On the Simulation-Extractability of Proof-Carrying Data

Behzad Abdolmaleki[1], Matteo Campanelli[2], Quang Dao[3], and Hamidreza Khoshakhlagh[4]

[1] University of Sheffield
behzad.abdolmaleki@sheffield.ac.uk
[2] Offchain Labs
binarywhalesinternaryseas@gmail.com
[3] Carnegie Mellon University
qvd@andrew.cmu.edu
[4] Aarhus University, Partisia
hamidreza@cs.au.dk

Abstract. With proof-carrying data (PCD), nodes in a distributed computation can certify its correct execution obtaining proofs with low-verification overhead (relative to the complexity of the computation). As PCD systems—and their special case, incrementally verifiable computation (IVC)—see rapid adoption in practice, understanding their robustness against malleability attacks becomes crucial. In particular, it remains unexplored whether recursive proof systems satisfy *simulation extractability* (SIM-EXT)—a property ensuring non-malleability and composability.
This work provides the first systematic study of simulation extractability for PCD. We begin by observing that the standard SIM-EXT notion for non-recursive zkSNARKs does not directly extend to PCD/IVC settings. To address this, we propose a new, tailored definition of SIM-EXT for proof-carrying data that accounts for their idiosyncratic features. Using this framework, we prove two general results: (1) that a simulation-extractable SNARK implies a simulation-extractable PCD when used recursively, and (2) that even lighter PCD constructions—built from a (not necessarily succinct) argument of knowledge (NARK) combined with a split-accumulation scheme—achieve SIM-EXT of PCD by requiring SIM-EXT only from the underlying NARK. Our results show that many modern PCD systems are already simulation-extractable by design.

1 Introduction

Proof-carrying data (PCD) [4, 24] has emerged as a powerful paradigm for the design of cryptographic proofs. PCD enables efficient verification of distributed computations among multiple distrusting parties. A special case of PCD, incrementally verifiable computation (IVC), represents one of the most promising solutions for scalability compared to *monolithic* proof systems[5]: they allow a prover to proceed incrementally and with very low memory footprint (see for example discussion in [47]). PCDs and IVCs are becoming more and more practical—thanks also to recent advances in accumulation- and folding-based recursion [9, 11, 47]— and are currently being deployed in real-world applications [5, 40, 43].

Stronger security properties in proof systems. In the common case in which proof schemes are deployed in larger protocols, the security guarantees we require on them go beyond the *bare minimum* ones, such as knowledge soundness and zero-knowledge. This includes *simulation extractability* (or SIM-EXT) introduced by De Santis et al. [26], 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 [16] and to prevent basic types of attacks on transaction blockchains (e.g. double spending). These attacks are not just hypothetical. The work of [27] observes that over three hundred thousand Bitcoins have been involved in malleability attacks. More specific to the PCD world, vulnerabilities related to malleability

[5] We call monolithic schemes where the prover needs access to the values of all the wires in a circuit C and performs a global computation over the entire trace.

have already been observed in prominent schemes, such as Nova [37, 45] (we remark that Nova is also already used in production environments, e.g. [40]).

Despite their wide applicability and the potential for real risks, to this day there exists no formal work on the non-malleability or simulation extractability of PCDs and IVCs. This stands in contrast to a recent line of works establishing SIM-EXT for several *non-recursive* zkSNARKs—such as Bulletproofs [6], Spartan [50], Sonic [41], PLONK [32], Marlin [22], Lunar [12], and Basilisk [49]—as well as to recent analyses of composable proof systems [13, 25, 29, 31, 33, 34, 36].

What should SIM-EXT require in the PCD setting? Another dimension in which the non-malleability of PCDs is understudied is from the point of view of *proper definitions*. In fact, the standard notion of SIM-EXT is not immediately applicable *as it is* to the PCD setting. In SIM-EXT, the adversary is given "additional powers" compared to knowledge soundness—the interaction with the simulator. This extra power jeopardizes the requirement of SIM-EXT (that is extracting a witness from such an adversary). Let us consider this example: let us imagine an IVC setting—where we are incrementally proving a chain of computations from an input z to an output z' (or $z \rightsquigarrow z'$)—and think of an adversary $\mathcal{A}$ asking the simulator for a proof that, in d steps we can proceed from input z_0 to output z by incrementally applying a function $F(\cdot,\cdot)$[6]. The local witnesses that allow going from z_0 to z may exist but they might be hard for a PPT machine to find (in other words we may not be able to extract them from the adversary). However, if they exist, the simulated proof will be valid. Imagine that, at this point, $\mathcal{A}$ proceeds honestly by incrementally proving that iterated applications of F to z lead to output z' *through honest witnesses* (witnesses which $\mathcal{A}$ actually *knows*). The adversary then returns the final proof to the challenger.

Will an extractor be able to produce *all* the witnesses for the target statement $z_0 \rightsquigarrow z'$? In general, it will plausibly not be able to. In our example, $\mathcal{A}$ could produce a proof for $z_0 \rightsquigarrow z$ only thanks to the simulator. Hence, while the extractor should be able to produce witnesses for the "segment" $z \rightsquigarrow z'$, it may not succeed in extracting the witnesses for the earlier segment (as the related proof was derived from the additional powers of the simulator). The standard notion of SIM-EXT, however, will require all such witnesses to be extracted. Standard SIM-EXT, therefore, is not a good fit for the PCD setting.

In view of this, it is natural to ask:

What does simulation extractability mean in the recursive setting, and under which conditions can it be preserved when proofs are composed across recursive layers?

1.1 Our Contributions

We provide the first study of simulation-extractability of PCD schemes. Our contributions include:

- **Modeling SIM-EXT for the PCD setting** We propose the first definition for SIM-EXT of PCD. We observe that the natural notion of SIM-EXT for PCD is not a direct translation of the one for zkSNARKs (as illustrated above). As a consequence, we provide a new definition that accommodates the idiosyncrasies of PCDs.
- **SIM-EXT SNARK ⇒ SIM-EXT PCD** We formalize the result that SIM-EXT zkSNARKs yield a SIM-EXT PCD when used recursively. Although not too surprising technically, the proof is involved due to the nature of PCDs.
- **KSND+Hid Split-Acc + SIM-EXT NARK ⇒ SIM-EXT PCD**. We then turn our attention to more lightweight template constructions for building PCDs that do not require succinct arguments of knowledge. These templates involve a split accumulation scheme and an argument of knowledge. While a naive proof may require simulation extractability of both underlying building blocks (split accumulation and NARK), we show that security goes through even if the accumulation scheme is "potentially malleable" (i.e., we do not require SIM-EXT of the split accumulation scheme). As a side contribution, we observe that the original definition of zero-knowledge in the paper can be

[6] Here the function is supposed to take a local non-deterministic witness w_i at every step. That is, for example, $z_1 = F(z_0, w_0), z_2 = F(z_1, w_1), \ldots$ and so on.

weakened for the PCD security proof (in particular, by removing the need for the simulator to explicitly reprogram the random oracle[7]).

1.2 Related Work

Simulation Extractability of zkSNARKs. Simulation-extractability (SIM-EXT) has been thoroughly explored for non-recursive proof systems. The notion, introduced by De Santis et al. [26], strengthens zero-knowledge proofs against adaptive and malleability attacks. Subsequent works established simulation-extractable arguments of knowledge for both pairing-based and transparent zkSNARKs [1, 2, 35]. Recent frameworks for polynomial IOP/PIOP-based systems—such as PLONK, Marlin, and Sonic—derive SIM-EXT via extractable commitment schemes, achieving non-malleability under composition [25, 29, 30, 33, 34, 36]. All these analyses, however, consider flat proofs where the statement excludes previous proofs. To our knowledge, no prior work formalizes SIM-EXT for recursive systems like PCD or Incremental Verifiable Computation (IVC), where proofs are nested and interdependent—requiring new definitions and analytical tools

Folding and Accumulation Schemes. Recursion in modern proof systems builds on two paradigms: accumulation and folding. The accumulation line, initiated by Bünz et al. [9] and extended by Arc [11], enables modular verification by aggregating constraints across recursive steps. Folding-based systems, starting with Nova [37] and refined through its successors [38, 46, 47, 53], achieve efficient recursion via algebraic compression of witness relations. These frameworks have made recursive SNARKs practical and widely deployed. Yet, while their algebraic soundness and efficiency are well understood, their behavior under simulation extractability—particularly when proofs compose recursively—remains unstudied. Our work addresses this gap by studying how folding and accumulation interact with extractability when proofs are nested across multiple recursive layers.

Security Analysis of PCD. Recent work has refined the quantitative security of PCD schemes derived from SNARKs with *straightline* extractors [21]. In particular, it shows that when the extractor is deterministic and non-adaptive, the PCD knowledge-soundness error equals that of the underlying SNARK, independent of recursion depth or graph size. Our setting aligns with this result: under analogous extractor assumptions, the SIM-EXT error of a recursive PCD remains strictly bounded by the SIM-EXT error of its base SNARK. Extending this equivalence to randomized or oracle-querying extractors remains an open direction. Other works exploring similar types of extractability for PCDs include [14, 39, 42]. The work in [15] explores the connections between depth and the security of IVC, a special case of PCD.

Recursive Proof Frameworks. Beyond individual constructions, recent advances in recursive proof systems have focused on efficiency and modularity rather than formal security composition. Frameworks based on accumulation [9, 11] and folding [38, 47, 52] demonstrate scalable recursion and are now widely deployed in practice. However, their security analyses primarily target algebraic soundness or knowledge soundness, leaving higher-level notions such as simulation extractability in recursive composition unaddressed. Our work complements these efforts by initiating a formal treatment of SIM-EXT in recursive proof settings like PCD and IVC.

1.3 Outline

We provide a technical overview in Section 2. In Section 3 we present basic preliminaries for non-interactive arguments of knowledge, accumulation schemes and PCDs. We provide and motivate our zero-knowledge definition for accumulation schemes on page 9. Formal treatment of simulation-extractability is deferred to Section 4. We present our new definition of SIM-EXT for PCD in Section 4.2. In Section 5 we prove the security of prominent PCD constructions.

[7] See discussion in Section 3.2.

2 Technical Overview

We now give an overview of our techniques. For simplicity, in it, we will focus on the special case of IVC (only one incoming node, with a path graph as the computation graph), and briefly note how our approach generalizes to the general case of PCD (incoming nodes of arbitrary arity) in the main body.

2.1 Templates for Constructing IVC/PCD and their Simulation Extractability

Recap: IVC. We first recall the general syntax of an IVC scheme. Given a compliance predicate Ψ on incoming data z_{in}, local witness w_{loc}, and outgoing data z_{out}, an IVC scheme $\mathsf{IVC} = (\mathbb{P}, \mathbb{V})$[8] consists of proving the Ψ-compliance of all data and local witnesses along the chain of computation. Specifically:

- $\mathbb{P}$ produces a proof π upon statement z_{out} and witness $(w_{\mathsf{loc}}, z_{\mathsf{in}})$; and
- $\mathbb{V}$ checks whether the proof π is valid with respect to statement z_{out}.

An IVC scheme is *knowledge sound* if, given a maliciously generated proof π for final output o, there is an extractor $\mathbb{E}$ that can output the entire history of the IVC computation leading to o; it is *zero knowledge* if there is a simulator $\mathbb{S}$ that can generate proofs π from just the statement z_{out}.

Defining Simulation Extractability for IVC. From the above syntax, we may try to define a $\mathsf{SIM\text{-}EXT}$ notion for IVC similar to the one defined for non-interactive arguments. The intuition for $\mathsf{SIM\text{-}EXT}$ is as follows: an adversary $\mathbb{A}$, even with access to a proof simulation oracle outputting valid proofs for arbitrary (even false) statements, must know a *witness* for any new accepting proof that it produces. In the case of IVC, the witness consists of all the previous intermediate values of the computation.

Hence, we have the following first attempt at the definition. In the $\mathsf{SIM\text{-}EXT}$ game for IVC, we have an adversarial prover $\mathbb{A}$ having oracle access to an oracle outputting simulated proofs on arbitrary statements (of the form (Ψ, z_{out})). Then there exists an extractor $\mathbb{E}$ such that whenever $\mathbb{A}$ returns an accepting proof π (with respect to some IVC predicate Ψ and statement z_{out}), then as long as π is not simulated, $\mathbb{E}$ can extract the entire history of the IVC computation.

However, this definition is too strong, since it cannot account for the following attack, which is only available in the recursive setting. In this attack, the adversary $\mathbb{A}$ queries for a simulated proof π, then *continues the computation* with that proof, producing a new step with a new honestly-generated valid proof π'. Since π' is new, the extractor will need to extract the entire history of the IVC computation, but it will get stuck when trying to recursively extract from π as it is simulated.

Such an attack is inevitable in general, and so we need to modify our $\mathsf{SIM\text{-}EXT}$ definition to cover it. The revised definition requires the extractor to produce a valid transcript that explains the adversary's claimed output, but allows the extractor to "stop" at simulated proofs encountered during recursive extraction.[9] Specifically, in the extracted transcript, any node associated with a simulated proof does not need to be further explained—the extractor is not required to provide witnesses for how that proof was derived. This captures the intuition that extraction should proceed as far as possible, stopping at either base cases of the IVC or at simulated proofs[10]

With the $\mathsf{SIM\text{-}EXT}$ definition now defined, we can look at how to prove particular IVC schemes satisfy the property. In fact, our approach is more general: since all known IVC (or PCD) schemes follow a small number of templates (in fact, just two),[11] we will first look at how proving $\mathsf{SIM\text{-}EXT}$ for IVC constructed using such a template may be reduced to proving properties of the individual components.

