Security Bounds for Proof-Carrying Data from Straightline Extractors

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Ziyi Guan
ziyi.guan@epfl.ch
EPFL

Shahar Samocha
shahars@starkware.co
StarkWare

Eylon Yogev
eylon.yogev@biu.ac.il
Bar-Ilan University

May 20, 2024

Abstract

Proof-carrying data (PCD) is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. Real-world deployments of PCD have sparked keen interest within the applied community and industry.

Known constructions of PCD are obtained by recursively-composing SNARKs or related primitives. Unfortunately, known security analyses incur expensive blowups, which practitioners have disregarded as the analyses would lead to setting parameters that are prohibitively expensive.

In this work we study the concrete security of recursive composition, with the goal of better understanding how to reasonably set parameters for certain PCD constructions of practical interest. Our main result is that PCD obtained from SNARKs with *straightline knowledge soundness* has essentially the same security as the underlying SNARK (i.e., recursive composition incurs essentially no security loss).

We describe how straightline knowledge soundness is achieved by SNARKs in several oracle models, which results in a highly efficient security analysis of PCD that makes black-box use of the SNARK's oracle (there is no need to instantiated the oracle to carry out the security reduction).

As a notable application, our work offers an idealized model that provides new, albeit heuristic, insights for the concrete security of *recursive STARKs* used in blockchain systems. Our work could be viewed as partial evidence justifying the parameter choices for recursive STARKs made by practitioners.

Keywords: proof-carrying data; succinct non-interactive arguments; relativization; concrete security

Contents

1 Introduction **3**
1.1 Our results 4

2 Techniques **8**
2.1 Security analysis of PCD from straightline extractors 8
2.2 Application: improved concrete security for black-box PCD constructions 11
2.3 Application: a paradigm to set security for hash-based PCD 12
2.4 Example: a real-world compliance predicate with unknown size and depth bound 15
2.5 Technical extension: a more general analysis 17

3 Preliminaries **19**
3.1 Non-interactive arguments in oracle models 19
3.2 Proof-carrying data in oracle models 20

4 From relativized ARG to PCD: construction **22**

5 From relativized ARG to PCD: security reduction **24**
5.1 Knowledge soundness error 24
5.2 Extraction time bound 28

6 Relativized non-interactive arguments with straightline extraction **31**
6.1 The arithmetized random oracle model 31
6.2 The signed random oracle model 31
6.3 The random oracle model (and any other oracle model) 31

7 Technical extension: probabilistic extractors with oracle access **34**
7.1 Probabilistic oracle extractor for PCD 36
7.2 Knowledge soundness error 37
7.3 Extraction query bound 42
7.4 Extraction time bound 42

Acknowledgments **43**

References **43**

1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. PCD generalizes the notion of incrementally-verifiable computation (IVC) [Val08], and has found applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], verifiable registries [TFZBT22], privacy pools [BF23], blockchains [Mina; BMRS20; CCDW20; KB23], and more.

Known PCD constructions (and practical IVC constructions) are obtained via *recursive proof composition*, a framework for building PCD from simpler primitives such as SNARKs [BCCT13; BCTV14; COS20] or accumulation schemes [BGH19; BCMS20; BDFG21; BCLMS21; KST22; KS22; BC23].[1] Constructions differ, but the high-level idea is similar: to prove the correctness of t computation steps given a correctness proof for $t-1$ steps, one proves that "step t is correct *and* there is a valid proof for the first $t-1$ steps".

There are several practically efficient constructions of PCD, which has sparked keen industry interest and led to real-world deployments [ML20; SW22; PL22; P23]. However, the concrete cost of the security reduction from PCD to the underlying primitive is not well understood: *there are no comprehensive guidelines for securely instantiating PCD constructions*. In fact, an initial motivation for this work was the desire to better understand the concrete security of recursive STARKs [SW22] used in blockchain systems.

The prevailing practice in real-world deployments is setting parameters so that the underlying SNARK or accumulation scheme achieves the desired security level, and then assuming that the resulting PCD construction inherits the same security level. *But this fails to account for the potential security loss in the security reduction from PCD to the primitive(s) underlying its construction.*

This state of affairs leads us to ask a basic question:

What is the concrete security cost of recursive proof composition?

Known security analyses. The security analysis of most PCD constructions works only for a constant number of recursions ([BCCT13; BCTV14; BGH19; COS20; BCMS20; BDFG21; BCLMS21; KST22; KS22; BC23]). Informally, this is because the security reduction recursively invokes an underlying knowledge extractor that, at each invocation, incurs a polynomial blowup in time/size relative to the prior invocation. Moreover, in many settings, this blow up is unknown because it originates from an underlying knowledge assumption (e.g., a knowledge-of-exponent assumption). Overall this state of affairs implies that one is unable to set security parameters, and that the security loss is exponential in the recursion depth. This is *considerably worse* than "the security of PCD is approximately that of the underlying primitive" (the prevailing practice).

What about oracle models? The aforementioned inefficiencies of knowledge extraction generally do not arise for SNARKs (or accumulation schemes) constructed in oracle models. This is because the knowledge extractor is explicit (it is constructed rather than assumed), and deduces a witness merely by analyzing the prover's queries to the oracle and their answers; this does not require any access to the prover itself, and avoids rerunning the prover multiple times (which incurs significant time or error overheads). Unfortunately, PCD constructions typically make a non-black-box use of the underlying SNARK or accumulation scheme,[2] which requires instantiating the oracle (a heuristic step), and so the security reduction cannot take advantage of the efficient knowledge extraction previously available in the oracle model. Instead, the security reduction assumes some (non-black-box) knowledge extractor for the heuristically derived scheme.

[1] A separate line of work constructs IVC for deterministic computations from falsifiable cryptographic assumptions using different tools (see [PP22] and references therein). These elegant constructions are less relevant to the motivation of this paper as, typically, applications of PCD and IVC require supporting nondeterministic computations.

[2] The statement that "there exists a valid proof" refers to the *verifier* of the underlying SNARK or accumulation scheme. As such, the resulting PCD scheme makes non-black-box use of the verifier for the underlying scheme.

Hope: black-box constructions of PCD. Several works construct PCD in oracle models *without* instantiating the oracle (that is, while making a black-box use of the oracle) [CT10; CCS22; CCGOS23]. The key step is obtaining a *relativized SNARK*, a SNARK in an oracle model that can prove computations that themselves involve calls to the oracle. Then recursive proof composition can be used to directly obtain PCD in the same oracle model, via a security analysis that involves an explicit extractor.

