SNARKs for Virtual Machines are Non-Malleable

Matteo Campanelli[1], Antonio Faonio[2], and Luigi Russo[2]

[1] Offchain Labs binarywhalesinternaryseas@gmail.com
[2] EURECOM, Sophia Antipolis, France {faonio,russol}@eurecom.fr

Abstract. Cryptographic proof systems have a plethora of applications: from building other cryptographic tools (e.g., malicious security for MPC protocols) to concrete settings such as private transactions or rollups. In several settings it is important for proof systems to be *non-malleable*: an adversary should not to be able to modify a proof they have observed into another for a statement for which they do not know the witness.

Proof systems that have been deployed in practice should arguably satisfy this notion: it is crucial in settings such as transaction systems and in order to securely compose proofs with other cryptographic protocols. As a consequence, results on non-malleability should keep up with designs of proofs being deployed.

Recently, Arun et al. proposed Jolt (Eurocrypt 2024), arguably the first efficient proof system whose architecture is based on the *lookup singularity* approach (Barry Whitehat, 2022). This approach consists in representing a general computation as a series of *table lookups*. The final result is a SNARK for a Virtual Machine execution (or SNARK VM). Both SNARK VMs and lookup-singularity SNARKs are architectures with enormous potential and will probably be adopted more and more in the next years (and they already are).

As of today, however, there is no literature regarding the non-malleability of SNARK VMs. The goal of this work is to fill this gap by providing both concrete non-malleability results and a set of technical tools for a more general study of SNARK VMs security (as well as "modular" SNARKs in general). As a concrete result, we study the non-malleability of (an idealized version of) Jolt and its fundamental building block, the lookup argument Lasso. While connecting our new result on the non-malleability of Lasso to that of Jolt, we develop a set of tools that enable the composition of non-malleable SNARKs. We believe this toolbox to be valuable in its own right.

Table of Contents

1 Introduction

A zero-knowledge proof (ZKP) is a privacy-enhancing cryptographic tool that allows to prove that a statement is true while preserving confidentiality of secret information [29]. A special class of ZKPs are the zkSNARKS [37] that are non-interactive, short, and efficiently verifiable, which make them a critically important tool in a wide range of applications.

SNARKs for VMs and the Lookup-Singularity. A popular approach to SNARKs is that of SNARKs for *Virtual Machines* (or SNARK VMs[3]), which at their heart consist of proving the execution of a computer program—expressed in a predetermined instruction set—over some CPU abstraction. This design has a number of attractive features: it makes available all the existing optimizing compilers for pre-existing instruction sets; it offers an excellent developer experience making SNARKs usable by anyone able to write a computer program [2,47]. Many SNARKs that are currently being deployed in practice follow this design pattern. Examples include the Cairo-VM [28], the RISC Zero project [54], Scroll's Ceno [35], Polygon Miden [34] and many others. Among these constructions, a notable example is Jolt [2], a SNARK for VMs that is based on the *lookup-singularity* approach [51], which consists in reducing execution of opcodes in a VM to a series of table lookups. This approach has huge potential for adoption being simple, as well as easy to extend and to audit. It is also leads to extremely fast provers (up to 2x faster than the current state of the art [46]).

Strong Security Properties in zkSNARKs. Most of the proposed constructions for zkSNARKs usually provide security for what we may consider *bare minimum* security properties, e.g., zero-knowledge and knowledge-soundness. However, when deployed in larger protocol it is important for cryptographic proof systems to satisfy stronger properties. This includes *simulation extractability* (or SIM-EXT) introduced by De Santis et al. [17], that requires that the knowledge extractor succeeds even when the malicious prover can request simulated proofs for arbitrary statements. This security notion implies *non-malleability*, where an accepted proof cannot be successfully tinkered with (*mauled*) into a different one without knowing the witness. This requirement is crucial for protocol composition in general [13] and to prevent basic types of attacks on transaction blockchains (e.g. double spending)[4].

A recent line of works [16,20,21,25,26,32] has shown simulation extractability of several zkSNARKs like Bulletproofs [9], Spartan [41], Sonic [36], PLONK [24], Marlin [15], Lunar [10] and Basilisk [38]. However, none of these results cover the case of zkVMs (we expand on the technical gap between these works and zkVMs in Section 1.4). Since zkVMs are behind the design of deployed systems with non-malleability requirements, this remains an urgent open problem.

1.1 This Work: Concrete Results & General Tools

Our general goal is to make progress on the problem above. The approach we take in this work is:

(i) to analyze the simulation extractability of a *concrete, representative zkVM design* to use as a case study.

(ii) to provide, at the same time, *a set of methodological tools* for the study of the simulation extractability of zkVMs *in general*—that is, beyond our specific choice of zkVM construction in item *(i)*. In fact, as we elaborate on below, we will provide a set of technical results useful for *an even broader* family of SNARK constructions, namely *Lego-ish* SNARKs (which we define below).

***(i)* SIM-EXT of Jolt** We will choose as a case study a design *loosely based on* Jolt, a lookup-singularity SNARK VM for the RISC-V instruction set, at the heart of which is Lasso, an argument for lookups with

[3] A note on terminology: in this paper we will not use the phrase "zk" unless we are talking about zero-knowledge. In particular: we use the phrase *SNARK for VM* or simply *SNARK VM* to mean "a succinct, scalable argument of knowledge for a VM architecture (which might or might not be zero-knowledge)"; we will apply the phrase *zkVM* only to denote the more specific notion of a "SNARK VM that also satisfies zero-knowledge" (i.e., that has hiding properties). Notice that we are diverging from a common usage which calls "zkVM" a SNARK VM without zero-knowledge features (or denotes by "zero-knowledge" a succinct SNARK).

[4] The work of [18] observes that over three hundred thousand Bitcoins have been involved in malleability attacks

attractive efficiency features. This makes Jolt/Lasso a likely adoption in different settings in the near future [45,48]. However, besides their efficiency, Jolt/Lasso constitute a natural choice since they together provide the first example of lookup-singularity SNARK VM having been concretely described and implemented. Finally, and crucially, Jolt [2] and Lasso [43], might be the most formal treatment of SNARKs for VMs in the literature at the present moment. This is important for us since otherwise we would not be able to carry out the type of formal analysis required by simulation extractability. To be more precise, the concrete design we will consider will not be exactly identical to that sketched in [2]. First off, the original description of Jolt and Lasso is not zero-knowledge. Since the framework of simulation extractability presupposes zero-knowledge, we have to naturally start from a zero-knowledge version of Lasso/Jolt. Second, for sake of generality and simplicity, we will abstract out some parts of the Jolt design. At the high-level, Jolt runs a VM dividing it into three parts[5] each proven by a different "sub-SNARKs": instruction execution (via Lasso), instruction-fetching and memory-checking (both proven via Spartan-like proof systems [41]). In our concrete result (Corollary 2) we assume that instruction execution applies (our variant of) Lasso, while we abstract out the remaining sub-SNARK specifying what properties they need to satisfy in order for the final zkVM to be simulation-extractable.

***(ii)* zkVMs through the lens of modularity** Our discussion above hints to how it may be possible to approach the simulation extractability of zkVMs in general: since SNARKs for VMs lend themselves to *modular* designs, this is potentially something we can leverage[6]. Thus, on our way towards our goal in item *(ii)* above, we tackle a develop a *more broadly interesting problem*: the non-malleability of *modular* (or *Lego-ish*) SNARKs [11], i.e. SNARKs that are obtained from the composition of several "sub-SNARKs", each possibly of a different design. In particular, we address this question:

What can we say about the non-malleability of a modular SNARK knowing that (some of) its building blocks are non-malleable?

Modular SNARKs have been identified as worth of a systematic investigation of their own because of their simplicity and efficiency [1,6,10,11]; general treatment of open problems in SNARKs designs—efficient distributed proving—have recently benefited from an explicit modular approach [39]. While we do have a general theoretical framework to reason about knowledge-soundness and zero-knowledge of Lego-ish SNARKs [11], to the best of our knowledge, no work prior to ours systematically studied the simulation extractability of modular SNARKs.

Challenges of Lego-ish SIM-EXT. We remark that composing non-malleable objects while maintaining their non-malleability does not come for free. For instance, as demonstrated in [20], there are *copy-paste* attacks when composing different Interactive Oracle Proofs (IOPs) (see Ben-Sasson, Chiesa and Spooner [5]) into one simulation-extractable zkSNARK. These attacks consider compositions of schemes for arbitrary relations without any shared knowledge. Briefly, our framework shows how to circumvent these attacks by "gluing" together the witnesses, either by considering a shared witness or by considering witnesses that are somehow logically linked (we will elaborate more in the next section and make these intuitions precise in our compilers in Section 6). To prove a statement composed of different relations, we will have to identify specific constraints for both the relations themselves as well as the sub-SNARKs used to prove each individual relation.

1.2 Our Results

A. SIM-EXT of Joltish Our main result consists in proving that a lookup-based singularity zkVM based on Jolt—that we call Joltish—is simulation-extractable.

[5] We stress that in the Jolt paper, this distinction is sketched and the reader can think of this paragraph as our own (intentionally fuzzy) paraphrase. A formal treatment of different components of a VM is highly dependent on the VM at hand. We will attempt a general formal treatment in Section 7.2.

[6] This modularity is not a mere technical artifact of the work in Jolt [2]. It has been used explicitly in other works [35] and it is a natural design approach: different sub-components of VMs will have distinct features where sub-SNARKs of different designs will shine. Arguably, a modular design is already *explicitly* at the core of "lookup-singularity" SNARKs since their defining principle is to use a specialized SNARK (a lookup argument) for a specific component (instruction execution).

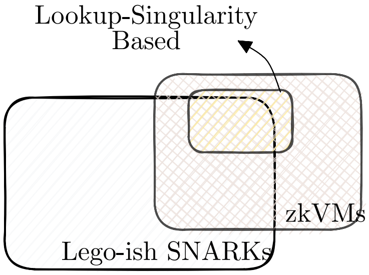

Fig. 1. Intuition for why SIM-EXT results for Lego-ish SNARKs are useful for zkVMs in general: zkVMs in general lend themselves easily to a modular design; this is especially true for lookup-singularity based ones. See also Section 7.

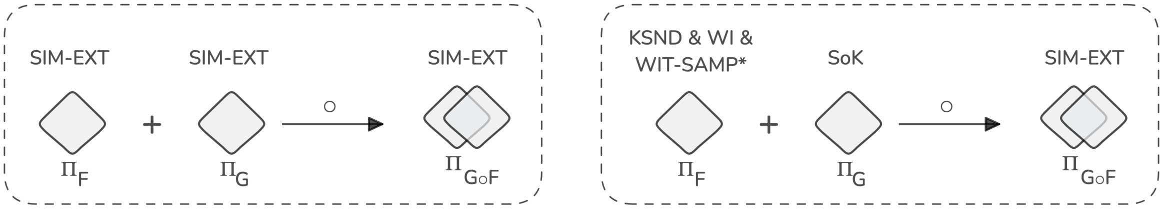

Fig. 2. Informal description of two of our composition results for SIM-EXT of Lego-ish SNARKs (Theorem 5). We mark WIT-SAMP with $*$ to hint that the requirements of the theorems are slightly more nuanced. See Theorem 5 for the full formal statement.

Theorem (informal) *Under the hardness of DLOG there exists a simulation-extractable lookup-singularity zkVM.*

Our Joltish is based on our simulation-extractable lookup argument zkLasso, which makes Joltish a lookup-singularity zkVM. In the technical overview in Section 1.3 we give more details on how we obtain Joltish.

B. A toolbox for SIM-EXT from commit-and-prove zkSNARKs A commit-and-prove argument of knowledge is an argument of knowledge where the witness is committed using a (non-interactive) commitment scheme. The work LegoSNARK of Campanelli, Fiore, and Querol [11] shows that commit-and-prove SNARKs are very useful for composing different SNARKs together in meaningful ways.

We show two compositions derived from commit-and-prove schemes that are simulation extractable. In particular, we provide two natural ways to compose schemes.

The first composition we consider is the conjunction of two relations. At first glance, given an argument for a relation $\mathcal{R}_F$ and an argument for a relation $\mathcal{R}_G$, we can realize an argument for "*their conjunction*" by running the two arguments independently. This composition is knowledge-sound; however, it is not simulation-extractable, as we can mount a copy-paste attack where the attacker knows a witness for the instance in $\mathcal{R}_F$ and uses a simulated proof for $\mathcal{R}_G$ (see [20] for more details). We avoid this attack by considering a conjunction of relations where the committed witness is shared between the two instances. Given this, we show that if the two arguments (for $\mathcal{R}_F$ and $\mathcal{R}_G$) are simulation-extractable, then their composition is also simulation-extractable.

The second composition is what we call function composition. Consider a SNARK for correct function execution, namely, a SNARK that proves $F(\mathbb{x}, \mathbb{w}) = \mathbb{y}$ for a function F, with public input $\mathbb{x}$, private input $\mathbb{w}$, and output $\mathbb{y}$. Consider two commit-and-prove schemes: one that proves $F(\mathbb{x}_f, \mathbb{w}_f) = \mathbb{y}_f$, and a second that proves $G(\mathbb{x}_g, \mathbb{w}_g) = \mathbb{y}_g$ and let us call them Π_F and Π_G respectively. Now we can compose them together to prove $G \circ F(\mathbb{x}_f, \mathbb{x}_g, \mathbb{w}_f) := G(\mathbb{x}_g, F(\mathbb{x}_f, \mathbb{w}_f))$. The idea is to generate the first and second instances so that they share the commitment to $\mathbb{y}_f$, thus linking the private output of F with the private input of G. Also in this case, we can show that if the two commit-and-prove schemes are simulation-extractable, then their composition is also simulation-extractable.

These two results are rather straightforward and rely only on the fact that when the two schemes share knowledge, one cannot mount the trivial copy-paste attack described above.

We can actually improve the conditions of the results by assuming an extra property from one of the relations, which we refer formally as *efficient witness computability* (WIT-SAMP). Loosely speaking, this property states that we can find easily valid witnesses for the (non-committed part of the) instances. For example, in the functional composition, if the prover has the freedom to sample the commitment to $\mathbb{y}_f = \mathbb{w}_g$, then the zero-knowledge simulator for the composed scheme could sample a dummy input $(\mathbb{x}_f, \mathbf{0})$ for F, run the honest prover for Π_F, and simulate the proof for Π_G. Since the simulator for the composed scheme does not use the simulator for Π_F, we can (1) reduce the simulation extractability of the composed scheme to the knowledge soundness of Π_F, and (2) reduce the zero-knowledge of the composed scheme to the witness indistinguishability of Π_F. There is a caveat in this composition: Π_F could be re-randomizable, allowing the adversary to create a forgery for an instance where it has already seen a simulated proof (i.e., we can only prove *weak* simulation extractability for the composed scheme). However, we can address this issue, and prove *full* simulation extractability for the composed scheme, by assuming that Π_G is a signature-of-knowledge (SoK) and by *signing* the proof for Π_F using Π_G. In Fig. 2 we give a graphical representations of our results on generic composition of commit-and-prove SNARKs and summarize in the following informal version of Theorem 5.

Theorem (informal) *There exists a black-box transformation from two SIM-EXT commit-and-prove SNARK Π_F, Π_G to a SIM-EXT conjunction (resp. composition) proof system $\Pi_{F \wedge G}$ (resp. $\Pi_{G \circ F}$). Moreover, there exists a black-box transformation to a SIM-EXT conjunction proof system $\Pi_{F \wedge G}$ (resp. for functions composition $\Pi_{G \circ F}$) from two commit-and-prove SNARKs Π_F, Π_G where (1) Π_F is KSND and (statistically) WI and $\mathcal{R}_F$ satisfies WIT-SAMP and (2) Π_G is a signature of knowledge.*

Recipes for parallelizable SIM-EXT SNARKs. A problem when using signature of knowledge is that we can call Π_G only after having computed the proof for Π_F, which forces sequentiality in the proof generation. To mitigate such a bottleneck, in Section 7.3 we describe a notion of signature of knowledge where, roughly speaking, the message can be fed at the very end of the prover's computations. We refer to this as a *signature of knowledge with delayed message.* We give two instantiations of SoK with delayed message. We show (1) that the classical Fiat-Shamir approach for signature of knowledge can be adapted to the delayed message setting extending the results on Fiat-Shamir-based simulation-extractable argument [16] and (2) we give a black-box construction of signature-of-knowledge with delayed message from (classical) signature-of-knowledge and one-time signatures.

C. Other contributions At the technical basis of our results on the non-malleable zkVMs lies a series of contributions that we are going to present in more detail in the next section. First, we give a zero-knowledge version of Lasso and provide the analysis of its simulation extractability. Second, we revisit the technical results of [16], weakening their requirements and achieving tighter bounds for Spartan and Bulletproofs. Finally, we give a proof of the simulation extractability of `HyraxPC`, which may be of independent interest.

1.3 Technical Overview

SIM-EXT of zkLasso. The technical core of our contribution is providing a simulation-extractable indexed lookup argument derived from Lasso. We take the work of [16] as our starting point. They prove the simulation extractability of (zero-knowledge variants of) schemes such as Bulletproofs and Spartan. Their work follows the results of simulation extractability for Fiat-Shamir based arguments inspired by the work of Faust et al. [22] and further investigated in [25,26,32].

Their approach works in three steps which together provide simulation extractability: *(i)* have ZK version of the protocols;[7] *(ii)* prove that all the inner (sub)protocols are *computational*[8] special-sound, i.e., it is possible to extract a witness from a sufficient number of valid proofs and whose transcript possibly satisfies some additional *predicate*; *(iii)* proving that for a specific k (where k is a round index) the protocol satisfies two properties referred as k-ZK and k-unique response (k-UR for short). k-ZK restricts the ZK simulator by allowing it to reprogram the random oracle only at the k-th round. k-UR states that the malicious prover's responses are uniquely determined after the k-th round.

To achieve step *(i)*, Dao and Grubbs need to replace all the occurrences of the inner protocols, such as the sum-check-based reductions, with their *blinded* versions. For example, if we consider the classical sum-check protocol which eventually evaluates on a random point $\boldsymbol{x}$ defined by the verifier's challenges a committed polynomial f, the blinded counterpart would instead commit to $f(\boldsymbol{x})$, for example using Pedersen, and then show in zero-knowledge that such a commitment opens to the evaluation of f on $\boldsymbol{x}$. Thus the scheme *blinds* the value y which might leak information about the witness.

While several of the building blocks of Lasso are common to those of Spartan, and we naturally use some of the same "low-level" technical tools, our analysis diverges substantially from that of [16] and requires to develop some more machinery. More in detail, we follow [16] and substitute the sub-protocols with their blinded versions. To do so, we need to define a blinded version of the grand product argument due to [44] and prove it computational special-sound. Once done that, we need a stronger analysis of the computational special soundness of the hash-based multi-set fingerprinting used in Spartan. Specifically, Lasso and Spartan use Spark as their underlying (sparse) polynomial commitment scheme; however, while in Spartan some of the sparse polynomials are committed honestly by the verifier, in Lasso these polynomials are committed by the untrusted prover. Crucially, in our case these sparse polynomials encode the matrix of the lookup indexes, i.e., the witness we wish to extract from the proof of the adversary, and this discrepancy introduces non-trivial differences between our work and [16] when analyzing the computational special soundness.

A second component of both Lasso and Spartan is (yet another) polynomial commitment called `HyraxPC` [49]. We improve the analysis for `HyraxPC`. Specifically, digging into the technical details of [16], to prove computational special soundness of Hyrax, we need first to define a tree of transcripts where the edges of each node satisfy a set of constraints that Dao and Grubbs formalize through a set of *predicates*. We show that we need to introduce one more predicate to fix the proof of computational special soundness of `HyraxPC`. Moreover, we additionally prove that `HyraxPC` achieves k-ZK and k-UR, and thus, as additional result, we can prove that this polynomial commitment is simulation-extractable.

By revisiting the techniques of [16], we also introduce some improvements that directly apply to Spartan and Bulletproofs, as well as to Lasso. First, we design a (slightly) tighter blinded sum-check protocol that only relies on the simple *distinctness* predicate, and for which it is sufficient to use the tree-builder of Attema et al. [3]. Second, and more importantly, we achieve a tighter bound in our extractor (cf. Theorem 2) avoiding a loss quadratic in the number of the prover's queries and by solving a problem left open in the previous work. We observe that our approach is still rewinding-based and so the provable SIM-EXT security we get is "low" in terms of security bits, however this seems inherent to this type of analysis.

From zkLasso to Joltish. We provide a model for arguments of knowledge for virtual machine execution. While similar formalizations exist in the literature [4,8,19,55], our framework focuses on abstracting zkVMs based on the lookup singularity. We isolate the logical components in the VM that lookup argument can handle from the rest and demonstrate that our compiler for conjunction, described in Section 6, is sufficient to achieve simulation-extractable zkVMs. More in detail, we adopt an indirect way to achieve such a formalization: we define a commit-and-prove relation $\mathcal{R}^\star$ as the series of logical and memory constraints and checks to perform to the trace of the *program execution* which, together with the correct *instructions execution* handled by the lookup argument, prove correct program execution. This abstraction results in a conjunction of a scheme for $\mathcal{R}^\star$ and a lookup argument. Thus we can use our general non-malleable composition results (see the informal theorem at page 6 and the associated Theorem 5). In particular, we can leverage on a

[7] The usual notion of simulation extractability makes sense for ZK protocols only.

[8] If the extractor fails to extract a witness, then we argue that the malicious prover is able to break some computationally-hard problem, e.g., finding a nontrivial discrete log relation between the Pedersen generators.

simulation-extractable lookup argument to weaken the necessary security properties of the scheme for $\mathcal{R}^\star$. To do so, we show that $\mathcal{R}^\star$ is WIT-SAMP, by showing how to derive a valid trace of a program execution that uses an invalid instruction set. As a consequence, the scheme for $\mathcal{R}^\star$ needs only to be WI and knowledge sound, which open the doors to many instantiations. Next, we demonstrate how to integrate our zkLasso into the framework, resulting in a broad class of zkVMs that, as we argue, includes Joltish, our zero-knowledge variant of Jolt [2]. This task is easier since we can use the knowledge-soundness results for the scheme(s) for $\mathcal{R}^\star$ of [2]. We emphasize that our composition theorem for zkVMs, Theorem 6, is general and allows for the replacement of components in Joltish with different SNARKs, which is why we refer to a large class of zkVMs.

1.4 Why new results for SNARK VMs?

One way to achieve zero-knowledge is to compose Jolt/Lasso with another zkSNARK, i.e., we could use a zkSNARK to prove the knowledge of a valid Jolt/Lasso proof (e.g., the recent work Testudo [12] composes Spartan with Groth16 [30], and some folding-based schemes such as Nova [33] follow this approach). If this zkSNARK is also simulation-extractable, then it seems we get the maximum result with the minimum effort. Despite viable, this approach of "adding" ZK by composition has some theoretical and practical drawbacks. In particular, it would require representing the Jolt verifier in a format like R1CS or Plonkish, which may be cumbersome and partially limit the benefits of the improved auditability depicted above. Furthermore, this *arithmetization* procedure incurs in a direct random oracle instantiation that hence becomes public to the adversary, which may lead to insecure schemes [14].

1.5 Related Work

Simulation extractability was first introduced by De Santis et al. [17], expanding the definition of simulation soundness of Sahai [40]. For zkSNARKs, this notion was studied by Groth and Maller [31] who proposed as an interesting application the succinct signatures of knowledge, or *Snarky signatures*. Recently, we had several results [16,20,21,25,26,32] about the simulation extractability of notable zkSNARKs, such as Bulletproofs [9], Spartan [41], Sonic [36], PLONK [24], Marlin [15], Lunar [10] and Basilisk [38].