[8] In this exposition, we omit the role of the generator $\mathbb{G}$ and indexer $\mathbb{I}$.

[9] We assume that the extractor will extract both prior data and associated proofs; this is the case for all known PCD schemes.

[10] There exist other notions in the literature that are philosophically close to our definition of SIM-EXT for PCD, for example the definition of *controlled malleability* developed in the context of NIZKs [17].

[11] This is the case for schemes supporting an arbitrary NP predicate Ψ; for more restricted predicates in P there may be alternate approaches [28, 44, 48].

Warm-up: IVC from SNARKs. In the first template [4, 51], an IVC/PCD scheme is constructed from a succinct non-interactive argument (SNARK), whose proof size and verification are sublinear in the size of the instance and witness. Using a SNARK that can prove the validity of its own verifier, we can construct an IVC scheme by proving the following statement at every step:

> *"Predicate Ψ is satisfied at this step, and the SNARK verifier accepts the previous proof."*

In this template, an IVC proof is precisely the same as a SNARK proof, and the IVC prover/verifier invokes the SNARK prover/verifier on the same recursive relation above. Thus, we can also see that SIM-EXT of the IVC scheme should be implied by SIM-EXT of the underlying SNARK. To make this intuition precise, we give a sketch of how this is proved (the details can be found in Theorem 5.2):

- An IVC adversary $\mathbb{A}$ against SIM-EXT can be seen as a SNARK adversary $\mathcal{A}$, outputting the same proof.
- The extractor $\mathbb{E}$ for the IVC will be recursively defined, with subextractor $\mathbb{E}_i$ outputting a partial transcript of depth i for every i.
- In the first layer, the extractor $\mathbb{E}_0$ will simply be the SIM-EXT extractor $\mathcal{E}$ corresponding to the SNARK adversary $\mathcal{A}$. In each subsequent layer, we will define a new SNARK adversary $\mathcal{A}_i$, from which we obtain a corresponding extractor $\mathcal{E}_i$.
- Finally, we note that all these extractors must be for the SIM-EXT game (not just for knowledge soundness), since both the SNARK adversary and the extractor expect to see simulated proofs (which can only be provided if there is a proof simulation oracle).

IVC from (non-succinct) NARK and accumulation. In the second template [8, 10], an IVC/PCD scheme is constructed from a non-interactive argument (NARK) that *does not necessarily need to be succinct*, and an associated *accumulation scheme* that can reduce the task of running multiple NARK verifiers to running only one.

In more detail, an accumulation scheme allows a prover P to accumulate prior *predicate inputs* $[\mathsf{q}_i = (\mathsf{qx}_i, \mathsf{qw}_i)]_{i\in[n]}$ (supposed to satisfy some predicate Φ) and prior *accumulators* $[\mathsf{acc}_j = (\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j\in[m]}$, into a single new accumulator acc and a proof pf of correct accumulation. The verifier V now checks for correct accumulation, taking as input only the instance parts of the predicate inputs $[\mathsf{qx}_i]_{i\in[n]}$, the old accumulator inputs $[\mathsf{acc}_j.\mathbb{x}]_{j\in[m]}$, the new accumulator input $\mathsf{acc}.\mathbb{x}$, and the accumulation proof pf. Finally, there is a decider D that checks whether an accumulator acc is correct.

Using an accumulation scheme whose predicate is the NARK verifier, one can construct an IVC scheme as follows. The proof accompanying each step has the form $\mathbb{m} = (\pi, \mathsf{acc})$, where π is a NARK proof and $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ is an accumulator. In each step, we prove and verify the following relation:

> *"1) the predicate Ψ is satisfied at this step; 2) the new accumulator acc is accepted by the decider D; and 3) there exist prior predicate instances $[\mathsf{qx}_i]_{i\in[m]}$, accumulator instances $[\mathsf{acc}_j.\mathbb{x}]_{j\in[m]}$, and a proof pf that make the accumulation verifier V accept."*

In this part, the new accumulator acc is produced by the accumulation prover P and checked by the decider D, while the NARK prover $\mathcal{P}$ produces a proof π_{ARG} certifying the correctness of items 1) and 3). If both the NARK and the accumulation scheme are zero-knowledge, then the resulting IVC scheme is also zero-knowledge, where the IVC simulator $\mathbb{S}$ first simulates the new accumulator acc and then simulates the NARK proof π_{ARG} with public input being $(\mathsf{ivk}, z, \mathsf{acc}.\mathbb{x})$. See Figure 1 for an illustration.

Proving SIM-EXT for the accumulation template. With this template in mind, we now see how one could reduce proving SIM-EXT of the resulting IVC/PCD scheme to proving individual properties of the components. A first attempt might be to reduce directly to SIM-EXT of the NARK and of the accumulation scheme. This is not quite enough, due to the potential existence of "mix-and-match" attacks as follows. A PCD adversary may produce a statement-proof pair z, $\mathbb{m} = (\pi_{\mathsf{ARG}}, \mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ that is overall non-simulated, but one of its components is part of a prior simulation query. Since the simulation

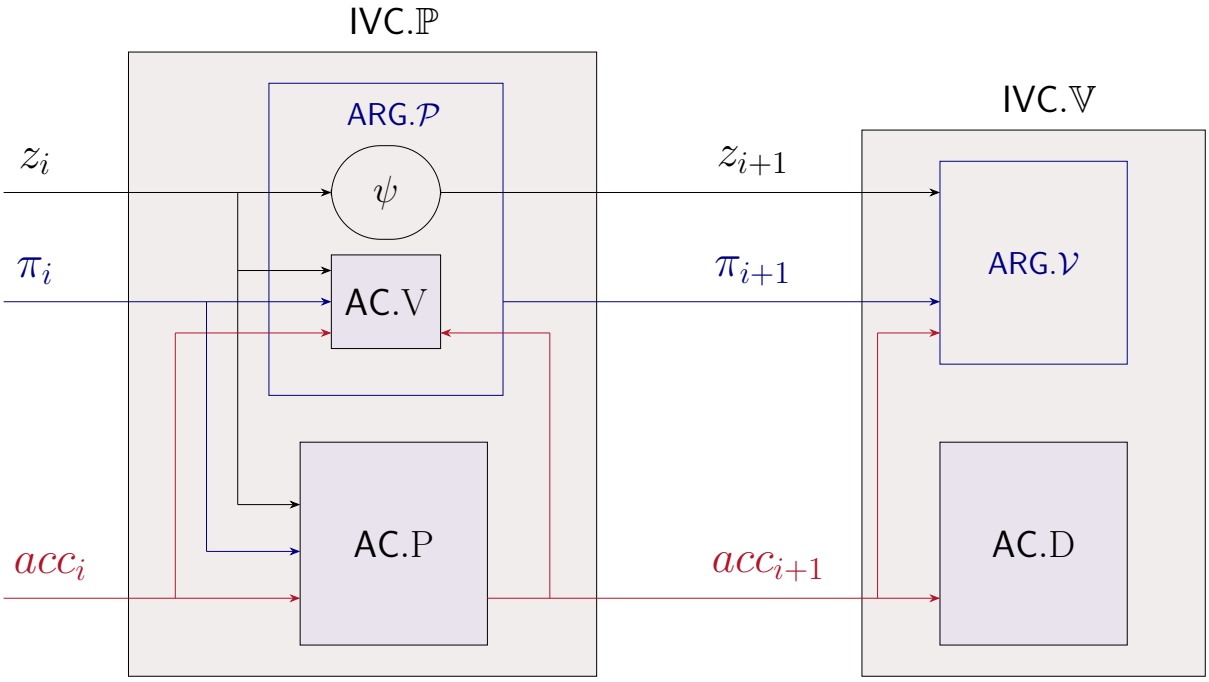

Fig. 1: IVC from NARK and accumulation schemes

of π_{ARG} relies on knowing z and $\mathsf{acc}.\mathbb{x}$, it follows that $\mathsf{acc}.\mathbb{x}$ is "bound" to π_{ARG}. Thus, the only potential problem is that $\mathsf{acc}.\mathbb{w}$ can be swapped.

In order to avoid this attack, we need an extra condition on the interaction between the NARK and the accumulation scheme. In known combinations, the instance part $\mathsf{acc}.\mathbb{x}$ is part of the NARK instance for the recursive relation (hence is bound to π_{ARG}), and $\mathsf{acc}.\mathbb{x}$ is also typically a hash of the witness part $\mathsf{acc}.\mathbb{w}$ (hence $\mathsf{acc}.\mathbb{w}$ is also bound to π_{ARG}). We call the second property *collision resistance* for the accumulation scheme.

From this intuition, we now sketch the main points of our reduction, going from SIM-EXT of the IVC to SIM-EXT of the NARK, plus knowledge soundness and collision resistance of the accumulation scheme.

3 Preliminaries

Oracle distributions. An oracle distribution $\mathcal{O} : \mathbb{N} \to \mathsf{Func}(X \to Y)$ is a distribution over functions $\mathsf{H} : X \to Y$. Here X is the domain and Y is the range. We denote a random oracle to be the uniform distribution over functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$.

Indexed oracle relations. An indexed relation $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance and $\mathbb{w}$ is the witness. The corresponding indexed language $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there is a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$.

Oracle algorithms. For a function $\theta : X \to Y$, we write A^θ for a (randomized) algorithm having oracle access to θ. We say that A is t-query if it makes at most t queries to θ (for any choice of its randomness).

Execution transcripts. For a t-query oracle algorithm A^θ, we denote by $\mathsf{tr} = [(q_i, a_i)]_{i=1}^t$ the transcript of query-answer pairs accessed by A. We denote by Trans_A the transcript of the full execution of A, which includes the collection of all inputs, outputs, random coins, and tr.

Adversaries. We model adversaries and extractors in this work as running in *(non-uniform) expected polynomial-time*. This means that they are modeled as oracle Turing machines receiving non-uniform polynomial-size advice, and has access to an infinite random tape, such that the average running time (over the choice of randomness for the random tape) is polynomial, for all possible advice.

Game $\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathsf{ARG},\mathcal{D}}(1^\lambda)$

$(\mathsf{pp},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda);\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp})$

$(\mathbb{i},\mathbb{x},\pi) \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{ai})$

$(\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp},\mathbb{i})$

$\mathbb{w} \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{Trans}_{\mathcal{A}})$

return $\left(\mathcal{V}(\mathsf{ivk},\mathbb{x},\pi) = 1 \wedge (\mathbb{i},\mathbb{x},\mathbb{w}) \notin \mathcal{R}_{\mathsf{pp}}\right)$

Fig. 2: Knowledge soundness for non-interactive arguments.

3.1 Non-Interactive Arguments of Knowledge

Definition 3.1 (Non-Interactive Arguments of Knowledge). *A (preprocessing) non-interactive argument of knowledge (NARK) is a tuple of PPT algorithms* $\mathsf{ARG} = (\mathcal{G},\mathcal{I},\mathcal{P},\mathcal{V})$ *with the following syntax:*

- $\mathcal{G}(1^\lambda) \to (\mathsf{pp},\mathsf{td})$. *On input the security parameter* 1^λ*, outputs public parameters* pp *and a trapdoor* td.
- $\mathcal{I}(\mathsf{pp},\mathbb{i}) \to (\mathsf{ipk},\mathsf{ivk})$. *On input the public parameters* pp *and index* $\mathbb{i}$*, deterministically outputs a proving key* ipk *and a verifying key* ivk.
- $\mathcal{P}(\mathsf{ipk},\mathbb{x},\mathbb{w}) \to \pi$. *On input the proving key* ipk*, instance* $\mathbb{x}$*, and witness* $\mathbb{w}$*, outputs a proof* π.
- $\mathcal{V}(\mathsf{ivk},\mathbb{x},\pi) \to b \in \{0,1\}$. *On input the verifying key* ivk*, instance* $\mathbb{x}$*, and proof* π*, outputs a bit* b *indicating accept/reject.*

We require ARG *to satisfy* completeness *and* knowledge soundness*, and define an optional notion of* zero knowledge.

