some small constant, e.g., $8 - -10$ in our arithmetic circuits encoding

We describe this approach in detail in Figure 13.

More deeply, the self-permutation protocol uses a probabilistic test to check that a vector $\vec{y}$ is a self permutation according to ϕ. In particular, if it is indeed a self-permutation then $\prod_i y'_i := \prod_i (y_i + r \cdot i - s) = \prod_i (y_i + r \cdot \phi(i) - s) := \prod_i y''_i$ (if it is not, this equality will hold with negligible probability by Schwartz-Zippel lemma). The protocol works by computing for each of the $\ell + 1$ power-of-2-length-blocks the value of the product of both binary subtrees using Thaler's approach. Then we compute iteratively the product of all of them and claim that both values coincide.

Proof Let $\mathcal{A}_{\mathsf{sfprm}}$ be the adversary against $\mathsf{CP}_{\mathsf{sfprm}}$ that, on input $(\mathsf{ck}, \mathsf{ek}_p)$ and interacting with the random oracle H, returns a statement $(\phi, x, (c_j)_{j\in[\ell]})$ and a proof π that verifies correctly. For any such $\mathcal{A}_{\mathsf{sfprm}}$ we can define a non-interactive adversary $\mathcal{A}^*_{\mathsf{sfprm}}$ that additionally takes as input a sequence of random values (r_i, s_i), for $i = 1$ to Q, such that (r_i, s_i) are used to answer the i-th query of $\mathcal{A}_{\mathsf{sfprm}}$ to the random oracle H. For any $\mathcal{A}_{\mathsf{sfprm}}$ making Q queries to H there exists an index $i \in [0, Q]$ such that for the relation statement $(\phi, x, (c_j)_{j\in[\ell]})$ returned at the end of its execution, $((c_{\phi,j})_{j\in[0,\ell]}, \vec{x}, (c_j)_{j\in[\ell]})$ was queried to H in the i-th query (letting $i = 0$ being the case in which they were not asked at all, and $c_{\phi,j}$ be deterministically derived from ϕ). From the above adversary $\mathcal{A}^*_{\mathsf{sfprm}}$ we can define $\mathcal{A}_{\mathsf{com}}$ as the non-uniform adversary that on input $(\mathsf{ck}, \mathsf{ek}_p, r_1, s_1, \ldots, r_{i-1}, s_{i-1})$ runs $\mathcal{A}_{\mathsf{sfprm}}$ (in the same way as $\mathcal{A}^*_{\mathsf{sfprm}}$ does) up to its i-th query $H((c_{\phi,j})_{j\in[0,\ell]}, \vec{x}, (c_j)_{j\in[\ell]})$ and returns $(c_j)_{j\in[\ell]}$. By the extractability of the commitment, for $\mathcal{A}_{\mathsf{com}}$ there exists an extractor $\mathcal{E}_{\mathsf{com}}$ that on the same input of $\mathcal{A}_{\mathsf{com}}$ outputs openings $(\tilde{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}$. We define the extractor $\mathcal{E}_{\mathsf{sfprm}}$ to be the one that runs $\mathcal{E}_{\mathsf{com}}$ and returns its output. Notice that by the extractability of PolyCom it holds $\mathsf{VerCommit}(\mathsf{ck}, c_j, \tilde{u}_j, o_j)$ for $j = 0, 1, 2$ with all but negligible probability.

Next, we need to argue that this adversary-extractor pair $(\mathcal{A}_{\mathsf{sfprm}}, \mathcal{E}_{\mathsf{sfprm}})$ has negligible probability of winning in the knowledge soundness experiment. Recall that we have $\mathsf{VerCommit}(\mathsf{ck}, c_j, \tilde{u}_j, o_j)$ for $j \in [\ell]$ and, by the linear homomorphic property of PolyCom, for all $j \in [0, \ell]$, c'_j and c''_j are commitments to the MLE of $\vec{y}'_j := \vec{y}_j + r \cdot \vec{v}_j - s \cdot \vec{1}_j$ and $\vec{y}''_j := \vec{y}_j + r \cdot \vec{\phi}_j - s \cdot \vec{1}_j$ respectively. Also, in order for the adversary to be successful it must be the case that the relation does not hold, i.e., $\vec{y}$ is *not* a self-permutation according to ϕ. Notice that the vector $\vec{y}$ is independent of (r, s) since it was returned by $\mathcal{E}_{\mathsf{com}}$ without having these values in its view. This allows us to argue that with overwhelming probability over the choice of r it is the case that at least one of the entries of $\vec{y} + r \cdot \vec{\phi}$ is not in $\vec{y} + r \cdot \vec{v}$. Moreover, when these vectors have different entries the equation $\prod_i (y_i + r \cdot i - s) = \prod_i (y_i + r \cdot \phi(i) - s)$ holds with negligible probability over the choice of s by the Schwartz-Zippel lemma.

Hence we have that with all but negligible probability $\prod_i (y_i + r \cdot i - s) \neq \prod_i (y_i + r \cdot \phi(i) - s)$, which means that at one of the statements in the $\mathsf{CP}_{\mathsf{ipd}}$, $\mathsf{CP}_{\mathsf{prd}}$ or $\mathsf{CP}_{\mathsf{eq}}$ proofs is not correct. We can reduce these cases to the knowledge soundness of $\mathsf{CP}_{\mathsf{ipd}}$, $\mathsf{CP}_{\mathsf{prd}}$ or $\mathsf{CP}_{\mathsf{eq}}$ using a fairly standard reduction, in which from an adversary $\mathcal{A}^*_{\mathsf{sfprm}}$ that falls into the above conditions (i.e., an (r, s) that cause the above inequality) we build either an adversary $\mathcal{A}_{\mathsf{ipd}}$ against $\mathsf{CP}_{\mathsf{ipd}}$, or an adversary $\mathcal{A}_{\mathsf{prd}}$ against $\mathsf{CP}_{\mathsf{prd}}$ or an $\mathcal{A}_{\mathsf{eq}}$ against $\mathsf{CP}_{\mathsf{eq}}$.

The zero-knowledge of $\mathsf{CP}_{\mathsf{sfprm}}$ follows from the hiding of PolyCom (for creating dummy commitments $(c_{z'_j}, c_{z''_j})_{j\in[0\ldots\ell]}$) and the zero-knowledge of all the underlying CP-SNARKs. □

```
$\mathsf{CP}_{\mathsf{sfprm}}.\mathsf{KeyGen}(\mathsf{ck}) \longrightarrow (\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_p), \mathsf{vk} := (\mathsf{cvk}, \mathsf{vk}_p))$ :
$(\mathsf{ek}_p, \mathsf{vk}_p) \leftarrow \mathsf{CP}_{\mathsf{ipd}}.\mathsf{KeyGen}(\mathsf{ck})$

$\mathsf{CP}_{\mathsf{sfprm}}.\mathsf{Derive}((\mathsf{ek}, \mathsf{vk}), R^{\mathsf{sfprm}}_{\phi}) \longrightarrow (\mathsf{ek}_\phi, \mathsf{vk}_\phi)$ :
for $j = 0, \ldots, \ell$ :
    $(c_{1,j}, o_{1,j}) \leftarrow \mathsf{ComPoly}^*(\mathsf{ck}, \tilde{1}_j)$
    $(c_{v,j}, o_{v,j}) \leftarrow \mathsf{ComPoly}^*(\mathsf{ck}, \tilde{v}_j)$
    $(c_{\phi,j}, o_{\phi,j}) \leftarrow \mathsf{ComPoly}^*(\mathsf{ck}, \tilde{\phi}_j)$
$\mathsf{ek}_\phi := (\mathsf{ek}, \{c_{1,j}, o_{1,j}, c_{v,j}, o_{v,j}, c_{\phi,j}, o_{\phi,j}\}_{j=0}^{\ell}, \phi)$
$\mathsf{vk}_\phi := (\mathsf{vk}, \{c_{1,j}, c_{v,j}, c_{\phi,j}\}_{j=0}^{\ell})$

$\mathsf{CP}_{\mathsf{sfprm}}.\mathsf{Prove}^*(\mathsf{ek}_\phi, \vec{x}, (c_j)_{j\in[\ell]}, (\vec{u}_j)_{j\in[\ell]}, (o_j)_{j\in[\ell]}) \longrightarrow \pi$ :
$(r, s) \leftarrow H((c_{\phi,j})_{j\in[0,\ell]}, \vec{x}, (c_j)_{j\in[\ell]})$ and let $\vec{\rho} = (1, r, -s)$
$(c_0, o_0) \leftarrow \mathsf{ComPoly}(\mathsf{ck}, \tilde{x})$
for $j = 0, \ldots, \ell$ :
$(c'_j, o'_j) \leftarrow \mathsf{HomEval}(\mathsf{ck}, \vec{\rho}, (c_j, c_{v,j}, c_{1,j}), (o_j, o_{v,j}, o_{1,j}))$
$(c''_j, o''_j) \leftarrow \mathsf{HomEval}(\mathsf{ck}, \vec{\rho}, (c_j, c_{\phi,j}, c_{1,j}), (o_j, o_{\phi,j}, o_{1,j}))$
$\vec{y}'_j := \vec{y}_j + r \cdot \vec{v}_j - s \cdot \vec{1}_j$   ;   $z'_j := \prod_{i=1}^{n_j} y'_{j,i}$
$\vec{y}''_j := \vec{y}_j + r \cdot \vec{\phi}_j - s \cdot \vec{1}_j$   ;   $z''_j := \prod_{i=1}^{n_j} y''_{j,i}$
$(c_{z'_j}, o_{z'_j}) \leftarrow \mathsf{ComVal}(\mathsf{ck}, z'_j)$ ; $(c_{z''_j}, o_{z''_j}) \leftarrow \mathsf{ComVal}(\mathsf{ck}, z''_j)$
$\pi'_j \leftarrow \mathsf{CP}_{\mathsf{ipd}}.\mathsf{Prove}(\mathsf{ek}_p, c_{z'_j}, (c'_{j,i})_{i\in[n_j]}, z'_j, (y'_{j,i})_{i\in[n_j]}, o_{z'_j}, (o'_{j,i})_{i\in[n_j]})$
$\pi''_j \leftarrow \mathsf{CP}_{\mathsf{ipd}}.\mathsf{Prove}(\mathsf{ek}_p, c_{z''_j}, (c''_{j,i})_{i\in[n_j]}, z''_j, (y''_{j,i})_{i\in[n_j]}, o_{z''_j}, (o''_{j,i})_{i\in[n_j]})$
if $j \overset{?}{=} 0$ : $w'_0 := z'_0$ ; $w''_0 := z''_0$ ; $c_{w'_0} := c_{z'_0}$ ; $c_{w''_0} := c_{z''_0}$ ; else :
    $w'_j \leftarrow w'_{j-1} \cdot z'_j$ ; $(c_{w'_j}, o_{w'_j}) \leftarrow \mathsf{ComVal}(\mathsf{ck}, w'_j)$
    $w''_j \leftarrow w''_{j-1} \cdot z''_j$ ; $(c_{w''_j}, o_{w''_j}) \leftarrow \mathsf{ComVal}(\mathsf{ck}, w''_j)$
    $\pi_{w'_j} \leftarrow \mathsf{CP}_{\mathsf{prd}}.\mathsf{Prove}(\mathsf{ck}, c_{w'_{j-1}}, c_{z'_j}, c_{w'_j}, w'_{j-1}, z'_j, w'_j, o_{w'_{j-1}}, o_{z'_j}, o_{w'_j})$
    $\pi_{w''_j} \leftarrow \mathsf{CP}_{\mathsf{prd}}.\mathsf{Prove}(\mathsf{ck}, c_{w''_{j-1}}, c_{z''_j}, c_{w''_j}, w''_{j-1}, z''_j, w''_j, o_{w''_{j-1}}, o_{z''_j}, o_{w''_j})$
endif ; endfor
$\pi_z \leftarrow \mathsf{CP}_{\mathsf{eq}}.\mathsf{Prove}(\mathsf{ck}, c_{w'_\ell}, c_{w''_\ell}, w'_\ell, w''_\ell, o_{w'_\ell}, o_{w''_\ell})$
return $\pi := (c_0, o_0, \{c_{z'_j}, c_{z''_j}, c_{w'_j}, c_{w''_j}, \pi'_j, \pi''_j, \pi_{w'_j}, \pi_{w''_j}\}_{j=0}^{\ell}, \pi_z)$

$\mathsf{CP}_{\mathsf{sfprm}}.\mathsf{VerProof}^*\left(\mathsf{vk}_\phi, \vec{x}, (c_j)_{j\in[\ell]}, \pi\right) \longrightarrow b$ :
$(r, s) \leftarrow H((c_{\phi,j})_{j\in[0,\ell]}, \vec{x}, (c_j)_{j\in[\ell]})$ and let $\vec{\rho} = (1, r, -s)$
$b \leftarrow \mathsf{VerCommit}(\mathsf{cvk}, c_0, \tilde{x}, o_0)$
for $j = 0, \ldots, \ell$ :
$(c'_j, \cdot) \leftarrow \mathsf{HomEval}(\mathsf{cvk}, \vec{\rho}, (c_j, c_{v,j}, c_{1,j}), \cdot)$
$(c''_j, \cdot) \leftarrow \mathsf{HomEval}(\mathsf{cvk}, \vec{\rho}, (c_j, c_{\phi,j}, c_{1,j}), \cdot)$
$b \leftarrow b \wedge \mathsf{CheckCom}(\mathsf{cvk}, c_{z'_j}) \wedge \mathsf{CP}_{\mathsf{ipd}}.\mathsf{VerProof}(\mathsf{vk}_p, c_{z'_j}, (c'_{j,i})_{i\in[n_j]})$
    $\wedge\ \mathsf{CheckCom}(\mathsf{cvk}, c_{z''_j}) \wedge \mathsf{CP}_{\mathsf{ipd}}.\mathsf{VerProof}(\mathsf{vk}_p, c_{z''_j}, (c''_{j,i})_{i\in[n_j]})$
    $\wedge\ \mathsf{CheckCom}(\mathsf{cvk}, c_{w'_j}) \wedge \mathsf{CheckCom}(\mathsf{cvk}, c_{w''_j})$
if $j \neq 0$ : $b \leftarrow b \wedge \mathsf{CP}_{\mathsf{prd}}.\mathsf{VerProof}(\mathsf{cvk}, c_{w'_{j-1}}, c_{z'_j}, c_{w'_j}, \pi_{w'_j})$
                $\wedge\ \mathsf{CP}_{\mathsf{prd}}.\mathsf{VerProof}(\mathsf{cvk}, c_{w''_{j-1}}, c_{z''_j}, c_{w''_j}, \pi_{w''_j})$
endif ; endfor
$b \leftarrow b \wedge \mathsf{CP}_{\mathsf{eq}}.\mathsf{VerProof}(\mathsf{cvk}, c_{w'_\ell}, c_{w''_\ell}, \pi_z)$
```