1.1 Our results

In this paper we show that PCD constructions obtained from SNARKs with *straightline knowledge soundness* have essentially the same security as the underlying SNARK. Afterwards, we explain how this setting arises in several constructions of interest, including in deployed systems. In particular, our work gives partial justification for parameter settings currently used by practitioners in certain deployed PCD constructions.[3]

PCD from straightline extraction. Suppose that we are given a relativized SNARK in a certain oracle model. The canonical construction of PCD from a SNARK in the standard model [BCCT13] straightforwardly extends to the relativized case. Indeed, the SNARK prover can prove the correctness of oracle computations, and in particular can prove the correctness of the SNARK verifier (which queries the oracle).

We show that if the relativized SNARK has a straightline extractor then the resulting PCD scheme has a straightline extractor with the same error as the underlying SNARK.

Theorem 1 (informal)**.** *Let* ARG *be a relativized non-interactive argument in an oracle model, and let* PCD *be the PCD scheme obtained from* ARG *via the canonical construction (adapted to the relativized setting).*

Suppose that ARG *has a straightline extractor with knowledge soundness error* $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s})$ *and extraction time* $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})$*, where* $\lambda \in \mathbb{N}$ *is the security parameter,* $\mathsf{q} \in \mathbb{N}$ *is the number of queries by the adversary to the oracle, and* s *is the size of the adversary. Then* PCD *has a straightline extractor with:*

- *knowledge soundness error* $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$ *where* $\mathsf{s}' := \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$*, and*
- *extraction time* $\mathsf{t}_{\mathsf{PCD}}(\lambda, \mathsf{q}, N) \leq O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$*.*

Above, N *is the maximum number of nodes in the PCD distributed computation.*

Above, the additive term $O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$ intuitively corresponds to the N extractions required to produce a PCD distributed computation of size at most N. Note that, since extraction is straightline, the extraction times $\mathsf{t}_{\mathsf{ARG}}$ and $\mathsf{t}_{\mathsf{PCD}}$ do not depend on adversary size.

We discuss two applications of Theorem 1: in Section 1.1.1 we discuss an application the black-box PCD constructions; and in Section 1.1.2 we discuss an application to partially justify parameter settings used in certain deployed PCD constructions.

1.1.1 Application: black-box PCD constructions

Several works [CT10; CCS22; CCGOS23] construct PCD in oracle models, with black-box security reductions to falsifiable cryptographic assumptions.[4] While [CCS22] constructs a rewinding knowledge extractor (and leaves open the question of constructing a straightline knowledge extractor), [CT10] and [CCGOS23] construct straightline knowledge extractors in their respective oracle models. These latter works roughly achieve the following: they construct a relativized SNARK with a certain (straightline) knowledge soundness

[3] There are other PCD constructions of practical interest that do not fit our setting (e.g., those based on knowledge-of-exponent assumptions). Achieving security reductions that yield useful concrete security bounds for these remains an open problem.

[4] Such reductions are unlikely to exist in the standard model [GW11], as PCD can be used to construct a SNARK [BCCT13].

error κ_{ARG} and then show that the resulting PCD scheme has (straightline) knowledge soundness error (roughly) $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$, where $\mathsf{s}' := \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$.

Our Theorem 1 offers a significant improvement for [CT10] and [CCGOS23]: the knowledge soundness error $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$, *eliminating the multiplicative factor* N (PCD distributed computation size). We suspect that this upper bound is tight, as the only overhead comes from increasing the adversary size from s to $\mathsf{s}' = \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$, which reflects the additive cost to recover a PCD distributed computation of size N by invoking the SNARK extractor (whose running time is $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})$) for N times.

1.1.2 Application: hash-based PCD

Hash-based SNARKs have found widespread deployment in practice. Security parameters of such SNARKs are heuristically set according to the random oracle methodology [BR93]: first, model the hash function as a random function (even though it is not), which results in a SNARK in the ROM that "idealizes" the given hash-based SNARK; second, establish concrete security bounds in the ROM; finally, set security parameters of the hash-based SNARK according to the analysis for the SNARK in the ROM.

The random oracle methodology applied to the hash-based SNARK is thus tantamount to a *conjecture*: attacks against the hash-based SNARK are no more effective than attacks against the corresponding SNARK in the ROM (hence it is reasonable to set security parameters of the former according to the latter). Such conjectures are generally believed to hold for "natural" cryptographic protocols that use hash functions.[5]

In certain applications, the hash-based SNARK is recursively composed, leading to deployments of hash-based PCD constructions. Known security analyses of these PCD constructions incur expensive blowups, which practitioners have disregarded as those security analyses would lead to setting parameters that are prohibitively expensive. In other words, the common practice is to set security parameters as if the security reduction incurred no costs. Below we elaborate on these challenges, and then we propose how the results in this paper can be viewed as providing partial justification for current parameter settings in practice.

Challenges. Once the SNARK in the ROM is heuristically instantiated as a hash-based SNARK, we lose the explicit knowledge extractor constructed in the ROM. Hence, to analyze the resulting hash-based PCD construction, prior work postulates the existence of a non-black-box knowledge extractor for the hash-based SNARK in the standard model. But such a knowledge extractor is weak (it leads to a PCD knowledge extractor whose time/size has an exponential dependence on the recursion depth), and also rules out any hope for concrete security because we know nothing of the postulated knowledge extractor.

Alternatively, why not apply the random oracle methodology? Idealize the hash-based PCD construction as a PCD in the ROM, then set security parameters of the hash-based PCD according to a concrete security analysis of the corresponding PCD in the ROM. Unfortunately, this is problematic because it would require the underlying SNARK in the ROM to be relativized; indeed, the SNARK prover would have to attest to computations involving the random oracle (namely, its own SNARK verifier). However known SNARKs in the ROM are not relativized and, in fact, relativized SNARKs in the ROM do not exist [BCG24].[6]

Our proposal. We propose a method, based on Theorem 1, that provides new insights into the concrete security cost of PCD obtained from hash-based SNARKs. Specifically, we *can* idealize the hash-based PCD construction in a less straightforward way, resulting in a PCD construction in the ROM that, albeit not succinct, is covered by Theorem 1 and could be reasonably conjectured to capture the security of the original

[5]The random oracle methodology is widely used across cryptography to set the security parameters of protocols that rely on cryptographic hash functions (and possibly other cryptographic building blocks). The methodology must, nevertheless, be applied with caution because it does not work for every protocol [CGH04].