- ***Completeness.*** *For every unbounded adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c|r} (\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}_{\mathsf{pp}} & (\mathsf{pp},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda) \\ \Downarrow & (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathcal{V}(\mathsf{ivk},\mathbb{x},\pi) = 1 & (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp},\mathbb{i}) \\ & \pi \leftarrow \mathcal{P}(\mathsf{ipk},\mathbb{x},\mathbb{w}) \end{array}\right] = 1.$$

- ***Knowledge Soundness.*** ARG *satisfies knowledge soundness with respect to an auxiliary input distribution* $\mathcal{D}$ *if for every expected polynomial-time adversary* $\mathcal{A}$*, there exists an expected polynomial-time extractor* $\mathcal{E}_{\mathcal{A}}$ *such that for every set* Z*,*

$$\mathbf{Adv}^{\mathsf{ARG\text{-}KS}}_{\mathcal{A},\mathcal{E}_{\mathcal{A}},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{ARG\text{-}KS}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}_{\mathsf{ARG},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

  The knowledge soundness security game is defined in Figure 2. We may define stronger notions of black-box *and* straightline *extraction by requiring the extractor* $\mathcal{E}_{\mathcal{A}}$ *to satisfy the corresponding property in Remark 3.2.*
- ***Zero Knowledge.*** ARG *satisfies* statistical *zero knowledge if there exists a PPT stateful simulator* $\mathcal{S}$ *with the following syntax:*
  - $\mathcal{S}(\mathsf{td},(\mathsf{pp},\mathbb{i},\mathbb{x})) \to \pi$ *returns a simulated proof* π *(using the trapdoor* td*) upon input* pp*, an index* $\mathbb{i}$*, and an instance* $\mathbb{x}$.

  such that for all unbounded distinguisher $\mathcal{A}$*, the following is negligible in* λ*:*

$$\mathbf{Adv}^{\mathsf{ARG\text{-}ZK}}_{\mathcal{A},\mathcal{S}}(\lambda) := \left|\Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},\mathcal{P}}_{0,\mathsf{ARG},\mathcal{R}}(1^\lambda)\right] - \Pr\left[\mathsf{ARG\text{-}ZK}^{\mathcal{A},\mathcal{S}}_{1,\mathsf{ARG},\mathcal{R}}(1^\lambda)\right]\right|.$$

  The zero-knowledge security games are defined in Figure 3.

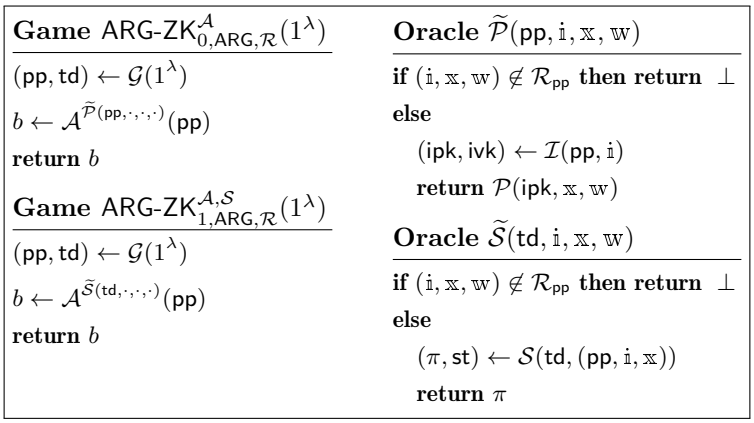

Fig. 3: Zero knowledge for non-interactive arguments.

Remark 3.2 (Notions of Extraction). Our definition of knowledge soundness follows [8], which is a multi-instance, non-black-box extraction definition. However, there are stronger variants of extraction that may lead to stronger security bounds:

- The default variant (given in Definition 3.1) is *non-black-box*, where $\mathcal{E}_{\mathcal{A}}$ may depend arbitrarily on the description of $\mathcal{A}$;
- A stronger variant is *black-box* extraction, e.g. there exists a universal extractor $\mathcal{E}$ such that $\mathcal{E}_{\mathcal{A}}$ consists of running $\mathcal{E}$ with black-box oracle access to $\mathcal{A}$. In particular, $\mathcal{E}$ may simulate any oracle that $\mathcal{A}$ has access to, and rewind $\mathcal{A}$ to any previous point in its execution with the same randomness;
- The strongest variant is *straightline* extraction, where the universal extractor $\mathcal{E}$ may only run $\mathcal{A}$ once without rewinding. This notion has been used in works on PCDs to show how they can achieve security with respect to compliance predicates of arbitrary (polynomial) depth [14, 21]. In some of our theorem statements we will invoke the fact that straightline extraction provides arbitrary depth (our proofs can be adapted in a straightforward manner and the technical details are by now folklore and well established).

Our definitions are written for non-black-box extraction; they can be modified to require stronger types of extraction by requiring the above properties.

Remark 3.3 (Succinctness). A NARK $\mathsf{ARG} = (\mathcal{G},\mathcal{I},\mathcal{P},\mathcal{V})$ is *succinct* (called a SNARK) if there exists a fixed polynomial p such that both the length of the proof and the running time of $\mathcal{V}$ are bounded by $p(\lambda,|\mathbb{x}|)$ and verifier time are sublinear

3.2 Accumulation Scheme

We recall the definition of a (split) accumulation scheme (which we simply call an accumulation scheme) as formalized in [8, 18].

Definition 3.4 (Accumulation Predicate). *An* accumulation predicate *is a tuple of PPT algorithms* $(\Phi,\mathcal{H})$ *with the following syntax:*

- $\mathcal{H}(1^\lambda) \to \mathsf{pp}_\Phi$. *On input the security parameter* 1^λ*, outputs predicate parameters* pp_Φ.
- $\Phi(\mathsf{pp}_\Phi,\mathsf{i}_\Phi,\mathsf{q}) \to b \in \{0,1\}$. *On input the predicate parameters* pp_Φ*, index* i_Φ*, and query* $\mathsf{q} = (\mathsf{qx},\mathsf{qw})$*, outputs a bit* b *indicating whether the predicate holds.*

Definition 3.5 (Admissible Aux. Input Distribution for Accumulation Schemes). *Let* $\mathcal{H}$ *be a predicate parameter sampler (see Definition 3.4). Let* $\mathcal{D}$ *be a PPT that syntactically outputs a pair* $(\mathsf{pp}'_\Phi,\mathsf{ai})$*. We say that* $\mathcal{D}$ *is admissible with respect to* $\mathcal{H}$ *if* $\{\mathsf{pp}_\Phi : \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda)\} \approx \{\mathsf{pp}'_\Phi : (\mathsf{pp}'_\Phi,\cdot) \leftarrow \mathcal{D}(1^\lambda)\}$.

Definition 3.6 (Accumulation Scheme). *An* accumulation scheme *for predicate* $(\Phi, \mathcal{H})$*, accepting* n *predicate inputs and* m *old accumulators, is a tuple of PPT algorithms* $\mathsf{AC} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *with the following syntax:*

- $\mathrm{G}(1^\lambda) \to \mathsf{pp}_{\mathsf{AC}}$*. On input the security parameter* 1^λ*, outputs public parameters* $\mathsf{pp}_{\mathsf{AC}}$*.*
- $\mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \to (\mathsf{apk}, \mathsf{avk}, \mathsf{dk})$*. On input the public parameters* $\mathsf{pp}_{\mathsf{AC}}$*, predicate parameters* pp_Φ*, and predicate index* i_Φ*, deterministically outputs a proving key* apk*, verifying key* avk*, and decision key* dk*.*
- $\mathrm{P}\left(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m\right) \to (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w}, \mathsf{pf})$*. On input the proving key* apk*, predicate inputs* $[(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n$*, and old accumulators* $[(\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})]_{j=1}^m$*, outputs a new accumulator* $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$ *and a proof* pf*.*
- $\mathrm{V}\left(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}\right) \to b \in \{0,1\}$*. On input the verifying key* avk*, predicate input instances* $[\mathsf{qx}_i]_{i=1}^n$*, old accumulator instances* $[\mathsf{acc}_j.\mathbb{x}]_{j=1}^m$*, new accumulator instance* $\mathsf{acc}.\mathbb{x}$*, and proof* pf*, outputs a bit* b *indicating accept/reject.*
- $\mathrm{D}(\mathsf{dk}, (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})) \to b' \in \{0,1\}$*. On input the decision key* dk *and new accumulator* $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$*, outputs a bit* b' *indicating accept/reject.*

We require AC *to satisfy* completeness *and* knowledge soundness*, and define an optional notion of* zero knowledge.

- ***Completeness.*** *For every (unbounded) adversary* A*,*

$$\Pr\left[\begin{array}{c} \mathrm{D}(\mathsf{dk}, \mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w}) = 1 \ \forall j \in [m] \\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{qx}_i, \mathsf{qw}_i) = 1 \ \forall i \in [n] \\ \Downarrow \\ \mathrm{V}\left(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}]_{j=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}\right) = 1 \\ \mathrm{D}(\mathsf{dk}, (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}(1^\lambda); \quad \mathsf{pp}_\Phi \leftarrow \mathcal{H}(1^\lambda) \\ \left(\mathsf{i}_\Phi, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m\right) \leftarrow \mathrm{A}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}\left(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m\right) \end{array}\right] = 1.$$

- ***Knowledge Soundness.*** *Let* AC *satisfies knowledge soundness with respect to an admissible auxiliary input distribution* $\mathcal{D}$ *w.r.t* $\mathcal{H}$ *if for every adversary* A *running in expected polynomial-time, there exists an extractor* E_A *running in expected polynomial-time such that for every set* Z*,*

$$\mathbf{Adv}^{\mathsf{AC\text{-}KS}}_{\mathrm{A},\mathrm{E}_\mathrm{A},\mathcal{D},Z}(\lambda) := \left| \begin{array}{l} \Pr\left[\mathsf{AC\text{-}KS}^{\mathrm{A}}_{0,\mathsf{AC},\mathcal{D},Z}(1^\lambda) = 1\right] \\ -\ \Pr\left[\mathsf{AC\text{-}KS}^{\mathrm{E}_\mathrm{A}}_{1,\mathsf{AC},\mathcal{D},Z}(1^\lambda) = 1\right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

  The knowledge soundness security game is defined in Figure 4. We define stronger notions of black-box and straightline extraction according to Remark 3.2.
- ***Zero Knowledge.*** AC *satisfies* computational *(resp.* statistical*) zero knowledge if there exists a PPT* S *with the following syntax:*
  - $\mathrm{S}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \to \mathsf{acc}$ *returns a new accumulator* acc *upon input* $\mathsf{pp}_{\mathsf{AC}}$*, predicate parameters* pp_Φ*, and index* i_Φ*;*

  such that for every expected poly-time (resp. unbounded) adversary A*, the following is negligible in* λ*:*

$$\mathbf{Adv}^{\mathsf{AC\text{-}ZK}}_{\mathrm{A},\mathrm{S}}(\lambda) := \left|\Pr\left[\mathsf{AC\text{-}ZK}^{\mathrm{A}}_{0,\mathsf{AC},\mathcal{R}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{AC\text{-}ZK}^{\mathrm{A},\mathrm{S}}_{1,\mathsf{AC},\mathcal{R}}(1^\lambda) = 1\right]\right|.$$

  The zero knowledge game is defined in Figure 5. In the game, both $\widetilde{P}$ *and* S' *implicitly gets access to the public parameters* $(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi)$*.*

Remark 3.7. We briefly mention how Definition 3.6 specializes to different variants in the literature.

Special case: accumulation scheme with no witnesses. In this case, the extractor does not need to extract anything, so knowledge soundness collapses into soundness.

Special case: folding schemes (folds one predicate input and one old accumulator, which are the same thing in this case, into a new accumulator)

Special case: accumulator for oracle queries, where V does not query the oracle

Game $\mathsf{AC\text{-}KS}^{\mathrm{A}}_{0,\mathsf{AC},\mathcal{D},Z}(1^\lambda)$

$\mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}(1^\lambda); \quad (\mathsf{pp}_\Phi, \mathsf{ai}) \leftarrow \mathcal{D}(\mathsf{pp}_{\mathsf{AC}})$

$$\left(\left[\mathsf{i}_\Phi^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{acc}^{(k)}, \mathsf{pf}^{(k)}\right]_{k=1}^{\ell}, \mathsf{ao}\right) \leftarrow \mathrm{A}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{ai})$$

$\forall k, (\mathsf{apk}^{(k)}, \mathsf{avk}^{(k)}, \mathsf{dk}^{(k)}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)})$

$$\textbf{return} \left(\begin{array}{c} \left(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{ai}, \left[\mathsf{i}_\Phi^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{acc}^{(k)}\right], \mathsf{ao}\right) \in Z \\ \wedge \\ \left\{ \begin{array}{c} \mathrm{V}\left(\mathsf{avk}^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{acc}.\mathbb{x}^{(k)}, \mathsf{pf}^{(k)}\right) = 1 \\ \mathrm{D}(\mathsf{dk}^{(k)}, \mathsf{acc}^{(k)}) = 1 \end{array} \right\}_{k=1}^{\ell} \end{array} \right)$$

Game $\mathsf{AC\text{-}KS}^{\mathrm{E_A}}_{1,\mathsf{AC},\mathcal{D},Z}(1^\lambda)$

$\mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}(1^\lambda); \quad (\mathsf{pp}_\Phi, \mathsf{ai}) \leftarrow \mathcal{D}(\mathsf{pp}_{\mathsf{AC}})$

$$\left(\left[\mathsf{i}_\Phi^{(k)}, [(\mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)})]_{i=1}^n, [\mathsf{acc}_j^{(k)}]_{j=1}^m, \mathsf{acc}^{(k)}\right]_{k=1}^{\ell}, \mathsf{ao}\right) \leftarrow \mathrm{E_A}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{ai})$$

$\forall k, (\mathsf{apk}^{(k)}, \mathsf{avk}^{(k)}, \mathsf{dk}^{(k)}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)})$

$$\textbf{return} \left(\begin{array}{c} \left(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi, \mathsf{ai}, \left[\mathsf{i}_\Phi^{(k)}, [\mathsf{qx}_i^{(k)}]_{i=1}^n, [\mathsf{acc}_j.\mathbb{x}^{(k)}]_{j=1}^m, \mathsf{acc}^{(k)}\right], \mathsf{ao}\right) \in Z \\ \wedge \\ \left\{ \begin{array}{c} \mathrm{D}(\mathsf{dk}^{(k)}, \mathsf{acc}_j^{(k)}) = 1 \ \forall j \in [m] \\ \Phi(\mathsf{pp}_\Phi, \mathsf{i}_\Phi^{(k)}, \mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)}) = 1 \ \forall i \in [n] \end{array} \right\}_{k=1}^{\ell} \end{array} \right)$$

Fig. 4: Knowledge soundness for accumulation schemes.

