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.

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]. Many SNARKs that are currently being deployed in practice follow this design pattern. Examples include the Cairo-VM [28], the RISC Zero project [50], Scroll's Ceno [35], Polygon Miden [34] and many others. Among these schemes, a notable example is Jolt [2], a SNARK for VMs that is based on the *lookup-singularity* approach [47], 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, easy to extend and to audit. It is also leads to extremely fast provers.

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. [18], 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 [14] and to prevent basic types of attacks on transaction blockchains (e.g. double spending).

A recent line of works [17,20,21,25,26,32] has shown simulation extractability of several zkSNARKs like Bulletproofs [9], Spartan [41], Sonic [36], PLONK [24], Marlin [16], 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:

[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).

(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 attractive efficiency features. This makes Jolt/Lasso a likely adoption in different settings in the near future. 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[4] 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 1) 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[5]. 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 [12], i.e. SNARKs that are

[4] 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 6.2.

[5] 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).

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 [1,6,10,12]; 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 [12], 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 5). 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, which we call Joltish, is simulation-extractable.

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 [12] 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. We summarize our results on generic composition of commit-and-prove SNARKs in the following informal version of Theorem 3.

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 [11] 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 [17] and (2) we give a black-box construction of signature-of-knowledge with delayed message from (classical) signature-of-knowledge and one-time signatures. For space reasons, we elaborate on this only in [11].

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 [17], 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 [17] 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;[6] *(ii)* prove that all the inner (sub)protocols are *computational*[7] 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

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

[7] If the extractor fails to extract a witness, then we argue that the malicious prover is able to break some computationally-hard problem.

(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 [17] and requires to develop some more machinery.

More in detail, we follow [17] 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 [17] when analyzing the computational special soundness.

A second component of both Lasso and Spartan is (yet another) polynomial commitment called `HyraxPC` [45]. We improve the analysis for `HyraxPC`. Specifically, digging into the technical details of [17], 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 [17], 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. [11]) avoiding a loss quadratic in the number of the prover's queries and by solving a problem left open in the previous work.

From zkLasso to Joltish We provide a model for arguments of knowledge for virtual machine execution. While similar formalizations exist in the literature [4,8,19,51], 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 5, 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 5 and the associated Theorem 3). In particular, we can leverage on a 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 4, 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 [13] 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 [15].

1.5 Related Work

Simulation extractability was first introduced by De Santis et al. [18], 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 [17,20,21,25,26,32] about the simulation extractability of notable zkSNARKs, such as Bulletproofs [9], Spartan [41], Sonic [36], PLONK [24], Marlin [16], 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 [8,51], 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 [17] 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 [17] 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.

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 all EPT adversary $\mathcal{A}$, there exists an EPT adversary $\mathcal{B}$, nearly as efficient as $\mathcal{A}$, such that:*

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

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

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: $\forall m, m'$, $\forall \mathrm{ck}$: $\{\mathbf{c} : (\mathbf{c}, \rho) \leftarrow \mathsf{Commit}(\mathrm{ck}, m)\}$ is indistinguishable from $\{\mathbf{c}' : (\mathbf{c}', o') \leftarrow \mathsf{Commit}(\mathrm{ck}, m')\}$

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 $\Pi \coloneqq (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ 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 propeties completeness, knowledge soundness and honest-verifier zero-knowledge.

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 [12]. We assume that there is only a single commitment

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

Fig. 1. 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.

(rather than an arbitrary number) and that this opens to the entirety of the witness (instead of allowing for uncommitted portions as in [12]).

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 $\Pi := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where: $\mathsf{Setup}(\mathrm{pp}_{\mathbb{G}}) \to \mathrm{pp}$ generates the public parameters $\mathcal{P}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \mathbb{w}) \to \pi$ generates a proof π $\mathcal{V}^{\mathsf{H}}(\mathrm{pp}, \mathbb{x}, \pi) \to b$ checks if a proof is valid or not and outputs a bit $b \in \{0, 1\}$ and H is a random oracle.[8] We consider non-interactive arguments that, besides the standard completeness, satisfy the following two properties.

(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:

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

[8] 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 $\mathsf{H}_i \colon \{0,1\}^* \to \mathcal{C}_i$ 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.

and the knowledge soundness games are defined in Fig. 1.