We mention some notable works related to SNARKs for virtual machine execution. Beginning with the pioneering work of [4], which required an expensive trusted setup, the field has advanced significantly. Subsequent works, such as [8,55], showed schemes with transparent setups and improved efficiency. More recent developments include Cairo-VM [28] and Ceno [35].

A technical tool we leverage is an efficient *tree-builder* to prove the knowledge soundness of computational special sound arguments compiled using the Fiat-Shamir transform, that was studied in the work of [16] in the wake of the results of [25,26].

1.6 Future Work

We foresee applications for our toolbox results beyond zkVMs. For example it could be used to provide alternative proofs for the SIM-EXT of Spartan and Bulletproofs potentially substantially simplifying the approach in [16] and our own approach for zkLasso with it. Spartan in particular is a good candidate for this given its several moving parts which can be seen as separate block (the Hyrax polynomial commitment, grand product arguments, etc.).

2 Preliminaries

A function f is negligible in λ (we write $f \in \mathsf{negl}(\lambda)$) if it approaches zero faster than the reciprocal of any polynomial. For an integer $n \geq 1$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We consider both strict polynomial time (PPT) and expected polynomial time (EPT) algorithms.

Cryptographic Assumptions Let $\mathsf{GroupGen}$ be some PPT algorithm than on input 1^λ, returns a description $\mathrm{pp}_{\mathbb{G}}$ of a group $\mathbb{G}$. Every element in $\mathbb{G}$ can be written as g^x for some generator $g \in \mathbb{G}$ and exponent $x \in \mathbb{F}$, but given g^x, it is in general hard to compute x (discrete logarithm problem).

Lemma 1 (Discrete Log Reduction, [27]). *For every EPT adversary $\mathcal{A}$, there exists an EPT adversary $\mathcal{B}$, nearly as efficient as $\mathcal{A}$, such that:*

$$\Pr[\textstyle\prod_{i=1}^{n} g_i^{a_i} = 1 \wedge (a_1, \dots, a_n) \neq \mathbf{0} \mid (a_1, \dots, a_n) \leftarrow \mathcal{A}(g_1, \dots, g_n)] \leq \mathbf{Adv}_{\mathbb{G}}^{\mathsf{DL}}(\mathcal{B}) + \frac{1}{|\mathbb{F}|}$$

where $g, g_1, \dots, g_n$ are random generators of $\mathbb{G}$ and we define the advantage of $\mathcal{B}$ as $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{DL}}(\mathcal{B}) := \Pr[g^x = h \mid h \leftarrow\!\!\$\, \mathbb{G}; x \leftarrow \mathcal{B}(g, h)]$.

Multilinear extensions For any function $f \colon \{0,1\}^\ell \to \mathbb{F}$, there exists a unique ℓ-variate multilinear polynomial $\widetilde{f}$ such that $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^\ell$. We refer to $\widetilde{f}$ as the multilinear extension of f. For a vector $a \in \mathbb{F}^n$, where n is a power of 2, we similarly define the multilinear extension $\widetilde{a} \colon \mathbb{F}^{\log n} \to \mathbb{F}$ as follows: we interpret a in the natural way as listing all n evaluations of a function with domain $\{0,1\}^{\log n}$, and define $\widetilde{a}$ to be the multilinear extension of this function.

Commitment schemes A commitment scheme with message space $\mathcal{M}$ is a tuple of algorithms $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCom})$ that works as follows: $\mathsf{Setup}(\mathrm{pp}_{\mathbb{G}}) \to \mathrm{ck}$ takes as input group parameters $\mathrm{pp}_{\mathbb{G}}$ and outputs a commitment key ck. $\mathsf{Commit}(\mathrm{ck}, m) \to (\mathbf{c}, \rho)$ takes as input the commitment key ck and a message $m \in \mathcal{M}$, and outputs a commitment $\mathbf{c}$ and an opening o $\mathsf{VerCom}(\mathrm{ck}, \mathbf{c}, m, \rho) \to b$ takes as input the commitment key ck, a commitment $\mathbf{c}$, a message $m \in \mathcal{M}$ and an opening o, and it accepts ($b = 1$) or rejects ($b = 0$). Besides correctness, a commitment scheme satisfies two more properties.

(Computational) Binding: no PPT adversary can find, unless with negligible probability, a commitment $\mathbf{c}$, two messages $m \neq m'$ and two openings o, o':

$$\mathsf{VerCom}(\mathrm{ck}, \mathbf{c}, m, \rho) = \mathsf{VerCom}(\mathrm{ck}, \mathbf{c}, m', \rho') = 1$$

(Statistical) Hiding: $\forall m, m'$, $\forall \mathrm{ck}$:

$$\{\mathbf{c} : (\mathbf{c}, \rho) \leftarrow \mathsf{Commit}(\mathrm{ck}, m)\} \approx \{\mathbf{c}' : (\mathbf{c}', o') \leftarrow \mathsf{Commit}(\mathrm{ck}, m')\}$$

Interactive Arguments An (NP-)relation $\mathcal{R}$ is a set of tuples $(\mathrm{pp}, \mathbb{x}, \mathbb{w})$ decided by a PT algorithm. Here pp are system-wide parameters, $\mathbb{x}$ is the public input (or instance), and $\mathbb{w}$ is the private input (or witness). We interchangeably represent a relation $\mathcal{R}$ either as an algorithm with boolean output or as a set, thus $\mathcal{R}(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \iff (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. Moreover, when clear from the context, we omit the parameters and simply write $\mathcal{R}(\mathbb{x}, \mathbb{w})$.

A public-coin interactive argument for a relation $\mathcal{R}$ is a tuple of PPT algorithms $\mathit{\Pi} := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where:

$\mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathcal{G}}) \to \mathrm{pp}$**:** outputs parameters pp given global parameters $\mathrm{pp}_{\mathcal{G}}$

$\langle \mathcal{P}(\mathbb{w}), \mathcal{V} \rangle(\mathrm{pp}, \mathbb{x}) \to \{0,1\}$**:** a public-coin interactive protocol whereby the prover $\mathcal{P}$, holding a witness $\mathbb{w}$, interacts with the verifier $\mathcal{V}$ on common input $(\mathrm{pp}, \mathbb{x})$ to convince $\mathcal{V}$ that $(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. At the i-th round, $\mathcal{V}$ samples its message uniformly at random from the challenge space $\mathcal{C}_i$. At the end, $\mathcal{V}$ outputs a bit to accept or reject.

We consider interactive arguments that satisfy the standard properties completeness, knowledge soundness and honest-verifier zero-knowledge.

(Completeness) For any adversary $\mathcal{A}$ we have that:

$$\Pr\left[\begin{array}{l} (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \vee \\ \langle \mathcal{P}(\mathbb{w}), V \rangle(\mathrm{pp}, \mathbb{x}) = 1 \end{array} \middle| \begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}}) \\ (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathrm{pp}) \end{array} \right] = 1$$

(Knowledge-Soundness) There exists an EPT extractor $\mathcal{E}$ such that for any stateful PPT adversary $\mathcal{P}^*$:

$$\Pr\left[b = 1 \land (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \;\middle|\; \begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}}) \\ (\mathbb{x}, \mathrm{st}_{\mathcal{P}^*}) \leftarrow \mathcal{P}^*(\mathrm{pp}) \\ b \leftarrow \langle \mathcal{P}^*(\mathrm{st}_{\mathcal{P}^*}), \mathcal{V} \rangle(\mathrm{pp}, \mathbb{x}) \\ \mathbb{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathrm{pp}, \mathbb{x}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{E}$ gets black-box access to each of the next-message functions of $\mathcal{P}^*$ in the interactive protocol and can rewind $\mathcal{P}^*$ to any point in the interaction.

(Honest-Verifier Zero-Knowledge) There exists a PPT simulator $\mathcal{S}$ such that for all $\mathrm{pp} \leftarrow \mathsf{Setup}(\mathrm{pp}_{\mathbb{G}})$ and $(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the following distributions are statistically indistinguishable:

$$\{\mathrm{View}_{\mathcal{V}} \langle \mathcal{P}(\mathbb{w}), \mathcal{V} \rangle(\mathrm{pp}, \mathbb{x})\} \approx_s \{\mathcal{S}(\mathrm{pp}, \mathbb{x})\}$$

where $\mathrm{View}_{\mathcal{V}} \langle \mathcal{P}(\mathbb{w}), \mathcal{V} \rangle(\mathrm{pp}, \mathbb{x})$ denotes the view of the verifier, consisting of the transcript and its own randomness.

Commit-and-Prove Arguments. Roughly speaking, a commit-and-prove argument of knowledge is an argument of knowledge whose witness is committed using a commitment scheme. We adapt a simpler definition of commit-and-prove SNARK than the one in [11]. We assume that there is only a single commitment (rather than an arbitrary number) and that this opens to the entirety of the witness (instead of allowing for uncommitted portions as in [11]).

Definition 1. *Given a relation $\mathcal{R}$ and a commitment scheme CS, the commit-and-prove argument for the relation $\mathcal{R}$ with commitment scheme CS is an argument for the relation $\hat{\mathcal{R}}$ such that $\hat{\mathcal{R}}((\mathrm{pp}, \mathrm{ck}), (\mathbf{c}, x), (w, \rho)) = 1$ if and only if $\mathbf{c}$ is commitment to w using CS with commitment key ck and opening material ρ, namely $\mathsf{VerCom}(\mathrm{ck}, \mathbf{c}, w, \rho)$, and $\mathcal{R}(x, w)$.*

Indexed Lookup Argument A lookup argument allows an untrusted prover to commit to a vector $a \in \mathbb{F}^m$ and prove that all entries of a reside in some predetermined table $T \in \mathbb{F}^n$. In an indexed lookup argument, in addition to a commitment to a, the verifier is handed a commitment to a second vector $b \in \mathbb{F}^m$. The prover claims that $a_i = T[b_i]$ for all $i \in [m]$. We refer to a as the vector of looked-up values, and b as the vector of indices. We can define the commit-and-prove relation:

$$\mathcal{R}_{\mathsf{lookup}}(\mathrm{pp} = (T, n, m), \mathbb{w} = (a, b)) \iff \forall i \in [m] : T[b_i] = a_i.$$

Non-Interactive Arguments in the ROM A non-interactive argument (in the ROM) for a relation $\mathcal{R}$ is a tuple of PPT algorithms $\mathit{\Pi} \coloneqq (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where: $\mathsf{Setup}(\mathrm{pp}_{\mathbb{G}}) \rightarrow \mathrm{pp}$ generates the public parameters $\mathcal{P}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \rightarrow \pi$ generates a proof π $\mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi) \rightarrow b$ checks if a proof is valid or not and outputs a bit $b \in \{0, 1\}$ and H is a random oracle.[9]

We consider non-interactive arguments that satisfy the following properties.

(Completeness) For any adversary $\mathcal{A}$ we have that:

$$\Pr\left[\begin{array}{l} (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \lor \\ \mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}}) \\ (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathrm{pp}) \\ \pi \rightarrow \mathcal{P}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \end{array} \right] = 1$$

[9] For public-coin $(2r + 1)$-message interactive arguments with challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_r$, we actually need r independent random oracles $\mathsf{H}_i \colon \{0,1\}^* \rightarrow \mathcal{C}_i$ with $i \in [r]$. For simplicity, we denote these by a single random oracle H, and it will be clear from context which random oracle is being used in a given round.

```
Game $\mathsf{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \pi) \leftarrow \mathcal{P}^{*\mathsf{H}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi)$
return $b$
```

```
Game $\mathsf{KS}^{\mathcal{E},\mathcal{P}^*}_{1,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \pi) \leftarrow \mathcal{P}^{*\mathsf{H}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi)$
$\mathbb{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathrm{pp}, \mathbb{x}, \pi)$
return $b \wedge (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$
```

Fig. 3. Knowledge soundness security games. The extractor $\mathcal{E}$ is given black-box access to $\mathcal{P}^*$, it simulates H and and can rewind $\mathcal{P}^*$ to any point.

(Knowledge-Soundness) There exists an EPT extractor $\mathcal{E}$ such that for any stateful PPT adversary $\mathcal{P}^*$, the following probability is negligible in λ:

$$\mathbf{Adv}^{\mathsf{KS}}_{\Pi_{\mathsf{FS}},\mathcal{R}}(\mathcal{E}, \mathcal{P}^*) := \left|\Pr\left[\mathsf{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)\right] - \Pr\left[\mathsf{KS}^{\mathcal{E},\mathcal{P}^*}_{1,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)\right]\right|$$

and the knowledge soundness games are defined in Fig. 3.

(Zero-Knowledge) There exists a PPT simulator $\mathcal{S}$ such that for $\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(\mathrm{pp}_{\mathbb{G}})$ and any unbounded adversary $\mathcal{A}$:[10]

$$\Pr\left[\mathcal{A}^{\mathsf{H}(\cdot),\mathcal{P}(\mathrm{pp},\cdot,\cdot)}(1^\lambda) = 1\right] \approx_s \Pr\left[\mathcal{A}^{\mathsf{H}(\cdot),\mathcal{S}^{\mathsf{RePro}}(\mathrm{pp},\cdot,\cdot)}(1^\lambda) = 1\right]$$

where RePro is an oracle that on input a pair (a, b) reprograms $\mathsf{H}(a) := b$.

We notice that zero-knowledge is defined in a model where the random oracle is *explicitly-programmable* [50] by the simulator: in particular, $\mathcal{S}$ can reprogram the random oracle H (using RePro).

To turn public-coin interactive arguments into their non-interactive versions, we can employ the Fiat-Shamir (FS) transform in a setting where $\mathcal{P}$ and $\mathcal{V}$ have black-box access to a random oracle H. We use Π_{FS} to denote the non-interactive argument derived by applying the FS transform to the argument Π.

Tree of Transcripts An $(n_1, \ldots, n_r)$-tree of transcripts for a $(2r+1)$-message public-coin protocol is a set of $\prod_{i\in[r]} n_i$ transcripts arranged in the following tree structure (see Fig. 6 for a graphical illustration):

- The nodes in this tree correspond to the prover's messages and the edges correspond to the verifier's challenges.
- Every node at depth i has precisely n_i children.
- Every transcript corresponds to exactly one path from the root to a leaf.

This notion, introduced by [3], was later generalized by [16] to support custom predicates for the verifier challenges. In particular, in the generalization of [16], the edges (i.e., the verifier's challenges) of each node need to be distinct and they also need to jointly satisfy a predicate ϕ_i where i is the depth of their corresponding node. In this work, we consider only the following predicates:

- $\phi_\pm$ that on input n field elements $(c_1, \ldots, c_n)$ returns 1 if and only if for all $i \in [n]$, there is not $j \neq i$ such that $c_i + c_j = 0$. We use the shortcut $n_\pm$ to indicate a node supporting this predicate.
- $\phi_{:k}$ that on input n challenges $(c_1, \ldots, c_n) \in \mathbb{F}^{n\cdot m}$ returns 1 if and only if all the inputs have different prefixes of length k. We use the shortcut $n_{:k}$ to indicate a node supporting this predicate.

We give a formal definition hereafter.

[10] Zero-knowledge is a security property that is only guaranteed for valid statements in the language, hence $\mathcal{A}$ never queries $\mathcal{P}/\mathcal{S}$ with a pair $(\mathbb{x}, \mathbb{w})$ such that $(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}$.

```
Game $\mathsf{SS}^{\mathcal{TE},\mathcal{A}}_{\mathit{\Pi},\mathcal{R},(\boldsymbol{\phi},\boldsymbol{n})}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^{\lambda}, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \mathsf{T}) \leftarrow \mathcal{A}(\mathrm{pp})$
$\mathbb{w} \leftarrow \mathcal{TE}(\mathrm{pp}, \mathbb{x}, \mathsf{T})$
return $(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \wedge \mathsf{IsAccepting}((\boldsymbol{\phi}, \boldsymbol{n}), \mathrm{pp}, \mathbb{x}, \mathsf{T})$
```

Fig. 4. Computational Special Soundness security game.

Definition 2 (Tree of Transcripts). *Let $\mathit{\Pi}$ be a $(2r+1)$-message public-coin interactive argument for a relation $\mathcal{R}$, with challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_r$. Let $\boldsymbol{n} := (n_1, \ldots, n_r) \in \mathbb{N}^r$, and let $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_r)$ with $\phi_i \colon \mathcal{C}_i \to \{0,1\}$, for all $i \in [r]$, we say that T is an $(\boldsymbol{\phi}, \boldsymbol{n})$-tree of accepting transcripts for* pp *if:*

1. *T is a tree of depth $r+1$,*
2. *For all $i \in [r+1]$, each vertex at depth i is labeled with a prover's message a_i, and if $i \leq r$ it has exactly n_i outgoing edges to its children, with each edge labeled with a verifier's challenge $c_{i,1}, \ldots, c_{i,n_i} \in \mathcal{C}_i^{n_i}$, satisfying $\phi_i(c_{i,1}, \ldots, c_{i,n_i}) = 1$. Additionally, the root label is prepended with $\mathbb{x}$ (its the label becomes $(\mathbb{x}, a_1)$),*
3. *The labels on any root-to-leaf path form a valid input-transcript pair $(\mathbb{x}, \mathrm{tr})$.*

We say that T is accepting *with respect to an input-transcript pair $(\mathbb{x}, \mathrm{tr})$ if $(\mathbb{x}, \mathrm{tr})$ corresponds to the left-most path of T. We define an acceptance predicate $\mathsf{IsAccepting}((\boldsymbol{\phi}, \boldsymbol{n}), \mathrm{pp}, \mathbb{x}, (\pi,)\mathsf{T})$ to check whether T is an $(\boldsymbol{\phi}, \boldsymbol{n})$-tree of accepting transcripts for* pp *and $\mathbb{x}$, and optionally π.*

We now define computational special soundness that essentially guarantees that there exists a tree-extractor algorithm $\mathcal{TE}$ that, given as input a tree of accepting transcripts produced by an efficient adversary, outputs a valid witness with high probability.

Definition 3 (Computational Special Soundness). *Let $\mathit{\Pi}$ be a $(2r+1)$-message public-coin interactive argument for a relation $\mathcal{R}$ with challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_r$. For any $\boldsymbol{n} := (n_1, \ldots, n_r) \in \mathbb{N}_r$ and any $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_r)$ with $\phi_i \colon \mathcal{C}_i^{n_i} \to \{0,1\}$, we say $\mathit{\Pi}$ is $(\boldsymbol{\phi}, \boldsymbol{n})$-computational special sound if there exists a PPT tree-extraction algorithm $\mathcal{TE}$ such that for every EPT adversary $\mathcal{A}$, the following probability is negligible in λ:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\mathit{\Pi},\mathcal{R},(\boldsymbol{\phi},\boldsymbol{n})}(\mathcal{TE}, \mathcal{A}) := \Pr\left[\mathsf{SS}^{\mathcal{TE},\mathcal{A}}_{\mathit{\Pi},\mathcal{R},(\boldsymbol{\phi},\boldsymbol{n})}(\lambda)\right]$$

and the special soundness game is defined in Fig. 4

Attema et al. prove the existence of an efficient *tree-builder* algorithm that can generate $\boldsymbol{n}$-trees of accepting transcripts having oracle access to a (malicious) prover $\mathcal{P}^*$. This result was later generalized by [16] to support partition predicates; in Section 3 we show how to adapt their result to achieve tighter bounds for the predicates needed to instantiate Spartan [41] and Bulletproofs [9].

Simulation extractability Simulation extractability requires that extractability holds even when the malicious prover is given access to simulated proofs, possibly for *false* statements.

Definition 4 (Simulation extractability). *Let $\mathit{\Pi} := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a public-coin zero-knowledge interactive argument for relation $\mathcal{R}$ with associated NIZK $\mathit{\Pi}_{\mathsf{FS}} := (\mathsf{Setup}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$. We say $\mathit{\Pi}_{\mathsf{FS}}$ is simulation extractable (with respect to a simulator $\mathcal{S}$) if there exists an EPT extractor $\mathcal{E}$ such that for every PPT adversary $\mathcal{P}^*$, the following probability is negligible in λ:*

$$\mathbf{Adv}^{\mathsf{SIM-EXT}}_{\mathit{\Pi}_{\mathsf{FS}},\mathcal{R}}(\mathcal{S}, \mathcal{E}, \mathcal{P}^*) := \left|\Pr\left[\mathsf{SE}^{\mathcal{S},\mathcal{P}^*}_{0,\mathit{\Pi}_{\mathsf{FS}}}(\lambda)\right] - \Pr\left[\mathsf{SE}^{\mathcal{E},\mathcal{S},\mathcal{P}^*}_{1,\mathit{\Pi}_{\mathsf{FS}}}(\lambda)\right]\right|$$

and the security games are defined in Fig. 5.

```
Game $\mathsf{SE}^{\mathcal{S},\mathcal{P}^*}_{0,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \pi) \leftarrow \mathcal{P}^{*\mathsf{H},\mathcal{S}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathsf{H}'}_{\mathsf{FS}}(\mathrm{pp}, \mathbb{x}, \pi)$
return $b \wedge (\mathbb{x}, \pi) \notin \mathcal{Q}_{\mathcal{S}}$
```

```
Game $\mathsf{SE}^{\mathcal{E},\mathcal{S},\mathcal{P}^*}_{1,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \pi) \leftarrow \mathcal{P}^{*\mathsf{H},\mathcal{S}}(\mathrm{pp})$
$b \leftarrow \mathcal{V}^{\mathsf{H}'}_{\mathsf{FS}}(\mathrm{pp}, \mathbb{x}, \pi)$
$\mathbb{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathrm{pp}, \mathbb{x}, \pi)$
return $b \wedge (\mathbb{x}, \pi) \notin \mathcal{Q}_{\mathcal{S}} \wedge (\mathrm{pp}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$
```

Fig. 5. Simulation extractability security games. $\mathcal{S}$ returns a proof π upon an input $\mathbb{x}$ (and may reprogram the random oracle), while $\mathcal{Q}_{\mathcal{S}}$ records all the pairs $(\mathbb{x}, \pi)$ queried by $\mathcal{P}^*$. H' denotes the modified RO after all the proof simulation queries. $\mathcal{E}$ is given black-box access to $\mathcal{P}^*$; in particular, it simulates H and $\mathcal{S}$ for $\mathcal{P}^*$ and can rewind $\mathcal{P}^*$ to any point in its execution (with same initial randomness).

Hereafter, we introduce two more properties, namely k-zero-knowledge and k-unique response. Roughly speaking, the former notion captures zero-knowledge when the simulator is only allowed to reprogram the random oracle in the k-th round, while the latter states that the malicious prover's responses are uniquely determined after the k-th round. These two properties together with knowledge-soundness imply simulation extractability [25,16].

Definition 5 (k-Zero-Knowledge, [16]). *Let $\Pi := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a $(2r+1)$-message public-coin interactive. We say that Π_{FS} satisfies (perfect) k-zero-knowledge, for some $k \in [r]$, if there exists a zero-knowledge simulator $\mathcal{S}_{\mathsf{FS},k}$ that only needs to program the random oracle in round k, and whose output is identically distributed to that of honestly generated proofs.*

Definition 6 (k-Unique Response, [16]). *Let $\Pi := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a $(2r+1)$-message public-coin interactive argument. We say that Π_{FS} satisfies k-unique response, for some $k \in [r]$, if for every PPT adversary $\mathcal{A}$:*

$$\Pr\left[b \wedge b' \wedge \pi \neq \pi' \wedge \pi|_k = \pi'|_k \;\middle|\; \begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}}) \\ (\mathbb{x}, \pi, \pi', c) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathrm{pp}) \\ b \leftarrow \mathcal{V}^{\mathsf{H}[(\mathrm{pp}, \mathbb{x}, \pi|_k) \rightarrow c]}_{\mathsf{FS}}(\mathrm{pp}, \mathbb{x}, \pi) \\ b' \leftarrow \mathcal{V}^{\mathsf{H}[(\mathrm{pp}, \mathbb{x}, \pi'|_k) \rightarrow c]}_{\mathsf{FS}}(\mathrm{pp}, \mathbb{x}, \pi') \end{array} \right] \in \mathsf{negl}(\lambda)$$

where $\mathsf{H}[x \rightarrow c]$ denotes the RO when the input x is reprogrammed to output c.