On Zero-Knowledge Definition of Accumulation Schemes. In our definition of accumulation schemes (Definition 3.6), we use a minor variant of the notion of zero-knowledge in papers that proposed accumulation schemes ([10] and [8]). The main difference lies in the fact that prior definitions give special powers to the simulator, in particular reprogramming the random oracle. We observe that this modeling choice in prior works is overly restrictive. In fact, all the constructions in those same papers actually do not require reprogramming of the random-oracle or any trapdoor of sort. This allowed us to provide the definition in Definition 3.6 where the simulator is essentially a *sampler* that can be run locally by any PPT on input the parameters of the scheme. Part of the reason for why this is possible may have to do with the condition for zero-knowledge in accumulation, which requires that only the *accumulators themselves* (rather than the *proofs*) are indistinguishable.

Below we discuss why the constructions in [10] and [8] already satisfy our notion given the simulation strategy already present in the security proofs in the respective papers. In general all the simulation strategies of these constructions involve sampling a random object (a vector or a polynomial) and whenever the simulator needs to provide a proof, this is (crucially) a proof on some *honest* statement related to that random object. Therefore the simulation strategy requires no "special powers".

◇ **Accumulation schemes (non-split) [10]**:

- Accumulation of polynomial commitments based on DLOG (Theorem 7.1 in [9]): Here the simulator samples a random polynomial and provides an opening proof. Although the proving of the polynomial evaluation is described with an invocation of the polynomial commitment simulator, we notice that this could be carried out by the honest polynomial commitment prover. The arguments presented in the proof in [8] would go through in the same way.

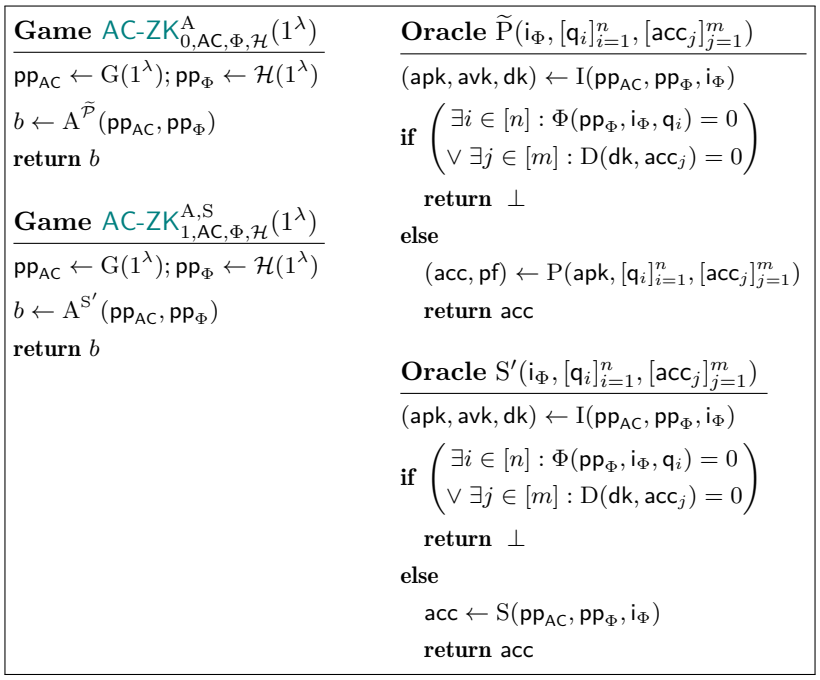

Fig. 5: Zero knowledge for accumulation schemes.

- Accumulation of polynomial commitments in the bilinear setting(Section 8 in [9]): the strategy for achieving zero-knowledge is not formalized but is described informally in the technical overview as close to the one from the previous item (a sampling of a random polynomial together with an honest evaluation proof related to it).
- Accumulation of verification in non-interactive arguments (Section 6 in [9]): the simulator here simply invokes the one for a "building block" accumulation scheme. This is instantiated with either of the two schemes above and the property we are interested in follows immediately.

◇ **Split-accumulation schemes [8]:**

- Split-accumulation of Hadamard products (Section 7 in [7]): the simulator already satisfies our notion (see simulator on page 45).
- Split-accumulation of (NARKs for) R1CS relations (Lemma 8.4 in [7]): the bulk of the simulator consists in sampling random vectors and invoking the simulator from the previous item.

We finally note that [7] presents an additional split-accumulation scheme (for "Pedersen polynomial commitments") whose zero-knowledge property is not described. This is not an issue for us since the two split-accumulation schemes described above are the only ones we need for our instantiations. In any event, this additional split-accumulation scheme follows similar techniques as the ones in the other constructions and probably admits a zero-knowledge variant satisfying our notion.

3.3 Proof-Carrying Data

We define PCD schemes with a stronger knowledge soundness guarantee than prior works, requiring that the extractor outputs prior proofs as well as prior data. This property is satisfied by all known PCD schemes.

Definition 3.8 (Proof-Carrying Transcript). *A* transcript T *is a directed acyclic graph where each vertex* $u \in V(\mathsf{T})$ *is labeled by local data* $w_{\mathsf{loc}}^{(u)} \in \{0,1\}^* \cup \{\perp\}$*, and each edge* $e \in E(\mathsf{T})$ *is labeled by a message* $z^{(e)} \neq \perp$*. The* output *of* T*, denoted* $\mathsf{o}(\mathsf{T})$*, is the message* $z^{(e)}$ *corresponding to the lexicographically first sink of* T*.*

A proof-carrying transcript PCT *has the same topology and format as* T*, except that each edge* $e \in E(\mathsf{PCT})$ *is further labeled with a proof* $\mathbb{m}^{(e)}$*.*

We denote by m the *fan-in* of PCT, i.e. the maximum number of incoming edges for any vertex $v \in V(\mathsf{PCT})$. We will consider classes of compliance predicates F where every $\Psi \in \mathsf{F}$ is of the form $\Psi : \{0,1\}^* \times (\{0,1\}^* \cup \{\bot\})^{m+1} \to \{0,1\}$, i.e. it takes in a tuple $(z, w_{\mathsf{loc}}, z_1, \ldots, z_{m'}, \bot, \ldots, \bot)$ for $m' \leq m$ and outputs accept/reject.

Definition 3.9 (Compliant Proof-Carrying Transcript). *Let* PCT *be a proof-carrying transcript. A vertex* $u \in V(\mathsf{PCT})$ *is* $(\Psi, \overline{\mathbb{V}}_{\mathbb{pp}})$-compliant *for a compliance predicate* $\Psi \in \mathsf{F}$ *and verifier circuit* $\overline{\mathbb{V}}_{\mathbb{pp}}$*, if for all outgoing edges* $e = (u, v) \in E(\mathsf{PCT})$*, if* $e_1, \ldots, e_{m'}$ *(with* $m' \leq m$*) are the incoming edges of* u*, then:*

i) $\Psi(z^{(e)}, z^{(u)}_{\mathsf{loc}}, z^{(e_1)}, \ldots, z^{(e_{m'})}, \bot, \ldots, \bot) = 1$,
ii) $\overline{\mathbb{V}}_{\mathbb{pp}}(\Psi, z^{(e)}, \mathbb{m}^{(e)}) = 1$.

We say that PCT *is* $(\Psi, \overline{\mathbb{V}}_{\mathbb{pp}})$-compliant *if all of its vertices are* $(\Psi, \overline{\mathbb{V}}_{\mathbb{pp}})$*-compliant.*

Definition 3.10 (PCD). *A* proof-carrying data *(PCD) scheme for a class of predicates* F *is a tuple of PPT algorithms* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *with the following syntax:*

- $\mathbb{G}(1^\lambda) \to (\mathbb{pp}, \mathbb{td})$*. On input security parameter* 1^λ*, outputs public parameters* $\mathbb{pp}$ *and a trapdoor* $\mathbb{td}$*.*
- $\mathbb{I}(\mathbb{pp}, \Psi) \to (\mathbb{ipk}, \mathbb{ivk})$*. On input the public parameters* $\mathbb{pp}$ *and a predicate* Ψ*, deterministically outputs a proving key* $\mathbb{ipk}$ *and a verifying key* $\mathbb{ivk}$*.*
- $\mathbb{P}(\mathbb{ipk}, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m) \to \mathbb{m}$*. On input the proving key* $\mathbb{ipk}$*, outgoing data* z*, local data* w_{loc}*, and incoming data-proof pairs* $[z_i, \mathbb{m}_i]_{i=1}^m$*, outputs an outgoing proof* $\mathbb{m}$*.*
- $\mathbb{V}(\mathbb{ivk}, z, \mathbb{m}) \to b \in \{0,1\}$*. On input the verifying key* $\mathbb{ivk}$*, outgoing data* z *and outgoing proof* $\mathbb{m}$*, outputs a bit* b *indicating accept/reject.*
- *We also define a* combined *verifier* $\overline{\mathbb{V}}_{\mathbb{pp}}(\Psi, z, \mathbb{m}) \to b \in \{0,1\}$*, which has* $\mathbb{pp}$ *hardwired and runs both* $\mathbb{I}(\mathbb{pp}, \Psi) \to (\mathbb{ipk}, \mathbb{ivk})$ *and* $\mathbb{V}(\mathbb{ivk}, z, \mathbb{m}) \to b$*.*

We require PCD *to satisfy* completeness *and* knowledge soundness*, and also define an optional notion of* zero knowledge*.*

- ***Completeness.*** *For every (unbounded) adversary* $\mathbb{A}$*,*

$$\Pr\left[\begin{array}{c} \Psi \in \mathsf{F} \\ \wedge\ \Psi(z, w_{\mathsf{loc}}, z_1, \ldots, z_m) = 1 \\ \wedge\ \mathbb{V}(\mathbb{ivk}, z_i, \mathbb{m}_i) = 1\ \forall\ i \in [m'] \\ \Downarrow \\ \mathbb{V}(\mathbb{ivk}, z, \mathbb{m}) = 1 \end{array} \middle| \begin{array}{r} (\mathbb{pp}, \mathbb{td}) \leftarrow \mathbb{G}(1^\lambda) \\ (\Psi, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m) \leftarrow \mathbb{A}(\mathbb{pp}) \\ (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \Psi) \\ \mathbb{m} \leftarrow \mathbb{P}(\mathbb{ipk}, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m) \end{array}\right] = 1.$$

- ***Knowledge Soundness.*** $\mathsf{PCD\text{-}KS}$ *has knowledge soundness with respect to auxiliary input distribution* $\mathcal{D}$ *if for any expected poly-time adversary* $\mathbb{A}$*, there exists an expected poly-time* $\mathbb{E}_{\mathbb{A}}$ *such that,*

$$\mathbf{Adv}^{\mathsf{PCD\text{-}KS}}_{\mathbb{A},\mathbb{E}_{\mathbb{A}},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{PCD\text{-}KS}^{\mathbb{A},\mathbb{E}_{\mathbb{A}}}_{\mathsf{PCD},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

  The knowledge soundness game is defined in Figure 6. We may define stronger notions of black-box and straightline extraction according to Remark 3.2.
- ***Zero Knowledge.*** PCD *satisfies* computational *(resp.* statistical*) zero knowledge if there exists a PPT stateful simulator* $\mathbb{S}$ *with the following syntax:*
  - $\mathbb{S}(\mathbb{td}, (\mathbb{pp}, \Psi, z)) \to \mathbb{m}$ *returns a proof* $\mathbb{m}$ *upon input* $\mathbb{pp}$*, predicate* Ψ*, and data* z*;*

  such that for every expected poly-time (resp. unbounded) adversary $\mathbb{A}$*, the following is negligible in* λ*:*

$$\mathbf{Adv}^{\mathsf{PCD\text{-}ZK}}_{\mathbb{A},\mathbb{S}}(\lambda) := \left|\Pr\left[\mathsf{PCD\text{-}ZK}^{\mathbb{A}}_{0,\mathsf{PCD},\mathsf{F}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{PCD\text{-}ZK}^{\mathbb{A},\mathbb{S}}_{1,\mathsf{PCD},\mathsf{F}}(1^\lambda) = 1\right]\right|.$$

  The zero knowledge game is defined in Figure 7.

Game $\mathsf{PCD\text{-}KS}^{\mathbb{A},\mathbb{E}_{\mathbb{A}}}_{\mathsf{PCD},\mathcal{D}}(1^\lambda)$

$(\mathbb{pp}, \mathbb{td}) \leftarrow \mathbb{G}(1^\lambda)$; $\mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp})$

$(\Psi, \mathsf{o}, \mathbb{m}) \leftarrow \mathbb{A}(\mathbb{pp}, \mathsf{ai})$

$(\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \Psi)$

$\mathsf{PCT} \leftarrow \mathbb{E}_{\mathbb{A}}(\mathsf{Trans}_{\mathbb{A}})$

return $\left(\begin{array}{c} \Psi \in \mathsf{F} \quad \wedge \\ \mathbb{V}(\mathbb{ivk}, \mathsf{o}, \mathbb{m}) = 1 \quad \wedge \\ (\mathsf{PCT} \text{ is not } (\Psi, \overline{\mathbb{V}}_{\mathbb{pp}})\text{-compliant} \vee \mathsf{o}(\mathsf{PCT}) \neq \mathsf{o}) \end{array}\right)$

Fig. 6: Knowledge soundness for proof-carrying data.