Figure 13: CP-SNARK for specializable universal R^{sfprm}

H.4 A CP-SNARK for Linear Properties of Committed Vector

In this section we show a CP-SNARK for PolyCom that has a specializable universal CRS for relations $R^{\mathsf{lin}}_{\mathbf{F}}(\vec{x}, \vec{u}) := \vec{x} \overset{?}{=} \mathbf{F} \cdot \vec{u}$ where $\mathbf{F} \in \mathbb{Z}_q^{n \times m}$, $\vec{x} \in \mathbb{Z}_q^n$ and $\vec{u} \in \mathbb{Z}_q^m$. More precisely, our $\mathsf{CP_{lin}}$ works for a family of relations $\mathcal{R}$ that includes all $R^{\mathsf{lin}}_{\mathbf{F}}$ for all matrices $\mathbf{F} \in \mathbb{F}^{m \times n}$.

The scheme is based on the interactive proof for Matrix multiplication of Thaler [63]. In a nutshell, we specialize this protocol to the case of a matrix-vector multiplication and we turn it into a ZK argument using ideas similar to those in [71].

Our scheme makes use of the building blocks defined in Section 5.1: a polynomial commitment scheme PolyCom, and CP-SNARKs $\mathsf{CP_{poly}}$ and $\mathsf{CP_{sc}}$ for the relations R_{poly} and R_{sc} respectively.

Review of Thaler's Matrix Multiplication protocol. We begin by reviewing the idea of Thaler's matrix multiplication protocol in our specific case of proving $\vec{x} = \mathbf{F} \cdot \vec{u}$. Let $\nu := \log n, \mu := \log m$. We let $\tilde{F} : \{0,1\}^\nu \times \{0,1\}^\mu \to \mathbb{Z}_q$ be the multilinear extension (MLE) of $\mathbf{F}$, i.e., the unique multilinear polynomial such that $\tilde{F}(i_1, \dots, i_\nu, j_1, \dots, j_\mu) = F_{i,j}$. Similarly, let $\tilde{u}$ and $\tilde{x}$ be the MLE of $\vec{u}$ and $\vec{x}$ respectively. The protocol exploits that the MLE $\tilde{x}$ can also be expressed as $\tilde{x}(\vec{R}) = \sum_{\vec{b} \in \{0,1\}^\mu} \tilde{F}(\vec{R}, \vec{b}) \cdot \tilde{u}(\vec{b})$. In particular, since this MLE is unique, if $\tilde{F}$ and $\tilde{u}$ are MLE of $\mathbf{F}$ and $\vec{u}$ respectively, then $\tilde{x}$ is a MLE of $\vec{x} = \mathbf{F} \cdot \vec{u}$. Next, starting from this observation, the verifier picks a random $\vec{r}$, and then starts a sum-check protocol where the prover convinces the verifier that $t = \tilde{x}(\vec{r}) = \sum_{\vec{b} \in \{0,1\}^\mu} g(\vec{b})$ for the polynomial $g(\vec{S}) := \tilde{F}(\vec{r}, \vec{S}) \cdot \tilde{u}(\vec{S})$. At the end of the sum-check the verifier instead of computing $g(\vec{s})$ directly, it gets it by evaluating $\tilde{F}(\vec{r}, \vec{s})$ and $\tilde{u}(\vec{s})$ and by computing their product.

The idea to turn the above protocol into a commit and prove argument is rather simple and consists into using a CP-SNARK for the sumcheck relation with a committed polynomial g, or more precisely for the case when a commitment to g is implicitly given through commitments to its factors (see the $\mathsf{CP_{sc}}$ scheme). To see this, let us write $g(\vec{S}) := \prod_0^p g_i(\vec{S})$, where $g_1(\vec{S}) := \tilde{F}(\vec{r}, \vec{S})$, $g_2(\vec{S}) = \tilde{u}(\vec{S})$, and $g_0(\vec{S}) := 1$ is the constant polynomial. A commitment to $\tilde{u}(\vec{S})$ is part of the statement, a commitment to $\tilde{F}(\vec{R}, \vec{S})$ can be generated when specializing the relation to $\mathbf{F}$ in the Derive algorithm. However, note that $\mathsf{CP_{sc}}$ expects a commitment to a μ-variate polynomial, whereas $\tilde{F}$ is in $\nu + \mu$ variables. For this, we let the prover commit to the partial evaluation of $\tilde{F}$ on $\vec{r}$, i.e., to the polynomial $g_1(\vec{S})$ and uses this commitment and polynomial in $\mathsf{CP_{sc}}$. Then, what is left to show is that such committed $g_1(\vec{S})$ is actually the partial evaluation of the other committed polynomial $\tilde{D}$. To prove this, the idea is that the verifier chooses a random $\sigma \leftarrow_\$ \mathbb{F}^\mu$, and the prover uses $\mathsf{CP_{poly}}$ to prove that $g_1(\vec{\sigma}) = \tilde{F}(\vec{r}, \vec{\sigma})$.

We show the full protocol $\mathsf{CP_{lin}}$ in Figure 14.

Theorem H.3. *In the random oracle model, assuming* PolyCom *is an extractable trapdoor commitment and* $\mathsf{CP_{poly}}$ *and* $\mathsf{CP_{sc}}$ *are zero-knowledge CP-SNARKs for* PolyCom, *then* $\mathsf{CP_{lin}}$ *in Figure 14 is a zero-knowledge CP-SNARK for* PolyCom *and relations* R^{lin}.

Proof Let $\mathcal{A}_{\mathsf{lin}}$ be the adversary against $\mathsf{CP_{lin}}$ that, on input $(\mathsf{ck}, \mathsf{ek}_s, \mathsf{ek}_p)$ and interacting with the random oracles H_1, H_2, returns a statement $(\mathbf{F}, \vec{x}, c_u)$ and a proof π that verifies correctly. For any such $\mathcal{A}_{\mathsf{lin}}$ we can define a non-interactive adversary $\mathcal{A}^*_{\mathsf{lin}}$ that additionally takes as input a sequence of random values $\{\vec{r}_i\}_i, \{\vec{\sigma}_j\}_j$, for $i = 1$ to Q_1 and $j = 1$ to Q_2, such that $\vec{r}_i$ (resp. $\vec{\sigma}_j$) is used to answer the i-th (resp. j-th) query of $\mathcal{A}_{\mathsf{lin}}$ to the random oracle H_1 (resp. H_2). For any $\mathcal{A}_{\mathsf{lin}}$ making Q_1 queries to H_1 there exists an index $i \in [0, Q_1]$ such that for the statement $(\mathbf{F}, \vec{x}, c_u)$ returned at the end of its execution the i-th query to H_1 (letting $i = 0$ being the case in which they were not asked at all) is $(c_F, \vec{x}, c_u)$. From the above adversary $\mathcal{A}^*_{\mathsf{lin}}$ we can define $\mathcal{A}_{com}$ as the non-uniform adversary that on input $(\mathsf{ck}, \mathsf{ek}_s, \vec{r}_1, \dots, \vec{r}_{i-1})$ runs $\mathcal{A}_{\mathsf{lin}}$ (in the same way as $\mathcal{A}^*_{\mathsf{lin}}$ does) up to its i-th query $H(c_F, \vec{x}, c_u)$ and returns c_u. By the extractability of the commitment, for $\mathcal{A}_{com}$ there exists an extractor Ext_{com} that on the same input of $\mathcal{A}_{com}$ outputs an opening $\tilde{u}, o_u$. We define the extractor $\mathcal{E}_{\mathsf{lin}}$ to be the one that runs Ext_{com} and returns its output. Notice that by the extractability of PolyCom it holds $\mathsf{VerCommit}(\mathsf{ck}, c_u, \tilde{u}, o_u)$ with all but negligible probability.

Next, we need to argue that this adversary-extractor pair $(\mathcal{A}_{\mathsf{lin}}, \mathcal{E}_{\mathsf{lin}})$ has negligible probability of winning in the knowledge soundness experiment.

In a similar way as we argued extractability of c_u, we can show that it is possible to extract the polynomial g_1 that correctly opens the commitment c_1.

Recall that the adversary is successful if the verifications pass and the relation does not hold, i.e., $\mathbf{F} \cdot \vec{u} \neq \vec{x}$. Considering MLEs, this means there is some $\vec{a} \in \{0,1\}^\nu$ such that

$$\tilde{x}(\vec{a}) \neq \sum\nolimits_{\vec{b} \in \{0,1\}^\mu} \tilde{F}(\vec{a}, \vec{b}) \tilde{u}(\vec{b}).$$

This means that the following polynomial inequality holds:

$$\tilde{x}(\vec{R}) \neq \sum\nolimits_{\vec{b} \in \{0,1\}^\mu} \tilde{F}(\vec{R}, \vec{b}) \cdot \tilde{u}(\vec{b})$$

First, we argue that with all but negligible probability over the choice of $\vec{r}$ we have $t = \tilde{x}(\vec{r}) \neq \sum_{\vec{b} \in \{0,1\}^\mu} \tilde{F}(\vec{r}, \vec{b}) \tilde{u}(\vec{b})$. Indeed, $\vec{r}$ is random and independent from $\vec{x}, \tilde{F}, \tilde{u}$ and the two polynomials would be equal when evaluated on $\vec{r}$ with probability at most $\nu/|\mathbb{F}|$ by Schwartz-Zippel. Thus we can continue the proof assuming that $t \neq \sum_{\vec{b} \in \{0,1\}^\mu} \tilde{F}(\vec{r}, \vec{b}) \tilde{u}(\vec{b})$.

Next, consider that for the extracted g_1 there are two possible cases: (i) $g_1(\vec{S}) = \tilde{F}(\vec{r}, \vec{S})$, and (ii) $g_1(\vec{S}) \neq \tilde{F}(\vec{r}, \vec{S})$.