[6]Relativization is distinct from other limitations of SNARKs in the presence of oracles: [FN16] studies limitations of knowledge extraction for adversaries that access oracles exogenous to the SNARK scheme itself (e.g., the signing oracle of a signature scheme).

hash-based PCD construction. Of course, the security of hash-based PCD constructions merits further study beyond this work, given the delicate nature of heuristic instantiations of SNARKs in the ROM [BBHMR19].

Briefly, the hash-based PCD construction makes a *specific* non-black-box use of the underlying hash function: it uses the underlying hash-based SNARK to prove correct execution of computations that involve the hash function itself. In the idealization, we can model this as a non-succinct SNARK in the ROM where query-answer pairs to the random oracle of the proved computation are simply included in the argument string as claims to be checked directly by the verifier (which has access to the oracle). This relativized "non-succinct NARK" in the ROM directly leads to a (non-succinct) PCD construction in the ROM, whose concrete security follows from Theorem 1. This latter PCD construction in the ROM closely models the original hash-based PCD construction, and, in analogy to the random oracle methodology, one may conjecture that attacks against the hash-based PCD construction are no more effective than attacks against the idealized (non-succinct) PCD construction in the ROM sketched above.

The above steps are summarized in Figure 1.

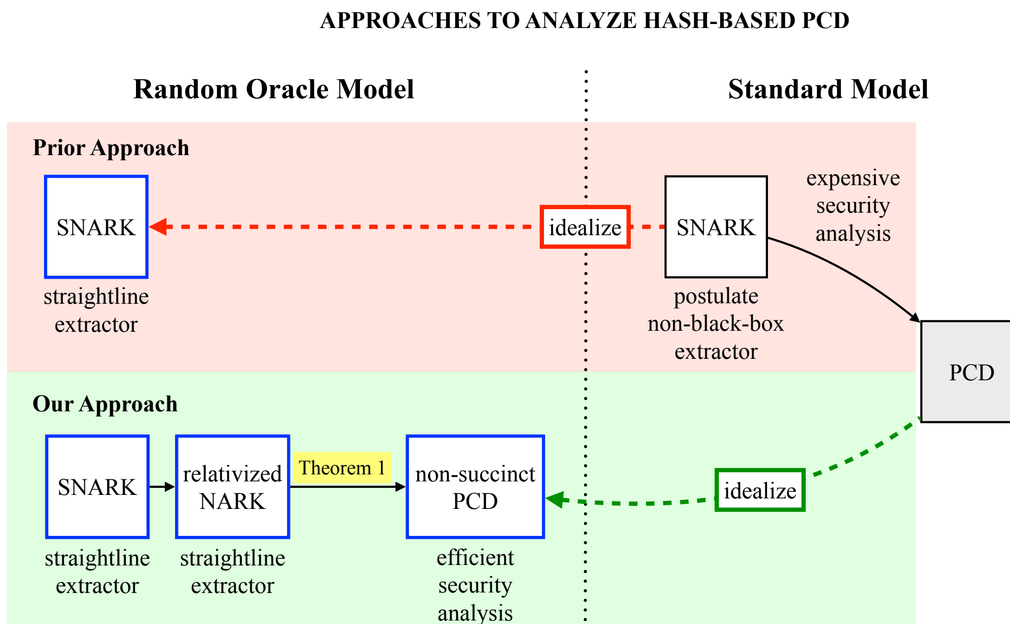

Figure 1: The grey box "PCD" on the right represents the hash-based PCD construction used in practice whose concrete security we wish to understand. *Top:* Prior work provides an expensive security analysis based on a hash-based SNARK, whose security is heuristically set by equating it to a corresponding idealized SNARK in the ROM (i.e., via the random oracle methodology). *Bottom:* We directly idealize the hash-based PCD construction, equating its security to a corresponding (non-succinct) PCD in the ROM whose security we establish.

In sum, our Theorem 1 provides new insights for practitioners: in the above heuristic sense (inspired by the random oracle methodology), one may conjecture that the concrete security of hash-based PCD constructions equals that of the underlying idealized SNARK in the ROM, matching widespread practices for recursive proof composition (and thus providing some justification for these practices).

See Section 2.3 for more discussion.

A-priori unknown N. In the (pure) ROM setting described above, the size of the adversary does not matter (only the query bound matters). Hence the knowledge soundness error simplifies to

$$\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$$

which is independent of N.[7] Therefore, with our improvement, suitably setting κ_{ARG} once suffices for *all* distributed computations (which may have arbitrarily large size that is unknown a priori), whereas with the prior results one would have to set κ_{ARG} depending on the pre-specified bound N on the size of the distributed computation. In Section 2.4 we describe a natural real-world example where there does not exist any pre-specified bound N on the size of a valid distributed computation.

[7]The extraction time is as before and, necessarily, depends on N, as the extractor outputs a distributed computation of size N.

2 Techniques

We overview the main ideas underlying our results. In Section 2.1 we discuss our improved security analysis for PCD constructed from relativized SNARKs with straightline extractors. In Section 2.2 we discuss how our improvement applies to prior relativized SNARKs in different oracle models. In Section 2.3 we discuss relativized "non-succinct NARKs" in the random oracle model, and implications to real-world constructions. In Section 2.4 we discuss a real-world compliance predicate that did not have security guarantees (in terms of knowledge soundness) prior to our work. In Section 2.5 we discuss how our security analysis extends to the case of straightline extractors that are probabilistic and may query the oracle.

2.1 Security analysis of PCD from straightline extractors

We elaborate on Theorem 1 and outline the main ideas of its proof. The technical details that make these discussions precise are provided in Sections 4 and 5.