Theorem 1 ([16]). *Let Π be a $(2r+1)$-message public-coin interactive argument. If Π_{FS} is knowledge-sound and there is $k \in [r]$ such that Π_{FS} satisfies k-zero-knowledge and k-unique response, then Π_{FS} is simulation extractable.*

Commitment Instantiations We mostly rely on the Pedersen commitment scheme with message space $\mathbb{F}^n$, for some $n \in \mathbb{N}$, that works as follows:

$\mathsf{Setup}(\mathrm{pp}_{\mathbb{G}})$ outputs $n+1$ random generators $g_1, \ldots, g_n, h$ of $\mathbb{G}$.
$\mathsf{Commit}(\mathrm{ck}, \boldsymbol{a}; \omega)$ parses ck as $(g_1, \ldots, g_n, h)$ and outputs the commitment $C := \prod_{i \in [n]} g_i^{a_i} h^{\omega}$ and the opening ω.
$\mathsf{VerCom}(\mathrm{ck}, C, \boldsymbol{a}, o)$ outputs 1 iff $\mathsf{Commit}(\mathrm{ck}, \boldsymbol{a}; o) = C$.

We will often use the shortcut $\boldsymbol{g}^{\boldsymbol{a}}$ to represent the multi-exponentiation $\prod_{i \in [n]} g_i^{a_i}$.

In this work, we make use of polynomial commitments, namely, commitment schemes with message space $\mathbb{F}[X_1, \ldots, X_\mu]$ for some $\mu \in \mathbb{N}$. In particular, we rely on the `HyraxPC` polynomial commitment scheme [49].

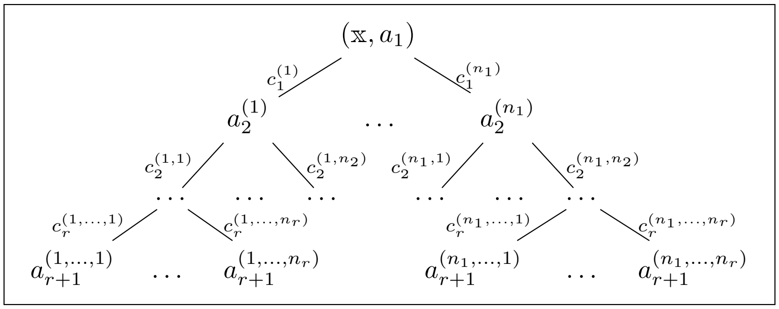

Fig. 6. An $(n_1, \dots, n_r)$-tree of transcripts for a $(2r + 1)$-message public-coin protocol.

3 A Tree Builder for Efficiently-decidable Partitions

A technical tool we leverage is an efficient tree-builder to prove the knowledge soundness of computational special sound arguments compiled using the Fiat-Shamir transform, that was studied in the work of [16] in the wake of the results of [25,26].

We start by introducing the notions of an abstract adversary and an abstract tree of transcripts that can be defined independently of any interactive argument Π.

Definition 7 (Abstract Adversary). *Let $\mathcal{S}_1, \dots, \mathcal{S}_r$ be finite sets and let $\mathsf{H} := (\mathsf{H}_1, \dots, \mathsf{H}_r)$ be a collection of random oracles $\mathsf{H}_i \colon \{0,1\}^* \to \mathcal{S}_i$. An r-round and Q-query random oracle adversary $\mathcal{A}$ against $(\mathcal{S}_1, \dots, \mathcal{S}_r)$ is a deterministic adversary having oracle access to H, making at most Q total accesses to these random oracles, and returning $((a_1, \dots, a_{r+1}), v)$ where $(a_i)_{i \in [r]}$ are strings and $v \in \{0,1\}$. The success probability of $\mathcal{A}$ is defined to be $\Pr\big[v = 1 \mid ((a_1, \dots, a_r), v) \leftarrow \mathcal{A}^{\mathsf{H}}\big]$ and this probability is defined over the randomness of choosing H.*

Definition 8 (Abstract Tree of Transcripts). *Let $\mathcal{S}_1, \dots, \mathcal{S}_r$ be finite sets, $\mathcal{A}$ be any abstract adversary against $\mathcal{S}_1, \dots, \mathcal{S}_r$, and $\boldsymbol{n} := (n_1, \dots, n_r) \in \mathbb{N}^r$. An $\boldsymbol{n}$-abstract tree of transcripts T for $\mathcal{A}$ and $\mathsf{H} := (\mathsf{H}_1, \dots, \mathsf{H}_r)$ is a labeled $\boldsymbol{n}$-tree where:*

- *Each vertex at depth $i \in [r+1]$ is labeled with a message a_i*
- *Each of the n_i edges coming from a vertex at depth $i \in [r]$ is labeled with a different element $s \in \mathcal{S}_i$*
- *For any root-to-leaf path, if the edges are labeled $(s1, \dots, s_r)$ and the vertices are labeled $(a_1, \dots, a_{r+1})$ then $((a_1, \dots, a_{r+1}), 1) \leftarrow \mathcal{A}^{\mathsf{H}'}$ where $\mathsf{H}' := (\mathsf{H}_1[a_1 \to s_1], \dots, \mathsf{H}_r[(a_1, \dots, a_r) \to s_r])$.*

Let Π be a $(2r + 1)$-message public-coin interactive argument with challenge sets $\mathcal{C}_1, \dots, \mathcal{C}_r$. From any deterministic adversary $\mathcal{P}^*$ against the knowledge-soundness of Π_{FS}, we can build an abstract adversary $\mathcal{A}$ against the sets $\mathcal{C}_1, \dots, \mathcal{C}_r$ by running $(x, (a_1, \dots, a_{r+1})) \leftarrow \mathcal{P}^{*\mathsf{H}}(\mathrm{pp})$ (with pp hard-coded) and also $v \leftarrow \mathcal{V}_{\mathsf{FS}}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi)$. $\mathcal{A}$ then outputs $(((\mathbb{x}, a_1), a_2, \dots, a_{r+1}), v)$. An $\boldsymbol{n}$-tree of accepting transcripts for $(\mathrm{pp}, \mathbb{x}, \pi)$ can be seen as an $\boldsymbol{n}$-abstract tree of transcripts for $\mathcal{A}$.

Definition 9 (Partition Predicates). *Let $\mathcal{C} := \bigcup_{i \in [C]} \mathcal{C}^{(i)}$ be a partition $\mathcal{P}$ of a set $\mathcal{C}$ into C blocks. We assume the partition is efficient, i.e. given an index $i \in [C]$, we can enumerate the set $\mathcal{C}^{(i)}$ in polynomial time. For $n \in \mathbb{N}$, we define the corresponding partition predicate $\phi_{\mathcal{P},n} \colon \mathcal{C}^n \to \{0,1\}$ to output 1 on input $(c_1, \dots, c_n)$ if and only if $c_1, \dots, c_n$ belong to distinct blocks of $\mathcal{C}$.*

We consider the following partition predicates:

- $\mathcal{C} := \mathbb{F}$ is partitioned into singletons $\{x\}$. This is the *distinctness* predicate, namely the one that outputs 1 if and only if all the inputs $c_1, \dots, c_r$ are distinct challenges. We implicitly assume that it is the default predicate and we may omit it, i.e., we abreviate a tree T supporting this predicate as an $\boldsymbol{n}$-tree of accepting transcripts

```
Game $\mathsf{TB}^{\mathcal{A},\mathcal{P}^*}_{\Pi_{\mathsf{FS}},(\boldsymbol{\phi},\boldsymbol{n})}(\lambda)$
$\mathrm{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathrm{pp}_{\mathbb{G}})$
$(\mathbb{x}, \pi) \leftarrow \mathcal{P}^{*\mathsf{H}}(\mathrm{pp})$
$\mathsf{T} \leftarrow \mathcal{A}^{\mathcal{P}^*}(\mathrm{pp}, \mathbb{x}, \pi)$
return $\mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi) = 1 \land \mathsf{IsAccepting}((\boldsymbol{\phi}, \boldsymbol{n}), \mathrm{pp}, \mathbb{x}, \mathsf{T})$
```

Fig. 7. Tree-building security game. $\mathcal{A}$ is given black-box access to $\mathcal{P}^*$.

- $\mathcal{C} := \mathbb{F}^m$, for some $m \in \mathbb{N}$, is partitioned into $\{(x, y) \mid y \in \mathbb{F}^{m-k}\}$ for all $x \in \mathbb{F}^k$. This is the *k-prefix distinctness* predicate, namely the one that outputs 1 if and only if all the inputs $c_1, \ldots, c_r$ have different prefixes of length k. We abbreviate this predicate into the number n of challenges as $n_{:k}$.
- $\mathcal{C} := \mathbb{F}$ is partitioned into $\{x, -x\}$ for all x. We abbreviate this predicate into the number n of challenges as $n_{\pm}$.
- $\mathcal{C} := \mathbb{F}^2$ is partitioned into $\{c \cdot x \mid c \in \mathbb{F}^*\}$ for all $x \in \{(0, 0), (0, 1)\} \cup \{(1, a) \mid a \in \mathbb{F}\}$ that captures the linear independence between two vectors. We abbreviate this predicate into the number n of challenges as n_{li} .

Definition 10 (ϵ-Uniform Partition). *Let $\mathcal{C} := \bigcup_{i=1}^{C} \mathcal{C}^{(i)}$. We say that $\{\mathcal{C}^{(i)}\}_{i\in[C]}$ is* ϵ-uniform *if there exists $\mathcal{I} \subseteq [C]$ such that $|\bigcup_{i\notin\mathcal{I}} \mathcal{C}^{(i)}| = \epsilon \cdot C$ and for all $i, j \in \mathcal{I}$: $|\mathcal{C}^{(i)}| = |\mathcal{C}^{(j)}|$.*

All the partitions defined in the above predicates satisfy this property. In particular, the distinctness predicate, the k-prefix distinctness (for all k) and the $n_{\pm}$ predicate use 0-uniform partitions, while n_{li} uses $1/(|\mathbb{F}| + 2)$-uniform partitions.

We now restate the guarantees of the (abstract) tree-builder of [3,16].

Theorem 2 (Efficient Abstract Tree Builder). *Consider any sets $\mathcal{S}_1, \ldots, \mathcal{S}_r$ that have an efficiently decidable partition $\mathcal{S}_i := \bigcup_{j=1}^{C_i} \mathcal{S}_{i,j}$, and any $\boldsymbol{n} := (n_1, \ldots, n_r) \in \mathbb{N}^r$ with $N := \prod_{i=1}^{r} n_i$. There exists a probabilistic algorithm $\mathcal{T}$ such that for any Q-query abstract adversary $\mathcal{A}$ with success probability $\nu_{\mathcal{A}}$ against $(\mathcal{S}_1, \ldots, \mathcal{S}_r)$, $\mathcal{T}$ outputs an $\boldsymbol{n}$-abstract tree of transcript T with probability*

$$\nu_{\mathcal{T}} \geq \nu_{\mathcal{A}} - \frac{(Q + 1)(\sum_{i=1}^{r} n_i - r)}{C}$$

where $C := \min_{i\in[r]} C_i$.

Finally, we restate a theorem asserting the existence of an efficient tree-builder that can generate $(\boldsymbol{\phi}, \boldsymbol{n})$-trees of accepting transcripts, where $\boldsymbol{\phi}$ consists of partition predicates as defined above. Similarly to [16], our proof relies on the tree-builder constructed in the work of [3], but we achieve a tighter bound since we do not incur in a quadratic dependence on the number of queries Q.

Theorem 3 (Efficient Tree Builder). *Let Π be a $(2r + 1)$-message public-coin interactive argument with challenge spaces $\mathcal{C}_1, \ldots, \mathcal{C}_r$. Consider any efficiently decidable and ϵ_i-uniform partition $\mathcal{C}_i := \bigcup_{j=1}^{C_i} \mathcal{C}_{i,j}$, with $\epsilon_i \in \mathsf{negl}(\lambda)$ for all $i \in [r]$, with minimum partition size $C := \min_i C_i$, and let $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_r)$ be the corresponding partition predicate. Consider the tree-building experiment in Fig. 7. There exists a probabilistic algorithm $\mathcal{A}$ such that for any $\boldsymbol{n} := (n_1, \ldots, n_r) \in \mathbb{N}^r$, with $N := \prod_{i=1}^{r} n_i$, and any (malicious) prover $\mathcal{P}^*$:*

$$\Pr\left[\mathsf{TB}^{\mathcal{A},\mathcal{P}^*}_{\Pi_{\mathsf{FS}},(\boldsymbol{\phi},\boldsymbol{n})}(\lambda)\right] \geq \Pr\left[\mathsf{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)\right] - \frac{(Q + 1)(\sum_{i=1}^{r} n_i - r)}{C} - Q \cdot \max_i \epsilon_i$$

where $\mathcal{A}$ makes in expectation at most $(Q + 1)(N - 1) + 1$ rewinding calls to $\mathcal{P}^*$, and Q is an upper bound to the number of RO queries of $\mathcal{P}^*$.

Proof. Without loss of generality, we assume that $\mathcal{P}^*$ is deterministic because if we can prove the theorem for every choice of $\mathcal{P}^*$'s randomness, then by averaging we also prove the theorem for arbitrary $\mathcal{P}^*$. Thus, the only source of randomness in the game $\mathsf{KS}^{\mathcal{P}^*}_{0,\Pi_{\mathsf{FS}},\mathcal{R}}(\lambda)$, and of the success probability of $\mathcal{P}^*$ is the choice of the random oracle H.

For all $i \in [r]$, let $\mathsf{H}_i \colon \{0,1\}^* \to [C_i]$. Moreover, for all $i \in [r]$, let $\mathcal{I}_r$ the subset of $[C_i]$ for which the challenge space $\mathcal{C}_i$ admits an ϵ_i-uniform partition.

We construct an abstract adversary $\mathcal{B}$ against the sets $[\mathcal{I}_1], \dots, [\mathcal{I}_r]$, having access to random oracles $\mathsf{H}^* \coloneqq (\mathsf{H}^*_1, \dots, \mathsf{H}^*_r)$ and to the malicious prover $\mathcal{P}^*$. It does the following:

- Get $\mathrm{pp} \leftarrow \mathsf{Setup}(\mathrm{pp}_{\mathbb{G}})$ and run $\mathcal{P}^*$ on input pp
- When $\mathcal{P}^*$ makes an oracle query to H_i on input a message a, search through the (initially empty) table T for an entry of the form $(i, a, (\cdot, c))$, and return c. If no such query exists, query $\mathsf{H}^*_i(a)$ and obtain the value j
  - If $j \in \mathcal{C}_i$ then sample $c \leftarrow\$\, \mathcal{C}_{i,j}$ uniformly at random, add $(i, a, (j, c))$ to T, and return c as the answer to $\mathcal{P}^*$
  - Otherwise abort
- When $\mathcal{P}^*$ outputs $(\mathbb{x}, \pi \coloneqq (a_1, \dots, a_{r+1}))$, run $v \leftarrow \mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi)$, where H is determined by T, and output $(((\mathbb{x}, a_1), a_2, \dots, a_{r+1}), v)$.

We now define our tree-builder algorithm $\mathcal{A}$. Given oracle access to $\mathcal{P}^*$, it emulates the abstract adversary $\mathcal{B}$, then run the abstract tree-builder $\mathcal{T}$ (cf. Theorem 2) on $\mathcal{B}$. If $\mathcal{T}$ returns an $\boldsymbol{n}$-abstract tree of transcripts T, then $\mathcal{A}$ returns a $(\boldsymbol{\phi}, \boldsymbol{n})$-tree of accepting transcripts T_{Π} for Π_{FS} as follows:

- For each vertex at depth $i \in [r+1]$ of T with label a_i, the same vertex for T_{Π} has label a_i too
- For each edge labeled j going from a vertex labeled a at depth $i \in [r]$, the same edge for T_{Π} has label c, where c is the unique challenge such that $(i, a, (c, j)) \in T$

First, we observe that the abstract adversary $\mathcal{A}$ is nearly as efficient as $\mathcal{P}^*$ since it runs $\mathcal{P}^*$ once and does some other tasks in comparable time (managing the table T, running the algorithm Setup and the verifier procedure $\mathcal{V}_{\mathsf{FS}}$). The tree-builder $\mathcal{A}$ invokes once on $\mathcal{B}$ the tree-builder $\mathcal{T}$ of [3] (cf. Theorem 2), hence inheriting its expected running time: concretely, the expected running time of $\mathcal{A}$ is at most $(Q-1)\cdot(N+1)+1$ times the running time of $\mathcal{P}^*$.

We show that if $\mathcal{B}$ does not abort, T_{Π} is indeed a $(\boldsymbol{\phi}, \boldsymbol{n})$-tree of accepting transcripts. It is clear that T_{Π} is of the right arity. For any vertex v at depth $i \in [r]$, we know that the edges coming from v are labeled with different $(j_{i,1}, \dots, j_{i,n_i})$ in T. This implies that for T_{Π}, the edges coming from the corresponding vertex v has challenges $(c_{i,1}, \dots, c_{i,n_i})$ satisfying $c_{i,k} \in \mathcal{C}_{i,j_{i,k}}$ for all $k \in [n_i]$. Hence T_{Π} satisfies the partition predicate $\boldsymbol{\phi}$.

Moreover, $\mathcal{B}$ perfectly simulates the random oracles H for $\mathcal{P}^*$. For all $i \in [r]$ it first samples a partition index j and then samples from the j-th partition $\mathcal{C}_{i,j}$ a random challenge: this procedure is equivalent to uniformly sampling a challenge from the challenge space $\mathcal{C}_i$ since the partitions are ϵ_i-uniform and, in particular, have the same size. Then we have that the winning probability of $\mathcal{A}$ is the same as $\mathcal{P}^*$, conditioned on the event that $\mathcal{B}$ does not abort. We now bound the probability that $\mathcal{B}$ aborts. Since for all $i \in [r]$ the partition $\{\mathcal{C}_{i,j}\}_{j\in[C_i]}$ is ϵ_i-uniform, the probability that on input a message a the abstract adversary $\mathcal{B}$ aborts is at most $\epsilon_i \in \mathsf{negl}(\lambda)$. By union bound on the number of RO queries we derive that $\mathcal{B}$ aborts with probability at most $Q \cdot \max_i \epsilon_i$.

4 Simulation extractability of Hyrax

4.1 An overview of Hyrax

We give a brief overview and provide some intuition on the multilinear polynomial commitment `HyraxPC` (see Fig. 8) that was first introduced in [49]. It is a polynomial commitment scheme, equipped with a $(\mu+1)$-rounds Eval protocol for a μ-variate multilinear polynomial.

- $\mathsf{Setup}(\mu, \mathrm{pp}_{\mathbb{G}})$: abort if μ is odd. Parse $\mathrm{pp}_{\mathbb{G}}$ as a group description $(\mathbb{G}, \mathbb{F})$. Sample $g_1, \ldots, g_{\mu/2}, h \leftarrow\!\!\$\ \mathbb{G}$ and output $\mathrm{pp} = (\mathbb{F}, \mathbb{G}, g, g_1, \ldots, g_{\mu/2}, h)$.
- $\mathsf{Commit}(\mathrm{pp}, p(X_1, \ldots, X_\mu); \boldsymbol{\omega})$: $\forall i \in [2^\mu]$ let $w_i \coloneqq p(\mathsf{bin}(i))$, define $\boldsymbol{T} \in \mathbb{F}^{2^{\mu/2} \times 2^{\mu/2}}$ s.t. $\forall i, j \in [2^{\mu/2}]$: $T_{i,j} \coloneqq w_{i+2^{\mu/2}\cdot(j-1)}$.
  (Namely, $\boldsymbol{w}$ is the column-major order of $\boldsymbol{T}$.)
  $\forall i \in [2^{\mu/2}]$, sample $\omega_i \leftarrow\!\!\$\ \mathbb{F}$ and compute $C_i \coloneqq \prod_{j=1}^{2^{\mu/2}} g_j^{T_{i,j}} \cdot h^{\omega_i}$.
  Output $\boldsymbol{C} \coloneqq (C_1, \ldots, C_{2^{\mu/2}})$ and opening $\boldsymbol{\omega} \coloneqq (\omega_i)_{i \in [2^{\mu/2}]}$.
- $\mathsf{Eval}\langle \mathcal{P}(p, \boldsymbol{\omega}, v, \omega_v), \mathcal{V}\rangle(\mathrm{pp}, \mathbf{C}, x, C_v)$: given a commitment C_v as public input, with an evaluation point $x \in \mathbb{F}^\mu$
  1. Let $\widetilde{eq}_L(Y) = \prod_{i=1}^{\mu/2} \widetilde{eq}(x_i, Y_i)$ and $\widetilde{eq}_R(Y) = \prod_{i=\mu/2+1}^{\mu} \widetilde{eq}(x_i, Y_i)$.
  2. $\mathcal{P}$ and $\mathcal{V}$ compute $P = C_v \cdot \prod_{k=1}^{2^{\mu/2}} C_k^{\widetilde{eq}_L(\mathsf{bin}(k))}$ and $\boldsymbol{r} = (\widetilde{eq}_R(k))_{k \in \{0,1\}^{\mu/2}}$.
  3. $\mathcal{P}$ also computes $\omega_P \coloneqq \omega_v + \sum_{k \in [2^{\mu/2}]} \omega_k \cdot \widetilde{eq}_L(\mathsf{bin}(k))$ and
  $$\boldsymbol{l} \coloneqq \left(\sum_{k \in [2^{\mu/2}]} T_{k,j} \cdot \widetilde{eq}_L(\mathsf{bin}(k)) \right)_{j \in [2^{\mu/2}]}$$
  4. $\mathcal{P}$ and $\mathcal{V}$ engage in `LogDotProd`, on input $((2^{\mu/2}, g, \mathbf{g}, h), (P, \mathbf{r}), (\mathbf{l}, v, \omega_P))$, to prove that $P = g^v \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot h^{\omega_P}$ and $v = \langle \boldsymbol{l}, \boldsymbol{r} \rangle$.
- $\mathsf{Open}\langle \mathcal{P}(p, \boldsymbol{\omega}), \mathcal{V}\rangle(\mathrm{pp}, \mathbf{C})$:
  1. $\mathcal{V}$ samples challenge $x \leftarrow \mathbb{F}^\mu$, $\mathcal{P}$ replies with $C_v = g^v h^{\omega_v}$, where $v = p(x)$
  2. $\mathcal{P}$ and $\mathcal{V}$ engage in Eval on input $(\mathrm{pp}, \mathbf{C}, x, C_v)$.

Fig. 8. Description of `HyraxPC`. The function $\mathsf{bin}\colon \mathbb{N} \to \{0,1\}^*$ computes the binary representation of an integer. The protocol `LogDotProd` is defined in Fig. 9.

Let $\mathsf{bin}\colon \mathbb{N} \to \{0,1\}^*$ be the function that computes the binary representation of an integer. Moreover, given a matrix $\boldsymbol{T}$ with n rows and m columns and a column vector $\boldsymbol{w}$ with $n \cdot m$ rows, we say that the *column-major order* of $\boldsymbol{T}$ is $\boldsymbol{w}$ if and only if $\forall i, j : T_{i,j} = w_{i+n\cdot(j-1)}$.