If (i) occurs, then we can immediately build an adversary against the soundness of $\mathsf{CP_{sc}}$.

If (ii) occurs, consider two subcases: (ii.a) $g_1(\vec{\sigma}) = \tilde{F}(\vec{r}, \vec{\sigma})$, and (ii.b) $g_1(\vec{\sigma}) \neq \tilde{F}(\vec{r}, \vec{\sigma})$. However, by Schwartz-Zippel (ii.a) occurs with negligible probability $\mu/|\mathbb{F}|$ over the choice of $\vec{\sigma}$. And if (ii.b) occurs then it is possible to do a reduction to the soundness of $\mathsf{CP_{poly}}$ (since at least one of the claims $y^* = g_1(\vec{\sigma})$ or $y^* = \tilde{F}(\vec{r}, \vec{\sigma})$ is false).

The zero-knowledge of $\mathsf{CP_{lin}}$ follows immediate from the zero-knowledge of $\mathsf{CP_{sc}}$. □

```
$\mathsf{CP_{lin}.KeyGen(ck)} \to (\mathsf{ek}, \mathsf{vk})$ :
    $(\mathsf{ek}_s, \mathsf{vk}_s) \leftarrow \mathsf{CP_{sc}.KeyGen(ck)}$
    $(\mathsf{ek}_p, \mathsf{vk}_p) \leftarrow \mathsf{CP_{poly}.KeyGen(ck)}$
    $\mathsf{ek} := (\mathsf{ck}, \mathsf{ek}_s, \mathsf{ek}_p)$
    $\mathsf{vk} := (\mathsf{cvk}, \mathsf{vk}_s, \mathsf{ek}_p)$

$\mathsf{CP_{lin}.Derive}((\mathsf{ek}, \mathsf{vk}), \mathbf{F})$ :
    $(c_F, o_F) \leftarrow \mathsf{ComPoly}^*(\mathsf{ck}, \tilde{F})$
    $\mathsf{ek}_F := (\mathsf{ek}, c_F, \mathbf{F}, o_F)$
    $\mathsf{vk}_F := (\mathsf{vk}, c_F)$
    return $(\mathsf{ek}_F, \mathsf{vk}_F)$

$\mathsf{CP_{lin}.Prove}^*(\mathsf{ek}_F, \vec{x}, c_u, \vec{u}, o_u) \to \pi$ :
    $\vec{r} \leftarrow H_1(c_F, c_u, \vec{x})$,  ;  $t \leftarrow \tilde{x}(\vec{r})$ ; $(c_t, o_t) \leftarrow \mathsf{ComVal}(\mathsf{ck}, t)$
    Let $g(\vec{S}) := \tilde{F}(\vec{r}, \vec{S}) \cdot \tilde{u}(\vec{S}) := g_1(\vec{S}) \cdot \tilde{u}(\vec{S})$
    $(c_1, o_1) \leftarrow \mathsf{ComPoly}(\mathsf{ck}, g_1)$ ; $\vec{\sigma} \leftarrow H_2(c_F, c_1, \vec{r})$;
    $y^* \leftarrow g_1(\vec{\sigma})$; $(c^*, o^*) \leftarrow \mathsf{ComVal}(\mathsf{ck}, y^*)$
    $\pi_1 \leftarrow \mathsf{CP_{poly}.Prove}(\mathsf{ek}_p, \vec{\sigma}, (c_1, c^*), (g_1, y^*), (o_1, o^*))$
    $\pi_F \leftarrow \mathsf{CP_{poly}.Prove}(\mathsf{ek}_p, (\vec{r}, \vec{\sigma}), (c_F, c^*), (\tilde{F}, \tilde{F}(\vec{r}, \vec{\sigma})), (o_F, o^*))$
    $\pi_{\mathsf{sc}} \leftarrow \mathsf{CP_{sc}.Prove}(\mathsf{ek}_s, g_0, (c_t, c_1, c_u), (t, \tilde{g}_1, \tilde{u}), (o_t, o_1, o_u))$
    $\pi := (c_t, o_t, c_1, c^*, y^*, o^*, \pi_1, \pi_F, \pi_{\mathsf{sc}})$

$\mathsf{CP_{lin}.VerProof}^*\left(\mathsf{vk}_F, \vec{x}, c_u, \pi\right) \to b \in \{0, 1\}$ :
    $\vec{r} \leftarrow H_1(c_F, c_u, \vec{x})$ ; $t \leftarrow \tilde{x}(\vec{r})$ ; $\vec{\sigma} \leftarrow H_2(c_F, c_1, \vec{r})$
    $b \leftarrow \mathsf{VerCommit}(\mathsf{cvk}, c_t, t, o_t) \wedge \mathsf{VerCommit}(\mathsf{cvk}, c^*, y^*, o^*)$
        $\wedge\ \mathsf{CP_{sc}.VerProof}(\mathsf{vk}_p, g_0, (c_t, c_1, c_u), \pi_{\mathsf{sc}})$
        $\wedge\ \mathsf{CP_{poly}.VerProof}(\mathsf{vk}_s, \vec{\sigma}, (c_1, c^*), \pi_1)$
        $\wedge\ \mathsf{CP_{poly}.VerProof}(\mathsf{vk}_p, (\vec{r}, \vec{\sigma}), (c_F, c^*), \pi_F)$
```

Figure 14: CP-SNARK for specializable universal R^{lin}

H.5 A CP-SNARK for data-parallel computations

In this section we discuss how a CP-SNARK for relations R^{par} and R^{parjnt}, and for the commitment scheme PolyCom of [71] can be obtained by merging ideas from [71] and [69]. Such a merge of techniques was hinted possible in [69]. Here we give more details on how such a scheme looks like. The main motivation of studying such a scheme is that the commitment part of the proof (and similarly a factor of the verification time) is $O(\log |w|)$, instead of $O(\sqrt{|w|})$.

An Abstract Version of Hyrax. Hyrax [69] is a zero-knowledge proof, based on discrete log in the random oracle model that is based on the CMT protocol [27]. Hyrax extends CMT, which is particularly suited for circuits composed of parallel identical basic blocks, by supporting non-determinism in zero-knowledge, as well as including other optimizations. Its basic structure as an interactive protocol: *(i)* the prover creates a commitment c_w to the witness $\vec{w}$ (a vector of field elements); *(ii)* the parties run a ZK variant of CMT (including optimizations from Giraffe++ [66]); *(iii)* the prover "links" together the outputs of steps *(i)* and *(ii)*. For this, it must prove that the MLE $\tilde{w}$ of the witness in c_w evaluated on a random point q_d is equal to another value y committed in ζ.

In Figure 15 we formalize this structure via a generic use of a commitment scheme for polynomials and a proof system for proving the correct evaluations of committed polynomials. For these two tools we use the syntax formalized in Appendix H. We call this scheme Hyrax-Abstract. It is clear from the security proof of [69] that one could rephrase their security statement so that Hyrax-Abstract has witness extended emulation if PolyCom is an extractable commitment and $\mathsf{CP_{poly}}$ is a NIZK argument of knowledge for polynomial evaluations.

Instantiating Hyrax-Abstract **with** PolyCom**.** We call Hyrax – PolyCom the instantiation of Hyrax-Abstract with the PolyCom commitment and $\mathsf{CP_{poly}}$ argument from [71] as described in Appendix G. This is essentially the only difference with the original Hyrax scheme that uses (an extension of) a matrix commitment of size $O(|w|^{1/l})$ and Bulletproof for proving polynomial evaluations with $O(|w|^{(l-1)/l})$ verification time. In HyrPoly there is instead a succinct commitment (of constant size) and a verification time, in step *(iii)*, of $O(\log(|w|))$.

Using HyrPoly **for Data-Parallel Computations.** Hyrax, and in particular its Gir^{++} core protocol, is designed to work on arithmetic circuits of fan-in two, consisting of N identical sub-computations, each having d layers and width at most G. For this class of circuits, considering Hyrax's cost analysis combined with the costs of PolyCom commitment and $\mathsf{CP_{poly}}$, we have that in HyrPoly: the verifier runs in time $O(|x| + |y| + dG + \lambda d \log(NG))$ and proofs have length $O(\lambda d \log(NG))$.

It is easy to see that the relation $R^{\mathsf{par}}((u_j)_{j\in[N]}) := \bigwedge_{j=1}^{N} R'(u_j)$ can be modeled with an arithmetic circuit C consisting of N copies of a circuit C' that outputs 0 on u_j iff $R'(u_j)$ holds.

If we instead consider a parallel relation with joint inputs, i.e., $R^{\mathsf{parjnt}}(u) := \bigwedge_{j=1}^{N} R'(u'_j)$ where each u'_j is a subset of the entries of u, a corresponding circuit can be built by taking the parallel C as for R^{par}, and by adding one layer – called *redistribution layer* (RDL) in [69] – that appropriately duplicates and redistributes wires from the input layer to the input wires of each C' copy. In the case of using an RDL, the verifier of Hyrax, and also in our HyrPoly scheme, incurs an additional overhead in running time of the verifier $O(|x|+|u|+NG)$. Essentially, for this break of parallelism the verifier must pay a cost in the total width of the circuit.

For the sake of our experiments, we call HyrPoly-Par the HyrPoly scheme executed on fully parallel circuits (no RDL), and we call HyrPoly-RDL the version of Hyrax – PolyCom executed with circuits with an RDL.

I A CP-SNARK FOR INTERNAL PRODUCTS FROM THALER'S PROTOCOL

In this section we show how to modify the zk-vSQL protocol of [71] in order to efficiently with a special class of circuits that simply consist of a tree of multiplications. The basic idea is to replace the CMT protocol over homomorphic commitment schemes proposed in [71] with an analogous version of the protocol proposed by Thaler [63] for the specific case of trees of multiplications. The advantage of this encoding is to bring the prover runtime linear in the number of gates in the circuit.

We first explain some preliminaries and then present this construction.

I.1 CMT Protocol

The CMT protocol [27] is a variant of the GKR protocol [37] where the prover runs in time $O(S \log S)$, where S is the size of the circuit.

```
Hyrax-Abstract.Setup($1^\lambda$) → ck :
ck ← PolyCom.Setup($1^\lambda$)
Hyrax-Abstract.KeyGen(ck) → (ek, vk) :
(ek, vk) ← $\mathsf{CP}_{\mathsf{poly}}$.KeyGen(ck)
Hyrax-Abstract.Prove(ek, $u$) → $\pi$ :
    $(c_{\tilde{u}}, o_{\tilde{u}})$ ← ComPoly(ck, $\tilde{u}$)
    $(\pi_{\mathsf{core}}, q_d, \zeta)$ ← ZK-Gir$^{++}$Core$_{\mathcal{P}}$(ek, $u$)
    $y \leftarrow \tilde{u}(q_d)$; $(c_y, o_y)$ ← ComVal(ck, $y$)
    $\pi_{\mathsf{eval}}$ ← $\mathsf{CP}_{\mathsf{poly}}$.Prove(ek, $q_d$, $(c_{\tilde{u}}, c_y)$, $(\tilde{u}, y)$, $(o_{\tilde{u}}, o_y)$)
    $\pi_{\mathsf{eq}}$ ← NIPoK-Eq$_{\mathcal{P}}(c_y, \zeta)$
    $\pi \leftarrow (c_{\tilde{u}}, \pi_{\mathsf{core}}, c_y, \pi_{\mathsf{eval}}, \pi_{\mathsf{eq}})$
Hyrax-Abstract.VerProof(vk, $c_{\tilde{u}}, \pi_{\mathsf{core}}, c_y, \pi_{\mathsf{eval}}, \pi_{\mathsf{eq}}$) :
    $(q_d, \zeta)$ ← ZK-Gir$^{++}$Core$_{\mathcal{V}}$(vk, $\pi_{\mathsf{core}}$)
    Run and test CheckCom(vk, $c_{\tilde{u}}$) and CheckCom(vk, $c_y$)
    Run and test $\mathsf{CP}_{\mathsf{poly}}$.VerProof(vk, $q_d$, $c_{\tilde{u}}$, $c_y$, $\pi_{\mathsf{eval}}$)
    Run and test NIPoK-Eq$_{\mathcal{V}}(\pi_{\mathsf{eq}}, c_y, \zeta)$
    Accept if all tests above pass
```