(Zero-Knowledge) There exists a PPT simulator $\mathcal{S}$ such that for $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(\mathsf{pp}_{\mathbb{G}})$ and any unbounded adversary $\mathcal{A}$:[9]

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

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* [46] 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:

- 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 [17] to support custom predicates for the verifier challenges. In particular, in the generalization of [17], 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 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 a predicate $\mathsf{IsAccepting}((\boldsymbol{\phi},\boldsymbol{n}),\mathsf{pp},\mathbb{x},(\pi,)\mathsf{T})$ to check whether T is an $(\boldsymbol{\phi},\boldsymbol{n})$-tree of accepting transcripts for pp and $\mathbb{x}$, and optionally π. We refer the reader to [11] for the formal definition.

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.

[9] 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 $(\mathsf{pp},\mathbb{x},\mathbb{w})\notin\mathcal{R}$.

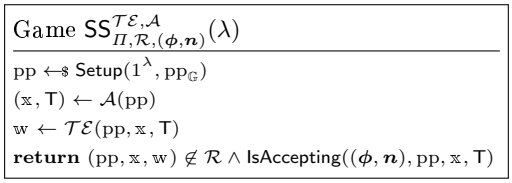

Fig. 2. Computational Special Soundness security game.

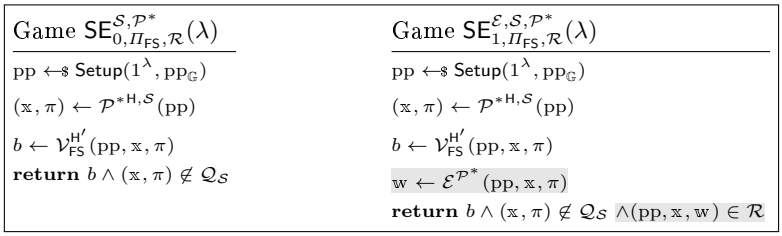

Fig. 3. Simulation extractability security games. $\mathcal{S}$ returns a proof π upon an input $\mathbb{x}$ (and may reprogram the random oracle), while $\mathcal{Q}_\mathcal{S}$ records all the pairs $(\mathbb{x}, \pi)$ queried by $\mathcal{P}^*$. H' denotes the modified RO after all the proof simulation queries. $\mathcal{E}$ is given black-box access to $\mathcal{P}^*$.

Definition 2 (Computational Special Soundness). *Let Π 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 Π 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:*

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

and the special soundness game is defined in Fig. 2

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 [17] to support partition predicates; in [11] 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 is a property that requires that extractability holds even when the malicious prover is given access to simulated proofs, possibly for *false* statements.

Definition 3 (Simulation extractability). *Let $\Pi := (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be a public coin zero-knowledge interactive argument for relation $\mathcal{R}$ with associated NIZK $\Pi_{\mathsf{FS}} := (\mathsf{Setup}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$. We say Π_{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:

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

and the security games are defined in Fig. 3.

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 [25,17].

Definition 4 (k-Zero-Knowledge, [17]). *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 5 (k-Unique Response, [17]). *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 $\mathcal{A}$:*

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

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

Theorem 1 ([17]). *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 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 [45].

3 Simulation extractability of Hyrax

HyraxPC is a commitment scheme, equipped with a $(\mu+1)$-rounds Eval protocol for a μ-variate multilinear polynomial that has been proved $(2^{\mu/2}, (4_{\pm})^{\mu/2}, 2)$ computational special sound in [17].[10] The protocol Eval is a public coin interactive argument for the relation:

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

In [17], 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 does not fulfill the formalism of Definition 2. 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, 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 HyraxPC is simulation-extractable. We refer the reader to [11] for the proofs.

4 Simulation extractability of Lasso

In this section, we show how we can apply Theorem 1 to prove that a zero-knowledge version of Lasso is simulation-extractable.