Game $\mathsf{PCD\text{-}ZK}^{\mathbb{A},\widetilde{\mathbb{P}}}_{0,\mathsf{PCD},\mathsf{F}}(1^\lambda)$

$(\mathbb{pp}, \mathbb{td}) \leftarrow \mathbb{G}(1^\lambda)$

$b \leftarrow \mathbb{A}^{\widetilde{\mathbb{P}}(\mathbb{pp},\cdot,\cdot,\cdot,\cdot)}(\mathbb{pp})$

return b

Game $\mathsf{PCD\text{-}ZK}^{\mathbb{A},\mathbb{S}}_{1,\mathsf{PCD},\mathsf{F}}(1^\lambda)$

$(\mathbb{pp}, \mathbb{td}) \leftarrow \mathbb{G}(1^\lambda)$

$b \leftarrow \mathbb{A}^{\widetilde{\mathbb{S}}(\mathbb{td},\cdot,\cdot,\cdot,\cdot)}(\mathbb{pp})$

return b

Oracle $\widetilde{\mathbb{P}}(\mathbb{pp}, \Psi, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m)$

if $(\Psi \notin \mathsf{F}) \ \vee \ \Psi(z, w_{\mathsf{loc}}, [z_i]_{i=1}^m) = 0$
$\vee \ \exists\, i \in [m], \ \mathbb{V}(\mathbb{ivk}, z_i, \mathbb{m}_i) = 0$
  return $\perp$
else
  $(\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \Psi)$
  return $\mathbb{P}(\mathbb{ipk}, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m)$

Oracle $\widetilde{\mathbb{S}}(\mathbb{td}, \Psi, z, w_{\mathsf{loc}}, [z_i, \mathbb{m}_i]_{i=1}^m)$

if $(\Psi \notin \mathsf{F}) \ \vee \ \Psi(z, w_{\mathsf{loc}}, [z_i]_{i=1}^m) = 0$
$\vee \ \exists\, i \in [m], \ \mathbb{V}(\mathbb{ivk}, z_i, \mathbb{m}_i) = 0$
  return $\perp$
else
  $(\mathbb{m}, \mathsf{st}) \leftarrow \mathbb{S}(\mathbb{td}, (\mathbb{pp}, \Psi, z))$
  return $\mathbb{m}$

Fig. 7: Zero knowledge for proof-carrying data.

As a special case, when T is the path graph then PCD reduces to incrementally verifiable computation (IVC) [51].

Remark 3.11 (Efficiency). For a standard PCD, we require that the proof size $|\mathbb{m}| = \mathsf{poly}(\lambda, |\Psi|)$ does not grow with each application of the PCD prover $\mathbb{P}$. However, we will also consider non-succinct PCD as in [20], where this requirement no longer holds. This relaxation was used in the same work [20] to provide heuristic concrete security bounds for PCD with straightline extraction.

4 Definitions of Simulation Extractability

4.1 Simulation Extractability for Non-Interactive Arguments

Definition 4.1. *Let* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *be a* NARK *for a relation family* $\mathcal{R}$*. Let* $\mathcal{S}$ *be a zero-knowledge simulator for* ARG*. We say that* ARG *satisfies* simulation extractability (ARG-SE) *if for every auxiliary input distribution* $\mathcal{D}$ *and every expected polynomial-time adversary* $\mathcal{A}$*, there exists an expected poly-time*

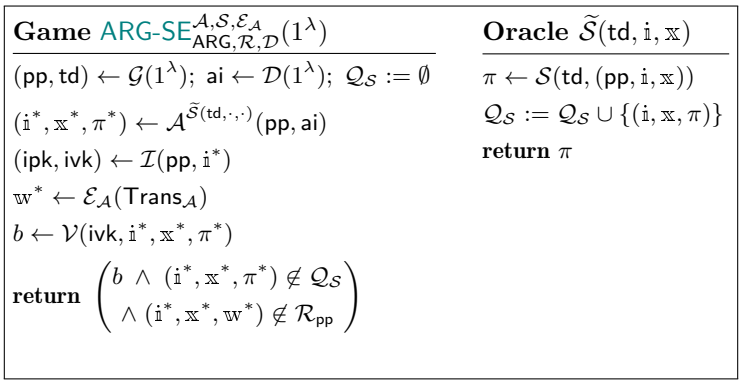

Fig. 8: Simulation extractability for non-interactive arguments.

extractor $\mathcal{E}_{\mathcal{A}}$ such that for every set Z:

$$\mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A},\mathcal{S},\mathcal{E}_{\mathcal{A}},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{ARG\text{-}SE}^{\mathcal{A},\mathcal{S},\mathcal{E}_{\mathcal{A}}}_{\mathsf{ARG},\mathcal{R},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

The simulation extractability game is defined in Figure 8. This game is identical to the knowledge soundness game (Figure 2) except that the adversary interacts with the simulation oracle. We may define stronger notions of black-box and straightline extraction according to Remark 3.2.

4.2 New Notion: Simulation Extractability for Proof-Carrying Data

Following our exposition in Section 5, we now define simulation extractability for proof-carrying data (PCD). An important subtlety is that, when recursively extracting, the PCD simulator-extractor may fail to extract from a simulated proof at a prior node. Since this is inevitable in general, we need to relax the success condition of the extractor.

Definition 4.2 (Relaxed Compliance for PCDSE). *Let* PCT *be a proof-carrying transcript, $\overline{\mathbb{V}}_{\mathbb{pp}}$ be the combined PCD verifier (see Definition 3.10), and $\mathcal{Q}_{\mathbb{S}}$ be the set of tuples $(\Psi', z', \mathbb{m}')$ returned by the simulation oracle $\mathbb{S}$. A vertex $u \in V(\mathsf{PCT})$ is* $(\overline{\mathbb{V}}_{\mathbb{pp}}, \mathcal{Q}_{\mathbb{S}})$-partially compliant *if for all outgoing edges $e = (u, v) \in E(\mathsf{PCT})$ labeled $(z^{(e)}, \mathbb{m}^{(e)})$, the following conditions hold* ***unless*** *$(\Psi_u, z^{(e)}, \mathbb{m}^{(e)}) \in \mathcal{Q}_{\mathbb{S}}$:*

1. *Let $e_1, ..., e_{m'}$ be the incoming edges to u labeled $(z^{(e_i)}, \mathbb{m}^{(e_i)})$. The local data $w^{(u)}_{\mathsf{loc}}$ must satisfy the predicate:*
$$\Psi_u(z^{(e)}, w^{(u)}_{\mathsf{loc}}, z^{(e_1)}, ..., z^{(e_{m'})}) = 1.$$
2. *The combined verifier must accept the outgoing proof:*
$$\overline{\mathbb{V}}_{\mathbb{pp}}(\Psi_u, z^{(e)}, \mathbb{m}^{(e)}) = 1.$$

(Here Ψ_u is the predicate associated with vertex u). We say PCT *is* $(\overline{\mathbb{V}}_{\mathbb{pp}}, \mathcal{Q}_{\mathbb{S}})$-partially compliant *if all its vertices u satisfy the condition above.*

Definition 4.3. *Let* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *be a zero-knowledge PCD scheme for a class of predicates* F*, with simulator* $\mathbb{S}$*. We say that* PCD *satisfies* simulation extractability (PCD-SE) *if for every auxiliary input distribution $\mathcal{D}$ and every expected polynomial-time adversary $\mathbb{A}$, there exists an expected polynomial-time extractor $\mathbb{E}_{\mathbb{A}}$ such that:*

$$\mathbf{Adv}^{\mathsf{PCD\text{-}SE}}_{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}},\mathcal{D}}(\lambda) := \Pr\left[\mathsf{PCD\text{-}SE}^{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}}}_{\mathsf{PCD},\mathsf{F},\mathcal{D}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

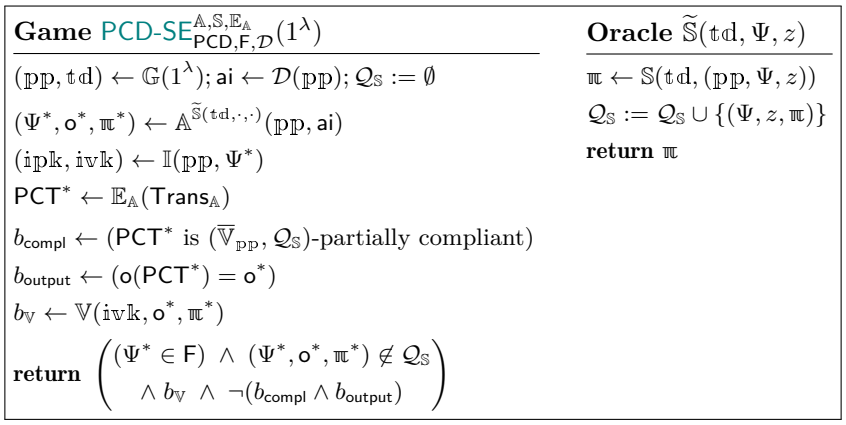

Fig. 9: Simulation extractability for proof-carrying data.

The simulation extractability game is defined in Figure 9. This game is identical to the knowledge soundness game (Figure 6) except that the adversary interacts with the simulation oracle, and the extractor's success relies on producing a partially compliant transcript according to Definition 4.2. The set $\mathcal{Q}_{\mathbb{S}}$ consists of the statement-proof pairs that are queried by $\mathbb{A}$ to $\mathbb{S}$. We may define stronger notions of black-box and straightline extraction according to Remark 3.2.

5 Templates for Constructing PCD and their Non-Malleability

In this section, we recall two general templates for constructing PCD, and show sufficient conditions for PCD constructed from either template to be simulation extractable.

5.1 PCD from SNARK

In the first template, a PCD can be constructed directly from a SNARK [4,18,23]. We recall this construction in Figure 10. We then show that the PCD constructed is simulation extractable (PCD-SE) assuming the underlying SNARK is simulation extractable (ARG-SE).

Theorem 5.1 (Theorem 9.2 in [18]). *Figure 10 gives a construction of a constant-depth PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ from a SNARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$. If ARG is black-box (resp. straightline) extractable, then so is PCD.*

Moreover, if ARG is zero-knowledge with stateful simulator $\mathcal{S}$, then PCD is zero-knowledge with the following simulator $\mathbb{S}$:

- $\mathbb{S}(\mathbb{td}, (\mathbb{pp}, \Psi, z))$: *run* $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R)$, *where* $R := R^{(\lambda,N,k)}_{\mathcal{V},\Psi,\mathsf{pp}}$, *then output* $\pi \leftarrow \mathcal{S}(\mathsf{td}, (\mathsf{pp}, R, (\mathsf{ivk}, z)))$.

Theorem 5.2. *In Figure 10, if the SNARK ARG satisfies ARG-SE with zero-knowledge simulator $\mathcal{S}$, then the resulting PCD scheme PCD satisfies PCD-SE (for constant-depth compliance predicates) with corresponding simulator $\mathbb{S}$ as described in Theorem 5.1. Furthermore, if ARG is black-box (resp. straightline) simulation-extractable, then so is PCD. If the PCD is straightline simulation-extractable, then it supports compliance predicates of arbitrary (polynomial) depth; otherwise it supports compliance predicates of constant depth.*

Notation.

- Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a SNARK relative to $\mathcal{O}$ for $\mathsf{NP}^{\mathcal{O}}$.
- Let $\mathcal{V}^{(\lambda,N,k)}$ be the circuit corresponding to the SNARK verifier $\mathcal{V}$ invoked on security parameter λ, index of size $\leq N$, and instance of size $\leq k$.

Recursion Circuit. Given a compliance predicate $\Psi : \{0,1\}^{(m+2)\ell} \to \{0,1\}$ for data of length ℓ, represented as a circuit, we define the following circuit:

$[R^{(\lambda,N,k)}_{\mathcal{V},\Psi,\mathsf{pp}}]\left((\mathsf{ivk}, z), (w_{\mathsf{loc}}, (z_i, \pi_i)_{i\in[m]})\right)$:[a]

1. Check that the compliance predicate $\Psi(z, w_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.
2. If there exists i such that $(z_i, \pi_i) \neq \perp$, check that the SNARK verifier $[\mathcal{V}^{(\lambda,N,k)}](\mathsf{ivk}, (\mathsf{ivk}, z_i), \pi_i)$ accepts for every $i \in [m]$.

PCD Construction. We define $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ as follows.