To evaluate on a point $x \in \mathbb{F}^\mu$ a multilinear polynomial $p(X_1, \ldots, X_\mu)$, given its evaluations $(w_i)_{i \in [2^\mu]}$ over the hypercube $\{0,1\}^\mu$, we can use the following formula:

$$\begin{aligned} p(x) &= \sum_{k \in \{0,1\}^\mu} p(k) \cdot \prod_{i=1}^{\mu} \widetilde{eq}(x_i, k_i) \\ &= \sum_{k \in [2^\mu]} w_k \cdot \prod_{i=1}^{\mu} \widetilde{eq}(x_i, \mathsf{bin}(k)_i) \\ &= \sum_{k \in [2^{\mu/2}]} \sum_{\ell \in [2^{\mu/2}]} w_{k+2^{\mu/2}\ell} \cdot \underbrace{\prod_{i=1}^{\mu/2} \widetilde{eq}(x_i, \mathsf{bin}(k)_i)}_{\boldsymbol{L}} \cdot \underbrace{\prod_{i=1}^{\mu/2} \widetilde{eq}(x_{\mu/2+i}, \mathsf{bin}(\ell)_i)}_{\boldsymbol{R}} \\ &= \boldsymbol{L} \cdot \boldsymbol{T} \cdot \boldsymbol{R}^\top \end{aligned}$$

where $\boldsymbol{T}$ is the $2^{\mu/2} \times 2^{\mu/2}$ matrix whose column-major order is $(w_i)_{i \in 2^\mu}$, i.e., $\forall i, j \in [2^\mu/2] : T_{i,j} \coloneqq w_{i+2^{\mu/2}\cdot(j-1)}$.

For $n = 2^k$

$$\mathcal{R}_{\texttt{LogDotProd}} = \{((n, g, \mathbf{g}, h), (P, \mathbf{a}), \mathbf{x}, y, r_P) : P = g^y \cdot \mathbf{g}^{\mathbf{x}} \cdot h^{r_P}, y = \langle \mathbf{x}, \mathbf{a} \rangle\}.$$

1. Set $n_0 \leftarrow n, \mathbf{g}^{(0)} \leftarrow \mathbf{g}, P^{(0)} \leftarrow P, \mathbf{a}^{(0)} \leftarrow \mathbf{a}, \mathbf{x}^{(0)} \leftarrow \mathbf{x}, y^{(0)} \leftarrow y, r_P^{(0)} \leftarrow r_P$.
   For $i = 1, \ldots, k$:
   (a) $L_i \leftarrow g^{y_L^{(i)}} \cdot \left(\mathbf{g}^{(i-1)}\right)^{\mathbf{x}_{[:n_i]}^{(i-1)}} \cdot h^{r_L^{(i)}}, R_i \leftarrow g^{y_R^{(i)}} \cdot \left(\mathbf{g}^{(i-1)}\right)^{\mathbf{x}_{[n_i:]}^{(i-1)}} \cdot h^{r_R^{(i)}}$.
   (b) $\mathcal{V}$ sends challenge $c_i \xleftarrow{\$} \mathbb{F}$.
   (c) $\mathcal{P}$ and $\mathcal{V}$ both compute $P^{(i)} \leftarrow L_i^{c_i^2} \cdot P^{(i-1)} \cdot R_i^{c_i^{-2}}$ and

$$\mathbf{a}^{(i)} \leftarrow c_i^{-1} \cdot \mathbf{a}_{[:n_i]}^{(i-1)} + c_i \cdot \mathbf{a}_{[n_i:]}^{(i-1)}, \quad \mathbf{g}^{(i)} \leftarrow \left(\mathbf{g}_{[:n_i]}^{(i-1)}\right)^{c_i^{-1}} \circ \left(\mathbf{g}_{[n_i:]}^{(i-1)}\right)^{c_i}.$$

   (d) $\mathcal{P}$ computes $\mathbf{x}^{(i)} \leftarrow c_i \cdot \mathbf{x}_{[:n_i]}^{(i-1)} + c_i^{-1} \cdot \mathbf{x}_{[n_i:]}^{(i-1)}$ and

$$y^{(i)} \leftarrow c_i^2 \cdot y_L^{(i)} + y^{(i-1)} + c_i^{-2} \cdot y_R^{(i)}, \quad r_P^{(i)} \leftarrow c_i^2 \cdot r_L^{(i)} + r_P^{(i-1)} + c_i^{-2} \cdot r_R^{(i)}.$$

2. Set $\hat{g} \leftarrow \mathbf{g}^{(k)}, \hat{P} \leftarrow P^{(k)}, \hat{a} \leftarrow \mathbf{a}^{(k)}, \hat{x} \leftarrow \mathbf{x}^{(k)}, \hat{y} \leftarrow y^{(k)}, \hat{r_P} \leftarrow r_P^{(k)}$. $\mathcal{P}$ samples $d, r_\beta, r_\delta \xleftarrow{\$}$ and sends $\beta \leftarrow g^d \cdot h^{r_\beta}, \delta \leftarrow \hat{g}^d \cdot h^{r_\delta}$.
3. $\mathcal{V}$ sends challenge $c \xleftarrow{\$} \mathbb{F}$.
4. $\mathcal{P}$ sends $z_1 \leftarrow d + c \cdot \hat{y}$ and $z_2 \leftarrow \hat{a} \cdot (c \cdot \hat{r_P} + r_\beta) + r_\delta$.
5. $\mathcal{V}$ checks that $\left(\hat{P}^c \cdot \beta\right)^{\hat{a}} \cdot \delta \overset{?}{=} \left(\hat{g} \cdot g^{\hat{a}}\right)^{z_1} \cdot h^{z_2}$.

Fig. 9. Description of `LogDotProd`.

The prover $\mathcal{P}$ commits individually to each row of $\boldsymbol{T}$, using `Pedersen`, and outputs a list of commitments $\boldsymbol{C} := (C_1, \ldots, C_{2^{\mu/2}})$. We observe that the verifier $\mathcal{V}$ can compute a commitment to $\boldsymbol{L} \cdot \boldsymbol{T}$, namely $C_{\boldsymbol{L}\cdot\boldsymbol{T}} \leftarrow \prod_{k=1}^{2^{\mu/2}} C_k^{L_k}$ since this just requires public information.

Finally, $\mathcal{P}$ and $\mathcal{V}$ can run an inner product argument to confirm that $(\boldsymbol{L} \cdot \boldsymbol{T}) \cdot \boldsymbol{R}^\top$ equals $p(x)$, supposedly committed in C_v, having access to the commitments $C_{\boldsymbol{L}\cdot\boldsymbol{T}}$ and $C_{\boldsymbol{R}}$. This part is handled by a logarithmic-size dot product proof `LogDotProd` (see Fig. 9) that is similar to the inner product argument of Bulletproofs [9], but also achieves zero-knowledge.

4.2 Proof of the simulation extractability of Hyrax

We recall that `HyraxPC` has been proved $(2^{\mu/2}, (4_\pm)^{\mu/2}, 2)$ computational special sound in [16].[11]

The protocol Eval is a public coin interactive argument for the relation:

$$\mathcal{R}_{\mathsf{Eval}} = \left\{ \mathrm{ck}, (C_p, x, C_v), (p, \omega_p, v, \omega_v) : \begin{array}{l} C_p = \mathsf{Commit}(\mathrm{ck}, p; \omega_p), \\ C_v = \mathsf{Commit}(\mathrm{ck}, v; \omega_v), \\ p(x) = v \wedge p \text{ is multilinear}, \end{array} \right\}$$

In [16], the authors prove the computational special soundness of Eval under the additional condition that the evaluation point $x \in \mathbb{F}^\mu$ is sampled uniformly at random. Although conceptually sound, their statement

[11] Their proof has some technical flaw, but we show how to fix it.

does not fulfill the formalism of Definition 3. We fix this inconsistency of the notation of Dao and Grubbs by, first, defining a different relation:

$$\mathcal{R}_{\mathsf{Open}} = \{\mathrm{ck}, (C), (p, \omega) : C = \mathsf{Commit}(\mathrm{ck}, p; \omega) \wedge p \text{ is multilinear}\}$$

We then define a protocol Open which, basically, runs Eval on a random challenge x (cf. Fig. 8), and we prove computational special soundness for Open: crucially, in the proof we can rewind the prover feeding different challenges x.

By additionally proving that the Open protocol achieves μ-zero-knowledge and μ-unique-response, we derive that $\texttt{HyraxPC}$ is simulation-extractable.

We start by proving that the Eval protocol of $\texttt{HyraxPC}$ achieves μ-ZK and μ-UR, where μ is the number of variables of the polynomials committed.

Lemma 2. $\texttt{HyraxPC}.\mathsf{Eval}$ *is μ-ZK and μ-UR.*

Proof. We start by proving the μ-ZK property.

The simulator $\mathcal{S}_{\texttt{HyraxPC}}$ for $\texttt{HyraxPC}.\mathsf{Eval}$, on input $(\mathrm{pp}, \boldsymbol{C}, x, C_v)$, computes the instance $(P, \boldsymbol{r})$ for $\texttt{LogDotProd}$, as the honest prover would do, and then invokes the simulator $\mathcal{S}_{\texttt{LogDotProd}}$ on input $((2^{\mu/2}, g, \boldsymbol{g}, h), (P, \boldsymbol{r}))$ that does the following:

1. For $i \in [\mu]$ samples the group elements L_i and R_i at random. Retrieves the challenge c_i (as the honest prover would do) and computes the values $P^{(i)}, \boldsymbol{a}^{(i)}$ and $\boldsymbol{g}^{(i)}$ accordingly.
2. Let $\hat{g} := \boldsymbol{g}^{(\mu)}$, $\hat{P} := P^{(\mu)}$ and $\hat{a} := \boldsymbol{a}^{(\mu)}$. Samples random field elements c, z_1, z_2 and a random group element β. Finally, computes $\delta := (\hat{g} \cdot g^{\hat{a}}) \cdot h^{z_2} / (\hat{P}^c \cdot \beta)^{\hat{a}}$ and invokes RePro to make c be the final challenge output by $\mathcal{V}$ on input the transcript, including β and δ.

$\mathcal{S}_{\texttt{HyraxPC}}$ only makes a single RO reprogramming, in particular when invokes the simulator for $\texttt{LogDotProd}$. The output of $\mathcal{S}_{\texttt{HyraxPC}}$ is indistinguishable from that of a real transcript: the random group elements L_i, R_i are indistinguishable from the hiding commitments used in a real proof; similarly, the distribution of the field elements z_1, z_2 and the elements β, δ is also indistinguishable from the one in a real proof.

As for the μ-UR property, it is sufficient to notice that once the transcript of a proof has been fixed up to μ-th round, if we are given two accepting last-round pairs $(z_1, z_2) \neq (z_1', z_2')$ we can always reduce to the discrete log problem as we can find a non-trivial relation between the generators g and h.

We observe that this result implies the following corollary.

Corollary 1. $\texttt{HyraxPC}.\mathsf{Open}$ *is $(\mu + 1)$-ZK and $(\mu + 1)$-UR.*

Proof. Since the Open protocol consists of a random challenge sent by the verifier followed by an execution of the protocol Eval, the proof of $(\mu + 1)$-UR follows directly by the μ-UR of Eval. Finally, it is easy to see that the we can define a simple simulator that frst obtains the random coin of the verifier and then runs the code of the simulator $\mathcal{S}_{\texttt{HyraxPC}}$ defined above that needs to reprogram the random oracle only at the μ-th round and produces transcripts indistinguishable from those of real proofs.

Finally, we show that $\texttt{HyraxPC}.\mathsf{Open}$ is computational special sound. Despite similar to the proof of special soundness of [16], we notice that we rely on the prefix distinctness predicate to extract the witness.

Lemma 3. *For all $\mu \in \mathbb{N}$, the protocol* $\texttt{HyraxPC}.\mathsf{Open}$ *(cf. Fig. 8) is computational special sound, i.e., there exist a tree extractor $\mathcal{TE}_{\texttt{HyraxPC}}$ and an EPT adversary $\mathcal{B}$ such that given an $\boldsymbol{n} := ((2^{\mu/2})_{:\mu/2}, (4_{\pm})^{\mu/2}, 2)$-tree of accepting transcripts (produced by an adversary $\mathcal{A}$) for the $(\mu + 2)$-rounds* Open *protocol, we have that:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{Open},\boldsymbol{n}}(\mathcal{TE}_{\mathsf{Open}}, \mathcal{A}) \leq 2^{\mu/2} \left(\mathbf{Adv}^{\mathsf{DL}}_{\mathbb{G}}(\mathcal{B}) + \frac{1}{|\mathbb{F}|} \right)$$

Proof. The first layer in the tree of transcripts consists of $2^{\mu/2}$ distinct verifier's challenges $((x_{i,\ell})_{\ell\in[\mu]})_{i\in[2^{\mu/2}]}$, each one corresponding to an evaluation point; the rest of the tree then corresponds to an instance of $\texttt{LogDotProd}$. The tree extractor $\mathcal{TE}_{\texttt{HyraxPC}}$ runs the tree extractor $\mathcal{TE}_{\texttt{LogDotProd}}$ (that is similar to the one of the inner-product argument of Bulletproofs, and so we refer to [9,16]) on each $((4^{\mu/2})_{\pm}, 2)$-subtree to recover the underlying linear combinations

$$\boldsymbol{l}^{(i)} := \left(\sum_{k\in[2^{\mu/2}]} w_{k+2^{\mu/2}(j-1)} \cdot \widetilde{eq}((x_{i,1}, \ldots, x_{i,\mu/2}), \mathsf{bin}(k)) \right)_{j\in[2^{\mu/2}]}$$

Here, the tree extractor $\mathcal{TE}_{\texttt{LogDotProd}}$ either succeeds or we can build an adversary $\mathcal{B}$ against the discrete log problem in $\mathbb{G}$.

Finally, for each $j \in [2^{\mu/2}]$, we can use the j-th entry of all the $\boldsymbol{l}^{(i)}$, for $i \in [2^{\mu/2}]$, corresponding to the $2^{\mu/2}$ different verifier's challenges, and we solve for $w_{k+2^{\mu/2}(j-1)}$ for all $k \in [2^{\mu/2}]$. This is possible because the Lagrange polynomials $\{\widetilde{eq}((x_{i,1}, \ldots, x_{i,\mu/2}), k)\}_{i,k}$ are independent since the challenges $((x_{i,\ell})_{\ell\in[\mu]})_{i\in[2^{\mu/2}]}$ crucially satisfy the $\mu/2$-prefix distinctness predicate $\phi_{:\mu/2}$.

5 Simulation extractability of Lasso

In this section, we recall the Lasso indexed lookup argument [43] and we show how we can apply Theorem 1 to prove that a zero-knowledge version of Lasso is simulation-extractable.

5.1 Overview of Lasso

The starting point of Lasso is to model the lookup argument in a sparse way, as it is done in schemes like Caulk [52] or Baloo [53]: given a commitment to a table $t \in \mathbb{F}^n$ and a a commitment to a vector $a \in \mathbb{F}^m$, the prover can prove to know a *sparse* matrix $M \in \mathbb{F}^{m\times n}$ such that (1) each row of M is a unit vector, i.e., there are $n-1$ zeroes and one cell is equal to 1, and (2) $M \cdot t = a$. This turns out to be equivalent, up to negligible soundness error $\log m \cdot |\mathbb{F}|^{-1}$, to check that:

$$\sum_{y\in\{0,1\}^{\log n}} \widetilde{M}(r, y) \cdot \tilde{t}(y) = \tilde{a}(r) \tag{1}$$

when $r \in \mathbb{F}^{\log m}$ is chosen uniformly at random by the verifier after the prover has sent (a commitment to) $\widetilde{M}$. The core idea of Lasso is to use Surge, a generalization of the Spark commitment scheme [41], to commit to $\widetilde{M}$ and then prove that Eq. (1) holds by evaluating $\widetilde{M}$ in a point (r, r_x) chosen by the verifier. To do that, the table t needs to be "decomposable" as we define hereafter.[12]

Definition 11 (Decomposable Table). *A table $t \in \mathbb{F}^n$ is* decomposable *if there is $k \geq 1$ and $\alpha := kc$ tables $t_1, \ldots, t_\alpha$, each of size $n^{1/c}$, as well as an α-variate multilinear polynomial $\hat{f}$ such that for every $(r_1, \ldots, r_c) \in (\{0,1\}^{\frac{1}{c}\log n})^c$:*

$$t[r_1, \ldots, r_c] = \hat{f}(t_1[r_1], \ldots, t_k[r_1], t_{k+1}[r_2], \ldots, t_{2k}[r_2], \ldots, t_{\alpha-k+1}[r-c], \ldots, t_\alpha[r_c])$$

Let $\mathrm{nz}(i)$ denote the (unique) column in the i-th row of M that contains the value 1. First, we observe that can rewrite Eq. (1) as:

$$\sum_{k\in\{0,1\}^{\log m}} \widetilde{eq}(k, r) \cdot t[\mathrm{nz}(i)] = \tilde{a}(r) \tag{2}$$

[12] In previous work, this is also referred to as *Spark-only structure* (SOS).

1. $P \rightarrow V$: The prover sends 3α different $(\log m)$-variate multilinear polynomials $E_1, \ldots, E_\alpha$, $\mathrm{dim}_1, \ldots, \mathrm{dim}_\alpha$, $\mathrm{read\_ts}_1, \ldots, \mathrm{read\_ts}_\alpha$ and α different $(\frac{1}{c}\log N)$-variate multilinear polynomials $\mathrm{final\_ts}_1, \ldots, \mathrm{final\_ts}_\alpha$, where $\forall i \in [\alpha]$: E_i is purported to specify the values of each of the m reads into T_i, dim_i is the multilinear extension of nz_i, while $\mathrm{read\_ts}_i$ and $\mathrm{final\_ts}_i$ are the "counter polynomials" for the i-th sub-table T_i.
2. $V \rightarrow P$: The verifier picks a random $r \in \mathbb{F}^{\log m}$ and sends it to $\mathcal{P}$. The verifier makes one evaluation query to $\widetilde{a}$ to learn $v := \widetilde{a}(r)$.
3. $P \leftrightarrow V$: sum-check protocol to check that $v = \sum_{k \in \{0,1\}^{\log m}} \widetilde{eq}(r, k) \cdot \hat{f}(E_1(k), \ldots, E_\alpha(k))$
   - $\log m$ rounds of interaction in which the prover sends univariate polynomials and the verifier replies with a random coin
   - The verifier checks that $E_i(r_z) = v_i$ for all $i \in [\alpha]$, where $(v_i)_i$ are values provided by the prover at the end of the sum-check protocol. The verifier checks the equation above with one oracle query to each E_i.
4. $V \rightarrow P$: The verifier picks two random field elements γ, τ
5. $P \leftrightarrow V$: α sum-check-based protocols (in parallel) for "grand products" to check that $H_{\tau,\gamma}(WS) = H_{\tau,\gamma}(RS) \cdot H_{\tau,\gamma}(S)$. The verifier checks the equations hold with an oracle query to each of $E_i, \mathrm{dim}_i, \mathrm{read\_ts}_i, \mathrm{final\_ts}_i$.

Fig. 10. A description of Lasso [43]. T is a decomposable lookup table of size N.

and if t is decomposable we can further rewrite it as:

$$\sum_{k \in \{0,1\}^{\log m}} \widetilde{eq}(k, r) \cdot \hat{f}(t_1[\mathrm{nz}_1(i)], t_k[\mathrm{nz}_1(i)], \ldots, t_{a-k+1}[\mathrm{nz}_c(i)], \ldots, t_\alpha[\mathrm{nz}_c(i)]) = \widetilde{a}(r) \tag{3}$$

for some polynomial $\hat{f}$, where $\mathrm{nz}_1(i), \ldots \mathrm{nz}_c(i)$ are the "chunks" in which $\mathrm{nz}(i)$ has been decomposed.

For all $j \in [c]$, let $\mathrm{dim}_j : \mathbb{F}^{\log m} \rightarrow \mathbb{F}$ be equal to $\widetilde{\mathrm{nz}}_j$. Moreover, for all $i \in [\alpha]$, let $E_i : \mathbb{F}^{\log m} \rightarrow \mathbb{F}$ be the $\log m$-variate multilinear polynomial that interpolates all the m lookups into t_i, namely $\forall k \in \{0,1\}^{\log m}$, we have that $E_i(k) := t_i[\mathrm{dim}_i(k)]$. Given this, we can rewrite Eq. (3) simply as:

$$\sum_{k \in \{0,1\}^{\log m}} \widetilde{eq}(k, r) \cdot \hat{f}(E_1(k), \ldots, E_\alpha(k)) = \widetilde{a}(r) \tag{4}$$

In Lasso, the prover commits to M sending commitments to $\mathrm{dim}_1, \ldots, \mathrm{dim}_c$, $E_i, \ldots, E_\alpha$ and the "counter polynomials" for the i-th sub-table T_i, $\mathrm{read\_ts}_i$ and $\mathrm{final\_ts}_i$. Then, the prover and the verifier engage in a sum-check protocol to check that Eq. (4) holds. Finally, the prover needs to convince the verifier that the polynomials E_i are actually encoding the values read from the (honest) memory t_i: to do that, they apply a memory checking procedure [7] that finally results into a sum-check-based grand products argument. More in detail, let WS and RS be two sets accounting for the write and read operations, respectively, and let S be the final state of the memory. Every time a read operation (i.e., a lookup) is issued, a write operation is performed too with the goal of updating the "counter" (i.e., the timestamp) associated with that memory location. The goal of the prover is to convince the verifier that the invariant "every value that has been read must have been written" is maintained at the end of the lookup process, i.e., $WS = RS \cup S$. Lasso is not zero-knowledge since a proof essentially leaks evaluations of $\widetilde{M}$ in some random coins sent by the verifier.

5.2 Zero-Knowledge Lasso

We define the main protocol in Fig. 11. It uses `Pedersen`, `HyraxPC`, three (2-perfect special sound) Σ-protocols (see also Fig. 12) sharing the same setup:

- `ProdPf` to prove that three commitments C_x, C_y, C_z satisfy $xy = z$,
- `DotProdPf` to prove that a multi-commitment $C_{\boldsymbol{x}}$ and a commitment C_y satisfy $y = \langle \boldsymbol{x}, \boldsymbol{a} \rangle$ for a public vector $\boldsymbol{a}$
- $\mathtt{GenPf}_{\hat{f},n}$ to prove that n commitments $(C_{v_i})_i$ satisfy $\hat{f}((v_i)_{i \in [n-1]}) = v_n$

and the following sub-protocols:

Setup Phase. Let $\mathsf{pp} \coloneqq (\mathsf{pp}_{\mathbb{G}}, g, g_1, \dots, g_{\mu/2}, h)$, where $\mu \coloneqq \max(\deg(\hat{f}), \log m, \frac{1}{c}\log N)$, $\deg(\hat{f})$ is the total degree of g and $(g_1, \dots, g_{\mu/2})$ are random generators of $\mathbb{G}$. Let $\mathsf{pp}_{\texttt{Pedersen},1} \coloneqq (\mathsf{pp}_{\mathbb{G}}, g, h)$ be the parameters for `Pedersen` with message space $\mathbb{F}$; for $\nu > 1$, let $\mathsf{pp}_{\texttt{Pedersen},\nu} \coloneqq (\mathsf{pp}_{\mathbb{G}}, g_1, \dots, g_\nu, h)$ be the parameters for `Pedersen` with message space $\mathbb{F}^\nu$. For $\nu \in \mathbb{N}$ let $\mathsf{pp}_{\texttt{HyraxPC},2\nu} \coloneqq (\mathsf{pp}_{\mathbb{G}}, g_1, \dots, g_\nu)$ be the parameters for `HyraxPC` with message space $\mathbb{F}[X_1, \dots, X_{2\nu}]$.

Interaction Phase.