The starting point of Lasso is to model the lookup argument in a sparse way, as it is done in schemes like Caulk [48] or Baloo [49]: 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:

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

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.[11]

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

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

Definition 6 (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 the l.h.s. of Eq. (1) as:

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

and if t is decomposable we can further rewrite Eq. (2) as:

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

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 $\mathrm{dim}_j \colon \mathbb{F}^{\log m} \to \mathbb{F}$ be equal to $\widetilde{\mathrm{nz}}_j$. Moreover, for all $i \in [\alpha]$, let $E_i \colon \mathbb{F}^{\log m} \to \mathbb{F}$ 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. (1) simply as:

$$\sum_{k \in \{0,1\}^{\log m}} \widetilde{eq}(k, r) \cdot \hat{f}(E_1(k), \ldots, E_\alpha(k)) = \widetilde{a}(r) \tag{3}$$

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. (3) 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.

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

Interaction Phase.

1. $\mathcal{P}$ sends to $\mathcal{V}$ `HyraxPC` commitments to 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. $\mathcal{V}$ 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 $u(X) := \widetilde{eq}(r, X) \cdot \hat{f}(E_1(X), \ldots, E_\alpha(X))$: 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 $C_{v_1}, \ldots, C_{v_\alpha}$ to values $v_1, \ldots, v_\alpha$, supposedly equal to $E_1(r_z), \ldots, E_\alpha(r_z)$
6. $\mathcal{P}$ and $\mathcal{V}$ engage in $\texttt{GenPf}_{g,\alpha}$ to check that $\hat{f}(v_1, \ldots, v_\alpha) = e_x \widetilde{eq}(r, r_z)^{-1}$
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. 4. The indexed lookup argument zkLasso.

4.1 Zero-Knowledge Lasso

We define the main protocol in Fig. 4. It uses `Pedersen`, `HyraxPC`, three (2-perfect special sound) Σ-protocols 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 $\boldsymbol{a}$
- $\texttt{GenPf}_{\hat{f},n}$ 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:

- A protocol `SumCheck` to reduce the task of proving that $\sum_{x \in \{0,1\}^\mu} p(x)$ equals 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 (see [11]).
- A sum-check-based protocol `GrandProd` for "grand products" (see Fig. 5).

4.2 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 [17,45]). 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.

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 $\sum_{p\in\{0,1\}^i} g^{(i)}_{z_{i-1}}(p) = \widetilde{V}_i(z_{i-1})$ to the claim that $g^{(i)}_{z_{i-1}}(r_i) = e_i$, 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. 5. 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 $g^{(i)}_z(p) \coloneqq \widetilde{eq}(z, p) \cdot \widetilde{V}_{i+1}(p, 0) \cdot \widetilde{V}_{i+1}(p, 1)$

Below we analyze the special soundness of GrandProd (cf. Fig. 5).

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

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

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 `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 $f_{d-i}(r_{d-1}^{(j)}) = f_z(r_{d-1}^{(j)})$ for all j. Since there is a unique $(d-1)$-variate polynomial, of individual degree at most 3, that "densely" interpolates the pairs $(r_{d-1}^{(j)}, f_z(r_{d-1}^{(j)}))$, 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 $r \coloneqq r_{d-1}^{(j)}$ such that $f_{d-1}(r) \neq f_z(r)$. In particular, this implies that $w_{1,d-1}^{(j)} \neq \widetilde{V}_d(r, 0) \vee w_{2,d-1}^{(j)} \neq \widetilde{V}_d(r, 1)$. Let ℓ be the unique line interpolating $((r,0), w_{1,d-1}^{(j)}), ((r,1), w_{2,d-1}^{(j)})$; 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 $\beta_{d-1}^{(j)}, \beta_{d-1}^{'(j)}$ 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 3. *zkLasso satisfies $\boldsymbol{n}$-computational special soundness, where $\boldsymbol{n}$ is:*

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

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 $\widetilde{f}(v_1, \ldots, v_\alpha) = e_x / \widetilde{eq}(r, r_z)$
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. [11]) 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 theorem on zkLasso, whose proof is in [11].

Theorem 2. *zkLasso is simulation-extractable.*

5 Modular Composition of Sim-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.

5.1 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 7. *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.

Domain separation: H_1 and H_2 are two random oracles.