Figure 15: Pseudocode for Hyrax-Abstract**.**

This protocol provides a proof that an element is the output of a circuit evaluated over a certain input. That is $y = C(\vec{x})$, where C is a circuit of depth d, $\vec{x}$ are the wires of layer d and y is claimed to be the output wire of the first layer 0. In short, the prover reduces recursively a claim on layer i to another claim on layer $i+1$, until he obtains a publicly verifiable claim on the input. In order to do that, both prover and verifier engage in a sum-check protocol for each layer, using one polynomial representing the values of the wires in layer i. Its multilinear extension links layer i (of size s_i) to layer $i+1$ by a summation of wiring predicates as follows

$$\tilde{V}_i(\vec{q}) = \sum_{\substack{\vec{b}\in\{0,1\}^{s_i} \\ \vec{l},\vec{r}\in\{0,1\}^{s_{i+1}}}} g_{\vec{q}}^{(i)}(\vec{b},\vec{r},\vec{l}) := \sum_{\substack{\vec{b}\in\{0,1\}^{s_i} \\ \vec{l},\vec{r}\in\{0,1\}^{s_{i+1}}}} \tilde{\beta}_i(\vec{q},\vec{b}) \cdot \dots$$
$$\left(\widetilde{\mathsf{add}}_{i+1}(\vec{l},\vec{r},\vec{b})\cdot(\tilde{V}_{i+1}(\vec{l}) + \tilde{V}_{i+1}(\vec{r})) + \widetilde{\mathsf{mul}}_{i+1}(\vec{l},\vec{r},\vec{b})\cdot\tilde{V}_{i+1}(\vec{l})\cdot\tilde{V}_{i+1}(\vec{r})\right)$$

where $\tilde{V}_i$ returns the value of one gate, $\tilde{\beta}_i(\vec{q},\vec{b}) = \vec{q} \overset{?}{=} \vec{b}$ is a selector function, and $\widetilde{\mathsf{opn}}_i(\vec{l},\vec{r},\vec{b})$ checks whether the value of gate $\vec{b}$ at layer i is the result of an opn $\in$ {add, mul} addition or multiplication gate with $\vec{l}$ and $\vec{r}$ being its left and right inputs in the $(i+1)^{th}$–layer.

The standard version of the protocol suggests that for each layer of the circuit the verifier has to check two claims. This results in $O(2^d)$ calls to the sum-check protocol. However, an ingenious technique shows how to use a single claim per layer using a line through both values. Then the verifier chooses one random point on which they perform a single sum-check invocation per layer, resulting in $O(d)$ calls.

I.2 Thaler's Protocol for Trees of Multiplications

In [63], Thaler proposes another variation of the CMT/GKR protocol [27, 37] for some specific classes of circuits, allowing for a logarithmic factor reduction in the prover's runtime. One of the his protocols takes advantage of circuits where all gates perform the same operation, and whose wires are settled in a binary tree structure. He denotes these regular circuits by *trees of multiplications* or *additions*. This section only shows the notation of the former one due to its suitability for our construction of $\mathsf{CP}_{\mathsf{sfprm}}$. Nonetheless, moving to the addition case is straightforward.

The main difference that will be discussed here is a different polynomial for sum-check, as well as the notation of the wiring predicates. Thaler's protocol assumes highly structured wiring in order to reduce the number of arguments of the predicates. Namely, given a gate at layer i with label $\vec{b} \in \{0,1\}^{s_i}$, we assume its value is the result of a multiplication of gates of layer $i+1$ with labels $(\vec{b}|0) \in \{0,1\}^{s_i+1}$ and $(\vec{b}|1) \in \{0,1\}^{s_i+1}$. This means, the number of inputs to the circuit is a power of two and each layer has half the size of its preceding one. On this basis, the resulting polynomial of each layer is much simpler as shown below:

$$\tilde{V}_i(\vec{q}) = \sum_{\vec{b}\in\{0,1\}^{s_i}} g_{\vec{q}}^{(i)}(\vec{b}) = \sum_{\vec{b}\in\{0,1\}^{s_i}} \tilde{\beta}_i(\vec{q},\vec{b}) \cdot \tilde{V}_{i+1}(\vec{b}|0) \cdot \tilde{V}_{i+1}(\vec{b}|1)$$

This tweak, together with a series of precomputations of $\tilde{\beta}_i(\vec{q},\vec{b})$ and $\tilde{V}_{i+1}(\vec{b})$ values allows to obtain a linear-time prover.

I.3 Adapting zk-vSQL to Thaler's Protocol

Here we show how to change the CMT protocol over homomorphic commitments in [71, Construction 3] in order to work with circuits that are a tree of multiplication gates using Thaler's representation ?? to achieve faster prover runtime. From the point of view of security, this modification of [71, Construction 3] does not require any significant change; essentially a proof would be a rewrite of the one in [71], and we leave it for an extended version of the paper. The precise description of the protocol is however interesting and therefore we give it for completeness in Figure 16.

Let $C : \mathbb{F}^m \to \mathbb{F}$ be a depth-d binary tree of multiplications such that $C(\vec{y}) = z$ represents the operation $z = \prod_{i=1}^{m} y_i$ where m is a power of two, and let ck ← Setup(1^λ) be a commitment key of a linearly homomorphic commitment scheme. The protocol in Figure 16 allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $C(\vec{y}) = z$ with respect to $\vec{y}$ and z committed in $\{c_{y_j}\}_{j\in\{1\dots m\}}$ and c_z.

As in [71], let $\mathsf{ZK}_{\mathsf{eq}}$ (resp. $\mathsf{ZK}_{\mathsf{prod}}$) be a zero-knowledge argument of knowledge for testing equality of two committed values (resp. the product relation between three commitments).

A Succinct Zero Knowledge Argument for R_{prd}. In Figure 17 we give the succinct version of the protocol $\mathsf{TTM}^{\mathsf{Com}}$ presented in Figure 16. The protocol is almost identical to Construction 4 in [71] except for a few simplifications due to the fact that in our case the input and output of the circuit are assumed to be already committed and these commitments are known to the verifier, and that all the input is committed (i.e., there is no public input). Basically, the idea is that prover and verifier run the $\mathsf{TTM}^{\mathsf{Com}}$ protocol until they

```
$\mathsf{TTM}^{\mathsf{Com}}$ :
1 : Common input: cvk ; $r_0 = 0$ ; $c_0 := c_z$ ; $(c_{y_j})_{j\in\{1\dots m\}}$
2 : $\mathcal{P}$ input: ck ; $t_0 := z$ ; $o_0 := o_z$ ; $\vec{y}$ ; $(o_{y_j})_{j\in\{1\dots m\}}$
3 : for $i = 0 \dots d-1$ :
4 :   Run Step 1 of Construction 2 [71] (sum-check over homomorphic
5 :   commitments) on the claim $t_i = V_i(\vec{r}_i) = \sum_{b\in\{0,1\}^{s_i}} g^{(i)}_{\vec{r}_i}(\vec{b})$
6 :   At the end of Step 1, $\mathcal{P}$ and $\mathcal{V}$ hold $\vec{r}'_i \in \mathbb{F}^{s_i}$,
7 :   and commitment $c'_i$ to $t'_i = g^{(i)}_{\vec{r}_i}(\vec{r}'_i)$
8 :   $\mathcal{P}$ : Claims that VerCommit(cvk, $c'_i$, $t'_1$, $o'_i$) = 1
9 :   $\mathcal{P}$ : $(c_R, o_R) \leftarrow$ ComVal$\big($ck, $v_R := \tilde{V}_{i+1}(\vec{r}'_i|0)\big)$
10 :   $\mathcal{P}$ : $(c_L, o_L) \leftarrow$ ComVal$\big($ck, $v_L := \tilde{V}_{i+1}(\vec{r}'_i|1)\big)$
11 :   $\mathcal{P}$ : $(c^*, o^*) \leftarrow$ ComVal$\big($ck, $v^* := v_L \cdot v_R\big)$
12 :   $\mathcal{P} \to \mathcal{V}$ : $c_R$, $c_L$, $c^*$
13 :   $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{ZK}_{\mathsf{prod}}\big($ck, $(c_L, c_R, c^*); ((v_L, v_R, v^*), (o_L, o_R, o^*))\big)$
14 :   $\mathcal{P}$ : $(c^*_i, o^*_i) \leftarrow$ HomEval(cvk, $\tilde{\beta}_i(\vec{r}_i, \vec{r}'_i)$, $c^*$, $o^*$)
15 :   $\mathcal{V}$ : $(c^*_i, \cdot) \leftarrow$ HomEval(cvk, $\tilde{\beta}_i(\vec{r}_i, \vec{r}'_i)$, $c^*$, $\cdot$)
16 :   $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{ZK}_{\mathsf{eq}}\big($ck, $(c'_i, c^*_i); (t'_i, (o'_i, o^*_i))\big)$
17 :   $\mathcal{P}$ : Compute $(c_{\ell_j}, o_{\ell_j}) \leftarrow$ ComVal(ck, $\ell_j$)
18 :      where $\ell(\rho) = \tilde{V}_{i+1}(\vec{r}'_i|\rho)$ for $\rho \in \mathbb{F}$ and $\ell_j$ its coefficients
19 :   $\mathcal{P} \to \mathcal{V}$ : $\{c_{\ell_j}\}_{j\in\{0\dots s_{i+1}\}}$
20 :   $\mathcal{P}$ : $c_{\ell(0)} \leftarrow c_{\ell_0}$
21 :      $(c_{\ell(1)}, o_{\ell(1)}) \leftarrow$ HomEval(cvk, $(1, \dots, 1)$, $\{c_{\ell_j}, o_{\ell_j}\}_{j\in[0, s_{i+1}]}$)
22 :   $\mathcal{V}$ : $c_{\ell(0)} \leftarrow c_{\ell_0}$ ; $(c_{\ell(1)}, \cdot) \leftarrow$ HomEval(cvk, $(1, \dots, 1)$, $\{c_{\ell_j}\}$, $\cdot$)
23 :   $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{ZK}_{\mathsf{eq}}\big($ck, $(c_R, c_{\ell(0)}); (v_R, o_R, o_{\ell_0})\big)$
24 :   $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{ZK}_{\mathsf{eq}}\big($ck, $(c_L, c_{\ell(1)}); (v_L, o_L, o_{\ell(1)})\big)$
25 :   $\mathcal{V} \to \mathcal{P}$ : $r''_i \leftarrow\!\!\$\, \mathbb{F}$ and define $\vec{r}_{i+1} \leftarrow (\vec{r}'_i|r''_i)$
26 :   $\mathcal{V}$ : $(c_{i+1}, \cdot) \leftarrow$ HomEval$\big($ck, $(1, r''_i, \dots, r''^{s_{i+1}}_i)$, $\{c_{\ell_j}\}_{j\in[0, s_{i+1}]}$, $\cdot\big)$
27 :   $\mathcal{P}$ : $\vec{r}_{i+1} \leftarrow (\vec{r}'_i|r''_i)$ ; $t_{i+1} \leftarrow \tilde{V}_{i+1}(\vec{r}_{i+1})$
28 :   $\mathcal{P}$ : $(c_{i+1}, o_{i+1}) \leftarrow$ HomEval$\big($ck, $(1, r''_i, \dots, r''^{s_{i+1}}_i)$, $\{c_{\ell_j}, o_{\ell_j}\}_{j\in[0, s_{i+1}]}\big)$
29 : endfor
30 : $\mathcal{P} \to \mathcal{V}$ : $\vec{y}$ ; $(o_{y_j})_{j\in\{1\dots m\}}$ ; $o_0$
31 : $\mathcal{V}$ : $\{$VerCommit(cvk, $c_{y_j}$, $y_j$, $o_{y_j}$)$\}_{j=1}^{m}$ ; VerCommit(cvk, $c_0$, $t_0$, $o_0$)
32 : $\mathcal{V}$ : $(c^*_y, o^*_y) \leftarrow$ ComVal(ck, $\tilde{V}_y(\vec{r}_d)$) where MLE$(V_y(j) = y_j) = \tilde{V}_y$
33 : $\mathcal{V} \to \mathcal{P}$ : $o^*_y$
34 : $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{ZK}_{\mathsf{eq}}\big($ck, $(c^*_y, c_d); (\tilde{V}_y(\vec{r}_d), o^*_y, o_d)\big)$
```