- $\mathbb{G}(1^\lambda) \to (\mathbb{pp}, \mathbb{td})$. Return $(\mathbb{pp}, \mathbb{td}) := (\mathsf{pp}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda)$.
- $\mathbb{I}(\mathsf{pp}, \Psi) \to (\mathbb{ipk}, \mathbb{ivk})$. Compute $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathbb{pp}, R)$, where $R := R^{(\lambda,N,k)}_{\mathcal{V},\Psi,\mathsf{pp}}$ is the recursion circuit defined above. Return $(\mathbb{ipk}, \mathbb{ivk}) := ((\mathsf{ipk}, \mathsf{ivk}), \mathsf{ivk})$.
- $\mathbb{P}(\mathbb{ipk}, z, w_{\mathsf{loc}}, (z_i, \pi_i)_{i\in[m]})$: return $\pi \leftarrow \mathcal{P}(\mathsf{ipk}, (\mathsf{ivk}, z), (w_{\mathsf{loc}}, (z_i, \pi_i)_{i\in[m]}))$.
- $\mathbb{V}(\mathbb{ivk}, z, \pi) \to b \in \{0,1\}$. Return $b \leftarrow \mathcal{V}(\mathsf{ivk}, (\mathsf{ivk}, z), \pi)$.

[a] where $k := |\mathsf{ivk}(\lambda, N)| + \ell$, and $N := N(\lambda, |\Psi|, m, \ell)$ is determined as in [23, Lemma 11.8].

Fig. 10: Construction of PCD from (preprocessing) SNARK, as in [23]

Proof. Let $\mathbb{A}$ be an adversary in the PCD-SE game against the PCD scheme constructed from a SNARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ as specified in Figure 10. Let ARG be simulation-extractable (ARG-SE) with respect to simulator $\mathcal{S}$. We construct a PCD-SE extractor $\mathbb{E}_{\mathbb{A}}$ that uses the ARG-SE extractor $\mathcal{E}$ for ARG.

SNARK Relation Definition. We first clarify the SNARK relation used in the PCD construction. For a PCD compliance predicate Ψ, let the corresponding SNARK relation R_Ψ be defined as follows. The instance is $x = z_{\mathsf{out}}$ (the outgoing data). The witness is $w = (w_{\mathsf{loc}}, [(z_i, \pi_i)]_{i=1}^{m'})$ where $m' \leq m$. The relation $R_\Psi(x, w)$ ($R^{(\lambda,N,k)}_{\mathcal{V},\Psi,\mathsf{pp}}$ from Figure 10) holds if and only if:

1. The compliance predicate holds: $\Psi(z_{\mathsf{out}}, w_{\mathsf{loc}}, z_1, \ldots, z_{m'}, \perp, \ldots, \perp) = 1$.
2. All incoming SNARK proofs verify: $\forall i \in [m'],\ \mathcal{V}(\mathsf{ivk}_i, z_i, \pi_i) = 1$, where ivk_i is the SNARK verification key for the predicate Ψ_i associated with the i-th incoming edge.[12]

A PCD proof $\mathbb{m}$ for statement z_{out} simply consists of the SNARK proof $\pi \leftarrow \mathcal{P}(\mathsf{ipk}, z_{\mathsf{out}}, w)$ for the relation R_Ψ defined above, where ipk is the SNARK proving key for R_Ψ. The PCD verifier $\mathbb{V}(\mathbb{ivk}, z_{\mathsf{out}}, \mathbb{m})$ checks if $\mathcal{V}(\mathsf{ivk}, z_{\mathsf{out}}, \pi) = 1$, where ivk is the SNARK verification key for R_Ψ.

We now construct the extractor $\mathbb{E}_{\mathbb{A}}$. The core idea is to define a sequence of SNARK adversaries $\mathcal{A}_k$ for each node k that might appear in the computation graph induced by $\mathbb{A}$'s output. We then use the ARG-SE extractor $\mathcal{E}_k$ for each $\mathcal{A}_k$ to extract the necessary witness information, which corresponds to the local data and incoming edges/proofs for that node in the PCD transcript.

Step 1: Constructing the Root SNARK Adversary $\mathcal{A}_{\mathbf{root}}$. We define a SNARK adversary $\mathcal{A}_{\mathrm{root}}$ that runs the PCD adversary $\mathbb{A}$, with respect to the *same* auxiliary input distribution (of $\mathsf{ai}_{\mathsf{PCD}}$) as $\mathbb{A}$. $\mathcal{A}_{\mathrm{root}}$ receives SNARK public parameters pp (derived from $\mathbb{pp}$) and auxiliary input $\mathsf{ai}_{\mathsf{ARG}}$ (distributed as $\mathsf{ai}_{\mathsf{PCD}}$). It simulates the environment for $\mathbb{A}$ as follows:

[12] We assume that the indexer has computed the relevant ivk_i.

1. **Input Handling:** $\mathcal{A}_{\text{root}}$ receives its own SNARK public parameters pp and auxiliary input $\mathsf{ai}_{\mathsf{ARG}}$. It passes on these inputs to $\mathbb{A}$ (since the SNARK public parameters are the same as the PCD public parameters).
2. **Oracle Simulation:** $\mathcal{A}_{\text{root}}$ uses its own simulation oracle $\mathcal{S}$ to answer $\mathbb{A}$'s queries to $\mathbb{S}$.
   (a) A query (Ψ, z) to $\mathbb{S}$ (PCD proof simulation) is answered by querying $\mathcal{S}(R_\Psi, z)$ (SNARK proof simulation) to get a simulated SNARK proof π. $\mathcal{A}_{\text{root}}$ returns $\mathbb{m} = (\pi)$ to $\mathbb{A}$ and records $(\Psi, z, \mathbb{m})$ in a list $\mathcal{Q}_{\mathbb{S}}$ of simulated outputs.
3. **Output Handling:** When $\mathbb{A}$ outputs $(\Psi^*, \mathsf{o}^*, \mathbb{m}^* = (\pi^*), \mathsf{ao}_{\mathsf{PCD}})$, $\mathcal{A}_{\text{root}}$ checks if $(\Psi^*, \mathsf{o}^*, \mathbb{m}^*)$ was previously returned by the simulated $\mathbb{S}$ oracle (i.e., if it's in $\mathcal{Q}_{\mathbb{S}}$).
   – If yes, $\mathcal{A}_{\text{root}}$ aborts (outputting $\perp$).
   – If no, $\mathcal{A}_{\text{root}}$ computes the SNARK relation index $\mathbb{i}^* = R_{\Psi^*}$ and outputs the tuple $(\mathbb{i}^*, \mathsf{o}^*, \pi^*, \mathsf{ao}_{\mathsf{ARG}})$, where $\mathsf{ao}_{\mathsf{ARG}}$ contains $\mathsf{ao}_{\mathsf{PCD}}$.

By the simulation extractability of ARG, there exists an expected polynomial-time extractor $\mathcal{E}_{\text{root}}$ for $\mathcal{A}_{\text{root}}$. Recall that $\mathcal{E}_{\text{root}}$ is given the full transcript $\mathsf{Trans}_{\mathcal{A}_{\text{root}}}$ of $\mathcal{A}_{\text{root}}$'s execution including queries to the simulation oracle. When $\mathcal{A}_{\text{root}}$ successfully outputs $(\mathbb{i}^*, \mathsf{o}^*, \pi^*, \mathsf{ao}_{\mathsf{ARG}})$, $\mathcal{E}_{\text{root}}$ outputs the same $(\mathbb{i}^*, \mathsf{o}^*, \mathsf{ao}_{\mathsf{ARG}})$, and also a SNARK witness $\mathbb{w}^* = (w^*_{\mathsf{loc}}, [(z_i, \pi_i)]_{i=1}^{m'})$ such that $R_{\Psi^*}(\mathsf{o}^*, \mathbb{w}^*) = 1$ with high probability.

Step 2: Constructing the Root PCD Extractor $\mathbb{E}_{\text{root}}$. The extractor $\mathbb{E}_{\text{root}}$ uses $\mathcal{E}_{\text{root}}$.

1. Run $\mathcal{E}_{\text{root}}$ (simulating oracles for it as needed, relying on the fact that $\mathcal{E}_{\text{root}}$'s proof simulation queries are in the PCD proof simulation format) to obtain $\mathbb{w}^*$. If $\mathcal{E}_{\text{root}}$ fails, $\mathbb{E}_{\text{root}}$ fails.
2. Parse $\mathbb{w}^*$ according to the structure defined by R_{Ψ^*}: $\mathbb{w}^* = (w^*_{\mathsf{loc}}, [(z_i, \pi_i)]_{i=1}^{m'})$.
3. Construct a partial proof-carrying transcript PCT_0 with a single root node labeled $(\mathsf{o}^*, \mathbb{m}^* = (\pi^*))$ and m' outgoing edges labeled $(z_i, \mathbb{m}_i = (\pi_i))$ respectively.
4. Output PCT_0 and $\mathsf{ao}_{\mathsf{PCD}} = \mathsf{ao}_{\mathsf{ARG}}$.

This PCT_0 represents the first level of the extracted transcript.

Step 3: Recursive Extraction for Inner Nodes. We now define the process to extend the transcript PCT_{k-1} to PCT_k. For each "leaf" edge $(z_k, \mathbb{m}_k = (\pi_k))$ in PCT_{k-1} corresponding to a node k that has *not* been marked as simulated:

1. **Check Simulation Log:** Check if $(z_k, \mathbb{m}_k)$ corresponds to an entry in $\mathcal{Q}_{\mathbb{S}}$ generated during the simulation for $\mathbb{A}$.
   – If yes, mark node k as "simulated" in the transcript and stop extraction down this path.
   – If no, proceed to extraction.
2. **Construct SNARK Adversary $\mathcal{A}_k$:** Define $\mathcal{A}_k$ which runs the process to generate PCT_{k-1} (this might involve running previous extractors or re-running parts of $\mathbb{A}$ depending on the extractor model). $\mathcal{A}_k$ identifies the non-simulated proof π_k for statement z_k associated with node k (predicate Ψ_k). It outputs $(\mathbb{i}_k = R_{\Psi_k}, z_k, \pi_k)$.
3. **Run SNARK Extractor $\mathcal{E}_k$:** Invoke the ARG-SE extractor $\mathcal{E}_k$ for $\mathcal{A}_k$. This yields a witness $\mathbb{w}_k = (w_{\mathsf{loc},k}, [(z_{child}, \pi_{child})])$ for the relation R_{Ψ_k} at instance z_k.
4. **Extend Transcript:** Update PCT_{k-1} by adding the local witness $w_{\mathsf{loc},k}$ to node k, and adding new child nodes corresponding to the edges $(z_{child}, \mathbb{m}_{child} = (\pi_{child}))$. This yields PCT_k.

This process is repeated until all non-simulated leaf nodes have been processed.

Step 4: The Final PCD Extractor $\mathbb{E}_{\mathbb{A}}$. The overall extractor $\mathbb{E}_{\mathbb{A}}$ runs the recursive extraction process described above, starting from $\mathbb{E}_{\text{root}}$, until no further non-simulated leaf nodes can be expanded. The final output is the fully constructed proof-carrying transcript PCT.

Success Probability Analysis (Sketch). Let $\mathbf{Adv}^{\mathsf{PCD\text{-}SE}}_{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}}}(\lambda)$ be the advantage of $\mathbb{A}$ in the PCD SIM-EXT game where $\mathbb{E}_{\mathbb{A}}$ fails to produce a compliant transcript for a non-simulated output $(\Psi^*, \mathsf{o}^*, \mathbb{m}^*)$. The extractor $\mathbb{E}_{\mathbb{A}}$ fails only if one of the underlying SNARK extractors $\mathcal{E}_k$ (for $\mathcal{A}_{\text{root}}$ or $\mathcal{A}_k$) fails. Let N be the number of nodes in the final extracted transcript PCT for which extraction is attempted (i.e., non-simulated nodes). We can use a hybrid argument. Let H_0 be the real PCD SIM-EXT game. Let H_k be a game where the first k SNARK extractions (in some canonical order) succeed, but the $(k+1)$-th may fail.

The difference between H_{k-1} and H_k is bounded by the ARG-SE advantage $\mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A}_k,\mathcal{S},\mathcal{E}_k}$ of the specific SNARK adversary $\mathcal{A}_k$ constructed for that step. By the union bound over the N potential extraction failures:

$$\mathbf{Adv}^{\mathsf{PCD\text{-}SE}}_{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}}}(\lambda) \leq \sum_{k \in \text{extracted nodes}} \mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A}_k,\mathcal{S},\mathcal{E}_k}(\lambda).$$

To relate this to the advantage of a single SNARK adversary $\mathcal{A}$, we construct $\mathcal{A}$ that internally simulates $\mathbb{A}$, picks a random node k from the potential extraction path, and runs the logic of $\mathcal{A}_k$. If $\mathbb{A}$ produces a valid output, the probability that $\mathcal{A}$ fails when $\mathcal{A}_k$ would have failed is roughly $1/N$ (needs formalization based on how N is defined and bounded). This leads to:

$$\mathbf{Adv}^{\mathsf{PCD\text{-}SE}}_{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}}}(\lambda) \leq N \cdot \mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A},\mathcal{S},\mathcal{E}_{\mathcal{A}}}(\lambda)$$

where $\mathcal{E}_{\mathcal{A}}$ is the extractor for the combined adversary $\mathcal{A}$. Since $\mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A},\mathcal{S},\mathcal{E}_{\mathcal{A}}}$ is negligible by the SNARK's SIM-EXT property and N is polynomially bounded (as $\mathbb{A}$ and $\mathcal{E}_k$ run in expected polynomial time), the overall PCD advantage is negligible.