// Conjunction proofs $\Pi_\wedge$ and $\bar{\Pi}_\wedge$ (including 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 $\pi_A \leftarrow \Pi_A.\mathcal{P}^{\mathsf{H}_1}(\mathrm{pp}, (\mathbf{c}, \mathbb{x}_A), (\mathbb{w}, \rho))$,
3. prove $\pi_B \leftarrow \Pi_B.\mathcal{P}^{\mathsf{H}_2}(\mathrm{pp}, (\boxed{\mathsf{msg}}, M_c(\mathbf{c}), \mathbb{x}_B), (M(\mathbb{w}), M_\rho(\rho)))$

   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 $\Pi_{g\circ f}$ and $\bar{\Pi}_{g\circ f}$ (including 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 $\pi_F \leftarrow \Pi_f.\mathcal{P}^{\mathsf{H}_1}(\mathbf{c}_i, \mathbf{c}', (x_{f,i}, x_{F_o}), (\mathbb{w}_i, w_{g,i}), \rho_i, \rho')$,
4. prove $\pi_G \leftarrow \Pi_g.\mathcal{P}^{\mathsf{H}_2}(\boxed{\mathsf{msg}}, \mathbf{c}', \mathbf{c}_o, (x_{g,i}, x_{g,o}), (w_{g,i}, w_o), \rho', \rho_o)$

   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 $\Pi_f.\mathsf{Vf}^{\mathsf{H}_1}((\mathbf{c}_i, \mathbf{c}', x_{f,i}, x_{f,o}), \pi_1) = 1$ and $\Pi_g.\mathsf{Vf}^{\mathsf{H}_2}((\mathsf{msg}, \mathbf{c}', \mathbf{c}_o, x_{g,i}, x_{g,o}), \pi_2) = 1$.

Fig. 6. Compiler to proofs for conjunction (top) and function composition (bottom). For $X \in \{A, B, f, g\}$ Π_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 $\bar{\Pi}_\wedge$ (resp. $\bar{\Pi}_{g\circ f}$) we additionally assume that Π_A (resp. Π_g) is a signature of knowledge.

(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 8. *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. 6.

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 $((x_{f,o}, x_{g,o}), \mathbb{w}_o)$.

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. 6.

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

Definition 9. *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* $\mathcal{R}(\text{pp}, \mathbb{x}, \mathcal{M}(\text{pp}, \mathbb{x})) = 1$ *or* $(\text{pp}, \mathbb{x}) \notin \mathcal{L}_{\mathcal{R}}$. *We say that a relation $\mathcal{R}$ is always satisfiable if, for any* pp*, the language $\mathcal{L}_{\mathcal{R},\text{pp}} = \{0,1\}^*$, where the latter is the language associated to the relation for given parameters* pp.

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[12] 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 Π_A (resp. Π_f) in the compilers to witness indistinguishability (WI)[13]. Furthermore, for WI to hold, it is not necessary to reprogram the random oracle.

Definition 10. *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* $(\text{pp}, \mathbb{x}, \mathbb{w}_i) \in \mathcal{R}$ *the distributions* $\mathcal{P}^{\mathsf{H}}(\text{pp}, \mathbb{x}, \mathbb{w}_i)$ *for* $i \in \{1,2\}$ *are statically close.*

Theorem 3. *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 Π_A and Π_B are trapdoorless zero-knowledge and simulation extractable then $\Pi_\wedge$ for the relation $\mathcal{R}^M_{A\wedge B}$ is simulation-extractable.*
2. *If Π_f and Π_g are trapdoorless zero-knowledge and simulation extractable then $\Pi_{g\circ f}$ is simulation-extractable.*
3. *For any PT M,* CS *is M-malleable, and if Π_A is knowledge sound and statistically witness indistinguishable, $\mathcal{R}_A$ is always satisfiable and efficiently witness computable and Π_B is trapdoorless zero-knowledge and a signature-of-knowledge then Π' is simulation-extractable.*
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.*

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

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

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).

5.2 Discussion and Applications

We note that, if we disregard the aspects of commitment malleability (see Definition 8), 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 9) 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 6, we only require the compilers for conjunction.

6 Simulation extractability of zkVMs

6.1 Preliminaries on SNARK VMs

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

Definition 11 (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* $\mathsf{Execute} : \{0,1\}^{\gamma\cdot(k+4)} \to \{0,1\}^{\gamma\cdot k}$.