1. $\mathcal{P}$ sends to $\mathcal{V}$ `HyraxPC` commitments to 3α different $(\log m)$-variate multilinear polynomials $E_1, \dots, E_\alpha$, $\mathrm{dim}_1, \dots, \mathrm{dim}_\alpha$, $\mathrm{read\_ts}_1, \dots, \mathrm{read\_ts}_\alpha$ and α different $(\frac{1}{c}\log N)$-variate multilinear polynomials $\mathrm{final\_ts}_1, \dots, \mathrm{final\_ts}_\alpha$, where $\forall i \in [\alpha]$: E_i is purported to specify the values of each of the m reads into T_i, dim_i is the multilinear extension of nz_i, while $\mathrm{read\_ts}_i$ and $\mathrm{final\_ts}_i$ are the "counter polynomials" for the i-th sub-table T_i.
2. $\mathcal{V}$ picks a random $r \in \mathbb{F}^{\log m}$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ sends a `Pedersen` commitment C_v to the value v supposedly equal to $\widetilde{a}(r)$.
4. $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check to check that $v = \sum_{k \in \{0,1\}^{\log m}} u(k)$, where $u(X) \coloneqq \widetilde{eq}(r, X) \cdot \hat{f}(E_1(X), \dots, E_\alpha(X))$: after $\log m$ rounds of interaction, the prover sends a `Pedersen` commitment C_{e_x} to the value e_x supposedly equal to $u(r_z)$
5. $\mathcal{P}$ sends the `Pedersen` commitments $C_{v_1}, \dots, C_{v_\alpha}$ to values $v_1, \dots, v_\alpha$, supposedly equal to $E_1(r_z), \dots, E_\alpha(r_z)$
6. $\mathcal{P}$ and $\mathcal{V}$ engage in $\texttt{GenPf}_{g,\alpha}$ to check that $\hat{f}(v_1, \dots, v_\alpha) = e_x \widetilde{eq}(r, r_z)^{-1}$
7. The verifier checks using `HyraxPC.Eval` that $E_i(r_z) = v_i$ for all $i \in [\alpha]$.
8. $\mathcal{V}$ picks two random field elements γ, τ.
9. For $i = 1$ to α:
   - $\mathcal{P}$ sends to $\mathcal{V}$ the `Pedersen` commitment C_{h_i} to the value h_i supposedly equal to $H_{\gamma,\tau}(WS_i)$ and $H_{\gamma,\tau}(RS_i) \cdot H_{\gamma,\tau}(S_i)$.
   - $\mathcal{P}$ and $\mathcal{V}$ engage in `GrandProd` to check that $H_{\tau,\gamma}(WS_i) = h_i$ and $H_{\tau,\gamma}(RS_i) \cdot H_{\tau,\gamma}(S) = h_i$.
10. $\mathcal{P}$ and $\mathcal{V}$ engage in `HyraxPC.Eval` to check that $v = \widetilde{a}(r)$

Fig. 11. The indexed lookup argument zkLasso.

- A protocol `SumCheck` (see Fig. 13) to reduce the task of proving that $\sum_{x \in \{0,1\}^\mu} p(x) = v$, given the commitments (C_p, C_v), to the claim that $p(r_x) = e_x$ for a random $r_x \in \mathbb{F}^\mu$ sampled randomly by the verifier, and some claimed value $e_x \in \mathbb{F}$, where C_{e_x} is provided by the prover at the end of the procedure.
- A sum-check-based protocol `GrandProd` for "grand products" (see Fig. 14).

1. $\mathcal{P}$ sends to $\mathcal{V}$: $\alpha \leftarrow g^{b_1} \cdot h^{b_2}, \beta \leftarrow g^{b_3} \cdot h^{b_4}, \gamma \leftarrow X^{b_3} \cdot h^{b_5}$, where $(b_1, \dots, b_5) \leftarrow\$ \mathbb{F}^5$
2. $\mathcal{V}$ responds with challenge $c \leftarrow\$ \mathbb{F} \setminus \{0\}$
3. $\mathcal{P}$ sends to $\mathcal{V}$: $z_1 \leftarrow b_1 + cx$, $z_2 \leftarrow b_2 + cr_x$, $z_3 \leftarrow b_3 + cy$, $z_4 \leftarrow b_4 + cr_y$, $z_5 \leftarrow b_5 + c(r_z - r_x y)$

$\mathcal{V}$ checks that $\alpha \cdot C_x^c = g^{z_1} \cdot h^{z_2}$, $\beta \cdot C_y^c = g^{z_3} \cdot h^{z_4}$ and $\delta \cdot C_z^c = C_x^{z_3} \cdot h^{z_5}$

Fig. 12. The Σ-protocol `ProdPf` to check that the prover knows (x, y, r_x, r_y, r_z) such that $C_x = g^x h^{r_x}$, $C_y = g^y h^{r_y}$, $C_z = g^{xy} h^{r_z}$, given the commitments (C_x, C_y, C_z) and generators g, h.

On the instantiation of `GenPf` and `GrandProd`. If $\hat{f}$ is a simple string concatenation, we can exploit the homomorphism of `Pedersen` and reduce `GenPf` to a single invocation of a Σ-protocol for the equality of two commitments (cf. `EqPf` in [16,49]). As for `GrandProd`, we use a commit-and-prove version of the Thaler's grand product argument [44] that is an optimized application of the GKR protocol for circuit evaluation to a circuit computing a binary tree of multiplication gates. Another possibility would be to use the protocol due to Setty and Lee [42] that reduces the communication cost, and hence the proof size, at the cost of committing to additional field elements.

Lemma 4 **([43]).** *Lasso has soundness error* $O(\frac{m + \log m}{|\mathbb{F}|})$.

Let $e_0 = v$. For $i = 1$ to μ:

1. $\mathcal{P}$ computes the polynomial $p_i(X) := \sum_{x \in \{0,1\}^{\mu-i}} p(r_1, \ldots, r_{i-1}, X, x)$, parses it as a vector of coefficients, then sends $C_{p_i} \leftarrow \mathsf{Commit}(\mathrm{pp}, p_i; \omega_{p_i})$ to $\mathcal{V}$.
2. $\mathcal{V}$ responds with challenge $r_i \leftarrow\!\!\$\ \mathbb{F}$.
3. $\mathcal{P}$ computes $e_i \leftarrow p_i(r_i)$, then sends $C_{e_i} \leftarrow \mathsf{Commit}(\mathrm{pp}, e_i; \omega_{e_i})$ to $\mathcal{V}$.
4. $\mathcal{V}$ responds with challenges $w_i \leftarrow\!\!\$\ \mathbb{F}$.
5. $\mathcal{P}$ and $\mathcal{V}$ compute $\boldsymbol{a} \leftarrow (1, \ldots, 1, 2) + w_i \boldsymbol{r}_i^k$ and $C_{y_i} \leftarrow C_{e_{i-1}} C_{e_i}^{w_i}$. In addition, $\mathcal{P}$ computes $y_i \leftarrow e_{i-1} + w_i e_i$ and $\omega_{y_i} \leftarrow \omega_{e_{i-1}} + w_i \omega_{e_i}$.
6. $\mathcal{P}$ and $\mathcal{V}$ engage in `DotProdPf` on input $(\mathrm{pp}, (C_{p_i}, C_{y_i}, \boldsymbol{a}), (p_i, \omega_{p_i}, y_i, \omega_{y_i}))$.

It is left to check that $p(r_1, \ldots, r_\mu) = e_\mu$.

Fig. 13. The protocol `SumCheck` to reduce the task of proving that $\sum_{x \in \{0,1\}^\mu} p(x) = v$, given the commitments (C_p, C_v), to the claim that $p(r_x) = e_\mu$ for a random $r_x \in \mathbb{F}^\mu$ sampled randomly by the verifier, and some claimed value $e_\mu \in \mathbb{F}$, where C_{e_μ} is provided by the prover at the end of the procedure.

Let $z_0 = r_1 = 0$. $\mathcal{P}$ also sets $e_1 \leftarrow v$. For $i = 1$ to $d-1$:

1. If $i > 1$ $\mathcal{P}$ and $\mathcal{V}$ engage in a (i rounds) sum-check to reduce the task of proving that $\sum_{p \in \{0,1\}^i} g_{z_{i-1}}^{(i)}(p) = \widetilde{V}_i(z_{i-1})$ to the claim that $g_{z_{i-1}}^{(i)}(r_i) = e_i$, for some r_i and $C_{e_i} \leftarrow \mathsf{Commit}(\mathrm{pp}, e_i; \omega_{e_i})$ provided by the prover by the end of the protocol.
2. $\mathcal{P}$ sends $C_{w_{1,i}} \leftarrow \mathsf{Commit}(\mathrm{pp}, \widetilde{V}_{i+1}(r_i, 0); \omega_{w_{1,i}})$ and $C_{w_{2,i}} \leftarrow \mathsf{Commit}(\mathrm{pp}, \widetilde{V}_{i+1}(r_i, 1); \omega_{w_{2,i}})$
3. $\mathcal{P}$ and $\mathcal{V}$ engage in `ProdPf` on input $(\mathrm{pp}, (C_{w_{1,i}}, C_{w_{2,i}}, C_{e_i}^{1/\widetilde{eq}(z_{i-1}, r_i)}), (w_{1,i}, \omega_{w_{1,i}}, w_{2,i}, \omega_{w_{2,i}}, e_i/\widetilde{eq}(z_{i-1}, r_i), \omega_{e_i}))$
4. $\mathcal{V}$ sends a challenge $\beta_i \leftarrow\!\!\$\ \mathbb{F}$
5. $\mathcal{P}$ and $\mathcal{V}$ set $z_i \leftarrow l_i(\beta_i)$, where $l_i(X)$ is the unique line such that $l_i(0) = (r_i, 0)$ and $l_i(1) = (r_i, 1)$
6. $\mathcal{P}$ and $\mathcal{V}$ set $C_{v_i} \leftarrow C_{w_{1,i}}^{(1-\beta_i)} \cdot C_{w_{2,i}}^{\beta_i}$. Additionally, $\mathcal{P}$ sets $v_i \leftarrow w_{1,i}(1 - \beta_i) + w_{i,2}\beta_i$

Finally, $\mathcal{P}$ and $\mathcal{V}$ engage in `HyraxPC.Eval` to prove that $\tilde{V}_d(z_{d-1}) = v_{d-1}$.

Fig. 14. The protocol `GrandProd` to prove that the product of 2^d inputs equals v, given a commitment C_v and a commitment to the MLE $\widetilde{V}_d$ of the input vector to a binary-tree circuit of depth d. The output gate is labelled 0, and the two inputs to a layer-i gate labelled $p \in \{0,1\}^i$ are labelled as $(p, 0)$ and $(p, 1)$ respectively; hence `GrandProd` allows to prove that $V_1(0) = v$. For all $i \in [d-1]$ and for all $p \in \{0,1\}^i$, we have that $g_z^{(i)}(p) := \widetilde{eq}(z, p) \cdot \widetilde{V}_{i+1}(p, 0) \cdot \widetilde{V}_{i+1}(p, 1)$

Lemma 5. *For all $\Pi \in \{$`ProdPf`$,$ `DotProdPf`$\}$, Π is 2-perfect special sound, i.e., there exists a tree-extraction algorithm that can extract a valid witness for Π given any 2-tree of accepting transcripts.*

We analyze the computational special soundness of the sumcheck (sub)protocol in Fig. 13. Altough very similar, our scheme is different from the one used in [16] since we change the way the prover computes the vector $\boldsymbol{a}$ of the batched evaluations. Besides a negligible improvement in the efficiency, this change allows us to provide a (tighter) extractor that relies only on the *distinctness* predicate.

Lemma 6. *For all $\mu \in \mathbb{N}$, the sum-check protocol* `SumCheck` *in Fig. 13 is computational special sound, i.e., there exist a tree extractor $\mathcal{TE}_{\mathtt{SumCheck}}$ and an EPT adversary $\mathcal{B}$ such that given an $\boldsymbol{n} := (1, 2, 2)^\mu$-tree of accepting transcripts (produced by an adversary $\mathcal{A}$) for the μ-rounds sum-check protocol, we have that:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\mathtt{SumCheck}, \boldsymbol{n}}(\mathcal{TE}_{\mathsf{SC}}, \mathcal{A}) \leq \mu \left(\mathbf{Adv}^{\mathsf{DL}}_{\mathbb{G}}(\mathcal{B}) + \frac{1}{|\mathbb{F}|} \right)$$

Proof. We construct a tree extractor $\mathcal{TE}_{\mathtt{SumCheck}}$ that does the following for each iteration $i \in [\mu]$. Given a $(1, 1, 2)$-tree of transcripts:

1. Run $\mathcal{TE}_{\mathtt{DotProdPf}}$ on each $(1, 1, 2)$-subtree (corresponding to an instance $C_{p_i}, C_{y_i}, \boldsymbol{a}$) to extract $(p_i, \omega_{p_i}, y_i, \omega_{y_i})$, where y_i is supposedly equal to $e_{i-1} + w_i e_i$

On input $(C_a, T_1, \dots, T_\alpha)$, the simulator does the following:

1. Sample a "dummy" witness $M \in \{0,1\}^{m \times n}$, such that all the rows of M are unit vectors.
2. Compute the vector of looked-up values $b \leftarrow M \cdot T$
3. Run Lasso prover, until the second to last round, on input $(C_b, T_1, \dots, T_\alpha)$ and witness M, where $C_b \leftarrow\!\$\ \mathsf{Commit}(\mathrm{pp}, \widetilde{b})$
4. To prove that $v = \widetilde{a}(r)$, use the ZK-simulator for $\texttt{HyraxPC.Eval}$ on input $(\mathrm{pp}, (C_a, r, C_v))$

Fig. 15. Our $(r-1)$-ZK Simulator $\mathcal{S}$ for Lasso, where r is the number of rounds.

2. Given two distinct challenges w_i, w_i', with extracted witnesses $(p_i, \omega_{p_i}, y_i, \omega_{y_i})$ and $(p_i', \omega_{p_i}', y_i', \omega_{y_i}')$ from the previous step, abort if $(p_i, \omega_{p_i}) \neq (p_i', \omega_{p_i}')$. Otherwise, solve for $e_{i-1}, e_i, \omega_{e_{i-1}}, \omega_{e_i}$ the system:

$$\begin{cases} y_i = e_{i-1} + w_i e_i \\ y_i' = e_{i-1} + w_i' e_i \end{cases} \qquad\qquad \begin{cases} \omega_{y_i} = \omega_{e_{i-1}} + w_i \omega_{e_i} \\ \omega_{y_i}' = \omega_{e_{i-1}} + w_i' \omega_{e_i} \end{cases}$$

The goal is to prove that $\mathcal{TE}_{\mathsf{SC}}$ either outputs polynomials $p_1(X), \dots, p_\mu(X)$ that satisfy the information-theoretic sumcheck protocol, or we can build an adversary $\mathcal{B}$, as efficient as $\mathcal{TE}_{\texttt{SumCheck}}$ and $\mathcal{A}$ combined, against the discrete log problem in $\mathbb{G}$.

We have that $\langle p_i, a_i \rangle = y_i$ and $\langle p_i, a_i' \rangle = y_i'$ by the guarantees of $\mathcal{TE}_{\texttt{DotProdPf}}$. We derive that, if $\mathcal{TE}_{\texttt{SumCheck}}$ does not abort, it would extract values such that $p_i(0) + p_i(1) = e_{i-1}$ and $p_i(r_i) = e_i$, i.e., it extracts valid polynomials for the information-theoretic sumcheck protocol. In this case, we have that $C_{p_i} = \boldsymbol{g}^{p_i} \cdot h^{\omega_{p_i}} = \boldsymbol{g}^{p_i'} \cdot h^{\omega_{p_i}'}$, and by Lemma 1 we conclude that the probability to abort is bound by the probability to solve the discrete log problem in $\mathbb{G}$. By union bound on the number of rounds, we derive the claimed bound.

Below we analyze the special soundness of $\texttt{GrandProd}$ (cf. Fig. 14).

Lemma 7. *For all $d > 1$, the protocol* $\texttt{GrandProd}$ *is computational special sound, i.e., there exist a tree extractor $\mathcal{TE}_{\texttt{GrandProd}}$ and EPT adversaries $\mathcal{B}, \mathcal{B}'$ such that given an $\boldsymbol{n}_{\texttt{GrandProd},d} \coloneqq ((\boldsymbol{n}_0, 2, 2), \dots, (\boldsymbol{n}_{d-2}, 2, 2))$-tree of accepting transcripts (produced by an adversary $\mathcal{A}$) for the grand product, we have:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\texttt{GrandProd},\boldsymbol{n}}(\mathcal{TE}_{\texttt{GrandProd}}, \mathcal{A}) \leq \mathbf{Adv}^{\mathsf{SS}}_{\texttt{HyraxPC.Open},((2^{d/2})_{:d/2},(4_\pm)^{d/2},2)}(\mathcal{TE}_{\texttt{HyraxPC}}, \mathcal{B}) + \sum_{i=1}^{d-2} 4^i \cdot \mathbf{Adv}^{\mathsf{SS}}_{\texttt{SumCheck},(1,2,2)^i}(\mathcal{TE}_{\texttt{SumCheck}}, \mathcal{B}')$$

where $\boldsymbol{n}_0$ is the empty string and $\boldsymbol{n}_i \coloneqq (4,2,2)^i$ for all $i > 0$.

Proof. We construct a tree extractor $\mathcal{TE}_{\texttt{GrandProd}}$ that does the following.

1. For each iteration $i \in [d-1]$:
   (a) If $i > 1$, run $\mathcal{TE}_{\texttt{SumCheck}}$ on each of the 4^i different $(1,2,2)^i$-subtrees, associated with the different random challenges $r_i^{(j)}$, to extract the polynomials sent during the sum-check
   (b) Run $\mathcal{TE}_{\texttt{ProdPf}}$ on each 2-subtree to extract the values $(w_{1,i}^{(j)}, w_{2,i}^{(j)}, e_i^{(j)})$ and let f_i be the polynomial that interpolates all the pairs $(r_i^{(j)}, e_i^{(j)})$
2. Extract the polynomial $\widetilde{V}_d$ running $\mathcal{TE}_{\texttt{HyraxPC}}$ on the subtree obtained by merging each $((4_\pm)^{d/2}, 2)$-subtree corresponding to a different challenge point

At each iteration, the protocol $\texttt{GrandProd}$ performs a sum-check to reduce the task of proving that a certain polynomial equals some claimed value over an hypercube of a given size, and a "reduction to a line" to batch two claims into one. Notice that the polynomial in the sum-check is only "virtually" represented and is never directly evaluated. We need to prove that at each iteration the prover performs a sum-check on a polynomial that is "consistent" wrt to the MLE of the input $\widetilde{V}_d$ that we extract using $\mathcal{TE}_{\texttt{HyraxPC}}$.

We start by focusing on the last iteration of the protocol. Let $f_z(X) \coloneqq \widetilde{eq}(z, X) \cdot \widetilde{V}_d(X, 0) \cdot \widetilde{V}(X, 1)$. We need to prove that the polynomial f_{d-1} extracted by $\mathcal{TE}_{\texttt{GrandProd}}$ at the $(d-1)$-th iteration is equal to $f_z(X)$ for some z (corresponding to an ancestor of the current subtree).

First, by the guarantees of the information-theoretic sum-check and the special soundness of `ProdPf`, we have that $\sum_{p \in \{0,1\}^{d-2}} f_i(p) = v_{d-2}$, for a value v_{d-2} that has been committed at the previous iteration. Second, we observe that f_{d-1} is a $(d-1)$-variate polynomial of individual degree at most 3: this is because the polynomials sent during the sum-check are univariate polynomials of maximum degree 3, due to the number of `Pedersen` generators in pp and later used to run `ProdPf`. Moreover, by definition f_z is a $(d-1)$-variate polynomial of individual degree 3. Let Agree be the event that $f_{d-i}(r^{(j)}_{d-1}) = f_z(r^{(j)}_{d-1})$ for all j. Since there is a unique $(d-1)$-variate polynomial, of individual degree at most 3, that "densely" interpolates the pairs $(r^{(j)}_{d-1}, f_z(r^{(j)}_{d-1}))$ [56], we conclude that, conditioned on Agree, $f_{d-1} \equiv f_z$. When Agree does not occur, we have that there is at least one challenge $r \coloneqq r^{(j)}_{d-1}$ such that $f_{d-1}(r) \neq f_z(r)$. In particular, this implies that $w^{(j)}_{1,d-1} \neq \widetilde{V}_d(r, 0) \vee w^{(j)}_{2,d-1} \neq \widetilde{V}_d(r, 1)$. Let ℓ be the unique line interpolating $((r, 0), w^{(j)}_{1,d-1}), ((r, 1), w^{(j)}_{2,d-1})$; then, there exists at most one field element β such that $\ell(\beta) = \widetilde{V}_d(r, \beta)$. However, when Agree does not occur, we can find in the corresponding subtrees two distinct challenges $\beta^{(j)}_{d-1}, \beta'^{(j)}_{d-1}$ such that the above equation holds, from which we conclude that $\Pr[\mathsf{Agree}] = 1$.

A similar analysis can be run for all the layers of the circuit. We do not need to run $\mathcal{TE}_{\mathsf{SC}}$ when we reach the first iteration since the protocol does not invoke the `SumCheck` protocol. At the first layer, we only rely on the special soundness of `ProdPf` to extract the value v consistent with the output $\widetilde{V}_1(0)$. □

Lemma 8. *zkLasso satisfies $\boldsymbol{n}$-computational special soundness, where*

$$\boldsymbol{n} = (2^{\log m}, (2,2,2)^{\log m}, 2, (4_{\pm})^{(\log\log m)/2}, 2, 3, \mu+1, (\boldsymbol{n}_{\texttt{GrandProd},\mu})^{\alpha}, (4_{\pm})^{(\log\log m)/2}, 2)$$

Proof. We construct a tree extractor $\mathcal{TE}_{\texttt{Lasso}}$ that, given an $\boldsymbol{n}$-tree of accepting transcripts, does the following:

1. Run $\mathcal{TE}_{\texttt{SumCheck}}$ on the first sum-check subprotocol on each $(1,2,2)^{\log m}$ subtree to extract the polynomials sent during the sum-check for $h(X)$
2. Run $\mathcal{TE}_{\texttt{GenPf}}$ on each corresponding 2-subtree to extract the values $v_1, \ldots, v_\alpha$ such that $\hat{f}(v_1, \ldots, v_\alpha) = e_x / \widetilde{eq}(r, r_z)$
3. Run $\mathcal{TE}_{\texttt{HyraxPC}}$ on the subtree obtained by merging each $((4_{\pm})^{\log m/2}, 2)$-subtree, corresponding to different challenge points, to extract α $\log m$-variate multilinear polynomials E_i such that $E_i(r_z) = v_i$ for all $i \in [\alpha]$
4. Run $\mathcal{TE}_{\texttt{GrandProd}}$ on each $\boldsymbol{n}_{\texttt{GrandProd},\mu}$-subtree to extract the multilinear polynomials $\mathrm{dim}_i, \mathrm{read\_ts}_i, \mathrm{write\_ts}_i$, for all $i \in [\alpha]$, corresponding to the MLE of the last layer of the circuit
5. Output matrix M derived from the encoding of its non-zero entries in dim_i

We show that, conditioned on the event that none of the sub-extractor fails, the matrix M extracted by $\mathcal{TE}_{\texttt{Lasso}}$ is a valid witness. In particular, from the guarantees of $\mathcal{TE}_{\texttt{GrandProd}}$ and $\mathcal{TE}_{\texttt{SumCheck}}$ and the soundness of the corresponding protocols, we have that the prover unconditionally passes the verifier's checks for the sum-check and the memory checking argument (cf. Lemma 9) and, moreover, the rows of M are unit vectors. Also, from the guarantees of $\mathcal{TE}_{\texttt{SumCheck}}, \mathcal{TE}_{\texttt{HyraxPC}}$ and the soundness of the sum-check protocol we have that $M \cdot t = a$ because the check holds for more than $\log m$ random rows. □

We are ready to present our main theorem on zkLasso.

Theorem 4. *zkLasso is simulation-extractable.*

Proof. We prove that zkLasso is $(r-1)$-ZK and $(r-1)$-UR, where r is the number of rounds. By combining Theorem 1 and Lemma 8, we derive a direct proof of the theorem.