Figure 16: Thaler's tree of multiplications over homomorphic commitment schemes. Main differences from [71, Construction 3] in blue

get to the end of the last round (line 29). Then the last lines of $\mathsf{TTM}^{\mathsf{Com}}$ – in which the prover opens the commitments to input and output and the verifier gets convinced that c_d opens to $\tilde{y}(\vec{r}_d)$ – are replaced with a step that does the same: the prover uses $\mathsf{CP}_{\mathsf{poly}}$ to prove that c_d opens to $\tilde{y}(\vec{r}_d)$ with respect to the commitment c_y. For the polynomial commitments and the proof system for their evaluations we use our notation of Section H.

```
SuccinctZK – TTM :
Preprocessing: generate the commitment key
  (ck, cvk) ← PolyCom.Setup($1^\lambda$)
   for m-variate multilinear polynomials.
  (ek, vk) ← $\mathsf{CP}_{\mathsf{poly}}$.Setup(ck)
Evaluation: on common input $(c_y, c_z)$ ; $\mathcal{P}$ input $(z, \tilde{y}, o_y)$
$\mathcal{V}$ : CheckCom(vk, $c_y$) ∧ CheckCom(vk, $c_z$)
$\mathcal{P}$, $\mathcal{V}$ : Execute $\mathsf{TTM}^{\mathsf{Com}}$ until line 29 :
  Both hold $\vec{r}_d$, $c_d$; $\mathcal{P}$ holds and opening $o_d$ of $\tilde{V}_d(\vec{r}_d) = \tilde{y}(\vec{r}_d)$
$\mathcal{P} \to \mathcal{V}$ : $\pi_y \leftarrow \mathsf{CP}_{\mathsf{poly}}$.Prove(ek, $\vec{r}_y$, $(c_y, c_d)$, $(\tilde{y}, \tilde{y}(\vec{r}_d))$, $(o_y, o_d)$)
$\mathcal{V}$ : $\mathsf{CP}_{\mathsf{poly}}$.VerProof(vk, $\vec{r}_d$, $c_{\tilde{V}_d}$, $c_d$, $\pi_d$)
```

Figure 17: Succinct zero-knowledge argument for $\mathsf{TTM}^{\mathsf{Com}}$

J CP-SNARKS FOR CIRCUITS

Here we formalize the ideas sketched in Section 6 in order to use the modular commit-and-prove approach to obtain new zkSNARKs for computation expressible by arithmetic circuits. Precisely, we show new CP-SNARKs for (1) arithmetic circuit satisfiability, and (2) parallel computation on joint inputs.

In both constructions the idea is to break the target problem into the conjunction of simpler relations with shared input. Once having done this, and assuming the existence of CP-SNARKs for these simpler relations and that share the same commitment scheme, we immediately obtain a CP-SNARK for the target problem by applying our composition Theorem 3.2. Furthermore, thanks to our lifting transformation of Section 3.5 sharing the same commitment scheme is not a restricting requirement.

Preliminaries: equalities among vector entries. A common building block in both schemes of this section is a system for proving that the entries of a vector satisfy a set of equalities between them. Namely, given a set S of pairs of indices (i, k), we define a relation R^{veq}_S that holds for a vector $\vec{u}$ iff $u_i = u_k$ for all $(i, k) \in S$. In what follows we discuss different ways to encode this relation:

Definition J.1 (Relation for equalities among vector entries). Let $\mathcal{D}$ be some domain (e.g., a finite field $\mathbb{F}$), let $n_0, \dots, n_\ell$ be positive integers such that $\mathcal{D}_j := \mathcal{D}^{n_j}$ and let $m = \sum_{j=0}^{\ell} n_j$. Given a set $S = \{(i_1, k_1), \dots, (i_l, k_l)\} \subset [m] \times [m]$, we define a relation R^{veq}_S over $\mathcal{D}_0 \times \cdots \times \mathcal{D}_\ell = \mathcal{D}^m$ such that: $R^{\mathsf{veq}}_S(\vec{y} := (\vec{x}, (\vec{u}_j)_{j\in[\ell]})) = 1 \iff \forall (i, k) \in S : y_i = y_k$.

In what follows we discuss different ways to encode the above relation.

Relation R^{veq}_S can be expressed using R^{sfprm}_ϕ (see Section 5.4)for an appropriate permutation ϕ that encodes S. The idea is that a set $S \subset [m] \times [m]$ can be seen as the description of an undirected graph with $2m$ vertices. From S it is possible to extract another set $S' \subset [m] \times [m]$ that contains a cycle $((i_1, k_1), \dots, (i_t, k_t))$ for every connected component of the graph represented by S. Taking the product of all the cycles in S' defines a permutation $\phi : [m] \to [m]$ such that $\forall (i, k) \in S : y_i = y_k$ iff $\forall j \in [m] : y_j = y_{\phi(j)}$.

Then for such ϕ computed from S we have $R^{\mathsf{veq}}_S(\vec{x}, (\vec{u}_j)_{j\in[\ell]}) \iff R^{\mathsf{sfprm}}_\phi(\vec{x}, (\vec{u}_j)_{j\in[\ell]})$.

At this point one can either design a proof system for R^{sfprm}_ϕ (as in Section 5.4) or use an alternative encoding of R^{sfprm}_ϕ based on linear constraints. The idea is to evaluate the relation on a vector $\vec{y} \in \mathbb{F}^m$ by checking that $\mathbf{Z} \cdot \vec{y} = \vec{0}$, where $\mathbf{Z} \in \mathbb{F}^{m'\times m}$, with $m' \leq m$, is the matrix obtained by removing the zero rows from $(\mathbf{I} - \Sigma_\phi) \in \mathbb{F}^{m\times m}$, where Σ_ϕ is the permutation matrix representing ϕ. Then clearly $R^{\mathsf{sfprm}}_\phi(\vec{x}, (\vec{u}_j)_{j\in[\ell]})$ holds iff $R^{\mathsf{lin}}_{\mathbf{Z}}(\vec{0}, \vec{x}, (\vec{u}_j)_{j\in[\ell]})$ holds, where the relation R^{lin}, modelling the linear property over (committed) vectors, is formally defined as follows.

Definition J.2 (Linear property relation). Let $n_1, n_2, m_1, \ldots, m_\ell$ be integers such that $\{\mathcal{D}_{x,j} := \mathcal{D}^{n_j}\}_{j\in[1,2]}$, $\{\mathcal{D}_{u,j} := \mathcal{D}^{m_j}\}_{j\in[\ell]}$, and $m = n_2 + \sum_{j=1}^{\ell} m_j$. Given a matrix $\mathbf{F} \in \mathcal{D}^{n_1\times m}$, we define a relation $R^{\mathsf{lin}}_{\mathbf{F}}$ over $\mathcal{D}_{x,1} \times \mathcal{D}_{x,2} \times \mathcal{D}_{u,1} \times \cdots \times \mathcal{D}_{u,\ell}$ such that:

$$R^{\mathsf{lin}}_{\mathbf{F}}(\vec{x}_1, \vec{x}_2, (\vec{u}_j)_{j\in[\ell]}) = 1 \iff \mathbf{F}\cdot\vec{y} = \vec{x}_1, \text{ where } \vec{y} := (\vec{x}_2, (\vec{u}_j)_{j\in[\ell]})$$

Note that the above relation R^{lin} is slightly different from the one supported by $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ of Section 4.2. The only difference is that in $\mathsf{CP}^{\mathsf{Ped}}_{\mathsf{lin}}$ the linear function is not applied over public inputs. However, this small discrepancy can be easily solved by adding a commitment to the additional public input $\vec{x}_2$ and opening this commitment.

J.1 Arithmetic Circuit Satisfiability

Let us consider the problem of arithmetic circuit satisfiability.

Definition J.3. Let $C : \mathbb{F}^{n_x} \times \mathbb{F}^{n_w} \to \mathbb{F}^l$ be an arithmetic circuit, where $n_x, n_w, l \in \mathbb{N}$ denote input, witness and output length. We define the arithmetic circuit satisfiability relation $R^{\mathsf{ac}}_C(\vec{x}, \vec{w})$ as the set of pairs such that $C(\vec{x}, \vec{w}) = \vec{0}^l$.

We show two solutions to model the above relation using a commit-and-prove paradigm. The first one is similar to that of Groth [39] (recently used in [19]) and encodes arithmetic circuit satisfiability using Hadamard products, additions and permutations of (committed) vectors. The second one relies on the encoding put forward by Bootle et al. [18] that reduces the relation R^{ac} to an Hadamard product and a set of linear constraints.

Arithmetic Circuit Satisfiability through Hadamard, Addition and Equalities. Any arithmetic circuit C consists of N_A addition gates, N_M multiplication gates, both of fan-in 2, and N_C multiplication-by-constant gates, of fan-in 1. Each gate has a left input, a right input and an output wire;[21] also each output wire can be input to another gate. This means that C can be described by integers N_A, N_M, N_C, a vector $\vec{c} \in \mathbb{F}^{N_C}$ of constants, and the wiring information saying that the output wire of addition/multiplication i is the left/right input of addition/multiplication gate j. With such a representation $\exists\vec{w} : C(\vec{x}, \vec{w}) = \vec{0}^l$ can be encoded by showing the existence of an assignment to the inputs and outputs of C's gates that satisfies every gate, that is consistent with the wiring of C as well as with the public input $\vec{x}$ and the output $\vec{0}$.

More formally, consider an arithmetic circuit $C : \mathbb{F}^{n_x} \times \mathbb{F}^{n_w} \to \mathbb{F}^l$ with N_A addition gates, N_M multiplication gates, and N_C multiplication by constant gates, and where we split the witness $\vec{w}$ between committed witness $\vec{u} \in \mathbb{F}^{n_u}$ and free witness $\vec{\omega} \in \mathbb{F}^{n_\omega}$. Assume we arrange the wires of C so as to have, orderly: the n_x input wires, the n_u committed witness wires, the l output wires, the $3N_A$ left, right and output wires of the addition gates, the $3N_M$ left, right and output wires of the multiplication gates, and the $2N_C$ input and output wires of the multiplication-by-constant gates. All these wires can be indexed by integers from 1 to $m = n_x + n_u + l + 3(N_A + N_M) + 2N_C$, and the wiring information of C can be described by a set S of pairs $(i, k) \in [m] \times [m]$ indicating that the wire at position i is connected to the wire at position k.

Therefore we model an arithmetic circuit C with a tuple $(n_x, n_u, l, N_A, N_M, N_C, \vec{c}, S)$. Then proving $\exists(\vec{u}, \vec{\omega})\ R^{\mathsf{ac}}_C(\vec{x}, \vec{u}, \vec{\omega})$ can be done by proving the existence of a vector $\vec{u}_w$, that is the concatenation of vectors $\vec{u}_w := (\vec{u}^A_L, \vec{u}^A_R, \vec{u}^A_O, \vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O, \vec{u}^C_I, \vec{u}^C_O)$, such that

$$\begin{aligned} R^{\mathsf{ac}}_C(\vec{x}, \vec{u}, \vec{u}_w) \quad &:= R_{\mathsf{add}}(\vec{u}^A_L, \vec{u}^A_R, \vec{u}^A_O) \wedge R_{\mathsf{had}}(\vec{u}^C_I, \vec{c}, \vec{u}^C_O) \\ &\wedge R_{\mathsf{had}}(\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O) \wedge R^{\mathsf{veq}}_S((\vec{x}, \vec{0}), \vec{u}, \vec{u}_w) \end{aligned}$$

where $R_{\mathsf{add}}(\vec{u}^A_L, \vec{u}^A_R, \vec{u}^A_O)$ is the relation expressing the predicate $\vec{u}^A_L + \vec{u}^A_R \overset{?}{=} \vec{u}^A_O$, and $R_{\mathsf{had}}(\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O)$ is the Hadamard product relation $\vec{u}^M_L \circ \vec{u}^M_R \overset{?}{=} \vec{u}^M_O$ (i.e., $u^M_{L,j} \cdot u^M_{R,j} = u^M_{O,j}$ for all $j \in [3N_M]$).

If Com is a linearly homomorphic and extractable commitment scheme, a proof system for R_{add} comes for free. Therefore, by definition of R^{ac}_C and our Theorem 3.2 we obtain the following corollary.