We want to describe the relation which describes a virtual machine execution. Consider the circuit in Fig. 7. This is parameterized by an *instruction set* 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 Execute, and simply

The virtual machine $\mathtt{VM}_{\mathsf{Execute}}(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{z})$:

Set $(\mathsf{sregs}, \mathsf{regs}) \leftarrow 0^{k+4}$, $\mathsf{mem} \leftarrow \mathbb{x} \| \mathbb{z}$.
Iterate 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. 7. The VM with parameters the instruction set 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 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.

write $\mathtt{VM}_{\mathsf{Execute}}$ whenever the parameters are clear from the context.) Following [2], we define the commit-and-prove relation:

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

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[14]. We define the CP relation below:

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

[14] 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

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 $(\mathbb{x}, \mathbb{z})$.

Definition 12. *The* zkVM-complementary relation *is the relation $\mathcal{R}^\star$ such that for any execution set* Execute*, and for any input $(\mathsf{P}_{\mathsf{code}}, \mathbb{x}, \mathbb{y}), \mathbb{z}$:*

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

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).

6.2 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 4. *For any instruction set* 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.*

We prove the theorem in [11]. Briefly, the theorem follows as an application of Theorem 3. 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.

6.3 The Lookup-Singularity is Non-Malleable

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

specialized versions of Spartan. Therefore, in spirit, our the instantiations we present in Section 6.3 are still applicable to the original presentation in [2].

table of the instruction set Execute as the table, and the execution trace $\mathbb{w}_{\mathsf{Execute}}$ as the subvector. Although the truth table of the instruction set Execute is exponentially large, [2] shows that the truth table for the instruction set of RISC-V is decomposable (Definition 6).

Below we use the concept that an instruction set is decomposable if it can be described by a decomposable table (Definition 6).

Theorem 5. *If* Execute *is a decomposable instruction set, then zkLasso (see Section 4) implies a simulation-extractable argument of knowledge and a signature of knowledge with delayed message for* $\mathcal{R}_{\mathsf{Execute}}$.

Proof. Fixed 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 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 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 2 and our theorem on the signature of knowledge [11] 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 13 (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 and Theorem 4 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 2 and 4, we have the following:

Corollary 1. *Let* Execute *be a decomposable instruction set, then there exists* $\Pi^\star$ *as in Definition 13 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 4 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[15] to read from or write to, τ is a timestamp, o is the operation (e.g., read or write),

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

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. 5, 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 3.

Acknowledgments. This work has received funding from the CHIST-ERA project PATTERN (ANR-23-CHR4-0008). The authors would like to thank Justin Thaler for general discussions about this work. We 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.

References

1. Aranha, D.F., Bennedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: Enhanced compiling method for pedersen-committed zk-SNARK engines. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part I. LNCS, vol. 13177, pp. 584–614. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97121-2_21
2. Arun, A., Setty, S.T.V., Thaler, J.: Jolt: SNARKs for virtual machines via lookups. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 3–33. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_1
3. Attema, T., Cramer, R., Kohl, L.: A compressed Σ-protocol theory for lattices. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 549–579. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_19
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_6
5. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116 (2016), https://eprint.iacr.org/2016/116
6. Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., Kolonelos, D.: Zero-knowledge proofs for set membership: Efficient, succinct, modular. In: Borisov, N., Díaz, C. (eds.) FC 2021, Part I. LNCS, vol. 12674, pp. 393–414. Springer, Berlin, Heidelberg (Mar 2021). https://doi.org/10.1007/978-3-662-64322-8_19
7. Blum, M., Evans, W.S., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: 32nd FOCS. pp. 90–99. IEEE Computer Society Press (Oct 1991). https://doi.org/10.1109/SFCS.1991.185352
8. Bootle, J., Cerulli, A., Groth, J., Jakobsen, S.K., Maller, M.: Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part I. LNCS, vol. 11272, pp. 595–626. Springer, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03326-2_20

9. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
10. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 3–33. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_1
11. Campanelli, M., Faonio, A., Russo, L.: SNARKs for virtual machines are non-malleable. Cryptology ePrint Archive, Paper 2024/1551 (2024), https://eprint.iacr.org/2024/1551
12. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2075–2092. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339820
13. Campanelli, M., Gailly, N., Gennaro, R., Jovanovic, P., Mihali, M., Thaler, J.: Testudo: Linear time prover SNARKs with constant size proofs and square root size universal setup. In: Aly, A., Tibouchi, M. (eds.) LATINCRYPT 2023. LNCS, vol. 14168, pp. 331–351. Springer, Cham (Oct 2023). https://doi.org/10.1007/978-3-031-44469-2_17
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
15. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: 30th ACM STOC. pp. 209–218. ACM Press (May 1998). https://doi.org/10.1145/276698.276741
16. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45721-1_26
17. Dao, Q., Grubbs, P.: Spartan and bulletproofs are simulation-extractable (for free!). In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 531–562. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30617-4_18
18. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Berlin, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_33
19. Delpech de Saint Guilhem, C., Orsini, E., Tanguy, T., Verbauwhede, M.: Efficient proof of RAM programs from any public-coin zero-knowledge system. Cryptology ePrint Archive, Report 2022/313 (2022), https://eprint.iacr.org/2022/313
20. Faonio, A., Fiore, D., Kohlweiss, M., Russo, L., Zajac, M.: From polynomial IOP and commitments to non-malleable zkSNARKs. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023, Part III. LNCS, vol. 14371, pp. 455–485. Springer, Cham (Nov / Dec 2023). https://doi.org/10.1007/978-3-031-48621-0_16
21. Faonio, A., Fiore, D., Russo, L.: Real-world universal zkSNARKs are non-malleable. Cryptology ePrint Archive, Report 2024/721 (2024), https://eprint.iacr.org/2024/721

22. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat-Shamir transform. In: Galbraith, S.D., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Berlin, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34931-7_5
23. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: 22nd ACM STOC. pp. 416–426. ACM Press (May 1990). https://doi.org/10.1145/100216.100272
24. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
25. Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., Zajac, M.: What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In: Galdi, C., Jarecki, S. (eds.) SCN 22. LNCS, vol. 13409, pp. 735–760. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-14791-3_32
26. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_14
27. Ghoshal, A., Tessaro, S.: Tight state-restoration soundness in the algebraic group model. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 64–93. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_3
28. Goldberg, L., Papini, S., Riabzev, M.: Cairo – a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive, Report 2021/1063 (2021), https://eprint.iacr.org/2021/1063
29. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC. pp. 291–304. ACM Press (May 1985). https://doi.org/10.1145/22145.22178
30. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
31. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
32. Kohlweiss, M., Pancholi, M., Takahashi, A.: How to compile polynomial IOP into simulation-extractable SNARKs: A modular approach. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023, Part III. LNCS, vol. 14371, pp. 486–512. Springer, Cham (Nov / Dec 2023). https://doi.org/10.1007/978-3-031-48621-0_17
33. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 359–388. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5_13
34. Labs, P.: Polygon Miden, https://github.com/0xPolygonMiden
35. Liu, T., Zhang, Z., Zhang, Y., Hu, W., Zhang, Y.: Ceno: Non-uniform, segment and parallel zero-knowledge virtual machine. Cryptology ePrint Archive, Paper 2024/387 (2024), https://eprint.iacr.org/2024/387

36. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339817
37. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS. pp. 436–453. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365746
38. Ràfols, C., Zapico, A.: An algebraic framework for universal and updatable SNARKs. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 774–804. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_27
39. Rosenberg, M., Mopuri, T., Hafezi, H., Miers, I., Mishra, P.: Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. Cryptology ePrint Archive, Paper 2024/1208 (2024), https://eprint.iacr.org/2024/1208
40. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814628
41. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_25
42. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020), https://eprint.iacr.org/2020/1275
43. Setty, S.T.V., Thaler, J., Wahby, R.S.: Unlocking the lookup singularity with Lasso. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 180–209. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_7
44. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 71–89. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_5
45. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2017/1132 (2017), https://eprint.iacr.org/2017/1132
46. Wee, H.: Zero knowledge in the random oracle model, revisited. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 417–434. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_25
47. Whitehat, B.: Lookup singularity (December 2022), https://zkresear.ch/t/lookup-singularity/65/7
48. Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: Lookup arguments in sublinear time. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 3121–3134. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560646
49. Zapico, A., Gabizon, A., Khovratovich, D., Maller, M., Ràfols, C.: Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565 (2022), https://eprint.iacr.org/2022/1565
50. Zero, R.: Universal zero knowledge, https://risczero.com/
51. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: Faster verifiable RAM with program-independent preprocessing. In: 2018 IEEE Symposium on Security and Privacy. pp. 908–925. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00013