Efficiency Analysis. The extractor $\mathbb{E}_{\mathbb{A}}$ involves running multiple SNARK extractors $\mathcal{E}_k$. Since each $\mathcal{E}_k$ runs in expected polynomial time, and the total number of nodes N is polynomially bounded in expectation (because $\mathbb{A}$ runs in expected polynomial time), the overall extractor $\mathbb{E}_{\mathbb{A}}$ also runs in expected polynomial time. □

Instantiations.

PCD on Cycles of Pairing-Friendly Elliptic Curves. Theorem 5.2 shows that to construct a simulation-extractable PCD scheme from a SNARK, we only need to ensure that the underlying SNARK is simulation-extractable. Several pairing-based SNARKs have been proven to be simulation-extractable, including the Groth-Maller SNARK [35]. When instantiated with this SNARK on cycles of pairing-friendly elliptic curves as in [3], we obtain a simulation-extractable PCD scheme.

Fractal. Fractal [23] is a SNARK that has been proven UC-secure in [19], which implies simulation extractability. By Theorem 5.2, the PCD scheme constructed from Fractal is also simulation-extractable.

5.2 PCD from NARK and Accumulation

In the second template, a PCD scheme is constructed by combining a NARK (that is not necessarily succinct) with an accumulation scheme that may accumulate the NARK's verifier circuit [8, 10]. We first state the compatibility requirement for NARK and AS to be able to compose them together.

Definition 5.3. *We say that* $\mathsf{AC} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *is a split accumulation scheme for the* NARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *if:*

1. *The* NARK *proof* π *can be split into two parts,* $\pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$*;*
2. AC *is a split accumulation scheme for the pair* $(\Phi_{\mathcal{V}}, \mathcal{H}_{\mathsf{ARG}} := \mathcal{G})$*, where:*
   $\Phi_{\mathcal{V}}(\mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = \mathbb{i}, \mathsf{qx} = (\mathbb{x}, \pi.\mathbb{x}), \mathsf{qw} = \pi.\mathbb{w})$:
   (a) Run $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i})$*;*
   (b) Output $\mathcal{V}(\mathsf{ivk}, \mathbb{x}, (\pi.\mathbb{w}, \pi.\mathbb{w}))$*.*

We recall the result from [8] showing that the construction in Figure 11 is secure, assuming appropriate properties of ARG and AC.

Theorem 5.4 (Theorem 5.3 in [8], adapted). *Figure 11 gives a construction of a constant-depth PCD scheme* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *from a* NARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *and an accumulation scheme* $\mathsf{AC} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$*. If both* ARG *and* AC *are black-box (resp. straightline) extractable, then so is* PCD*.*

Moreover, if ARG *is zero-knowledge with simulator* $\mathcal{S}$*, and* AC *is zero-knowledge with simulator* S*, then* PCD *is zero-knowledge with the following simulator* $\mathbb{S}$:

Notation.

- Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a NARK for NP.
- Let $\mathsf{AC} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for ARG.
- Let $\mathrm{V}^{(\lambda,m,N,k)}$ to be the circuit corresponding to the AC verifier V invoked on security parameter λ, number of instance-proof pairs and accumulators $\leq m$, index of size $\leq N$, and instance of size $\leq k$.

Recursion Circuit. Given a compliance predicate $\Psi : \{0,1\}^{(m+2)\ell} \to \{0,1\}$ for data of length ℓ, represented by a circuit, we define the following circuit:

$[R^{(\lambda,N,k)}_{\mathrm{V},\Psi,\mathsf{pp}}]\left((\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (w_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i\in[m]}, \mathsf{pf})\right)$:[a]

1. Check that the compliance predicate $\Psi(z, w_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.
2. For every $i \in [m]$ with $z_i \neq \perp$, check that

$$[\mathrm{V}^{(\lambda,m,N,k)}]\,(\mathsf{avk}, [\mathsf{qx}_i]_{i=1}^m, [\mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{acc}.\mathbb{x}, \mathsf{pf}) = 1,$$

   where $\mathsf{qx}_i := ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$.
3. If all checks hold, output 1, otherwise output 0.

PCD Construction. We define $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ as follows.

- $\mathbb{G}(1^\lambda)$: Compute $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda)$, $\mathsf{pp}_{\mathsf{AC}} \leftarrow \mathrm{G}(1^\lambda)$, and output $\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AC}})$ and $\mathbb{td} = \mathsf{td}$.
- $\mathbb{I}(\mathbb{pp}, \Psi) \to (\mathbb{ipk}, \mathbb{ivk})$.
  1. Compute $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R)$, where $R := R^{(\lambda,N,k)}_{\mathrm{V},\Psi,\mathsf{pp}}$ as defined above.
  2. Compute $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R)$.
  3. Output $\mathbb{ipk} = (\mathsf{ipk}, \mathsf{apk})$ and $\mathbb{ivk} = (\mathsf{ivk}, \mathsf{dk}, \mathsf{avk})$.
- $\mathbb{P}(\mathbb{ipk}, z, w_{\mathsf{loc}}, [z_i, (\pi_i, \mathsf{acc}_i)]_{i=1}^m) \to \mathbb{m}$.
  1. If $z_i = \perp$ for all $i \in [m]$ then sample $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, \perp)$.
  2. If $z_i \neq \perp$ for some $i \in [m]$ then:
     (a) Set $\mathsf{qx}_i = ((\mathsf{avk}, z_i, \mathsf{acc}_i.\mathbb{x}), \pi_i.\mathbb{x})$.
     (b) Set $\mathsf{qw}_i = (\mathsf{acc}_i.\mathbb{w}, \pi_i.\mathbb{w})$.
     (c) Sample $(\mathsf{acc}, \mathsf{pf}) \leftarrow \mathrm{P}(\mathsf{apk}, [(\mathsf{qx}_i, \mathsf{qw}_i)]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
  3. Compute $\pi_{\mathsf{ARG}} \leftarrow \mathcal{P}\,(\mathsf{ipk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), (w_{\mathsf{loc}}, [z_i, \pi_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{x}]_{i=1}^m, \mathsf{pf}))$.
  4. Output $\mathbb{m} = (\pi_{\mathsf{ARG}}, \mathsf{acc})$.
- $\mathbb{V}(\mathbb{ivk}, z, \mathbb{m}) \to b \in \{0,1\}$. Parse $\mathbb{m} = (\pi_{\mathsf{ARG}}, \mathsf{acc})$. Accept if and only if both $\mathcal{V}(\mathsf{ivk}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}), \pi_{\mathsf{ARG}})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ accept.

[a] where $k := |\mathsf{avk}(\lambda, N)| + |\mathsf{acc}.\mathbb{x}(\lambda, m, N)| + \ell$, and $N := N(\lambda, |\Psi|, m, \ell)$ is determined as in [8].

Fig. 11: Construction of PCD from NARK and AC, as in [8]

- *Let* $(\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AC}}), \mathbb{td} := \mathsf{td}) \leftarrow \mathbb{G}(1^\lambda)$.
- $\mathbb{S}(\mathbb{td}, (\mathbb{pp}, \Psi, z))$:
  1. *Let* $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R)$, *where* $R := R^{(\lambda,N,k)}_{\mathrm{V},\Psi,\mathsf{pp}}$.
  2. *Compute* $\mathsf{acc} \leftarrow \mathrm{S}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R)$.
  3. *Compute* $\pi \leftarrow \mathcal{S}(\mathsf{td}, (R, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x})))$.
  4. *Output* (π, acc).

Theorem 5.5. *In Figure 11, let* ARG *be zero-knowledge with simulator* $\mathcal{S}$*, and* AC *be zero-knowledge with simulator* S*. Let* $\mathcal{IG}(\Psi, z) \to (R^{(\lambda,N,k)}_{\mathrm{V},\Psi,\mathsf{pp}}, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}))$ *be the instance generator, where* avk *and* acc.$\mathbb{x}$ *is obtained similar to how they are obtained by* $\mathbb{S}$*.*

If ARG *satisfies* $(\mathcal{IG}, \mathcal{S})$-ARG-SE *and* AC *satisfies* S-AC-SE*, then the resulting PCD scheme* PCD *satisfies* PCD-SE *with corresponding simulator* $\mathbb{S}$ *as described in Theorem 5.4. Furthermore, if* ARG *and* AC *are both black-box (resp. straightline) simulation-extractable, then so is* PCD*. If the PCD is straightline simulation-extractable, then it supports compliance predicates of arbitrary (polynomial) depth; otherwise it supports compliance predicates of constant depth.*

Proof. Let d be the depth of the proof-carrying transcript PCT we need to extract. Without loss of generality, we may assume that the extracted transcript PCT will be a tree. Thus we may associate the data $(z^{(u,v)}, \mathbb{m}^{(u,v)})$ of the unique outgoing edge from a node u with the node u itself, writing instead $(z^{(u)}, \mathbb{m}^{(u)})$.

Construction Overview. Given an adversary $\mathbb{A}$ in the PCD-SE game, we will define a corresponding extractor $\mathbb{E}_{\mathbb{A}}$ for transcripts up to depth d. The key insight is to build this extractor recursively, layer by layer, using the simulation extractability of both the NARK and the accumulation scheme.

For each layer $i \in [0, \ldots, d]$, we construct an extractor $\mathbb{E}_i$ that extracts a proof-carrying tree of depth i. The base case $\mathbb{E}_0$ is essentially the adversary $\mathbb{A}$ itself (as it produces the output node without explaining its derivation). For each subsequent layer, we build upon the previous one by extracting witnesses for nodes at that depth. The final extractor $\mathbb{E}_{\mathbb{A}}$ will be $\mathbb{E}_d$.

Recursive Construction. Assuming we have defined $\mathbb{E}_i$, we will define $\mathbb{E}_{i+1}$ through the following steps:

Step 1: Construct a NARK adversary. We first construct an adversary $\mathcal{A}_i$ against the ARG-SE property of the NARK, utilizing $\mathbb{E}_i$:

$\mathcal{A}_i^{\tilde{\mathbb{S}}}(\mathsf{pp}, (\mathsf{pp}_{\mathsf{AC}}, \mathsf{ai}))$:

1. Run $\mathbb{A}^{\tilde{\mathbb{S}}}(\mathbb{pp}, \mathsf{ai})$, simulating oracle accesses for $\mathbb{A}$ as follows:
   (a) To simulate $\tilde{\mathbb{S}}$ on input (Ψ, z):
      i. Let $R = R^{(\lambda,N,k)}_{\mathrm{V},\Psi,\mathsf{pp}}$. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R)$.
      ii. Generate $\mathsf{acc} \leftarrow \mathrm{S}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}, R)$.
      iii. Obtain $\pi_{\mathsf{ARG}} \leftarrow \tilde{\mathcal{S}}(\mathsf{pp}, R, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}))$.
      iv. Return $\mathbb{m} = (\pi_{\mathsf{ARG}}, \mathsf{acc})$.
2. Run $(\Psi^*, \mathsf{PCT}_i) \leftarrow \mathbb{E}_i(\mathsf{Trans}_{\mathbb{A}})$, where $\mathsf{Trans}_{\mathbb{A}}$ is the transcript resulting from running $\mathbb{A}^{\tilde{\mathbb{S}}}(\mathbb{pp}, \mathsf{ai})$.
3. For each node $u \in \ell_{\mathsf{PCT}_i}(i)$ at depth i in PCT_i, extract the associated data $(z^{(u)}, \mathbb{m}^{(u)})$ where $\mathbb{m}^{(u)} = (\pi^{(u)}_{\mathsf{ARG}}, \mathsf{acc}^{(u)})$.
4. For each such node where $(\Psi^*, z^{(u)}, \mathbb{m}^{(u)}) \notin \mathcal{Q}_{\mathbb{S}}$ (i.e., it's not a simulated proof), output:

$$(R^{(\lambda,N,k)}_{\mathrm{V},\Psi^*,\mathsf{pp}}, (\mathsf{avk}, z^{(u)}, \mathsf{acc}^{(u)}.\mathbb{x}), \pi^{(u)}_{\mathsf{ARG}}.\mathbb{x})$$

Let $\mathsf{Trans}_{\mathcal{A}_i}$ be $\mathcal{A}_i$'s transcript containing all of its inputs and outputs, including random coins. By the simulation extractability of the NARK, there exists an extractor $\mathcal{E}_{\mathcal{A}_i}$ that, given $\mathsf{Trans}_{\mathcal{A}_i}$, extracts the witness for $\mathcal{A}_i$. For each node u at depth i with non-simulated proof, this extractor produces a witness of the form:

$$(w^{(u)}_{\mathsf{loc}}, [z^{(u)}_j, \pi^{(u)}_j.\mathbb{x}, \mathsf{acc}^{(u)}_j.\mathbb{x}]_{j\in[m]}, \mathsf{pf}^{(u)})$$

Step 2: Construct an accumulation scheme adversary. We next define an adversary A_i against the $\mathsf{AC\text{-}SE}$ property of the accumulation scheme:

$\mathrm{A}_i(\mathsf{pp}_{\mathsf{AC}}, \mathsf{ai})$:

1. Let $\mathsf{ai}_{\mathsf{ac}}$ be the part of the auxiliary that concerns only the accumulation scheme (without any of the NARK simulation responses)
2. Run the NARK adversary $\mathcal{A}_i^{\tilde{\mathcal{S}}}(\mathsf{pp}, \mathsf{ai}_{\mathsf{ac}})$, simulating oracle accesses for $\mathcal{A}_i^{\tilde{\mathcal{S}}}$ as follows:
   (a) To simulate $\tilde{\mathcal{S}}$ on input $(\mathbb{i} = (\mathsf{pp}, R), \mathbb{x} = (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}))$:
      i. Let $Q_{\mathbb{S}}$ denote the list of queries made by $\mathbb{A}$ to its simulation oracle $\tilde{\mathbb{S}}$, as recorded in $\mathsf{Trans}_{\mathbb{A}}$.
      ii. Scan $Q_{\mathbb{S}}$ for the unique item whose R matches $\mathbb{i}$'s R, and whose avk, z, and $\mathsf{acc}.\mathbb{x}$ match $\mathbb{x}$.
      iii. Return the answer π_{ARG} from that same PCD entry.
3. Let $\mathsf{Trans}_{\mathcal{A}}$ be the transcript resulting from the execution of $\mathcal{A}$.
4. Run $\mathcal{E}_{\mathcal{A}_i}(\mathsf{Trans}_{\mathcal{A}})$
5. For each node at depth i, $\mathcal{E}_{\mathcal{A}_i}$ then outputs the local data w_{loc}, the prior $[z_j, \pi_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{x}]_{j=1}^m$ (at depth $i+1$), and the accumulation proof pf.
6. For each node u at depth i with non-simulated proof and its extracted witness containing $\mathsf{pf}^{(u)}$:
   (a) For each incoming edge j, extract $\mathsf{qx}_j^{(u)} = ((\mathsf{avk}, z_j^{(u)}, \mathsf{acc}_j^{(u)}.\mathbb{x}), \pi_j^{(u)}.\mathbb{x})$.
   (b) Output $(R_{\mathrm{V},\Psi^*,\mathsf{pp}}^{(\lambda,N,k)}, [\mathsf{qx}_j^{(u)}]_{j\in[m]}, [\mathsf{acc}_j^{(u)}.\mathbb{x}]_{j\in[m]}, \mathsf{acc}^{(u)}.\mathbb{x}, \mathsf{pf}^{(u)})$

The extractor $\mathcal{E}_{\mathcal{A}_i}(\mathsf{Trans}_{\mathcal{A}})$ is the non-black-box extractor for $\mathcal{A}_i$, as guaranteed by the simulation extractability of the NARK. By the knowledge soundness of the accumulation scheme, there exists an extractor $\mathrm{E}_{\mathrm{A}_i}$ that extracts, for each node, the witness components:

$$([\mathsf{qw}_j^{(u)}]_{j\in[m]}, [\mathsf{acc}_j^{(u)}.\mathbb{w}]_{j\in[m]})$$

where $\mathsf{qw}_j^{(u)} = (\mathsf{acc}_j^{(u)}.\mathbb{w}, \pi_j^{(u)}.\mathbb{w})$, and the overall relation $R_{\mathrm{V},\Psi^*,\mathsf{pp}}^{(\lambda,N,k)}$.

Step 3: Define the extractor for the next layer. We can now define $\mathbb{E}_{i+1}$ as follows:

$\mathbb{E}_{i+1}(\mathsf{Trans})$:

1. Run
$$\left(\left[\mathbb{i}_\Phi^{(k)}, [(\mathsf{qx}_i^{(k)}, \mathsf{qw}_i^{(k)})]_{i=1}^n, [\mathsf{acc}_j^{(k)}]_{j=1}^m, \mathsf{acc}^{(k)}\right]_{k=1}^{\ell}, \mathsf{ao}'\right) \leftarrow \mathrm{E}_{\mathrm{A}_i}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{Trans})$$
2. Parse ao' as (Ψ^*, PCT_i). Abort if PCT_i is not a transcript of depth i.
3. For each node u at depth i:
   (a) If $(\Psi^*, z^{(u)}, \mathbb{w}^{(u)}) \in \mathcal{Q}_{\mathbb{S}}$ (i.e., it's a simulated proof), mark it as a leaf node (not to be extracted further).
   (b) Extend PCT_i by adding these child nodes and connecting them to node u.
4. Return $(\Psi^*, \mathsf{PCT}_{i+1})$, where PCT_{i+1} is the extended transcript.

The extractor $\mathrm{E}_{\mathrm{A}_i}$ is the one guaranteed by the knowledge soundness of the accumulation scheme instantiated against A_i and the auxiliary distribution $\mathcal{D}(\mathsf{pp}_{\mathsf{AC}})$ defined as follows:

- Sample $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda)$.
- Emulate the $\mathsf{SIM\text{-}EXT}$ game for $\mathbb{A}$. To simulate $\tilde{\mathbb{S}}$ on input (Ψ, z):
  1. Let $R = R_{\mathrm{V},\Psi,\mathsf{pp}}^{(\lambda,N,k)}$. Compute $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}_\Phi = \mathsf{pp}, \mathsf{i}_\Phi = R)$.
  2. Generate $\mathsf{acc} \leftarrow \mathrm{S}(\mathsf{pp}_{\mathsf{AC}}, \mathsf{pp}, R)$.
  3. Obtain $\pi_{\mathsf{ARG}} \leftarrow \tilde{\mathcal{S}}(\mathsf{pp}, R, (\mathsf{avk}, z, \mathsf{acc}.\mathbb{x}))$.
  4. Return $\mathbb{w} = (\pi_{\mathsf{ARG}}, \mathsf{acc})$.
- Output the transcript $\mathsf{Trans}_{\mathbb{A}}$.

Analysis of the Extractor. We now show that the constructed extractor $\mathbb{E}_{\mathbb{A}} := \mathbb{E}_d$ succeeds with high probability. The key insight is that if $\mathbb{A}$ produces a valid proof that is not simulated, then the recursive extraction process will produce a $(\Psi^*, \overline{\mathbb{V}}_{\mathrm{pp}}, \mathcal{Q}_{\mathbb{S}})$-compliant transcript with the same output.

By construction, our extractor satisfies the requirements for a compliant transcript with simulated proofs:

1. For non-simulated nodes, we extract valid witnesses and verify that the compliance predicate is satisfied. 2. For simulated nodes, we stop extraction as permitted by the definition. 3. We preserve the output of the original adversary.

The success probability analysis follows from the simulation extractability of the NARK and the knowledge soundness of the accumulation scheme. If the NARK extractor $\mathcal{E}_{\mathcal{A}_i}$ succeeds in extracting valid witnesses, and the accumulation scheme extractor $\mathrm{E}_{\mathrm{A}_i}$ succeeds in extracting valid witness components, then our PCD extractor $\mathbb{E}_{\mathbb{A}}$ will succeed.

Specifically, the advantage of an adversary against our extractor is bounded by:

$$\mathbf{Adv}^{\mathsf{PCD\text{-}SE}}_{\mathbb{A},\mathbb{S},\mathbb{E}_{\mathbb{A}}}(\lambda) \leq \sum_{i=0}^{d-1} \left(\mathbf{Adv}^{\mathsf{ARG\text{-}SE}}_{\mathcal{A}_i,\mathcal{S},\mathcal{E}_{\mathcal{A}_i}}(\lambda) + \mathbf{Adv}^{\mathsf{AC\text{-}KS}}_{\mathrm{A}_i,\mathrm{E}_{\mathrm{A}_i}}(\lambda) \right)$$

Since ARG and AC satisfy simulation extractability and knowledge soundness, respectively, both terms are negligible, and so is their sum for any constant depth d. □

References

1. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1987–2005. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417228
2. Baghery, K., Kohlweiss, M., Siim, J., Volkhov, M.: Another look at extraction and randomization of groth's zk-SNARK. In: Borisov, N., Díaz, C. (eds.) FC 2021, Part I. LNCS, vol. 12674, pp. 457–475. Springer, Berlin, Heidelberg (Mar 2021). https://doi.org/10.1007/978-3-662-64322-8_22
3. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 276–294. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_16
4. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 111–120. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488623
5. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352 (2020), https://eprint.iacr.org/2020/352
6. 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
7. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Report 2020/1618 (2020), https://eprint.iacr.org/2020/1618
8. Bünz, B., Chiesa, A., Lin, W., Mishra, P., Spooner, N.: Proof-carrying data without succinct arguments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 681–710. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_24
9. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499 (2020), https://eprint.iacr.org/2020/499
10. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 1–18. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_1
11. Bünz, B., Mishra, P., Nguyen, W., Wang, W.: Arc: Accumulation for reed–solomon codes. Cryptology ePrint Archive, Paper 2024/1731 (2024), https://eprint.iacr.org/2024/1731
12. 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
13. Campanelli, M., Faonio, A., Russo, L.: SNARKs for virtual machines are non-malleable. pp. 153–183. LNCS, Springer, Cham (Jun 2025). https://doi.org/10.1007/978-3-031-91134-7_6
14. Campanelli, M., Fiore, D., Pancholi, M.: On composing AGM-secure functionalities with cryptographic proofs: Applications to unbounded-depth IVC and more. Cryptology ePrint Archive, Paper 2025/2086 (2025), https://eprint.iacr.org/2025/2086
15. Campanelli, M., Fiore, D., Pancholi, M.: When can we incrementally prove computations of arbitrary depth? Cryptology ePrint Archive, Paper 2025/1413 (2025), https://eprint.iacr.org/2025/1413
16. 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
17. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Berlin, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_18
18. Chen, M., Chiesa, A., Spooner, N.: On succinct non-interactive arguments in relativized worlds. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 336–366. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_12
19. Chiesa, A., Fenzi, G.: zkSNARKs in the ROM with unconditional UC-security. In: TCC 2024, Part I. pp. 67–89. LNCS, Springer, Cham (Nov 2024). https://doi.org/10.1007/978-3-031-78011-0_3
20. Chiesa, A., Guan, Z., Samocha, S., Yogev, E.: Security bounds for proof-carrying data from straightline extractors. Cryptology ePrint Archive, Report 2023/1646 (2023), https://eprint.iacr.org/2023/1646
21. Chiesa, A., Guan, Z., Samocha, S., Yogev, E.: Security bounds for proof-carrying data from straightline extractors. In: TCC 2024, Part II. pp. 464–496. LNCS, Springer, Cham (Nov 2024). https://doi.org/10.1007/978-3-031-78017-2_16

22. 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
23. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45721-1_27
24. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: Yao, A.C.C. (ed.) ICS 2010. pp. 310–331. Tsinghua University Press (Jan 2010)
25. 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
26. 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
27. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014, Part II. LNCS, vol. 8713, pp. 313–326. Springer, Cham (Sep 2014). https://doi.org/10.1007/978-3-319-11212-1_18
28. Devadas, L., Goyal, R., Kalai, Y., Vaikuntanathan, V.: Rate-1 non-interactive arguments for batch-NP and applications. In: 63rd FOCS. pp. 1057–1068. IEEE Computer Society Press (Oct / Nov 2022). https://doi.org/10.1109/FOCS54457.2022.00103
29. 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
30. Faonio, A., Fiore, D., Kohlweiss, M., Russo, L., Zajac, M.: From polynomial IOP and commitments to non-malleable zkSNARKs. Cryptology ePrint Archive, Report 2023/569 (2023), https://eprint.iacr.org/2023/569
31. 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
32. 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
33. 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
34. 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
35. 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
36. 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
37. 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
38. Kothapalli, A., Setty, S.T.V.: HyperNova: Recursive arguments for customizable constraint systems. pp. 345–379. LNCS, Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_11
39. Lee, H., Seo, J.H.: On the security of nova recursive proof system. Cryptology ePrint Archive, Paper 2024/232 (2024), https://eprint.iacr.org/2024/232
40. Lurk lab. https://github.com/lurk-lab (2024)
41. 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
42. Maller, M., Mohnblatt, N., Zapico, A.: IVC in the open-and-sign random oracle model. Cryptology ePrint Archive, Paper 2025/1663 (2025), https://eprint.iacr.org/2025/1663
43. Mina protocol. https://minaprotocol.com/ (2024)
44. Naor, M., Paneth, O., Rothblum, G.N.: Incrementally verifiable computation via incremental PCPs. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 552–576. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_21

45. Nguyen, W., Boneh, D., Setty, S.: Revisiting the nova proof system on a cycle of curves. Cryptology ePrint Archive (2023)
46. Nguyen, W., Boneh, D., Setty, S.: Revisiting the nova proof system on a cycle of curves. Cryptology ePrint Archive, Report 2023/969 (2023), https://eprint.iacr.org/2023/969
47. Nguyen, W.D., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based SNARKs. pp. 308–344. LNCS, Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68403-6_10
48. Paneth, O., Pass, R.: Incrementally verifiable computation via rate-1 batch arguments. In: 63rd FOCS. pp. 1045–1056. IEEE Computer Society Press (Oct / Nov 2022). https://doi.org/10.1109/FOCS54457.2022.00102
49. 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
50. 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
51. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Berlin, Heidelberg (Mar 2008). https://doi.org/10.1007/978-3-540-78524-8_1
52. Zhao, J., Setty, S., Cui, W., Zaverucha, G.: MicroNova: Folding-based arguments with efficient (on-chain) verification. Cryptology ePrint Archive, Paper 2024/2099 (2024), https://eprint.iacr.org/2024/2099
53. Zhao, J., Setty, S.T.V., Cui, W., Zaverucha, G.: MicroNova: Folding-based arguments with efficient (on-chain) verification. In: 2025 IEEE Symposium on Security and Privacy. pp. 1964–1982. IEEE Computer Society Press (May 2025). https://doi.org/10.1109/SP61157.2025.00168