Review: PCD. Proof-carrying data (PCD) is a cryptographic primitive that enables untrusted provers to efficiently demonstrate the correctness of a distributed computation. A distributed computation T is viewed as a directed acyclic graph in which each vertex is labeled with *local data* and each edge is labeled with a *message*; the computation *output* is the message on the lexicographically-first edge into a sink. Correctness is determined by a given *compliance predicate* ϕ: T is ϕ-compliant if, for every vertex in T, ϕ outputs 1 when given as input the vertex's output message, local data, and input messages. The *transcript size* and *transcript depth* of ϕ are the largest size and largest depth of any ϕ-compliant distributed computation T.[8] A *PCD scheme* is a tuple $\mathsf{PCD} = (\mathbb{P}, \mathbb{V})$ for proving/verifying ϕ-compliance of distributed computations, as follows.

- The PCD prover $\mathbb{P}$ receives an output message z, local data w_{loc}, and input messages $(z_i)_i$ together with PCD proofs $(\mathbb{\Pi}_i)_i$ (each proof $\mathbb{\Pi}_i$ attests to the ϕ-compliance of the corresponding message z_i), and produces a PCD proof $\mathbb{\Pi}$ for the ϕ-compliance of the output message z.
- The PCD verifier $\mathbb{V}$ receives a message z and PCD proof $\mathbb{\Pi}$, and outputs a decision bit.

The PCD scheme is complete if proofs for compliant messages are accepted by the PCD verifier. The PCD scheme is knowledge sound if every malicious PCD prover producing a message and proof accepted by the PCD verifier "knows" a compliant distributed computation whose output is that message, up to some error. This error is bounded by a knowledge soundness error function $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N)$, which depends on the security parameter λ, number of queries q by the adversary to the oracle, size s of the adversary, largest size N of any ϕ-compliant distributed computation, and other parameters that we omit here for simplicity. Our Theorem 1 establishes an improved bound on the knowledge soundness error $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N)$ of PCD schemes obtained from non-interactive arguments with straightline knowledge soundness. See Section 3.2 for a formal definition of a PCD scheme.

Limitations of PCD from SNARKs in the standard model. A PCD scheme in the standard model can be constructed from any SNARK (with adaptive security) in the standard model [BCCT13]. Informally, the PCD prover uses the SNARK prover to produce a short proof attesting that (i) the compliance predicate accepts the output message, local data, and input messages, and (ii) input messages carry valid SNARK proofs; the PCD verifier uses the SNARK verifier to check the proof accompanying a message.

However, the security analysis works only for compliance predicates with *constant* transcript depth. This is because SNARKs in the standard model satisfy a modest notion of adaptive knowledge soundness: *non-black-box knowledge soundness*. Informally, for every SNARK adversary there exists a knowledge

[8] In particular, the transcript size and transcript depth may or may not be bounded for a particular compliance predicate ϕ.

extractor, whose size is polynomially-related to the adversary size, such that, whenever the SNARK adversary convinces the SNARK verifier, the knowledge extractor outputs a valid witness (up to some error). If the polynomial blowup from adversary to extractor is $n \mapsto n^c$ then the security reduction for a PCD prover of size n would yield a PCD extractor of size (roughly) n^{c^d}, where d is the transcript depth of the compliance predicate.[9] This size blowup is huge in concrete terms, and asymptotically this requires d to be constant.

PCD from relativized SNARKs. In the *relativized setting*, we consider SNARKs that are constructed in an idealized model where the SNARK can prove/verify computations involving *the same oracle*. In other words, all (honest and malicious) parties have access to an oracle sampled according to a certain distribution and, in particular, the SNARK prover and SNARK verifier may query the oracle; crucially, the SNARK is required to work even for relations that are defined relative to the same oracle. See Section 3.1 for a formal definition of a relativized non-interactive argument in an oracle model.

The aforementioned canonical PCD construction in the standard model extends naturally to the relativized setting. Indeed, in the recursive step of the construction, the PCD prover produces a SNARK proof attesting the computation of the SNARK verifier, which in turn involves oracle calls. A relativized SNARK prover possesses the capability to generate such a proof. Furthermore, the security analysis of the relativized PCD construction can be carried over but presents the same blowup encountered in the standard model.

Enabler: straightline extraction. In the relativized setting, we have the additional benefit of a stronger knowledge soundness property: many SNARKs in oracle models have a (universal) *straightline extractor*, a notion that is uniquely defined within an oracle model and lacks an equivalent counterpart in the standard model. A straightline extractor does *not* get access to the malicious SNARK prover; instead, it produces a witness by examining the following: the instance and argument string output by the malicious SNARK prover, the sequence of queries to the oracle performed by the malicious SNARK prover, and the corresponding query answers. We refer to the list of query-answer pairs as the *query-answer trace* tr of the SNARK prover.

Definition 2.1 (informal)**.** $\mathsf{ARG} = (\mathcal{P}, \mathcal{V})$ *for a relation R has* **straightline knowledge soundness error** κ_{ARG} *if there exists a polynomial-time deterministic extractor $\mathcal{E}$ such that, for every security parameter $\lambda \in \mathbb{N}$ and q-query s-size prover $\widetilde{\mathcal{P}}$,*

$$\Pr\left[\begin{array}{c|c} (\mathbb{x}, \mathbb{w}) \notin R & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, \mathcal{V}^f(\mathbb{x}, \pi) = 1 & (\mathbb{x}, \pi) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}^f \\ & \mathbb{w} \leftarrow \mathcal{E}(\mathbb{x}, \pi, \mathsf{tr}) \end{array}\right] \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}) \ .$$

Above tr *denotes the query-answer trace of* $\widetilde{\mathcal{P}}$ *with the oracle* f.

The running time of the straightline extractor $\mathcal{E}$ does *not* depend on the running time of the malicious SNARK prover, but only on the number of query-answer pairs in the SNARK prover's trace tr (as well as the instance $\mathbb{x}$ and SNARK proof π). Straightline extractors are common in oracle models, and as we explain shortly *they will enable us to avoid the blowup in the PCD extractor size discussed above*.

We begin by sketching a (straightline) PCD extractor $\mathbb{E}$ that is naturally obtained from the given straightline SNARK extractor $\mathcal{E}$, by recursively extracting prior messages (and SNARK proofs), one vertex at a time. The straightline extractor $\mathbb{E}$ receives as input the compliance predicate ϕ, the message z_{out} and proof $\mathbb{\Pi}_{\mathrm{out}}$ output by the malicious PCD prover, and the query-answer trace tr of the malicious PCD prover; $\mathbb{E}$ aims to output a ϕ-compliant PCD transcript T whose output is z_{out}.