Corollary J.4. *If there exist CP-SNARKs* $\mathsf{CP}_{\mathsf{had}}$ *and* $\mathsf{CP}_{\mathsf{veq}}$ *for a linearly-homomorphic extractable commitment scheme* Com *and for relations* R_{had} *and* R^{veq} *respectively, then there is a CP-SNARK* $\mathsf{CP}_{\mathsf{ac}}$ *for* Com *and relation* R^{ac}_C.

Arithmetic Circuit Satisfiability through Hadamard and Linear Constraints. Following [18, 23], an arithmetic circuit C can be described by a tuple $(n_x, n_u, N, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_x, \mathbf{W}_U, \vec{c})$ where n_x and n_u are the input and (committed) witness lengths respectively, N is the number of multiplication gates, and the matrices $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{F}^{Q\times N}, \mathbf{W}_x \in \mathbb{F}^{Q\times n_x}, \mathbf{W}_U \in \mathbb{F}^{Q\times n_u}$ and vector $\vec{c} \in \mathbb{F}^Q$ describe a system of linear equations over the wires of C. Using such a definition, C is satisfied by $(\vec{x}, \vec{u})$ if there exist three vectors $\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O \in \mathbb{F}^N$ such that

$$\vec{u}^M_L \circ \vec{u}^M_R = \vec{u}^M_O \ \wedge\ \mathbf{W}_L \cdot \vec{u}^M_L + \mathbf{W}_R \cdot \vec{u}^M_R + \mathbf{W}_O \cdot \vec{u}^M_O + \mathbf{W}_x \cdot \vec{x} + \mathbf{W}_U \cdot \vec{u} = \vec{c}$$

which for $\mathbf{F} = (\mathbf{W}_x, \mathbf{W}_U, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O) \in \mathbb{F}^{Q\times(n_x+n_u+3N)}$ means

$$R^{\mathsf{ac}}_C(\vec{x}, \vec{u}, \vec{u}_w) := R_{\mathsf{had}}(\vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O) \wedge R^{\mathsf{lin}}_{\mathbf{F}}(\vec{c}, (\vec{x}, \vec{u}, \vec{u}^M_L, \vec{u}^M_R, \vec{u}^M_O))$$

By the above definition of R^{ac}_C and our Theorem 3.2 we obtain:

Corollary J.5. *If there exist CP-SNARKs* $\mathsf{CP}_{\mathsf{had}}$ *and* $\mathsf{CP}_{\mathsf{lin}}$ *for a commitment scheme* Com *and for relations* R_{had} *and* R^{lin} *respectively, then there is a CP-SNARK* $\mathsf{CP}_{\mathsf{ac}}$ *for* Com *and relation* R^{ac}_C.

J.2 Parallel Computation on Joint Inputs

Consider relations $R^{\mathsf{parjnt}}(u) := \bigwedge_{j=1}^{N} R'(u'_j)$ where each u'_j is a subset of the entries of u.

One way to deal with R^{parjnt} is by defining the arithmetic circuit that computes it (cf. Fig. 2b). The Hyrax system is particularly designed for parallel circuits [69]; they deal with non-parallel input by introducing a (non-parallel) redistribution layer (RDL) layer that *redistributes* the input and feeds it to the identical sub-circuits at

[21] We model gates of fan-in 1 as having only a left input.

the next level. Unfortunately an effect of using an RDL is that the verifier must pay an additional cost linear in the total width of the circuit. This makes verification time pretty high in applications like the Merkle tree example above.

Here we propose another natural modeling of relations with joint inputs, that is the simple conjunction of two relations: R^{par} that models fully parallel checks of some R' on *disjoint* inputs, and another relation that models the consistency of the shared inputs across the (fully) parallel executions. The advantage of this encoding is that R^{par} is now fully parallel and one could use for it a system for parallel computation without any caveat, whereas to check the consistency of shared input one can use a system for the R^{veq} relation from Definition J.1.

More formally, we define a parallel relation on disjoint inputs as follows.

Definition J.6 (Parallel relation on disjoint inputs). For a relation R' over $\mathcal{D}'$ and an integer $N \geq 1$, a parallel relation $R^{\mathsf{par}}_{R'}$ on *disjoint* inputs is defined as $R^{\mathsf{par}}_{R'}(\vec{u} := (u_j)_{j\in[N]} \in (\mathcal{D}')^N) := \bigwedge_{j=1}^{N} R'(u_j)$.

From R^{par} and R^{veq} we define a relation for parallel checks on joint inputs.

Definition J.7 (Parallel relation on joint inputs). Let $n_0, n_1, n', N \in \mathbb{N}$ be integers such that $n', N \geq 1$ and $n_0, n_1 \geq 0$, and let $m = n_0 + n_1 + N \cdot n'$. Let $\mathcal{D}$ be some domain, R' be a relation over $\mathcal{D}' := \mathcal{D}^{n'}$, and a set $S = \{(i_1, k_1), \ldots, (i_l, k_l)\} \subset [m] \times [m]$. $R^{\mathsf{parjnt}}_{R',S}$ is a relation over $\mathcal{D}_x \times \mathcal{D}_1 \times \mathcal{D}_2$, with $\mathcal{D}_x := \mathcal{D}^{n_0}$, $\mathcal{D}_1 := \mathcal{D}^{n_1}$ and $\mathcal{D}_2 := \mathcal{D}^{Nn'}$, such that: $R^{\mathsf{parjnt}}_{R',S}(\vec{x}, \vec{u}_1, \vec{u}_2) := R^{\mathsf{par}}_{R'}(\vec{u}_2) \wedge R^{\mathsf{veq}}_{S}(\vec{x}, \vec{u}_1, \vec{u}_2)$

Basically, $R^{\mathsf{parjnt}}_{R',S}$ models the parallel checking of R' on N different subsets of the entries of $(\vec{x}, \vec{u}_1)$ (consisting of a public $\vec{x}$ and committed $\vec{u}_1$) where such subsets are defined by the set S, and their concatenation is the vector $\vec{u}_2$. Alternatively, if $\vec{x}, \vec{u}_1$ are empty, $R^{\mathsf{parjnt}}_{R',S}$ models the parallel checking of R' on N different sets of inputs with some shared values (as specified by S).

From the definition of R^{parjnt} and our Theorem 3.2 we obtain the following corollary.

Corollary J.8. *If there exist CP-SNARKs* $\mathsf{CP}_{\mathsf{par}}$ *and* $\mathsf{CP}_{\mathsf{veq}}$ *for a commitment scheme* Com *relations* R^{par} *and* R^{veq} *respectively, then there is a CP-SNARK* $\mathsf{CP}_{\mathsf{parjnt}}$ *for* Com *and relations* R^{parjnt}.

K COMMIT AND PROVE SNARKS FROM EXISTING SCHEMES

In this section we give details supporting our claims of Section 3.4.

Background on Quadratic Arithmetic Programs. Since several of the SNARKs considered in this section rely on quadratic arithmetic programs [33] here we recall this notion.

Definition K.1 (QAP [33]). A Quadratic Arithmetic Program (QAP) $Q = (\mathcal{A}, \mathcal{B}, C, t(Z))$ of size m and degree d over a finite field $\mathbb{F}$ is defined by three sets of polynomials $\mathcal{A} := \{a_i(Z)\}_{i=0}^{m}, \mathcal{B} := \{b_i(Z)\}_{i=0}^{m}, C := \{c_i(Z)\}_{i=0}^{m}$ of degree $\leq d-1$, and a target degree-d polynomial $t(Z)$. Given Q we define a relation R_Q over pairs $(\vec{x}, \vec{w}) \in \mathbb{F}^n \times \mathbb{F}^{m-n}$ that holds iff there exists a polynomial $h(X)$ (of degree at most $d-2$) such that:

$$\left(\sum_{k=0}^{m} y_k \cdot a_k(Z)\right) \cdot \left(\sum_{k=0}^{m} y_k \cdot b_k(Z)\right) = \left(\sum_{k=0}^{m} y_k \cdot c_k(Z)\right) + h(Z)t(Z) \tag{1}$$

where $y_0 = 1$, $y_k = x_k$ for all $k = 1$ to n, and $y_k = w_{k-n}$ for $k = n+1$ to m.

K.1 "Adaptive Pinocchio" [65]

The Adaptive Pinocchio scheme proposed in [65] yields a CP-SNARK for QAP relations $R_Q(\vec{x}, \vec{u}, \vec{\omega})$. First, note that [65] already presents the scheme as a commit-and-prove SNARK for QAP relations $R_Q(\vec{u}_1, \ldots, \vec{u}_\ell, \vec{\omega})$, and for an extractable trapdoor commitment scheme, which is the one proposed by Groth in [40]. Second, observe that the commitment key consists of two vectors $\vec{S} := [1, s, s^2, \ldots, s^d]_1$, $\vec{S}' := [\alpha, \alpha s, \alpha s^2, \ldots, \alpha s^d]_2$, for random $s, \alpha \leftarrow\!\!\$\, \mathbb{Z}_q$, and the commitment to $\vec{u}_j$ is a pair $(C, C') = (r, \vec{u}_j^{\top}) \cdot (\vec{S}, \vec{S}')$. To see how this implies a CP-SNARK for $R_Q(\vec{x}, \vec{u}, \vec{\omega})$, consider $\ell = 2$ so that the first input $\vec{u}_1$ is used for the public input $\vec{x}$ (the corresponding commitment can be a dummy one) and the second one for the actual committed value $\vec{u}$. Also, to fit our syntax let C be the actual commitment whereas C' is part of the proof.

K.2 Lipmaa's Hadamard Product Argument [51]

The product argument proposed by Lipmaa in [51] is a commit-and-prove SNARK for the Hadamard product relation $R_{\mathsf{had}}(\vec{a}, \vec{b}, \vec{c})$. In this case the commitment key ck are two vectors $\vec{S} := [Z(\chi), \ell_1(\chi), \ldots, \ell_m(\chi)]_1^{\top}$ and $\vec{S}' := [\gamma Z(\chi), \gamma \ell_1(\chi), \ldots, \gamma \ell_m(\chi)]_2^{\top}$, for random $\chi, \gamma \leftarrow\!\!\$\, \mathbb{Z}_q$, where, for m a power of two and ω the m-th root of unity modulo q, $Z(X) = \prod_{i=1}^{m}(X - \omega^{i-1})$ and $\ell_i(X)$ is the i-th Lagrange basis polynomial (such distribution of ck guarantees binding under the m-PDL assumption [50, 51]). A commitment to $\vec{a}$ is a pair $(A_1, A_2) = (r_a, \vec{a}^{\top}) \cdot (\vec{S}, \vec{S}')$ (and similarly to $\vec{b}, \vec{c}$). As in the previous section, to fit our CP-SNARK syntax we can think of A_1, B_1, C_1 as the actual commitments and let their "knowledge components" as part of the proof.

K.3 zk-vSQL [71]

The zk-vSQL protocol [71] is a CP-SNARK for relations $R((u_j)_{j\in[\ell]})$[22] where R is an arithmetic circuit (that we assume to output some constant, e.g., 0, on acceptance), and for the commitment scheme PolyCom introduced in [71] and recalled in Appendix G.[23] The commit and prove capability is immediate by the construction and security of [71]. In what follows we observe that their commitments can also be seen as a variant of extended Pedersen commitment. This observation is crucial to see that we can apply our lifting transformation using our $\mathsf{CP}_{\mathsf{link}}$ scheme to zk-vSQL. Let us recall that for an input $\vec{u} \in \mathbb{Z}_q^m$ (for some $m = 2^{\mu}$), its commitment is ComPoly(ck, $\tilde{u}$) where $\tilde{u}$ is the multilinear extension of $\vec{u}$ (cf. section 5.1 about multilinear extensions). In particular, such MLE is

[22] Precisely, although the scheme in [71] is described with a single u, the same technique used in its predecessor [70] trivially allows to let it work with multiple commitments.
[23] Here we are considering the non-interactive version in the random oracle model obtained after applying the Fiat-Shamir transform.

the following μ-variate multilinear polynomial

$$\tilde{u}(X_1, \ldots, X_\mu) = \sum_{i=0}^{m-1} \chi_i(X_1, \ldots, X_\mu) \cdot u_{i+1}$$

Since c returned by ComPoly(ck, $\tilde{u}, \rho$) is defined as $[\tilde{u}(s_1, \ldots, s_\mu) + \rho s_{\mu+1}]_1$ and the common reference string includes the monomials $[\prod_{j \in W} s_j]_1$ for all possible subsets of indices W needed to evaluate such a $\tilde{u}$, c can also be seen as a Pedersen commitment $c = (\rho, u^\top) \cdot [s_{\mu+1}, \chi_0(s_1, \ldots, s_\mu), \ldots, \chi_{m-1}(s_1, \ldots, s_\mu)]_1^\top = (\rho, u^\top) \cdot \mathsf{ck}$, where the elements $[\chi_i(s_1, \ldots, s_\mu)]_1$ can be publicly computed from the existing key. Note that this commitment is binding. This can be seen via a simple reduction to the soundness of the polynomial delegation protocol in [71]. The idea is that from an adversary that opens the commitment to two different polynomials $\tilde{u}_1, \tilde{u}_2$ one can sample a random t such that with overwhelming probability $y_1 = \tilde{u}_1(t) \neq \tilde{u}_2(t) = y_2$, honestly compute a proof for the evaluation of $y_1 = \tilde{u}_1(t)$ and then claim this is an evaluation for $\tilde{u}_2(t)$.