We leverage a simple $(r-1)$-ZK simulator $\mathcal{S}$ for zkLasso that, at at high-level, executes all subprotocols using a dummy witness and invoke the simulator for the final `HyraxPC.Eval`. For sake of completeness, we give the code of this simulator in Fig. 15.

First, by inspection, it is clear that the proofs produced are accepting: this is because the verifier accepts if both the Lasso proof (until the last round) is valid (let call this proof π_1), and if the final proof π_2

for `HyraxPC.Eval` is valid too. In fact, π_1 is a composition of honestly generated (sub)proofs, and by the completeness of Lasso, we derive that the verifier accepts all of them. The last proof π_2 is generated by invoking the ZK simulator for `HyraxPC.Eval` (cf. Lemma 2), and in this case the validity follows from the NIZK guarantees. Second, we observe that $\mathcal{S}$ only makes a single RO reprogramming, in particular when invokes the ZK simulator for `HyraxPC.Eval`. Finally, we need to prove that the output of $\mathcal{S}$ is indistinguishable from that of a real transcript. Since the Lasso prover only employs hiding commitments as inputs to the inner subprotocols and, additionally, all the subprotocols used in Lasso are zero-knowledge, we conclude that the honestly generated proofs made by our simulator are identically distributed to the proofs in a real transcript. In the final (sub)protocol `HyraxPC.Eval`, indistinguishability is due to the guarantees of the ZK simulator.

The last subprotocol of Lasso consists of an invocation of the $(\mu+1)$-rounds `HyraxPC.Eval` protocol that satisfies computational μ-UR (cf. Lemma 2). Hence, we conclude that zkLasso satisfies perfect $(r-1)$-UR. □

5.3 On Multi-Set Fingerprinting

Lemma 9. *The extracted multi sets in Step 9 of Fig. 11 are the same except with negligible probability.*

Proof. Let A and B be the two multisets and let $n := |A| = |B|$ (we can exclude that they have different cardinalities from the extraction procedure). Recall that each element in A or B is a tuple of three elements (x, v, t). In order to check their equality we check the equality of their fingerprints: $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$, where $\mathcal{H}_{\tau,\gamma}(A) = \prod_{(x,v,t)\in A}(h_\gamma(x,v,t) - \tau)$, and $h_\gamma(x,v,t) = x\cdot\gamma^2 + v\cdot\gamma + t$ denotes the Reed-Solomon fingerprinting.

Now assume that $A \neq B$ and let us bound the probability that the fingerprint test verifies. Below we denote by $\boldsymbol{\alpha}$ (resp. $\boldsymbol{\beta}$) the elements of the tuple $(h_\gamma(A_j))_{j\in[n]}$ (resp. B_j) where, in the indexing, we assume the elements of A (resp. B) are lexicographically ordered[13].

We observe that:

$$\begin{aligned}
&\Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)\right] \\
=&\Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B) \wedge \boldsymbol{\alpha} \neq \boldsymbol{\beta}\right] + \Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B) \wedge \boldsymbol{\alpha} = \boldsymbol{\beta}\right] \\
\leq&\Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B) \mid \boldsymbol{\alpha} \neq \boldsymbol{\beta}\right] + \Pr\left[\boldsymbol{\alpha} = \boldsymbol{\beta}\right]
\end{aligned}$$

Intuitively the first summand in the last line refers to the event where the final product check passes *even though* the Reed-Solomon fingerprints somehow differ. The second summand in the last line is the probability that all the Reed-Solomon fingerprints are the same (despite A and B being distinct).

We observe:

$$\Pr\left[\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B) \mid \boldsymbol{\alpha} \neq \boldsymbol{\beta}\right] = \Pr\left[\prod_j(\alpha_j - \tau) = \prod_j(\beta_j - \tau) \mid \boldsymbol{\alpha} \neq \boldsymbol{\beta}\right]$$

and in order to bound the last probability we can simply apply Schwartz-Zippel (the left- and right-hand side are two distinct polynomials of degree n evaluated in a random point τ) and conclude that it is lower or equal to $\frac{n}{|\mathbb{F}|}$.

We now bound the other summand. We first observe that, from our assumption $A \neq B$ there must exist some index j^* such that $A_{j^*} \neq B_{j^*}$ (recall we are assuming a lexicographic ordering of the multisets). Therefore:

$$\Pr\left[\boldsymbol{\alpha} = \boldsymbol{\beta}\right] = \Pr\left[\bigwedge_j \alpha_j = \beta_j\right] \leq \Pr\left[\alpha_{j^*} = \beta_{j^*}\right]$$

[13] It could in fact be *any* canonical ordering. Having some ordering is going to simplify some observations in our proof.

By the definition of h_γ we can then apply Schwartz-Zippel again and conclude that the last event can occur with probability at most $\frac{2}{|\mathbb{F}|}$.

We showed that if $A \neq B$ then the probability that the test passes is at most $\frac{n}{|\mathbb{F}|} + \frac{2}{|\mathbb{F}|}$. Since this quantity is negligible this concludes the proof. □

6 Modular Composition of Simulation-Extractable Arguments

We describe two variations of two compilers for modular compositions of non-interactive arguments of knowledge. The first compilers handles conjunction of relations with shared witness; the other two handle functional compositions.

6.1 General Results on Conjunction and Functional Composition

In both cases, the compilers start from commit-and-prove arguments that are simulation-extractable. However, for two of the compilers, we require the slightly more general notion of signature-of-knowledge.

Definition 12. *We say that a non-interactive argument Π is a signature-of-knowledge for a relation $\mathcal{R}$, if Π is a complete, simulation extractable and zero-knowledge non-interactive argument for the (augmented) relation $\mathcal{R}'$ such that:*

$$\forall \mathsf{msg} \in \{0,1\}^\lambda : \mathcal{R}(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \iff \mathcal{R}'(\mathrm{pp}, (\mathsf{msg}, \mathbb{x}), \mathbb{w}),$$

where msg *is referred to as the* signed message.

(Generalized) Conjunction of arguments. We consider two compilers for conjunction of relations with common witnesses with different trade-offs. Additionally, we generalize the notion of conjunction with common witness by assuming a (possible) processing through a function M to such a common witness. Specifically, given relation $\mathcal{R}_A$ and $\mathcal{R}_B$ we define $\mathcal{R}^M_{A\wedge B}$ the relation such that $\mathcal{R}^M_{A\wedge B}(\mathrm{pp}, \mathbb{x}_A, \mathbb{x}_B, \mathbb{w}) \iff \mathcal{R}_A(\mathrm{pp}, \mathbb{x}_A, \mathbb{w}) \wedge \mathcal{R}_B(\mathrm{pp}, \mathbb{x}_B, M(\mathbb{w}))$.

Definition 13. *Let M be a polynomial time function, we say that a commitment scheme* CS *is M-malleable if there exist efficiently computable functions M_c, M_ρ such that, for any commitment $\mathbf{c}$ to $\mathbb{w}$ with opening ρ we have that $M_c(\mathbf{c})$ is a valid commit to $M(\mathbb{w})$ with opening $M_\rho(\rho)$. Namely $\forall \mathrm{pp}, \mathbf{c}, \mathbb{w}, \rho : \mathsf{VerCom}(\mathrm{pp}, \mathbf{c}, \mathbb{w}, \rho) \Rightarrow \mathsf{VerCom}(\mathrm{pp}, M_c(\mathbf{c}), M(\mathbb{w}), M_\rho(\rho))$.*

We define a compiler from simulation-extractable arguments (resp. signature-of-knowledge) $\Pi_\wedge$ (resp. $\bar{\Pi}_\wedge$) for $\mathcal{R}^M_{A\wedge B}$ in Fig. 16.

Functional composition of arguments. For any polynomial-time function f let the relation $\mathcal{R}_f$ be such that $\mathcal{R}_f(\mathrm{pp}, (\mathbb{x}_i, \mathbb{x}_o), (\mathbb{w}_i, \mathbb{w}_o)) \iff f(\mathbb{x}_i, \mathbb{w}_i) = (\mathbb{x}_o, \mathbb{w}_o)$. We define $g \circ f$ to be the functional composition of g and f, namely, the function that on input $((x_{f,i}, x_{g,i}), w_{f,i})$ computes $(x_{f,o}, w_{g,i}) \leftarrow f(x_{f,i}, w_{f,i})$, computes $(x_{g,o}, \mathbb{w}_o) \leftarrow g(x_{g,i}, w_{g,i})$ and outputs $((x_{f,o}, x_{g,o}), \mathbb{w}_o)$. See Fig. 17 for a graphical representation of functional composition.

We define a compiler to functional composition from simulation-extractable arguments $\Pi_{g\circ f}$ and from a signature of knowledge $\bar{\Pi}_{g\circ f}$ for $\mathcal{R}_{g\circ f}$ in Fig. 16.

Additional Definitions and Theorem on Compilers Security. We are almost ready to state the theorem. We first need two additional definitions.

Definition 14. *We say that a relation $\mathcal{R}$ is efficiently witness computable if there exists a EPT algorithm $\mathcal{M}$ such that for any* pp *and* $\mathbb{x}$ *we have either $\mathcal{R}(\mathrm{pp}, \mathbb{x}, \mathcal{M}(\mathrm{pp}, \mathbb{x})) = 1$ or $(\mathrm{pp}, \mathbb{x}) \notin \mathcal{L}_\mathcal{R}$. We say that a relation $\mathcal{R}$ is always satisfiable if, for any* pp*, the language $\mathcal{L}_{\mathcal{R},\mathrm{pp}} = \{0,1\}^*$, where the latter is the language associated to the relation for given parameters* pp.

```
Domain separation: $\mathsf{H}_1$ and $\mathsf{H}_2$ are two random oracles.
// Conjunction proofs $\mathit{\Pi}_\wedge$ and $\bar{\mathit{\Pi}}_\wedge$ (including $\boxed{\text{boxed instructions}}$)
Prover: $\mathcal{P}^{\mathsf{H}_1,\mathsf{H}_2}(\mathrm{pp}, (\mathbb{x}_A, \mathbb{x}_B), \mathbb{w})$ does:
  1. commit $\mathbf{c}, \rho \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \mathbb{w})$,
  2. prove $\pi_A \leftarrow \mathit{\Pi}_A.\mathcal{P}^{\mathsf{H}_1}(\mathrm{pp}, (\mathbf{c}, \mathbb{x}_A), (\mathbb{w}, \rho))$,
  3. prove $\pi_B \leftarrow \mathit{\Pi}_B.\mathcal{P}^{\mathsf{H}_2}(\mathrm{pp}, (\boxed{\mathsf{msg}}, M_c(\mathbf{c}), \mathbb{x}_B), (M(\mathbb{w}), M_\rho(\rho)))$
       $\boxed{\text{where } \mathsf{msg} := \mathbb{x}_A \| \pi_A}$
Verifier: $\mathsf{Vf}^{\mathsf{H}_1,\mathsf{H}_2}(\mathrm{pp}, (\mathbb{x}_A, \mathbb{x}_B), \pi)$ parses $\pi = (\mathbf{c}, \pi_A, \pi_B)$, return 1 if and only if $\mathsf{Vf}^{\mathsf{H}_1}(\mathrm{pp}, (\mathbf{c}, \mathbb{x}_A), \pi_A) = 1$ and $\mathsf{Vf}^{\mathsf{H}_2}(\mathrm{pp}, (\boxed{\mathsf{msg},} M_c(\mathbf{c}), \mathbb{x}_B), \pi_B) = 1$

// Composition proofs $\mathit{\Pi}_{g \circ f}$ and $\bar{\mathit{\Pi}}_{g \circ f}$ (including $\boxed{\text{boxed instructions}}$)
Prover: $\mathcal{P}^{\mathsf{H}_1,\mathsf{H}_2}(\mathbf{c}_i, \mathbf{c}_o, \mathbb{x}_i, \mathbb{x}_o, \mathbb{w}_i, \mathbb{w}_o, \rho_i, \rho_o)$ does
  1. parse $\mathbb{x}_i = x_{f,i} \| x_{g,i}$ and $\mathbb{x}_o = x_{f,o} \| x_{g,o}$ and $x_f = (x_{f,i}, x_{f,o})$,
  2. let $(x_{f,o}, w_{g,i}) \leftarrow f(x_{f,i}, w_{f,i})$ and $\mathbf{c}', \rho' \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, w_{g,i})$.
  3. prove $\pi_F \leftarrow \mathit{\Pi}_f.\mathcal{P}^{\mathsf{H}_1}(\mathbf{c}_i, \mathbf{c}', (x_{f,i}, x_{F_o}), (\mathbb{w}_i, w_{g,i}), \rho_i, \rho')$,
  4. prove $\pi_G \leftarrow \mathit{\Pi}_g.\mathcal{P}^{\mathsf{H}_2}(\boxed{\mathsf{msg}}, \mathbf{c}', \mathbf{c}_o, (x_{g,i}, x_{g,o}), (w_{g,i}, w_o), \rho', \rho_o)$
       $\boxed{\text{where } \mathsf{msg} := \mathbf{c}_i \| x_F \| \pi_1}$
Verifier: $\mathsf{Vf}((\mathbf{c}_i, \mathbf{c}_o, \mathbb{x}_i, \mathbb{x}_o), \pi)$ parses $\pi = (\mathbf{c}', \pi_1, \pi_2)$, return 1 if and only if $\mathit{\Pi}_f.\mathsf{Vf}^{\mathsf{H}_1}((\mathbf{c}_i, \mathbf{c}', x_{f,i}, x_{f,o}), \pi_1) = 1$ and $\mathit{\Pi}_g.\mathsf{Vf}^{\mathsf{H}_2}((\mathsf{msg}, \mathbf{c}', \mathbf{c}_o, x_{g,i}, x_{g,o}), \pi_2) = 1$.
```

Fig. 16. Compiler to proofs for conjunction (top) and function composition (bottom). For $X \in \{A, B, f, g\}$ $\mathit{\Pi}_X$ is assumed to be a commit-and-prove non-interactive argument over commitment scheme CS (assumed to be M-malleable for the compiler for conjunction). For $\bar{\mathit{\Pi}}_\wedge$ (resp. $\bar{\mathit{\Pi}}_{g \circ f}$) we additionally assume that $\mathit{\Pi}_A$ (resp. $\mathit{\Pi}_g$) is a signature of knowledge.

The definition above indicates that the relation $\mathcal{R}$ can be decided by an expected polynomial-time algorithm. At first glance, one might consider an argument of knowledge for a relation in P[14] to be somewhat trivial. However, the scenario becomes more compelling in the context of commit-and-prove relations. In this case, while $\mathcal{R}$ is decidable, the corresponding commit-and-prove relation $\hat{\mathcal{R}}$ is not, unless, we allow the prover to sample the commitment to the witness.

Nicely, when the relation $\mathcal{R}_A$ (resp. $\mathcal{R}_f$) is efficiently witness computable we can weaken the zero-knowledge property of $\mathit{\Pi}_A$ (resp. $\mathit{\Pi}_f$) in the compilers to witness indistinguishability (WI)[15]. Furthermore, for WI to hold, it is not necessary to reprogram the random oracle.

Definition 15. *An non-interactive argument for $\mathcal{R}$ is statistically witness indistinguishable (WI) if for any* pp *and any* $\mathbb{x}, \mathbb{w}_1, \mathbb{w}_2$ *such that* $(\mathrm{pp}, \mathbb{x}, \mathbb{w}_i) \in \mathcal{R}$ *the distributions* $\mathcal{P}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \mathbb{w}_i)$ *for* $i \in \{1, 2\}$ *are statically close.*

Theorem 5. *Assuming that the commitment scheme* CS *is hiding and binding, the following statements hold true:*

1. *For any PT M, if* CS *is M-malleable, and $\mathit{\Pi}_A$ and $\mathit{\Pi}_B$ are trapdoorless zero-knowledge and simulation extractable then $\mathit{\Pi}_\wedge$ for the relation $\mathcal{R}^M_{A \wedge B}$ is simulation-extractable.*
2. *If $\mathit{\Pi}_f$ and $\mathit{\Pi}_g$ are trapdoorless zero-knowledge and simulation extractable then $\mathit{\Pi}_{g \circ f}$ is simulation-extractable.*
3. *For any PT M,* CS *is M-malleable, and if $\mathit{\Pi}_A$ is knowledge sound and statistically witness indistinguishable, $\mathcal{R}_A$ is always satisfiable and efficiently witness computable and $\mathit{\Pi}_B$ is trapdoorless zero-knowledge and a signature-of-knowledge then $\mathit{\Pi}'$ is simulation-extractable.*

[14] More precisely, the class $AvgP$.

[15] Zero-knowledge implies witness indistinguishability, see Feige and Shamir [23].

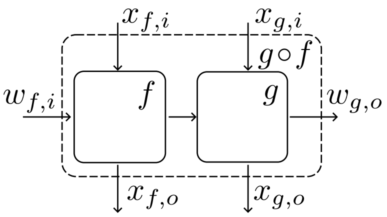

Fig. 17. Graph for the functional composition $g \circ f$.

4. *If Π_f is knowledge sound and statistically witness indistinguishable, $\mathcal{R}_f$ is always satisfiable and efficiently witness computable or the public output of f is the empty string, namely for any $x_{f,i}, \mathbb{w}_i$ we have $|x_{f,o}| = 0$ where $x_{f,o}, w_{f,o} = f(x_{f,i}, \mathbb{w}_i)$, and Π_g is trapdoorless zero-knowledge and a signature-of-knowledge then $\bar{\Pi}_{g \circ f}$ is simulation-extractable.*

Before proving the theorem we remark that the notion of *trapdoorless* zero-knowledge is key for the four statements to hold. This is evident, for example, in the proof of the fourth statement, where we can invoke the knowledge soundness of Π_f in the presence of simulated proofs for Π_g. We can do this because to simulate proofs we only need to reprogram the random oracle H_2 which does not interfere with Π_f. On the other hand, if we needed a trapdoor for the simulations then we would need to make sure that the knowledge sound of Π_f held in the presence of such a trapdoor (for example, by sampling independent reference strings for the two schemes, which is unnatural and cumbersome in many practical scenarios).

Proof (of Theorem 5). We proceed statement by statement.

First statement. Completeness of Π_A and Π_B, together with the M-malleability of the commitment scheme, imply the completeness of $\Pi_\wedge$.

As for simulation extractability, let $\mathcal{P}^*$ be an adversary for the simulation extractability of $\Pi_\wedge$. Given the set $\mathcal{Q}_{\mathsf{sim}}$ of queries and answers to the simulation oracle, we can derive the set $\mathcal{Q}_{\mathsf{sim},i}$ of queries and answers to Π_i for $i \in \{A, B\}$. Specifically, if $(\mathbb{x}, \pi) \in \mathcal{Q}_{\mathsf{sim}}$ and $\mathbb{x} = (\mathbb{x}_A, \mathbb{x}_B)$ and $\pi = (\mathbf{c}, \pi_A, \pi_B)$ then $((\mathbf{c}, \mathbb{x}_i), \pi_i) \in \mathcal{Q}_{\mathsf{sim},i}$ for $i \in \{A, B\}$. As a shortcut, given a tuple $(\mathbb{x}, \pi)$ for $\Pi_\wedge$, we can define $(\mathbb{x}, \pi)_i$ the derived tuple of instance and proof for Π_i.

Let $(\tilde{\mathbb{x}}, \tilde{\pi})$ be the forgery of the adversary, where $\tilde{\mathbb{x}} := (\tilde{\mathbb{x}}_A, \tilde{\mathbb{x}}_B)$ and $\tilde{\pi} := (\tilde{\mathbf{c}}, \tilde{\pi}_1, \tilde{\pi}_B)$, and consider the event bad:

$$(\tilde{\mathbb{x}}, \tilde{\pi}) \notin \mathcal{Q}_{\mathsf{sim}} \wedge \left(\bigwedge_{i \in \{A,B\}} (\tilde{\mathbb{x}}, \tilde{\pi})_i \in \mathcal{Q}_{\mathsf{sim},i} \right) \tag{5}$$

It is easy to check that the $\Pr[\mathsf{bad}] = 0$. In fact, if $(\tilde{\mathbb{x}}, \tilde{\pi})_1 \in \mathcal{Q}_{\mathsf{sim},1}$ then either $\tilde{\mathbb{x}}_B$ or $\tilde{\pi}_B$ are *fresh*, namely either $\forall \pi_2' : (\tilde{\mathbf{c}}, \tilde{\mathbb{x}}_B, \pi_2') \notin \mathcal{Q}_{\mathsf{sim},2}$ or $\forall \mathbb{x}_2' : (\tilde{\mathbf{c}}, \mathbb{x}_2', \tilde{\pi}_B) \notin \mathcal{Q}_{\mathsf{sim},2}$, as otherwise $(\tilde{\mathbb{x}}, \tilde{\pi}) \in \mathcal{Q}_{\mathsf{sim}}$. The other alternative is $(\tilde{\mathbb{x}}, \tilde{\pi})_2 \in \mathcal{Q}_{\mathsf{sim},2}$, which is handled similarly.

First we show that $\Pi_\wedge$ is indeed zero-knowledge. Let $\mathcal{S}_i$ be the zero-knowledge simulator for Π_i, and consider the zero-knowledge simulator $\mathcal{S}_\wedge$ that runs $\mathcal{S}_A, \mathcal{S}_B$ in parallel, in particular the simulator provides three interfaces to the adversary, the simulation oracle query on the appropriate instances the simulators $\mathcal{S}_A$ and $\mathcal{S}_B$ (following the specification as in the prover), while the other two oracles are the simulator for the random oracles, in particular, $\mathcal{S}_\wedge$ queries $\mathcal{S}_A$ for the queries directed to H_1 and $\mathcal{S}_B$ for the queries directed to H_2. Because of the domain separation, the simulators can handle the RO-queries independently. More in detail, the simulation oracles are handled as follows:

- Parse the instance as $\mathbb{x}$ as $(\mathbb{x}_A, \mathbb{x}_B)$.
- Sample a commitment to a dummy value, namely $\mathbf{c}, \rho \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \bar{0})$.
- Run the simulator $\mathcal{S}_A$ on $(\mathbf{c}, \mathbb{x}_A)$ and $\mathcal{S}_B$ on $(M_c(\mathbf{c}), \mathbb{x}_B)$.

It is rather straight-forward to show that if $\mathit{\Pi}_A$ and $\mathit{\Pi}_B$ are zero-knowledge and the commitment scheme is hiding, then $\mathit{\Pi}_\wedge$ is zero-knowledge.

Let $\mathcal{P}^*_i$ be the adversary for the simulation extractability of $\mathit{\Pi}_i$ that internally runs $\mathcal{P}^*$ and the simulator $\mathcal{S}_{\bar{i}}$ where $i \in \{A, B\}$ and $\bar{i}$ is set to B if $i = A$ and to B otherwise. Specifically, the adversary does:

- Upon simulation query $\mathbb{x}$ for $\mathit{\Pi}_\wedge$, similarly to the simulator $\mathcal{S}$, it samples a commitment $\mathbf{c}, \rho \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \bar{0})$, runs $\mathcal{S}_{\bar{i}}$ on the derived instance $(\mathbf{c}, \mathbb{x}_{\bar{i}})$ and queries the simulation oracle for the instance $(\mathbf{c}, \mathbb{x}_i)$.
- Forward the query to the random oracle appropriately: either internally handled by $\mathcal{S}_{\bar{i}}$, or externally forwarded the queries to $\mathcal{P}^*_i$'s random oracle.
- Upon forgery $(\tilde{\mathbb{x}}, \tilde{\pi})$ output the forgery $(\tilde{\mathbb{x}}, \tilde{\pi})_i$.