$\mathbb{E}(\phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}, \mathsf{tr})$:

[9] The dependence is on transcript depth rather than transcript size because the security reduction simultaneously extracts from all SNARK proofs at the same transcript depth (see, e.g., [COS20]).

1. Initialize a PCD transcript T as an empty graph.
2. Add to T vertices v_0 and v_1, and add to T the edge (v_1, v_0) with label $(z_{\text{out}}, \mathbb{\Pi}_{\text{out}})$.
3. Initialize an extraction queue $\mathcal{L}$ with the vertex v_1.
4. While the extraction queue $\mathcal{L}$ is not empty:
   (a) Pop the first vertex v from the queue $\mathcal{L}$.
   (b) Let $(z, \mathbb{\Pi})$ be the label of the unique outgoing edge of v.
   (c) Run the SNARK knowledge extractor $\mathcal{E}((\phi, z), \mathbb{\Pi}, \mathsf{tr})$ to obtain a witness $\mathbb{w}$.
   (d) Parse $\mathbb{w}$ to obtain local data w_{loc} and input messages and proofs $((z_i, \mathbb{\Pi}_i))_i$ for v.
   (e) Label v the vertex by w_{loc}.
   (f) For each message-proof pair $(z_i, \mathbb{\Pi}_i)$: add a new child vertex of v, label the new edge with $(z_i, \mathbb{\Pi}_i)$, and add the new vertex to the extraction queue $\mathcal{L}$.
5. Output the PCD transcript T.

For the rest of this section, let λ be the security parameter, q an upper bound on the number of oracle queries made by the malicious PCD prover $\widetilde{\mathbb{P}}$, s an upper bound on the size of the malicious PCD prover, and N an upper bound on the size of any ϕ-compliant transcript.

Security analysis inspired by prior work. Security analyses of PCD in prior works based on straightline extractors [CT10; CCGOS23] bound the knowledge error (roughly) as $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$ where $\mathsf{s}' := \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$. Intuitively, each recursion incurs the SNARK knowledge soundness error of $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$. In more detail, the i-th extraction is achieved by invoking the SNARK knowledge extractor for a corresponding i-th SNARK prover $\widetilde{\mathcal{P}}_i^f$, which outputs the message and proof in the label of the outgoing edge of the i-th vertex considered by $\mathbb{E}$. Note that $\widetilde{\mathcal{P}}_i^f$ runs in time $\mathsf{s} + O(i \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$ because, to perform the extraction associated to the i-th vertex, it first has to perform the extractions associated to the first $i-1$ vertices (as $\mathbb{E}$ does). Using a union bound, the success probability of the PCD adversary can be upper bounded by the sum of the success probabilities of all these argument adversaries (one per vertex in the transcript), yielding the aforementioned upper bound $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \lambda, \mathsf{q}, \mathsf{s}')$.

This bound (obtained via straightline knowledge extraction) is a significant improvement over the exponential blow-up incurred when only relying on non-black-box knowledge extraction, and as discussed, is achieved by prior works on PCD in oracle models [CT10; CCGOS23]. However, the multiplicative factor of N impacts concrete security, and, in fact, is unacceptable when N is unknown a priori. (There are compliance predicates deployed in the real world for which N is unknown a priori; see Section 2.4.)

Our security analysis. We improve the security analysis of PCD from straightline knowledge extraction: we avoid paying for the multiplicative factor of N, bounding the PCD knowledge soundness error as $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$ where $\mathsf{s}' := \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$. Intuitively, we force the SNARK adversary to pinpoint the problematic vertex (if any) in a PCD distributed computation transcript T by running the PCD adversary $\widetilde{\mathbb{P}}$ *once*.

In particular, our SNARK adversary $\widetilde{\mathcal{P}}$ follows the PCD knowledge extractor $\mathbb{E}$. If the PCD transcript T extracted by $\mathbb{E}$ is not compliant with the predicate ϕ, there must exist at least one problematic vertex in T. The SNARK adversary $\widetilde{\mathcal{P}}$ reconstructs T and searches for this problematic vertex along the way.

$\widetilde{\mathcal{P}}^f$:
1. Run the PCD adversary $\widetilde{\mathbb{P}}^f$ to obtain its output $(\phi, z_{\text{out}}, \mathbb{\Pi}_{\text{out}})$, and its query-answer trace tr.
2. Initialize a PCD transcript T as an empty graph.
3. Add to T vertices v_0 and v_1, and add to T the edge (v_1, v_0) with label $(z_{\text{out}}, \mathbb{\Pi}_{\text{out}})$.
4. Initialize an extraction queue $\mathcal{L}$ with the vertex v_1.
5. While $\mathcal{L}$ is not empty:

(a) Pop the first vertex v from the queue $\mathcal{L}$.
(b) Let $(z, \mathbb{\Pi})$ be the label of the unique outgoing edge of v.
(c) Run the SNARK knowledge extractor $\mathcal{E}((\phi, z), \mathbb{\Pi}, \mathsf{tr})$ to obtain a witness $\mathbb{w}$.
(d) Parse $\mathbb{w}$ to obtain local data w_{loc} and input messages and proofs $((z_i, \mathbb{\Pi}_i))_i$ for v.
(e) For each message-proof pair $(z_i, \mathbb{\Pi}_i)$: add a new child vertex of v, label the new edge with $(z_i, \mathbb{\Pi}_i)$, and add the new vertex to the extraction queue $\mathcal{L}$.
(f) If at least one of following is true, output $(z, \mathbb{\Pi})$:
  i. $\phi(z, w_{\mathrm{loc}}, (z_i)_i) \neq 1$ (i.e., v is not ϕ-compliant).
  ii. There exists i such that $\mathcal{V}^f(z_i, \mathbb{\Pi}_i) \neq 1$ (i.e., the SNARK verifier rejects $(z_i, \mathbb{\Pi}_i)$).

6. If no output so far, output an arbitrary message-proof pair.

The malicious SNARK prover $\widetilde{\mathcal{P}}$ above follows the PCD extractor $\mathbb{E}$, with additional checks in Item 5f. If the PCD transcript T is not ϕ-compliant, there must be at least one vertex v in T such that the outgoing message of v is inconsistent with the incoming messages to v (and the local data at v), and the SNARK verifier does not catch this. In other words, computation at vertex v successfully fools the SNARK verifier. The label corresponding to the outgoing edge of the first such v (in the order $\mathbb{E}$ extracts) is output by $\widetilde{\mathcal{P}}$.