K.4 Geppetto [28]

The Geppetto scheme [28] yields a cc-SNARK for QAP relations $R_Q(\vec{x}, \vec{w})$ where $\vec{x} \in \mathbb{Z}_q^n$ and $\vec{w} = (\vec{u}, \vec{\omega})$ with $\vec{u} \in \mathbb{Z}_q^{n'}, \vec{\omega} \in \mathbb{Z}_q^{m-n-n'}$ for some integers n, n'. We recall that Geppetto is a SNARK for MultiQAP relations. A polynomial MultiQAP is a tuple $\mathcal{MQ} = (\ell, \mathcal{J}, \mathcal{A}, \mathcal{B}, \mathcal{C}, t(Z))$ such that $(\mathcal{A}, \mathcal{B}, \mathcal{C}, t(Z))$ is a QAP, and $\mathcal{J} = \{I_0, \ldots, I_{\ell-1}\}$ is a partition of $[m]$. Let $R_{\mathcal{MQ}}$ denote the relation corresponding to $\mathcal{MQ}$. To model $R_Q(\vec{x}, \vec{u}, \vec{\omega})$ we consider a MultiQAP where $\ell = 3$ and where the partition $\mathcal{J}$ consists of $I_0 = [n]$, $I_1 = \{n+1, \ldots, n+n'\}$ and $I_2 = \{n+n'+1, \ldots, m\}$ such that I_0 and I_1 are in the binding subset S.

To see how Geppetto yields a cc-SNARK for such family of relations, we consider the following straightforward modification:

ccGep.KeyGen(R_Q) $\rightarrow$ (ck, ek, vk): run $(EK, VK) \leftarrow$ Geppetto.KeyGen($R_{\mathcal{MQ}}$); set ek $= EK$, vk $= VK$ and let ck be subset of EK consisting of $[r_y t(s), r_c c_{n+1}(s), \ldots, r_c c_{n+n'}(s)]_1^\top \in \mathbb{G}_1^{n'+1}$.

ccGep.VerCommit(ck, $c, \vec{u}, o$) $\rightarrow b$: output 1 iff $(o, \vec{u}^\top) \cdot \mathsf{ck} = c$.

ccGep.Prove(ek, $\vec{x}, \vec{u}, \vec{\omega}$) $\rightarrow (c, \pi; o)$: compute commitments
$C_0 \leftarrow$ Geppetto.Commit($EK_0, \vec{x}, 0$),[24]
$C_1 \leftarrow$ Geppetto.Commit($EK_1, \vec{u}, o_1$),
$C_2 \leftarrow$ Geppetto.Commit($EK_2, \vec{\omega}, o_2$); compute the proof
$\pi' \leftarrow$ Geppetto.Prove($EK, (\vec{x}, \vec{u}, \vec{\omega}), (0, o_1, o_2)$).
Parse C_1 as $(C_{1,1}, C_{1,\alpha}, C_{1,\beta}) \in \mathbb{G}_1^3$.
Output $c = C_{1,1}$, $\pi = (C_{1,\alpha}, C_{1,\beta}, C_2, \pi')$, and $o = o_1$.

ccΠ.VerProof(vk, $\vec{x}, c, \pi$) $\rightarrow b$: recompute $C_0 \leftarrow$ Geppetto.Commit($EK_0, \vec{x}, 0$); reconstruct $C_1 \leftarrow (c, C_{1,\alpha}, C_{1,\beta})$; for $j = 1, 2$ check Geppetto.Verify(VK_j, C_j); check Geppetto.Verify(VK, C_0, C_1, C_2, π').

We claim that assuming Geppetto is a commit-and-prove SNARK for MultiQAPs (according to the commit-and-prove definition in [28]), then the scheme ccGep described above is a cc-SNARK for QAP relations $R_Q(\vec{x}, \vec{u}, \vec{\omega})$.

The correctness of ccGep immediately follows from the one of Geppetto, and the same holds for knowledge soundness. Indeed, notice that the knowledge soundness satisfied by Geppetto provides extractability of the commitment's openings. The perfect zero-knowledge of ccGep follow from the zero-knowledge of Geppetto and the perfect hiding of its commitments. Finally, we observe that by Def. 10 in [28] the polynomials $\{c_k(x)\}_{k \in I_1}$ are linearly independent; thus for a random s, the vector $[r_c t(s), r_c c_{n+1}(s), \ldots, r_c c_{n+n'}(s)]_1$ defines a Pedersen commitment key whose distribution guarantees the binding property under the d-SDH assumption.

K.5 cc-SNARKs based on Groth's SNARK

In this section we show that the SNARK of [41] is a *weak* cc-SNARK, and then that it can be modified to obtain an efficient cc-SNARK (with binding commitments). Below we start by giving a background on non-interative linear proofs, that are instrumental for presenting the scheme.

Split Non-Interactive Linear Proofs of Degree 2. This notion, dubbed NILP for brevity, was introduced by Groth [41] as a refinement of the linear interactive proofs defined in [17]. A NILP is a triple of algorithms (LinSetup, ProofMatrix, Test) working as follows. LinSetup takes in a relation R (e.g., a QAP) and outputs two vectors $\vec{\sigma}_1 \in \mathbb{F}^{\mu_1}, \vec{\sigma}_2 \in \mathbb{F}^{\mu_2}$. ProofMatrix on input a relation R and a pair (x, w) outputs two matrices $(\Pi_1, \Pi_2) \in \mathbb{F}^{k_1 \times \mu_1} \times \mathbb{F}^{k_2 \times \mu_2}$ so that a proof $(\vec{\pi}_1, \vec{\pi}_2)$ is computed as $(\Pi_1 \cdot \vec{\sigma}_1, \Pi_2 \cdot \vec{\sigma}_2)$. Test on input a relation R and a statement x outputs a collection of matrices $T_1, \ldots T_\eta \in \mathbb{F}^{(\mu_1+k_1) \times (\mu_2+k_2)}$ such that a proof $(\vec{\pi}_1, \vec{\pi}_2)$ is accepted iff $(\vec{\sigma}_1^\top, \vec{\pi}_1^\top) \cdot T_i \cdot (\vec{\sigma}_2^\top, \vec{\pi}_2^\top) = 0$ for all $i = 1$ to η. A NILP is required to satisfy completeness, statistical knowledge soundness and zero-knowledge. Informally, completeness says that honestly computed proofs for true statements are accepted. Knowledge soundness says that there must exist an extractor algorithm that on input R, x and a prover strategy (Π_1, Π_2) outputs a witness w such that the probability that $(\Pi_1 \cdot \vec{\sigma}_1, \Pi_2 \cdot \vec{\sigma}_2)$ is accepted while $R(x, w) = 0$ is negligible (over the random choices of LinSetup). Finally, (perfect) zero-knowledge states requires to show a simulator that with knowledge of $(\vec{\sigma}_1, \vec{\sigma}_2, R, x)$) outputs proofs $(\vec{\pi}_1, \vec{\pi}_2)$ that have the same distribution as honestly generated ones.

Groth's zkSNARK [41] is a weak cc-SNARK for QAP relations $R_Q(\vec{u})$. First, we recall the scheme from [41], which is the one obtained by applying the construction of Figure 18 to the Non-Interactive Linear Proof (NILP) in Figure 19.

Recall that we consider the case when $\vec{x}$ is void and the witness $\vec{w} = \vec{u}$ (i.e., the commitment is to the entire witness). To see why this scheme is a weak cc-SNARK for QAP relations $R_Q(\vec{u})$ we make the following observations.

First, let the commitment c to $\vec{u}$ be the value $[A]_1 = r[\delta]_1 + \sum_{k=0}^m u_k \cdot [a_k(\tau)]_1 + [\alpha]_1$; this means that ck is $[\delta, \{a_k(\tau)\}, \alpha]_1$ where α, δ, τ are random. Second, for knowledge soundness we observe that from the existing security proof we can also extract the opening r of $[A]_1$. What is left to argue is the binding of such commitment. Since the $\{a_k(Z)\}_k$ polynomials are not necessarily linearly independent (see, e.g., [57]) the commitment key ck does not guarantee binding. However, we can show that the scheme satisfies *weak binding*. In a nutshell, this means that it is computationally infeasible to open $[A]_1$ to two different witnesses $\vec{u}$ and $\vec{u}'$ such that $R_Q(\vec{u}) \neq R_Q(\vec{u}')$. We show this as follows.

[24] Setting randomness 0 here is essentially a trick to let this commitment correspond to the public input of the relation.

```
$\mathsf{KeyGen}(R_Q)$
    $(\vec{\sigma}_1, \vec{\sigma}_2) \leftarrow\!\!\$\ \mathsf{LinSetup}(R_Q)$
    return $\sigma := ([\vec{\sigma}_1]_1, [\vec{\sigma}_2]_2)$
$\mathsf{Prove}(\sigma, R_Q, x, w)$
    $(\Pi_1, \Pi_2) \leftarrow\!\!\$\ \mathsf{ProofMatrix}(R_Q, x, w)$
    $[\vec{\pi}_1]_1 \leftarrow \Pi_1 \cdot [\vec{\sigma}_1]_1$
    $[\vec{\pi}_2]_2 \leftarrow \Pi_2 \cdot [\vec{\sigma}_2]_2$
    return $\pi = ([\vec{\pi}_1]_1, [\vec{\pi}_2]_2)$
$\mathsf{VerProof}(\sigma, x, \pi)$
    $T_1, \ldots, T_\eta \leftarrow\!\!\$\ \mathsf{Test}(R_Q, x)$
    return 1 iff $\forall i \in [\eta]$ :
        $\begin{pmatrix} [\vec{\sigma}_1]_1 \\ [\vec{\pi}_1]_1 \end{pmatrix} \cdot T_i \cdot \begin{pmatrix} [\vec{\sigma}_2]_2 \\ [\vec{\pi}_2]_2 \end{pmatrix} = [0]_T$
```

Figure 18: Groth's generic SNARK in asymmetric groups from a split NILP.

```
$\mathsf{LinSetup}(R_Q) \to (\vec{\sigma}_1, \vec{\sigma}_2)$
    $\alpha, \beta, \gamma, \delta, \tau \leftarrow\!\!\$\ \mathbb{F}^*$
    $\vec{\sigma}_1 := \Big(1, \alpha, \delta, \{\tau^i\}_{i=0}^{d-1}, \Big\{\frac{1}{\gamma}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\Big\}_{i=0}^{n},$
        $\Big\{\frac{1}{\delta}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\Big\}_{i=n+1}^{m}, \{\frac{1}{\delta}\tau^i t(\tau)\}_{i=0}^{d-2}\Big)$
    $\vec{\sigma}_2 := \Big(1, \beta, \gamma, \delta, \{\tau^i\}_{i=0}^{d-1}\Big)$
$\mathsf{ProofMatrix}(R_Q, \vec{x}, \vec{w}) \to (\Pi_1, \Pi_2)$
    Compute $h(Z)$ and define $\vec{y}$ from $(\vec{x}, \vec{w})$ as in (1)
    $r, s \leftarrow\!\!\$\ \mathbb{F}$
    Let $\Pi_1 \in \mathbb{F}^{3\times(m+2d+3)}$, $\Pi_2 \in \mathbb{F}^{1\times(d+4)}$ s.t.
        $(A, C)^\top = \Pi_1 \cdot \vec{\sigma}_1$, $B = \Pi_2 \cdot \vec{\sigma}_2$ with
    $A := \alpha + \sum_{k=0}^{m} y_k \cdot a_k(\tau) + r\delta; \qquad B := \beta + \sum_{k=0}^{m} y_k \cdot b_k(\tau) + s\delta$
    $C := \sum_{k=n+1}^{m} y_k \cdot \frac{\beta a_k(\tau) + \alpha b_k(\tau) + c_k(\tau)}{\delta} +$
        $+ \sum_{i=0}^{d-2} h_i \frac{\tau^i t(\tau)}{\delta} + As + Br - rs\delta$
$\mathsf{Test}(R_Q, \vec{x}) \to T$
    Define $T \in \mathbb{F}^{(m+2d+5)\times(d+5)}$ encoding the following quadratic test
    $A \cdot B = \alpha \cdot \beta + C \cdot \delta + (\sum_{k=0}^{n} x_k \cdot \frac{1}{\gamma}(\beta a_k(\tau) + \alpha b_k(\tau) + c_k(\tau))) \cdot \gamma$
```

Figure 19: Groth's NILP for a QAP relation $R_Q(\vec{x}, \vec{w})$.