Since the event bad defined in Eq. (5) never happens, from a valid forgery for $\mathit{\Pi}_\wedge$ we can derive either valid forgery for $\mathit{\Pi}_A$ or for $\mathit{\Pi}_B$, thus:

$$\Pr\left[\mathsf{SE}^{\mathcal{S},\mathcal{P}^*}_{0,\mathit{\Pi}_\wedge}(\lambda)\right] \leq \Pr\left[\mathsf{SE}^{\mathcal{S}_A,\mathcal{P}^*_A}_{0,\mathit{\Pi}_A}(\lambda)\right] + \Pr\left[\mathsf{SE}^{\mathcal{S}_B,\mathcal{P}^*_B}_{0,\mathit{\Pi}_B}(\lambda)\right] \tag{6}$$

We define the knowledge extractor $\mathcal{E}$ for $\mathit{\Pi}_\wedge$:

- For $i \in \{A, B\}$, run $\mathcal{E}_i$ interacting with $\mathcal{P}^*_i$ and let $\hat{\mathbb{w}}_i := (\mathbb{w}_i, \rho_i)$ be the output of $\mathcal{E}_i$.
- If $(M(\mathbb{w}_A), M_o(\rho_A)) \neq (\mathbb{w}_B, \rho_B)$ abort, otherwise output $\mathbb{w}_A$.

Notice, the description above is incomplete because we did not describe how $\mathcal{E}$ provides the interaction between $\mathcal{E}_i$ and $\mathcal{P}^*_i$. More in detail, the extractor can provide a virtual interface to $\mathcal{P}^*_i$ given oracle access to $\mathcal{P}^*$ using the same strategy we define the adversary $\mathcal{P}^*_i$ using only oracle access to $\mathcal{P}^*$. Moreover, the two (internal) extractors are run with independent randomness. We can show that:

$$\begin{aligned}\Pr\left[\mathsf{SE}^{\mathcal{E},\mathcal{S},\mathcal{P}^*}_{1,\mathit{\Pi}_\wedge}(\lambda)\right] &= \Pr\left[\mathsf{SE}^{\mathcal{S},\mathcal{P}^*}_{0,\mathit{\Pi}_\wedge}(\lambda) \wedge \neg\left(\wedge_i(\tilde{\mathbf{c}}, \tilde{\mathbb{x}}_i, \hat{\mathbb{w}}_i) \in \hat{\mathcal{R}}_i \wedge \neg\mathtt{Abort}\right)\right] \\ &\leq \sum_i \Pr\left[\mathsf{SE}^{\mathcal{E}_i,\mathcal{S}_i,\mathcal{P}^*_i}_{1,\mathit{\Pi}_i}(\lambda)\right] + \Pr[\mathsf{Abort}]\end{aligned} \tag{7}$$

The running time of the extractor is the sum of the running times of $\mathcal{E}_A$ and $\mathcal{E}_B$, and when Abort happens, we can break the binding property of the commitment scheme. Putting Eqs. (6) and (7) together we have:

$$\mathbf{Adv}^{\mathsf{SIM-EXT}}_{\mathit{\Pi}_\wedge,\mathcal{R}}(\mathcal{S}, \mathcal{E}, \mathcal{P}^*) \leq \sum_{i \in \{A,B\}} \mathbf{Adv}^{\mathsf{SIM-EXT}}_{\mathit{\Pi}_i,\mathcal{R}_i}(\mathcal{S}_i, \mathcal{E}_i, \mathcal{P}^*_i) + \mathsf{negl}(\lambda)$$

Second Statement. The proof for this statement is almost the same as the previous proof. The main difference is the definition of the extractor which aborts in case it finds two different openings for the commitment $\mathbf{c}'$. We give more details on the extractor in the proof of the forth statement.

Third Statement. Similarly to the proof of the first statement, we consider $\mathcal{P}^*$ be an adversary for the simulation extractability of $\bar{\mathit{\Pi}}_\wedge$. Given the set $\mathcal{Q}_{\mathsf{sim}}$ of queries and answers to the simulation oracle, we can derive the set $\mathcal{Q}_{\mathsf{sim},i}$ of queries and answers to $\mathit{\Pi}_i$ for $i \in \{A, B\}$. The only difference is that from a tuple $(\mathbb{x}, \pi)$ for $\bar{\mathit{\Pi}}_\wedge$ we derive the tuple $(\mathbb{x}, \pi)_B = (\mathsf{msg}, \mathbf{c}, \mathbb{x}_B, \pi_B)$ where $\mathsf{msg} = \mathbb{x}_A \| \pi_A$ for $\mathit{\Pi}_B$. Thanks to this difference, if the forgery $(\tilde{\mathbb{x}}, \tilde{\pi}) \notin \mathcal{Q}_{\mathsf{sim}}$ then the derived forgery $(\tilde{\mathbb{x}}, \tilde{\pi})_2 \notin \mathcal{Q}_{\mathsf{sim},2}$.

First we show that $\bar{\mathit{\Pi}}_\wedge$ is indeed zero-knowledge. Let $\mathcal{S}_B$ be the zero-knowledge simulator for $\mathit{\Pi}_B$, and consider the zero-knowledge simulator $\mathcal{S}_\wedge$ that:

- Parse the instance as $\mathbb{x} = (\mathbb{x}_A, \mathbb{x}_B)$.
- Let $\mathcal{M}$ be the algorithm satisfying the efficiently witness computability of $\mathcal{R}_A$, compute $\mathbb{w}_A \leftarrow \mathcal{M}(\mathbb{x}_A)$.
- Sample ρ_A and computes honest proof π_A for $(\mathbf{c}, \mathbb{x}_A, \mathbb{w}, \rho_A) \in \hat{\mathcal{R}}_A$ where $\mathbf{c}, \rho_A \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \mathbb{w}_A)$.
- Run the simulator $\mathcal{S}_B$ on $(\mathsf{msg}, \mathbf{c}, \mathbb{x}_B)$.

We show that if $\mathit{\Pi}_A$ is statistical witness-indistinguishable, $\mathit{\Pi}_B$ is zero-knowledge, and the commitment scheme is hiding, then $\mathit{\Pi}_\wedge$ is zero-knowledge. We start from the real-world distribution of honestly generated proofs and move to the ideal distribution of simulated proofs through an hybrid argument.

- The first hybrid $\mathbf{H}_1$ is the same as the real-world distribution but the proof π_B is computed using the simulator $\mathcal{S}_B$ on message $(\mathsf{msg}, \mathbf{c}, \mathbb{x}_B)$. It is easy to show that the real world and the hybrid $\mathbf{H}_1$ are statistically close thanks to the (statistical) zero-knowledge property of $\mathit{\Pi}_B$.
- In the second hybrid $\mathbf{H}_2$, the prover, on input $(\mathbb{x}, \mathbb{w})$, additionally computes $\mathbb{w}' \leftarrow \mathcal{M}(\mathbb{x}_A)$ and breaks the binding of the commitment $\mathbf{c}$ finding ρ' such that $\mathsf{VerCom}(\mathrm{ck}, \mathbf{c}, \mathbb{w}', \rho') = 1$. It aborts if it cannot find such an opening ρ'. The difference between the two hybrids is the event that $\mathbf{H}_2$ might abort. We can show that, since the commitment scheme is statistically hiding, the event happens with negligible probability. Briefly, the reduction fixes messages $\mathbb{w}$ and $\mathbb{w}' \leftarrow \mathcal{M}(\mathbb{x}_A)$ and, given a challenge commitment $\mathbf{c}$, it outputs 1 if it can brute-force the commitment on a valid opening w.r.t. $\mathbb{w}'$. Notice, if the challenge commitment is a commitment to $\mathbb{w}$, the reduction outputs 0 with the same probability of the aborting event, while if $\mathbf{c}$ is a commitment to $\mathbb{w}'$ there exists always a valid opening so the reduction eventually outputs 1.
- The hybrid $\mathbf{H}_3$ is the same as $\mathbf{H}_2$ but the proof π_A is computed using witness $(\mathbb{w}', \rho')$. The two hybrid are statistically close thanks to the statistical witness indistinguishability of $\mathit{\Pi}_A$.
- The last hybrid $\mathbf{H}_4$ is the same as $\mathbf{H}_3$ but the commitment is computed directly as a commitment to $\mathbb{w}'$. Again, we can reduce to the hiding of the commitment scheme. Also notice, this last hybrid is equivalent to the simulated world.

We are ready to prove simulation extractability. Let $\mathcal{P}^*_A$ be an adversary for the knowledge extractability of $\mathit{\Pi}_A$ that internaly runs $\mathcal{P}^*$ and $\mathcal{S}_2$. Notice that, even if $\mathcal{R}_A$ is efficiently witness computable, the relation $\hat{\mathcal{R}}_A$ (proved by $\mathit{\Pi}_A$) is not polynomially decidable, thus the notion of knowledge extractability is still meaningful.

Specifically the adversary $\mathcal{P}^*_A$ does:

- Upon simulation query $\mathbb{x}$ for $\bar{\mathit{\Pi}}_\wedge$, similarly to the simulator $\mathcal{S}$ described above, it computes $\mathbb{w}_A \leftarrow \mathcal{M}(\mathbb{x}_A)$ and samples commitment $\mathbf{c}, \rho_A \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \mathbb{w}_A)$, it runs $\mathcal{S}_2$ on the derived instance $(\mathsf{msg}, \mathbf{c}, \mathbb{x}_2)$.
- Internally forward the query to H_2 to $\mathcal{S}_B$, and (externally) forward the queries to H_1.
- Upon forgery $(\tilde{\mathbb{x}}, \tilde{\pi})$ output the forgery $(\tilde{\mathbb{x}}, \tilde{\pi})_A$.

Almost identically, let $\mathcal{P}^*_B$ be an adversary for the simulation extractability of $\mathit{\Pi}_B$ that internally runs $\mathcal{P}^*$. Specifically the adversary $\mathcal{P}^*_B$ does:

- Upon simulation query $\mathbb{x}$ for $\bar{\mathit{\Pi}}_\wedge$, similarly to the simulator $\mathcal{S}$ described above, it computes $\mathbb{w}_A \leftarrow \mathcal{M}(\mathbb{x}_A)$ and samples commitment $\mathbf{c}, \rho_A \leftarrow \mathsf{CS.Commit}(\mathrm{ck}, \mathbb{w}_A)$, and it queries the simulation oracle on the derived instance $(\mathsf{msg}, \mathbf{c}, \mathbb{x}_B)$.
- Forward the query to H_i for $i \in \{A, B\}$ to the appropriate oracles.
- Upon forgery $(\tilde{\mathbb{x}}, \tilde{\pi})$ output the forgery $(\tilde{\mathbb{x}}, \tilde{\pi})_B$.

Identically to the proof of the first statement, we define the knowledge extractor $\mathcal{E}$ for $\bar{\mathit{\Pi}}_\wedge$ to:

- For $i \in \{A, B\}$, run $\mathcal{E}_i$ interacting with $\mathcal{P}^*_i$ and let $\hat{\mathbb{w}}_i = (\mathbb{w}_i, \rho_i)$ be the output of $\mathcal{E}_i$.
- If $M(\mathbb{w}_A), M_o(\rho_A) \neq \mathbb{w}_B, \rho_B$ abort, otherwise output $\mathbb{w}_A$.

As in the proof of the first statement, the extractor can provide a virtual interface to $\mathcal{P}^*_i$ given oracle access to $\mathcal{P}^*$ using the same strategy we define the adversary $\mathcal{P}^*_i$ using only oracle access to $\mathcal{P}^*$. The only difference is that $\mathcal{P}^*_1$ is an adversary for the knowledge extractability (it does not need simulation queries). Moreover, the two (internal) extractors are run with independent randomness. It is rather straight-forward to show that:

$$\begin{aligned}\Pr\Big[\mathsf{SE}^{\mathcal{E},\mathcal{S},\mathcal{P}^*}_{1,\mathit{\Pi}_\wedge}(\lambda)\Big] &= \Pr\Big[\mathsf{SE}^{\mathcal{S},\mathcal{P}^*}_{0,\mathit{\Pi}_\wedge}(\lambda) \wedge \neg\Big(\wedge_i(\tilde{\mathbf{c}}, \tilde{\mathbb{x}}_i, \hat{\mathbb{w}}_i) \in \hat{\mathcal{R}}_i \wedge \neg\mathsf{Abort}\Big)\Big] \leq \\ &\leq \Pr\Big[\mathsf{SE}^{\mathcal{E}_B,\mathcal{S}_B,\mathcal{P}^*_B}_{1,\mathit{\Pi}_B}(\lambda)\Big] + \Pr\Big[\mathsf{KS}^{\mathcal{E}_A,\mathcal{P}^*_A}_{A,\mathit{\Pi}_A}(\lambda)\Big] + \Pr[\mathsf{Abort}] \qquad (8)\end{aligned}$$

The running time of the extractor is the sum of the running times of $\mathcal{E}_A$ and $\mathcal{E}_B$, and when Abort happens, we can break the binding property of the commitment scheme. Putting things together we have:

$$\mathbf{Adv}^{\mathsf{SIM-EXT}}_{\Pi_\wedge,\mathcal{R}}(\mathcal{S},\mathcal{E},\mathcal{P}^*) \le \mathbf{Adv}^{\mathsf{KS}}_{\Pi_A,\mathcal{R}_A}(\mathcal{E}_A,\mathcal{P}^*_A) + \mathbf{Adv}^{\mathsf{SIM-EXT}}_{\Pi_B,\mathcal{R}_B}(\mathcal{S}_B,\mathcal{E}_B,\mathcal{P}^*_B) + \mathsf{negl}(\lambda). :$$

Fourth Statement. Given the set $\mathcal{Q}_{\mathsf{sim}}$ of queries and answers to the simulation oracle, we can derive the sets $\mathcal{Q}_{\mathsf{sim},f}$ and $\mathcal{Q}_{\mathsf{sim},g}$ of queries and answers to Π_f and Π_g respectively. Specifically, if $\mathbb{x},\pi \in \mathcal{Q}_{\mathsf{sim}}$ and $\mathbb{x} = (\mathbf{c}_i,\mathbf{c}_o,\mathbb{x}_i,\mathbb{x}_o)$ and $\pi = (\mathbf{c}',\pi_f,\pi_g)$ then $(\mathbf{c}_i,\mathbf{c}',x_{f,i},x_{f,o}),\pi_f \in \mathcal{Q}_{\mathsf{sim},f}$ and $(\mathsf{msg},\mathbf{c}',\mathbf{c}_o,x_{g,i},x_{g,o}),\pi_g \in \mathcal{Q}_{\mathsf{sim},g}$ where $\mathsf{msg} = \mathbf{c}' \| x_f \| \pi_f$. As a shortcut, given a tuple $\mathbb{x},\pi$ for $\Pi_{g\circ f}$, we can define $(\mathbb{x},\pi)_X$ the derived tuple of instance and proof for Π_X for $X \in \{f,g\}$.

Similarly to the simulator for zero-knowledge in the proof of the third statement. We can define a simulator for $\Pi_{g\circ f}$ that makes use of the efficient witness computability of $\mathcal{R}_f$. Additionally, we give a second simulator for the special case where $x_{f,o}$ is the empty string for any assignments of the public and private inputs $x_{f,i}, w_{f,i}$.

Let $\mathcal{S}_g$ the zero-knowledge simulator for Π_g, and consider the zero-knowledge simulator $\mathcal{S}$ (resp. the zero-knowledge simulator $\mathcal{S}'$ that executes Item 2b instead of Item 2a) that:

1. Parse the instance as $\mathbb{x} = (\mathbb{x}_f,\mathbb{x}_g)$.
2. Execute one of the two steps:
   (a) Let $\mathcal{M}$ be the algorithm satisfying the efficient witness computability of $\mathcal{R}_f$, compute $\mathbb{w}_f = (w_{f,i},w_{f,o}) \leftarrow \mathcal{M}(\mathbb{x}_f)$.
   (b) Set $w_{f,i} := \bar{0}$ compute $w_{f,o} \leftarrow f(x_{f,i},w_{f,o})$ and let $\mathbb{w}_f := (w_{f,i},w_{f,o})$.
3. Compute honest proof π_F for $(\mathbf{c}_i,\mathbf{c}',\mathbb{x}_f,\mathbb{w}_f) \in \hat{\mathcal{R}}_f$ where $\mathbf{c}',\rho' \leftarrow \mathsf{CS.Commit}(\mathrm{ck},w_{f,o})$ and $\mathbf{c}_i,\rho_i \leftarrow \mathsf{CS.Commit}(pp,w_{f,i})$.
4. Sample dummy commitment $\mathbf{c}_o,\rho_o \leftarrow \mathsf{CS.Commit}(\mathrm{ck},\bar{0})$, run the simulator $\mathcal{S}_g$ on $(\mathsf{msg},\mathbf{c}',\mathbf{c}_o,\mathbb{x}_g)$.

The proofs of zero-knowledge w.r.t. the two simulators follow similarly to the proof of zero-knowledge in the third statement. In particular, in both cases, we first switch to simulated proofs for Π_g and then use the hybrid argument that use a combination of the hiding property and the witness indistinguishability property. Also to prove simulation extractability we proceed similarly to the proof of the third statement. We omit the details as the proof is almost identical.

6.2 Discussion and Applications

We note that, if we disregard the aspects of commitment malleability (see Definition 13), the compilers for functional composition are more general than those for conjunction. Specifically, we could think of the function f as computing the relation $\mathcal{R}_A$ and passing the witness, unchanged, to the next function g, which in turn computes the relation $\mathcal{R}_B$.

We chose to present two distinct types of compilation (conjunctions and functional compositions) because this approach arguably makes it easier to present our results. Additionally, the simpler compiler (for conjunction) allows us to handle the commitment malleability aspects more directly.

In terms of assumptions, the third and fourth results trade the (additional) efficient witness sampleability property (see Definition 14) for weaker assumptions on the security of the arguments of knowledge. While the assumption of efficient witness sampleability might seem strong, for functional composition, we can omit this assumption by requiring a structural property on f. This is another reason why we include the fourth result, even though in the following discussion on zkVM in Section 7, we only require the compilers for conjunction.

7 Simulation extractability of zkVMs

7.1 Preliminaries on SNARK VMs

Here we provide an abstract treatment of virtual machines. We start from this general definition:

```
The virtual machine $\mathtt{VM}_{\mathsf{Execute}}(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{z})$:
Set $(\mathsf{sregs}, \mathsf{regs}) \leftarrow 0^{k+4}$, $\mathsf{mem} \leftarrow \mathbb{x} \| \mathbb{z}$.
Iteratre for $t$ times the following:
  – Update Program-Counter: $\mathsf{sregs}[0] \leftarrow \mathsf{regs}[0]$.
  – Fetch: $\mathsf{sregs}[1] \leftarrow \mathsf{P}_{\mathsf{code}}[\mathsf{sregs}[0]]$.
  – Read-and-Write Operations:
      • $\mathsf{sregs}[2] \leftarrow \mathsf{mem}[\mathsf{regs}[1]]$,    //read from memory
      • $\mathsf{sregs}[3] \leftarrow \mathsf{regs}[3]$,              //load to special register
      • $\mathsf{mem}[\mathsf{regs}[2]] \leftarrow \mathsf{sregs}[3]$,   //write to memory
  – Execute: $\mathsf{regs} \leftarrow \mathsf{Execute}(\mathsf{sregs})$
Output $\mathbb{y} = \mathsf{mem}[0 : o]$.
```

Fig. 18. The VM with parameters the instruction set $\mathsf{Execute}$ and a time bound t. The inputs are the program code $\mathsf{P}_{\mathsf{code}}$, a public input $\mathbb{x}$ and a private input $\mathbb{z}$, the output of the VM is $\mathbb{y}$. The machines load on the memory the inputs and executes t steps, the output $\mathbb{y}$ of the machine is the state of the memory after t steps. There are four *special* registers: $\mathsf{sregs}[0]$ stores the current program counter, $\mathsf{sregs}[1]$ stores the next instruction, while $\mathsf{sregs}[1]$ and $\mathsf{sregs}[2]$ store the (two) operands for the next instruction, in particular, $\mathsf{sregs}[1]$ stores data fetched from the main memory and $\mathsf{sregs}[2]$ stores data from the result of the previous instruction. The instructions in $\mathsf{Execute}$ do not change the content of the special registers and update the program counter for the fetch of the next instruction in $\mathsf{regs}[0]$. The VM, at any iteration, writes in memory at location $\mathsf{regs}[2]$ the content of $\mathsf{sregs}[3]$ and at $\mathsf{sregs}[3]$ the content of $\mathsf{regs}[3]$, these are (somewhat arbitrary) operations to allow flow of information from regs to sregs and from sregs to memory: notice that different architectures performing additional reading/writing operations are theoretically (and practically) equivalent.

Definition 16 (Instruction Set (Execution)). *Let $\gamma, k \in \mathbb{N}$. An instruction set for a virtual machine with k registers and codewords of size γ is an efficiently computable function* $\mathsf{Execute} : \{0,1\}^{\gamma \cdot (k+4)} \rightarrow \{0,1\}^{\gamma \cdot k}$.

We want to describe the relation which describes a virtual machine execution. Consider the circuit in Fig. 18. This is parameterized by an *instruction set* $\mathsf{Execute}$, an execution bound t, a bound on the number of register k, codewords of size γ, and a bound on the output size o. We denote the circuit thus parametrized as $\mathtt{VM}_{\mathsf{Execute},t,o}$. (For simplicity, we hide all the parameters but $\mathsf{Execute}$, and simply write $\mathtt{VM}_{\mathsf{Execute}}$ whenever the parameters are clear from the context.) Following [2], we define the commit-and-prove relation:

$$\mathcal{R}^{\mathsf{Execute}}_{\mathsf{zkVM}}((t,o), (\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y}), \mathbb{z}) \iff \quad \mathtt{VM}_{\mathsf{Execute}}(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{z}) = \mathbb{y} \tag{†}$$

Splitting $\mathcal{R}_{\mathsf{zkVM}}$ in its logical components. We now show how to approach proving the relation $\mathcal{R}_{\mathsf{zkVM}}$ from a modular perspective. The way we "split" relation $\mathcal{R}_{\mathsf{zkVM}}$ will roughly follow the lookup-singularity approach in [2]. For this reason we will isolate an execution component (which in [2] is performed through the lookup argument Lasso) and "anything else" (in relation $\mathcal{R}^{\star}$) roughly consisting of instruction fetching (which we abstracted out in our VM model) and memory checking. For simplicity we do not break this second part further; our goal is to showcase the modular flavor of zkVMs and to provide a blueprint that can be specialized in follow-up works[16]. We thus define the commit-and-prove relation below:

$$\mathcal{R}_{\mathsf{Execute}}(\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}) \iff \forall i \in [t-1] : \mathsf{Execute}(\mathbb{w}_{\mathsf{sregs}}[i]) = \mathbb{w}_{\mathsf{regs}}[i+1]$$

Here $(\mathbb{w}_{\mathsf{sregs}}[i], \mathbb{w}_{\mathsf{regs}}[i])$ is the state of the registers of the virtual machine at the i-th step of computation. Looking ahead, we associate the tuple $(\mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{regs}})$ with the trace of the virtual machine in the computation of the program $\mathsf{P}_{\mathsf{code}}$ on input $(\mathbb{x}, \mathbb{z})$.

[16] The work in [2] actually logically separates memory checking and instruction fetching. Both the components they use for these modules can be thought of more or less specialized versions of $\mathsf{Spartan}$. Therefore, in spirit, our the instantiations we present in Section 7.4 are still applicable to the original presentation in [2].