The number of queries made by $\widetilde{\mathcal{P}}$ equals the number of queries made by $\widetilde{\mathbb{P}}$, plus the additional queries by the SNARK verifier $\mathcal{V}$ in Item 5(f)ii (which is invoked at most N times). In fact, in the technical sections, we explain how to avoid this latter additive cost, observing that it suffices to run only a "part" of the SNARK verifier $\mathcal{V}$ that does not query the oracle (see details in Section 5).

Lastly, the size of $\widetilde{\mathcal{P}}$ is the sum of the size s of $\widetilde{\mathbb{P}}$, the size of the N invocations of the argument extractor $\mathcal{E}$ (and some processing in between), all of which is upper bounded by $\mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))$.

We conclude $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s}')$ as desired.

The preprocessing setting. For wider applicability of our results, in the technical sections, we work in a more general setting. We consider SNARKs (and PCD) in the *preprocessing model*, which means that an additional algorithm known as the *indexer* may do an offline computation on the "offline" part of the instance, producing a corresponding proving key and verification key to be used for proving and verifying proofs.

2.2 Application: improved concrete security for black-box PCD constructions

We discuss two oracle models from prior work where one can construct relativized SNARKs with straightline knowledge soundness. Our Theorem 1 yields a security bound for PCD schemes obtained in these oracle models that is a significant improvement over previously-known bounds.

Arithmetized random oracle. [CCGOS23] constructs PCD in the *arithmetized random oracle model* (AROM), which is an idealization of capabilities associated to the arithmetization of a hash function. In this model, all parties have access to a random oracle, which as usual can be viewed as the idealization of some concrete hash function h; in addition, all parties have access to an associated arithmetization oracle, which can be viewed as an idealization of a low-degree polynomial p_h that "encodes" the circuit of h.

Briefly, [CCGOS23] shows that queries to the AROM can be "accumulated", and they show how this implies that any SNARK in the ROM can be transformed into a relativized SNARK in the AROM; moreover, the relativized SNARK has a straightline extractor if the given SNARK has a straightline extractor. In turn, this implies a construction of PCD in the AROM (that makes a black-box use of the AROM).

The analysis in [CCGOS23] implies an error bound for PCD that is (roughly) $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$ where κ_{ARG} is the (straightline) knowledge soundness error of the underlying relativized SNARK in the AROM. Our Theorem 1 improves the error bound for PCD to $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$.

Signed random oracle. [CT10] constructs PCD in an oracle model that combines the random oracle model and a signature scheme, which here we refer to as the *signed random oracle model* (SROM). All parties have access to an oracle that, on a new input x, samples a random answer y, generates a signature σ on (x, y) under a secret signing key embedded in the oracle, and outputs (y, σ); repeated inputs have the same answers.

Intuitively, this model facilitates a PCD construction because the SNARK verifier does not need to query the oracle: to check that the oracle answers x with (y, σ), one can verify that σ is a valid signature on the message (x, y) using the oracle's public key; there is no need to query the oracle at x.

More generally, any SNARK in the ROM (with straightline extraction) directly implies a relativized SNARK in the SROM (with straightline extraction), up to an error that depends on the security of the signature scheme. To prove an oracle computation, invoke the prover of the SNARK in the ROM on the computation where all oracle calls are replaced with sub-computations that verify signatures on the relevant messages; to verify the corresponding SNARK proof, invoke the verifier of the SNARK in the ROM.

The aforementioned relativized SNARK implies a corresponding PCD construction in the SROM (which is essentially the one studied in [CT10] but reinterpreted through the relativization lens). The analysis in [CT10] implies an error bound that is (roughly) $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$ where κ_{ARG} is the (straightline) knowledge soundness error of the underlying relativized SNARK in the SROM. Our Theorem 1 improves this error bound to $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$.

Remark 2.2. [CCS22] constructs PCD in the *low-degree random oracle model (LDROM)*, where all parties have access to a random low-degree extension of a random oracle. Specifically they construct a relativized SNARK in the LDROM, and from there obtain PCD in the LDROM. However, the relativized SNARK in [CCS22] is only shown to have a rewinding extractor, and because of this they show security of the PCD construction only for compliance predicates with constant transcript depth. Constructing a straightline extractor for the relativized SNARK in the LDROM in [CCS22] (or, indeed, any relativized SNARK in the LDROM) remains an open problem, which precludes our Theorem 1 from use in the LDROM setting.

2.3 Application: a paradigm to set security for hash-based PCD

Relativized SNARKs in the ROM do not exist [BCG24]. Nevertheless, one can achieve a weak form of relativized SNARKs in the ROM that implies a corresponding weak form of PCD in the ROM (using no assumptions or heuristics), for which our Theorem 1 gives concrete security bounds. This weak form of PCD in the ROM can be (heuristically) viewed as an idealization of an important class of (succinct) hash-based PCD constructions used in practice. In turn, we gain new insights into the concrete security of these hash-based PCD constructions. We elaborate on this below.

"Weak" relativized SNARKs in the ROM. One can construct relativized SNARKs in the ROM for relations decidable via computations that perform few queries to the random oracle. The construction below remains secure regardless of the number of queries. However, if the number of queries to the oracle is large, then the resulting argument system is a relativized "non-succinct NARK" rather than a relativized SNARK.

Suppose we have a non-relativized SNARK in the ROM [Mic00; BCS16; CY21a; CY21b; CY24] with proof size ℓ. Consider an oracle relation $R^{\mathcal{U}}$ whose decision involves q queries to the random oracle. We can construct a SNARK in the ROM for $R^{\mathcal{U}}$ with proof size $\ell + O(\mathsf{q} \cdot \lambda)$, where λ is the output size of the random oracle. We modify the circuit that checks whether a given instance-witness pair is in the relativized relation: remove each oracle gate and instead read a corresponding query-answer pair from an augmented instance (which now additionally stores the list of all query-answer pairs for the computation). The new circuit is proved using the given non-relativized SNARK in the ROM; and the resulting SNARK proof of size ℓ is

accompanied by the list of query-answer pairs, increasing its size to $\ell + O(\mathsf{q} \cdot \lambda)$. The new SNARK verifier checks the SNARK proof and checks that the list of query-answer pairs is consistent with the random oracle.