Notice that from the two different valid openings $(\vec{u},r)$ and $(\vec{u}',r')$ of $[A]_1$ we can easily rule out two cases. The first case is the one where $r \neq r'$: this can be immediately reduced to finding the discrete log δ. The second case is the one when $r = r'$ and $\sum_k (u_k - u'_k) a_k(Z)$ is a nonzero polynomial: this can be reduced to finding the discrete log τ (which is known as PDL problem [50]), as τ can be computed by factoring this polynomial. Therefore we are left with the case when $\sum_k (u_k - u'_k) a_k(Z)$ is the zero polynomial, yet $\vec{u} \neq \vec{u}'$. We argue that it cannot be that $R_Q(\vec{u}) \neq R_Q(\vec{u}')$. Indeed, the existing proof [41][Theorem 1] shows that equalities $A = \alpha + r\delta + \sum_{k=0}^{m} C_k \cdot a_k(\tau)$ and $B = \beta + s\delta + \sum_{k=0}^{m} C_k \cdot b_j(\tau)$ hold, where $\{C_k\}_{k=0}^{m}$ are the same coefficients of the term $\sum_{k=0}^{m} C_k \cdot \frac{\alpha b_k(\tau) + \beta a_k(\tau) + c_k(\tau)}{\delta}$ in C. Therefore, if the commitment A opens to $\vec{u}'$ then it must be the case that $C_k = u'_k$, but in this case the QAP would be satisfied (i.e., $R_Q(\vec{u}') = 1$) contradicting that u' is an invalid witness for R_Q.

A new cc-SNARK for QAP relations $R_Q(\vec{u}, \vec{\omega})$. Here we show how we can modify the zkSNARK of [41] in order to obtain a cc-SNARK for proving the satisfiability of QAP relations of the form $R_Q(\vec{u}, \vec{\omega})$, that is a scheme where the commitment is to a portion, $\vec{u}$, of the witness and where the public input is void.

In our construction we consider an *augmented QAP* (in the sense of [13]), which is a QAP as in Definition K.1 with the additional property that the polynomials $a_k(X)$ for $k = 0$ to n are *linearly independent.*

Our new cc-SNARK is the scheme obtained by applying the generic SNARK construction of [41] recalled in Figure 18 to the NILP in Figure 20. To match the cc-SNARK syntax we let the commitment be the proof element $[D]_1$. Clearly, $[D]_1$ can be seen as a Pedersen commitment for the key $\mathsf{ck} = [\frac{\eta}{\gamma}, \{\frac{1}{\gamma}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\}_{i=0}^{n}]_1$. By the linear independence of the $a_i(Z)$ polynomials the binding of this commitment can be reduced to the PDL assumption. Correctness and knowledge soundness follow from the proof of the generic construction in [41], assuming that the construction in Figure 20 is a NILP. We show that this is the case in the following theorem.

Theorem K.2. *The construction in Figure 20 is a NILP with perfect completeness, perfect zero-knowledge and statistical knowledge soundness against affine provers.*

Proof Perfect completeness is easy to verify. For perfect zero-knowledge, we define the simulator that samples $A, B, D \leftarrow\!\!\$\ \mathbb{F}$ at random and then finds C so that the verification test is satisfied. This shows that real and simulated proofs are identically distributed.

For knowledge soundness, let $(\Pi_1, \Pi_2) \in \mathbb{F}^{3\times(m+2d+5)} \times \mathbb{F}^{1\times(d+4)}$ be an affine prover strategy. First, we define the extractor as the one that returns as witness component $\vec{\omega}$ the entries of the second row of Π_1 corresponding to the terms $\big\{\frac{1}{\delta}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\big\}_{i=n+1}^{m}$, and as witness component $\vec{u}$ the entries in the third row of Π_1 corresponding to the terms $\Big\{\frac{1}{\gamma}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\Big\}_{i=1}^{n}$. For the cc-SNARK knowledge soundness, we need to additionally extract the commitment opening that can be taken from the entry in the third row of Π_1 corresponding to the term $\frac{\eta}{\gamma}$.

Once defined the extractor, we need to show that the probability that the proof verifies and the relation $R_Q(\vec{u}, \vec{\omega})$ does not hold is negligible. This proof is essentially identical to the one used for the NILP of [41]; we defer a complete proof to an extended version of the paper. ☐

$\mathsf{LinSetup}(R_Q) \to (\vec{\sigma}_1, \vec{\sigma}_2)$

$\alpha, \beta, \gamma, \delta, \eta, \tau \leftarrow\$ \mathbb{F}^*$

$$\vec{\sigma}_1 := \left(1, \alpha, \delta, \{\tau^i\}_{i=0}^{d-1}, \left\{\frac{1}{\gamma}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\right\}_{i=0}^{n}, \frac{\eta}{\gamma}, \left\{\frac{1}{\delta}(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))\right\}_{i=n+1}^{m}, \{\frac{1}{\delta}\tau^i t(\tau)\}_{i=0}^{d-2}, \frac{\eta}{\delta}\right)$$

$$\vec{\sigma}_2 := \left(1, \beta, \gamma, \delta, \{\tau^i\}_{i=0}^{d-1}\right)$$

$\mathsf{ProofMatrix}(R_Q, \vec{w}) \to (\Pi_1, \Pi_2)$

Let $\vec{w} := (\vec{u}, \vec{\omega})$. Compute $h(Z)$ as in (1)

$r, s, v \leftarrow\$ \mathbb{F}$

Let $\Pi_1 \in \mathbb{F}^{3\times(m+2d+5)}$, $\Pi_2 \in \mathbb{F}^{1\times(d+4)}$ s.t.

$(A, C, D)^\top = \Pi_1 \cdot \vec{\sigma}_1$, $B = \Pi_2 \cdot \vec{\sigma}_2$ and

$$A := \alpha + \sum_{k=0}^{m} w_k \cdot a_k(\tau) + r\delta; \qquad B := \beta + \sum_{k=0}^{m} w_k \cdot b_k(\tau) + s\delta$$

$$C := \sum_{k=n+1}^{m} w_k \cdot \frac{\beta a_k(\tau) + \alpha b_k(\tau) + c_k(\tau)}{\delta} + v\frac{\eta}{\delta} + \sum_{i=0}^{d-2} h_i \frac{\tau^i t(\tau)}{\delta} + As + Br - rs\delta$$

$$D := \sum_{k=0}^{n} w_k \cdot \frac{1}{\gamma}(\beta a_k(\tau) + \alpha b_k(\tau) + c_k(\tau)) + v\frac{\eta}{\gamma}$$

$\mathsf{Test}(R_Q) \to T$

Define $T \in \mathbb{F}^{(m+2d+7)\times(d+5)}$ encoding the following quadratic test

$$A \cdot B = \alpha \cdot \beta + C \cdot \delta + D \cdot \gamma$$

Figure 20: NILP for a QAP relation $R_Q(\vec{w})$.

L EXPERIMENTS DETAILS

In this section we provide more detailed data on our experiments.

L.1 LegoGro16 for Commit-and-Prove.

The following table shows our experimental results that compare the schemes LegoGro16 and CPGro16 with respect to the overhead for dealing with data committed using a Pedersen vector commitment. The experiments considered vectors of different length n.

In the case of LegoGro16, such overhead in proving time is essentially that of creating the additional element D of the proof that contains a commitment to to the data (see Appendix K.5) and to create a $\mathsf{CP}_{\mathsf{link}}$ proof to link D to the external commitment. The LegoGro16 proof is longer because of these two additional elements of $\mathbb{G}_1$. And for verification, the $\mathsf{CP}_{\mathsf{link}}$ verification must be executed. With respect to the CRS, in LegoGro16 we have the additional elements of the CRS needed to create D and the $\mathsf{CP}_{\mathsf{link}}$ CRS, that is essentially one vector of n elements of $\mathbb{G}_1$. In CPGro16, all the overhead in proving time and CRS is related to the size and degree of the QAP that models the computation of the Pedersen commitment. This was done by selecting an appropriate gadget in libsnark, which optimizes the task by selecting a suitable elliptic curve.

n	CPGro16			LegoGro16		
	$\mathcal{KG}$ (s)	$\mathcal{P}$ (s)	crs (MB)	$\mathcal{KG}$ (ms)	$\mathcal{P}$ (ms)	crs (KB)
8	3.928	1.185	3.653	3.677	3.044	0.51
16	7.307	2.252	7.305	5.949	4.202	1.02
32	13.78	4.461	14.61	10.90	5.201	2.04
64	26.04	8.685	29.22	19.37	8.979	4.08
128	50.69	16.50	58.44	32.49	15.58	8.16
256	102.8	33.02	116.9	57.76	19.50	16.32
512	292.0	65.42	233.7	117.8	30.84	32.64
1024	876.3	133.3	467.5	241.2	55.35	65.28
2048	1011	428.7	935.0	466.6	84.09	130.6

	CPGro16	LegoGro16
$\lvert\pi\rvert$ (B)	127.38	191.13
$\mathcal{V}$ (ms)	3.4	4.129

Table 2: Performance of CPGro16 in comparison to LegoGro16. We remark that the numbers for the two schemes are in different units and that those for CPGro16 are three orders of magnitude larger.

L.2 LegoAC1 for Arithmetic Circuits.

Here we show our experimental results that compare our LegoAC1 commit-and-prove zkSNARK against the Groth16 scheme, in the SHA256 and matrix factoring applications explained in Section 7.

For SHA256, Groth16 needs $1.9s$ for key generation of a CRS of $5.1MB$, $0.7s$ for proving and $0.9ms$ for verification; LegoAC1 needs $7.9s$ for key generation of a CRS of $6.2MB$, $0.9s$ for proving and $1.8ms$ for verification.

For matrix factoring, we used $n \times n$ matrices of 32-bit integers with $n \in \{16, 32, 64, 128\}$. Detailed timings appear in the table below.

Overall, these experiments show that LegoAC1 is about $5-6\times$ slower in key generation and $1.5-1.8\times$ slower in proving; verification is nearly the same and improves with larger inputs. Noteworthy that most of LegoAC1 key generation time (about 70%) is taken by the corresponding algorithm for $\mathsf{CP}_{\mathsf{lin}}^{\mathsf{Ped}}$; this is mainly due to an unoptimized technique for dealing with sparse matrices like the ones that encode the linear constraints $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O$, and we expect this to be improved in the future.

Finally, we remark that LegoAC1 is commit-and-prove, which means its proofs are done with respect to matrices that committed in a Pedersen commitment (in a canonical vectorized form).

n	Groth16			LegoAC1		
	$\mathcal{KG}$ (s)	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	$\mathcal{KG}$ (s)	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)
16	0.210	0.1496	1.662	1.105	0.2779	3.097
32	1.227	0.9568	3.696	7.569	1.6803	4.697
64	8.848	7.1774	9.686	52.86	11.904	10.73
128	69.21	58.60	34.83	419.8	89.704	35.71

	Groth16	LegoAC1
$\lvert\pi\rvert$ (B)	127.38	350.25

Table 3: Performance of LegoAC1 in comparison to Groth16