Definition 17. *The* zkVM-complementary relation *is the relation* $\mathcal{R}^\star$ *such that for any execution set* $\mathsf{Execute}$*, and for any input* $(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y}), \mathbb{z}$*:*

$$\begin{aligned}\mathcal{R}^{\mathsf{Execute}}_{\mathsf{zkVM}}((\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y}), \mathbb{z}) \iff \; & \exists(\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{mem}}) : \\ & \mathcal{R}_{\mathsf{Execute}}(\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}) \wedge \\ & \mathcal{R}^\star(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y}, (\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{mem}}))\end{aligned}$$

where $\mathcal{R}^{\mathsf{Execute}}_{\mathsf{zkVM}}$ *is the relation defined as in Eq. (†).*

Intuitively, the relation $\mathcal{R}^\star$ needs to handle the logic of the virtual machine and make sure that the memory accesses, during the execution of the program, are consistent (namely, that we read the correct instructions from $\mathsf{P}_{\mathsf{code}}$, we perform the read and write operations, and that if the virtual machine reads from the memory the value v at location i, it means that the last time the virtual machine wrote at location i, it wrote the value v).

7.2 A General Theorem on the Non-Malleability of SNARK VMs

We say that a commit-and-prove argument of knowledge for $\mathcal{R}^\star$ has *separate commitments* (for CS) if the witnesses $\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}$ and $\mathbb{w}_{\mathsf{mem}}$ are committed separately. Namely, the witness $\mathbb{w} := (\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{mem}})$ for $\mathcal{R}^\star$ is committed as $\mathbf{c}_X, \rho_X \leftarrow \mathsf{Commit}(\mathrm{ck}, \mathbb{w}_X)$ for $X \in \{\mathsf{regs}, \mathsf{sregs}, \mathsf{mem}\}$ and $\mathbf{c} := (\mathbf{c}_{\mathsf{regs}}, \mathbf{c}_{\mathsf{sregs}}, \mathbf{c}_{\mathsf{mem}})$.

Theorem 6. *For any instruction set* $\mathsf{Execute}$*, let* Π *be a zero-knowledge argument of knowledge for* $\mathcal{R}_{\mathsf{Execute}}$ *that is simulation extractable, and let* $\Pi^\star$ *be an argument of knowledge for* $\mathcal{R}^\star$ *that has separate commitments. There exists a simulation-extractable zkVM if one of the following holds:*
1. $\Pi^\star$ *is simulation-extractable and zero-knowledge.*
2. $\Pi^\star$ *is witness-hiding and* Π *is a signature-of-knowledge.*

The theorem follows as an application of Theorem 5. The interesting case is when $\Pi^\star$ is WI. In this case, we additionally need to prove efficient witness sampleability by showing an *altered* instruction set that simply prints the output $\mathbb{y}$ into memory.

Proof (of Theorem 6). The theorem follows as an application of Theorem 5 and in particular of the first statement assuming the conditions in item (1) and third statement assuming the condition in item (2). We start with the more interesting case where we combine a knowledge-sound and witness-hiding scheme for $\mathcal{R}^\star$ with a simulation-extractable scheme for $\mathcal{R}_{\mathsf{Execute}}$. We can assume that the witness $\mathbb{w} := (\mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{mem}})$ for $\mathcal{R}^\star$ is committed separately, namely $\mathbf{c}_X, \rho_X \leftarrow \mathsf{Commit}(\mathrm{ck}, \mathbb{w}_X)$ for $X \in \{\mathsf{regs}, \mathsf{sregs}, \mathsf{mem}\}$ and $\mathbf{c} := (\mathbf{c}_{\mathsf{regs}}, \mathbf{c}_{\mathsf{sregs}}, \mathbf{c}_{\mathsf{mem}})$.

We define the functions $M_c(\mathbf{c}_{\mathsf{sregs}}, \mathbf{c}_{\mathsf{regs}}, \mathbf{c}_{\mathsf{mem}}) = (\mathbf{c}_{\mathsf{sregs}}, \mathbf{c}_{\mathsf{regs}})$, and similarly, $M(\mathbb{w}) = (\mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{regs}})$ and $M_\rho(\rho_{\mathsf{sregs}}, \rho_{\mathsf{regs}}, \rho_{\mathsf{mem}}) = (\rho_{\mathsf{sregs}}, \rho_{\mathsf{regs}})$. It is trivial to show that the commitment for $\Pi^\star$ is M-malleable and $M(\mathbb{w})$ is a witness for $\mathcal{R}_{\mathsf{Execute}}$ as required by the third statement of Theorem 5. We define the simulation-extractable zkVM as the composition $\bar{\Pi}_\wedge$ in Theorem 5 with M-malleable commitment between Π and $\Pi^\star$. We need to show that $\mathcal{R}^\star$ is efficiently witness sampleable and always satisfiable. For the former, consider the following "altered" virtual machine $\mathtt{VM}'_{\mathsf{Execute}}$:

```
– Run for $(t - o)$ iterations the code of $\mathtt{VM}_{\mathsf{Execute}}(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{z})$ with $\boxed{\mathbb{z} \leftarrow \bar{0}}$.
– For $i \in [o]$ runs the following:
    • Update Program-Counter: $\mathsf{sregs}[0] \leftarrow \mathsf{regs}[0]$.
    • Fetch: $\mathsf{sregs}[1] \leftarrow \mathsf{P}_{\mathsf{code}}[\mathsf{sregs}[0]]$.
    • Read-and-Write Operations:
        ∗ $\mathsf{sregs}[2] \leftarrow \mathsf{mem}[\mathsf{regs}[1]]$,    //read from memory
        ∗ $\mathsf{sregs}[3] \leftarrow \mathsf{regs}[3]$,             //load to special register
```

```
            * $\mathsf{mem}[\mathsf{regs}[2]] \leftarrow \mathsf{sregs}[3]$,   //write to memory
        • Execute: $\boxed{\mathsf{regs} \leftarrow (0, 0, i, \mathbb{y}_i, \bar{0})}$
    – Output $\mathbb{y} = \mathsf{mem}[0 : o]$.
```

We can compute the trace $\mathbb{w} := (\mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{regs}}, \mathbb{w}_{\mathsf{mem}})$ associated with the execution of the altered virtual machine $\mathtt{VM}'_{\mathsf{Execute}}$. In the code of $\mathtt{VM}'_{\mathsf{Execute}}$, the only difference w.r.t. an execution of $\mathtt{VM}_{\mathsf{Execute}}(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \bar{0})$ is that at the end we force to write to the first o locations of the memory the value $\mathbb{y}$, thus forcing the output of $\mathtt{VM}'_{\mathsf{Execute}}$ to $\mathbb{y}$. Notice that $\mathbb{w}$ is a valid witness for $\mathcal{R}^\star$ on instance $(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y})$, this is because $\mathcal{R}^\star$ does not enforce the consistency of the registers regs between two consecutive steps and, in particular, during the last o iterations.

Additionally, we notice that, if the commitment scheme is perfectly hiding, then language $\mathcal{L}_{\hat{\mathcal{R}}^\star}$ is always satisifiable because we can execute the procedure above to create a valid witness $\mathbb{w}$ and, although inefficiently, we can always find ρ such that the commitment opens to $\mathbb{w}$ with opening ρ.

If we assume (2) we can define the simulation-extractable zkVM as the composition in Theorem 5 between Π and $\Pi^\star$. □

7.3 Signature-of-Knowledge with delayed message

Theorem 5 highlights that, in many scenarios, we can obtain simulation extractability even when one of the components of the composed argument is malleable. However, the caveat is that we need to require that the second component is not only simulation extractable but also a signature of knowledge. It is rather easy to instantiate a signature of knowledge from a FS-based simulation extractable argument of knowledge, by including the message to the hashed view. However, there is an efficiency bottleneck in doing so in our compilers from Theorem 5. In fact, for example in the third statement, the message contain the proofs π_A, which enforce a sequentiality in the proofs' generation by the prover, namely π_A needs to be generated before π_B.

To mitigate such a bottleneck, we describe a notion of signature of knowledge where, roughly speaking, the message can be fed at the very end of the prover's computations. We refer to this as a *signature of knowledge with delayed message*. Informally, the prover's algorithm $\mathcal{P}$ can be divided into two procedures $\mathcal{P}_1$ and $\mathcal{P}_2$: the first procedure $\mathcal{P}_1$ takes as input the instance and witness (thus it is independent of the message), while $\mathcal{P}_2$ receives the internal state of $\mathcal{P}_1$ and the message, namely $\mathcal{P}(\mathrm{pp}, \mathsf{msg}, \mathbb{x}, \mathbb{w}) = \mathcal{P}_2(\mathsf{msg}, \mathcal{P}_1(\mathrm{pp}, \mathbb{x}, \mathbb{w}))$. The efficiency property we are interested in is that non-trivially $t(\mathcal{P}_2) < t(\mathcal{P}_1)$ where, very roughly speaking, $t(A)$ is the computational complexity of the algorithm A.

Fiat-Shamir-based Approach. We show that in Fiat-Shamir-based signature-of-knowledge the message does not need to be hashed until the round k where k-zero-knowledge and k-unique-response hold. This might enable for delayed message when the index k is the last (or more generally, when all the commitments have been computed and sent).

Theorem 7. *Let Π be a $(2r+1)$-message public-coin interactive argument. Let $\Pi_{\mathsf{FS}^*,k}$ be the Fiat-Shamir transform where the k-th challenge is derived as $\mathsf{H}(\mathrm{pp}, \mathsf{msg}, \mathbb{x}, \pi|_k)$ for an input message msg. If there is $k \in [r]$ such that $\Pi_{\mathsf{FS}^*,k}$ satisfies knowledge-soundness, k-zero-knowledge and k-unique response, then $\Pi_{\mathsf{FS}^*,k}$ is a signature of knowledge.*

Proof (Sketch.). The proof proceeds exactly as Theorem 3.4 in [16]. In particular we can define a knowledge-sound adversary $\mathcal{B}$ for $\Pi_{\mathsf{FS}^*,k}$ from the sim-ext adverary $\mathcal{A}$ for $\Pi_{\mathsf{FS}^*,k}$ by internally program the random oracle only on the input defined by the k-th round when running the zero-knowledge simulator and reply all the other queries using the random oracle interface.

Eventually $\mathcal{A}$ outputs its forgery. Such a forgery is considered valid for $\mathcal{B}$ if the verifier does not need to query the random oracle at the input programmed by $\mathcal{B}$ when verifying the forgery. When such an event happens we say that $\tilde{\pi}$ contains a *critical* RO-query.

Let $(\tilde{\mathsf{msg}}, \tilde{\mathbb{x}}, \tilde{\pi})$ be the forgery of $\mathcal{A}$ and $(\tilde{\mathsf{msg}}, \tilde{\mathbb{x}}, \tilde{\pi}) \notin \mathcal{Q}_{\mathsf{sim}}$, we can proceed with a case analysis:

- If $(*, \tilde{\mathbb{x}}, *) \notin \mathcal{Q}_{\mathsf{sim}}$ then $\tilde{\pi}$ does not contain any critical queries and we can reduce directly to the knowledge soundness.
- Otherwise, if $\tilde{\pi}_{|k} = \pi_{|k}$ and $\widetilde{\mathsf{msg}} \neq \mathsf{msg}$ for simulated $(\mathsf{msg}, \tilde{\mathbb{x}}, \pi)$, then we can break k-UR since $\mathsf{H}(\widetilde{\mathsf{msg}}, \pi_{|k}) \neq \mathsf{H}(\mathsf{msg}, \pi_{|k})$ with overwhelming probability.
- Finally if $(\widetilde{\mathsf{msg}}, \tilde{\mathbb{x}}, *) \in \mathcal{Q}_{\mathsf{sim}}$ but $\tilde{\pi}_{|k} \neq \pi_{|k}$ then $\tilde{\pi}$ does not contain any critical queries and we can reduce directly to the knowledge soundness.

□

Black-box approach. A second, black-box approach is based on a technique (which we believe to be folklore) relying on one-time signature.

Definition 18. *We say that $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is a one-time signature iff:*

- **Syntax.** *The three algorithms are PPT where $\mathsf{KGen}(1^\lambda)$ returns a pair pk, sk of public and secret keys, $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ with $\mathsf{msg} \in \{0,1\}^\lambda$ returns a signature σ_{msg}, and $\mathsf{Vf}(\mathsf{pk}, \mathsf{msg}, \sigma)$ returns a decision bit.*
- **Correctness.** *For any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KGen}(1^\lambda)$ and any $\mathsf{msg} \in \{0,1\}^\lambda$ we have $\mathsf{Vf}(\mathsf{pk}, \mathsf{msg}, \mathsf{Sign}(ss, \mathsf{msg})) = 1$.*
- **One-time unforgeability.** *For any PT adversary $\mathcal{A}$ that upon input the public key and an adaptively chosen message msg (and a signature σ for it) outputs $(\widetilde{\mathsf{msg}}, \tilde{\sigma})$, with $(\widetilde{\mathsf{msg}}, \tilde{\sigma}) \neq (\mathsf{msg}, \sigma)$:*

$$\Pr\big[\mathsf{Vf}(\mathsf{pk}, \mathcal{A}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, \mathsf{msg}))) = 1 : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)\big] \in \mathsf{negl}(\lambda)$$

Let Π be a signature of knowledge for $\mathcal{R}$ and Σ a one-time signature, consider the protocol $\Pi' \coloneqq (\mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ where:

- $\mathcal{P}(\mathrm{pp}, \mathsf{msg}, \mathbb{x}, \mathbb{w})$ samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$, computes $\pi \leftarrow \Pi.\mathcal{P}(\mathrm{pp}, \mathsf{pk}, \mathbb{x}, \mathbb{w})$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg} \| \mathbb{x})$, returns $\pi' \coloneqq (\pi, \mathsf{pk}, \sigma)$.
- $\mathcal{V}(\mathrm{pp}, \mathsf{msg}, \mathbb{x}, \pi')$ returns $\Pi.\mathcal{V}(\mathsf{pk}, \mathbb{x}, \pi)$ and $\mathsf{Vf}(\mathrm{pp}, \mathsf{pk}, \mathbb{x}, \sigma)$.

Theorem 8. *If Π is a signature-of-knowledge for $\mathcal{R}$ and Σ is a one-time signature then Π' is a signature-of-knowledge for $\mathcal{R}$.*

Proof (sketch). The event that there exist two simulation queries that have the same public key for the one-time signature scheme is negligible, as otherwise we can break one-time unforgeability. Let $(\widetilde{\mathsf{msg}}, \tilde{\mathbb{x}}, (\widetilde{\mathsf{pk}}, \tilde{\pi}, \tilde{\sigma}))$ be the forgery of the adversary. If $\widetilde{\mathsf{pk}}$ is not fresh, i.e., there exists a simulated proof $(\mathsf{pk}, \pi, \sigma)$ such that $\mathsf{pk} = \widetilde{\mathsf{pk}}$, then it must be $(\widetilde{\mathsf{msg}}, \tilde{\sigma}) = (\mathsf{msg}, \sigma)$, as otherwise we break one-time unforgeability, but then $(\tilde{\mathbb{x}}, \tilde{\pi}) \neq (\mathbb{x}, \pi)$ which implies that $(\widetilde{\mathsf{msg}}, \tilde{\mathbb{x}}, \tilde{\pi})$ is a valid forgery for the inner-scheme Π. On the other hand, when $\widetilde{\mathsf{pk}}$ is fresh, $(\widetilde{\mathsf{pk}}, \tilde{\mathbb{x}}, \tilde{\pi})$ is a valid forgery for Π independently of the signed message $\widetilde{\mathsf{msg}}$.

7.4 The Lookup-Singularity is Non-Malleable (or, Joltish is SIM-EXT)

As already mentioned in this section, we can realize an argument of knowledge for $\mathcal{R}_{\mathsf{Execute}}$ using a lookup argument. The basic idea is to consider the truth table of the instruction set $\mathsf{Execute}$ as the table, and the execution trace $\mathbb{w}_{\mathsf{Execute}}$ as the subvector. Although the truth table of the instruction set $\mathsf{Execute}$ is exponentially large, [2] shows that the truth table for the instruction set of RISC-V is decomposable (Definition 11).

Below we use the concept that an instruction set is decomposable if it can be described by a decomposable table (Definition 11), that is

Theorem 9. *If $\mathsf{Execute}$ is a decomposable instruction set, then zkLasso (see Section 5) implies a simulation-extractable argument of knowledge and a signature of knowledge with delayed message for $\mathcal{R}_{\mathsf{Execute}}$.*

Proof. Fixed $\mathsf{Execute}$, we can define the argument system for $\mathcal{R}_{\mathsf{Execute}}$ that runs the prover and verifier of zkLasso with parameter a table E that encodes the truth table of $\mathsf{Execute}$. Namely, E is the table such that $E[\mathsf{sregs}] = \mathsf{Execute}(\mathsf{sregs})$ for any $\mathsf{sregs} \in \{0,1\}^{3 \cdot \gamma}$. Recall that the truth table of $\mathsf{Execute}$, namely E, is decomposable. To prove $\mathcal{R}_{\mathsf{Execute}}(\mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{regs}})$ we prove that $\mathcal{R}_{\mathsf{lookup}}(E, \mathbb{w}_{\mathsf{sregs}}, \mathbb{w}_{\mathsf{regs}})$ where $\mathbb{w}_{\mathsf{sregs}}$ defines the committed indexes and $\mathbb{w}_{\mathsf{regs}}$ the committed sub-table.

Additionally, we notice that Theorem 4 and Theorem 7 imply we can apply the FS-transform to zkLasso to create a signature-of-knowledge with delayed messages and thus a signature-of-knowledge with delayed messages for $\mathcal{R}_{\mathsf{Execute}}$.

Definition 19 (Joltish). *Let* Execute *be a decomposable instruction set and let $\Pi^\star$ be an argument of knowledge for $\mathcal{R}^\star$. We call Joltish (instantiated with $\Pi^\star$) the argument for $\mathcal{R}_{\mathsf{zkVM}}^{\mathsf{Execute}}$ derived from the one of the compilers for conjunction in Fig. 17 and Theorem 6 where Π_{Execute} for $\mathcal{R}_{\mathsf{Execute}}$ is zkLasso.*

An efficient SIM-EXT zkVM for RISC-V Following [2], and as a corollary of Theorems 4 and 6, we have the following:

Corollary 2. *Let* Execute *be a decomposable instruction set, then there exists $\Pi^\star$ as in Definition 19 s.t. Joltish instantiated with $\Pi^\star$ is simulation-extractable zkVMs for $\mathcal{R}_{\mathsf{zkVM}}$ yielded by* Execute*.*

To argue that Jolt, or more precisely its zero-knowledge version, is simulation-extractable, it remains to show that the hypotheses of Theorem 6 hold for Jolt's implementation of the argument of knowledge for $\mathcal{R}^\star$.

In detail, in [2], Arun, Setty, and Thaler show how to realize a succinct argument of knowledge for $\mathcal{R}^\star$ using a commit-and-prove argument of knowledge for R1CS (they use Spartan [41]) and a commit-and-prove argument of knowledge for memory consistency based on the grand-product argument and memory checking techniques from [7].

More specifically, the latter parses $\mathbb{w}_{\mathsf{mem}}$ as a list of memory operations of the form (M, τ, o, l, v), where $M \in \{\mathsf{P}_{\mathsf{code}}, \mathsf{mem}\}$ indicates which of the memories[17] to read from or write to, τ is a timestamp, o is the operation (e.g., read or write), l is a location, and v is a value. The former proves that, assuming the memory accesses are consistent, the logic of the virtual machine is executed correctly; namely, the fetch and read-and-write operations (on the registers) are executed and iterated t times.

In Fig. 14, we show a zero-knowledge variant of the grand-product argument, which allows us to state that the sub-scheme for $\mathcal{R}^\star$ in Joltish is both knowledge-sound and zero-knowledge, thus enabling us to use the result from our theorem Theorem 5.

Acknowledgements

This work has received funding from the CHIST-ERA project PATTERN (ANR-23-CHR4-0008). The authors would like to thank Justin Thaler for innumerous clarifications about Jolt and Lasso and for general discussions about this work. We also thank Ben Livshits for insightful conversations about non-malleability and about this work in general. We thank Mahak Pancholi for providing feedback on early drafts on this work and for occasional oracle access on the topic of simulation-extractability. This work was partly carried out while some of the authors were at Matter Labs.

[17] The $\mathsf{P}_{\mathsf{code}}$ is a read-only memory, thus additional optimizations are available, while mem is a read-and-write memory.

References

1. Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: Enhanced compiling method for pedersen-committed zkSNARK engines. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 584–614. Springer, Cham, March 2022.
2. Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 3–33. Springer, Cham, May 2024.
3. Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 549–579, Virtual Event, August 2021. Springer, Cham.
4. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Berlin, Heidelberg, August 2013.
5. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116, 2016.
6. Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 393–414. Springer, Berlin, Heidelberg, March 2021.
7. Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *32nd FOCS*, pages 90–99. IEEE Computer Society Press, October 1991.
8. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Cham, December 2018.
9. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.
10. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Cham, December 2021.
11. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.
12. Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover SNARKs with constant size proofs and square root size universal setup. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *LATINCRYPT 2023*, volume 14168 of *LNCS*, pages 331–351. Springer, Cham, October 2023.
13. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
14. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.
15. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.
16. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 531–562. Springer, Cham, April 2023.
17. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001.
18. Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and MtGox. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 313–326. Springer, Cham, September 2014.

19. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Titouan Tanguy, and Michiel Verbauwhede. Efficient proof of RAM programs from any public-coin zero-knowledge system. Cryptology ePrint Archive, Report 2022/313, 2022.
20. Antonio Faonio, Dario Fiore, Markulf Kohlweiss, Luigi Russo, and Michal Zajac. From polynomial IOP and commitments to non-malleable zkSNARKs. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 455–485. Springer, Cham, November / December 2023.
21. Antonio Faonio, Dario Fiore, and Luigi Russo. Real-world universal zkSNARKs are non-malleable. Cryptology ePrint Archive, Report 2024/721, 2024.
22. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012.
23. Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.
24. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.
25. Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 735–760. Springer, Cham, September 2022.
26. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Cham, May / June 2022.
27. Ashrujit Ghoshal and Stefano Tessaro. Tight state-restoration soundness in the algebraic group model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Cham.
28. Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo – a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive, Report 2021/1063, 2021.
29. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.
30. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.
31. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017.
32. Markulf Kohlweiss, Mahak Pancholi, and Akira Takahashi. How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 486–512. Springer, Cham, November / December 2023.
33. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022.
34. Polygon Labs. Polygon Miden. `https://github.com/0xPolygonMiden`.
35. Tianyi Liu, Zhenfei Zhang, Yuncong Zhang, Wenqing Hu, and Ye Zhang. Ceno: Non-uniform, segment and parallel zero-knowledge virtual machine. Cryptology ePrint Archive, Paper 2024/387, 2024.
36. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.
37. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.
38. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Cham.
39. Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. Cryptology ePrint Archive, Paper 2024/1208, 2024.
40. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.
41. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020.

42. Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.
43. Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 180–209. Springer, Cham, May 2024.
44. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 71–89. Springer, Berlin, Heidelberg, August 2013.
45. Justin Thaler. FAQ on Jolt's initial implementation. https://a16zcrypto.com/posts/article/faqs-on-jolts-initial-implementation, April 2022.
46. Justin Thaler. Accelerating the World Computer. https://a16zcrypto.com/posts/article/accelerating-the-world-computer-implementing-jolt/, April 2024.
47. Justin Thaler. A New Era in SNARK Design. https://a16zcrypto.com/posts/article/a-new-era-in-snark-design-releasing-jolt/, April 2024.
48. Justin Thaler. Understanding Jolt: Clarifications and reflections. https://a16zcrypto.com/posts/article/understanding-jolt-clarifications-and-reflections, June 2024.
49. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2017/1132, 2017.
50. Hoeteck Wee. Zero knowledge in the random oracle model, revisited. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 417–434. Springer, Berlin, Heidelberg, December 2009.
51. Barry Whitehat. Lookup singularity. https://zkresear.ch/t/lookup-singularity/65/7, December 2022.
52. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022.
53. Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565, 2022.
54. Risc Zero. Universal zero knowledge. https://risczero.com/.
55. Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018.
56. Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3):375–403, 1990. Computational algebraic complexity editorial.