Say that the non-relativized SNARK $(\mathcal{P}_1, \mathcal{V}_1)$ is for the circuit satisfiability relation R_{CSAT}. We construct a (weak) relativized SNARK $(\mathcal{P}_2, \mathcal{V}_2)$ for the oracle relation $R^f_{\mathrm{CSAT}} := \{(C, \mathbb{x}, \mathbb{w}) : C^f(\mathbb{x}, \mathbb{w}) = 1\}$.

- $\mathcal{P}_2^f(C, \mathbb{x}, \mathbb{w})$:
  1. Run $C^f(\mathbb{x}, \mathbb{w})$ to obtain its query-answer trace tr_C.
  2. Construct the new (non-oracle) circuit C' that, on input $((\mathbb{x}, \mathsf{tr}_C), \mathbb{w})$, computes $C(\mathbb{x}, \mathbb{w})$ by answering C's queries to f with the query-answer pairs in tr_C.
  3. Run the SNARK prover for R_{CSAT}: $\pi \leftarrow \mathcal{P}_1^f(C', (\mathbb{x}, \mathsf{tr}_C), \mathbb{w})$.
  4. Output (π, tr_C).
- $\mathcal{V}_2^f(C, \mathbb{x}, (\pi, \mathsf{tr}_C))$:
  1. Construct the new (non-oracle) circuit C' from C like $\mathcal{P}_2$ does.
  2. Check that $\mathcal{V}_1^f(C', (\mathbb{x}, \mathsf{tr}_C), \pi) = 1$.
  3. Check that tr_C is consistent with f (by directly querying f for each query in tr_C).

The security of the SNARK for R^f_{CSAT} follows from the security of the SNARK for R_{CSAT}. Specifically, the transformation *preserves straightline extraction*: if the SNARK for R_{CSAT} has a straightline extractor (with a knowledge error),[10] then so does the SNARK for R^f_{CSAT} constructed above (with the same knowledge error).

"Weak" PCD in the ROM. The weak relativized SNARK $(\mathcal{P}_2, \mathcal{V}_2)$ in the ROM directly leads to a weak PCD scheme $(\mathbb{P}, \mathbb{V})$ in the ROM. The PCD construction invokes the SNARK for relations that involve the SNARK verifier, which makes a small number of queries to the random oracle. Hence the above relativized SNARK can be used to recursively prove the correctness of the SNARK verifier. Our Theorem 1 provides a bound on the knowledge soundness error of the resulting PCD scheme, thanks to the straightline knowledge soundness of the SNARK. However, with each recursive step, proof size increases, leading to a PCD construction that is *not succinct*, aligning with the limitations of PCD in the ROM [CL20; HN23; BCG24].

The silver lining. There is a silver lining between the limitations of PCD in the ROM and the aforementioned non-succinct construction of PCD in the ROM, which improves our understanding of security bounds in practice. Specifically, *the non-succinct construction of PCD in the ROM described above can be viewed as an idealization of succinct hash-based constructions of PCD in practice*, as we now explain.

The random oracle methodology tells us that the (non-relativized) SNARK $(\mathcal{P}_1, \mathcal{V}_1)$ in the ROM can be viewed as an idealization of a hash-based SNARK $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$ in the standard model, namely, the scheme $(\mathcal{P}_1, \mathcal{V}_1)$ where the random oracle is instantiated via a concrete hash function. Crucially, in a similar (though formally distinct) way, we can view $(\mathcal{P}_2, \mathcal{V}_2)$ as an idealization of $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$ when used to prove computations that involve calls to the concrete hash function. Indeed, $(\mathcal{P}_2, \mathcal{V}_2)$ equals $(\mathcal{P}_1, \mathcal{V}_1)$ up to the fact that calls to the random oracle are included as explicit input-output claims in the output argument string.

Next, let $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ be the hash-based PCD scheme in the standard model that is obtained by recursively composing the hash-based SNARK $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$. The PCD scheme $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ is the real-world hash-based construction whose concrete security we wish to understand. The key point in this discussion is that we can view the weak PCD scheme $(\mathbb{P}, \mathbb{V})$ in the ROM mentioned above as an idealization of $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$. This is because:
- in the standard model, $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ is obtained via recursive composition of $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$;
- in the ROM, $(\mathbb{P}, \mathbb{V})$ is obtained via recursive composition of $(\mathcal{P}_2, \mathcal{V}_2)$;
- $(\mathcal{P}_2, \mathcal{V}_2)$ is an idealization of $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$ when used for computations involving calls to the hash function.

[10] Achieving straightline extraction in the ROM is straightforward; see prior work [Mic00; BCS16; CY21a; CY21b; CY24].

In sum, it may be reasonable to set the security parameters of $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ according to the security parameters of $(\mathbb{P}, \mathbb{V})$ (whose security is establish by our Theorem 1). This is tantamount to conjecturing that attacks against $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ are no more effective than attacks against $(\mathbb{P}, \mathbb{V})$. More precisely, either an attack against $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ reduces to an attack against $(\mathbb{P}, \mathbb{V})$ (inheriting its security), or an attack (usefully) exploits the instantiation (and non-black-box use of) the concrete hash function, which remains an open problem.

The above reasoning is summarized in Figure 1 (and compared to prior approaches). This approach avoids the need to replace the random oracle with a hash function in the middle of the construction and its analysis; instead, all heuristics are deferred to the very end. (Deferring all heuristics to the very end has several advantages, articulated in [CCS22]; indeed, other works achieving PCD in oracle models [CT10; CCGOS23] also benefit from the ability to defer any heuristics till the end.)

Instantiating the SNARK. The aforementioned PCD construction in the ROM is based on a given SNARK in the ROM. There are several SNARKs in the ROM (with unconditional security) [Mic00; BCS16; CY21a; CY21b; CY24]. These constructions follow a common paradigm: they compile a probabilistic proof (a PCP or an IOP) into a SNARK by using a vector commitment scheme in the ROM and other ROM techniques. If the underlying probabilistic proof has a straightline extractor (the vast majority of relevant probabilistic proofs do) then the resulting SNARK in the ROM has one as well. Indeed, these constructions preserve straightline extractability (e.g., the vector commitment scheme in the ROM is straightline extractable).

For example, the SNARK in the ROM in [BBHR19] (known as *STARK*) is widely deployed in practice (along with various optimizations), including with recursion [SW22]. It is based on an IOP that is far more practical than any known PCP (and admits a straightline extractor). By our Theorem 1, the resulting PCD from the relativized version of the IOP-based SNARK has knowledge soundness error

$$\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}) \leq \mathsf{q} \cdot \kappa_{\mathsf{IOP}} + \frac{4\mathsf{q}^2}{2^\lambda} \ ,$$

where κ_{IOP} is the (straightline) knowledge soundness error of the IOP. Note that in the ROM, κ_{ARG} does not depend on the adversary size.

Our bound provides partial justification for the security parameters for PCD based on this SNARK used in practice. (This was one of our initial motivations to study concrete security bounds for PCD in the ROM.)

Remark 2.3 (compatibility with zero knowledge)**.** Recursively composing a zero-knowledge SNARK (that can prove correctness of its own verifier) yields zero-knowledge PCD. Concrete bounds on the zero-knowledge error of the PCD construction (in terms of the zero-knowledge error of the underlying SNARK) are known, including for the hash-based constructions of interest to us [COS20]. (Analyzing zero knowledge is "easy" because only the last recursion matters.)

Our proposal for a heuristic security analysis of the knowledge soundness error of hash-based PCD constructions is compatible with zero knowledge, in the following sense. Observe that if the (non-relativized) SNARK $(\mathcal{P}_1, \mathcal{V}_1)$ is zero knowledge, the transformation from $(\mathcal{P}_1, \mathcal{V}_1)$ to the relativized NARK $(\mathcal{P}_2, \mathcal{V}_2)$ does not necessarily maintain zero knowledge (due to the inclusion of query-answer pairs in the argument string). However, this is not a problem because our goal is to establish security bounds on the knowledge soundness error: the (non-succinct) PCD construction $(\mathbb{P}, \mathbb{V})$ obtained from $(\mathcal{P}_2, \mathcal{V}_2)$ remains an idealization of the hash-based PCD construction $(\hat{\mathbb{P}}, \hat{\mathbb{V}})$ that is obtained via recursive composition of $(\hat{\mathcal{P}}_1, \hat{\mathcal{V}}_1)$ (the hash-based instantiation of $(\mathcal{P}_1, \mathcal{V}_1)$ that heuristically remains zero knowledge). Hence, our proposal, in particular, also could be used as a guide for parameters of hash-based PCD constructions that are zero knowledge.

2.4 Example: a real-world compliance predicate with unknown size and depth bound

We describe a compliance predicate with unbounded transcript size and depth that is an illustrative simplification of a compliance predicate deployed in a real-world application. Prior security analyses of PCD constructions do not provide any security guarantees (in terms of knowledge soundness) for such predicates. In contrast, the discussion in Section 2.3 for the ROM (where, in particular, adversary size does not matter) explains how our Theorem 1 can be used to partially justify security parameters currently used in practice for this example. We elaborate on this below.

Motivation: recursive STARKs. Computation in the Ethereum smart contract system is expensive: informally, each computation step is re-executed by every node in the network, and so the system charges users for each computation step that they want to execute (e.g., by calling a smart contract). A class of architectures known as *layer 2 proof-based rollups* [E23] moves computation off-chain, in the sense that users send their computation requests to an aggregator who then periodically produces a SNARK proof about batches of user computations; the Ethereum smart contract system then verifies the SNARK proof and makes a state transition reflecting all the computations in the batch. The SNARK's succinctness property ensures that checking a SNARK proof is exponentially cheaper than checking the computation it attests to. These savings in on-chain computation are the motivation behind layer 2 proof-based rollups.

Producing SNARK proofs for large batches is expensive, but efficiency can be improved if the SNARK proof is itself produced via a PCD distributed computation that involves separately proving and aggregating small sub-computations, following a "proof tree" approach common in PCD applications [Val08; BCCT13]. This approach is taken by several systems, including one produced by StarkWare [GPR21; SW22].

Informally, a smart contract on Ethereum [SW21] is a PCD verifier that enables the recursive proof composition of "STARK proofs" according to a compliance predicate described below.[11] The users submit computation requests by providing a piece of code to run and an input for it, which are the local data in the distributed computation. Messages, on the other hand, are hashes of outputs of computations.

In that system, security in the recursive composition of the STARK is assumed to equal the security of a standalone (non-recursive) use of the STARK (i.e., no security loss is accounted for in the security reduction from PCD to the STARK). Is this assumption (at least heuristically) justified?

The compliance predicate. As mentioned in Section 2.1, a compliance predicate receives as input, for a given vertex v in the graph (of the distributed computation), an output message z, some local data w_{loc}, and (in the recursive case) a list of input messages $(z_i)_i$. Let $h\colon \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant hash function, M be a universal Turing machine (on input a program P and an input x, M outputs $P(x)$), and $T \in \mathbb{N}$ be a maximum time bound. Below we describe a compliance predicate $\phi_{h,M,T}\colon \{0,1\}^* \to \{0,1\}$.

- *Formats:*
  - Local data w_{loc} is a tuple (P, x), where P is a program and x is an input.
  - A message z is a pair (y, t), where y is a claimed output (or hash value) and t is a time bound.
- *Base case: v is a source vertex.*

  $\phi_{h,M,T}(z, w_{\mathrm{loc}}, \bot)$:
  1. Parse z as (y, t).
  2. Parse w_{loc} as (P, x).
  3. Check that $t \le T$, $M(P, x) = y$, and $M(P, x)$ runs in t steps.
- *Recursive case: v is an internal node.*

  $\phi_{h,M,T}(z, w_{\mathrm{loc}}, (z_i)_i)$:

[11] A STARK (as deployed in that system) is the heuristic instantiation (via the random oracle methodology) of a SNARK in the ROM, with straightline knowledge soundness, that is based on a certain IOP.

1. Parse z as (y, t).
2. Check that $t = 0$ and $w_{\mathrm{loc}} = \bot$.
3. For each i, parse z_i as (y_i, t_i) and check that $t_i \leq T$.
4. Check that $h((y_i)_i) = y$.

The base case in $\phi_{h,M,T}$ represents user computation requests, and the recursive case represents aggregation. In practice, h is set to a concrete hash function (e.g., blake2s in [SW21]), M is set to a specific universal machine (e.g., a machine that executes Cairo instructions [GPR21]), and T to some large upper bound.

Moreover, $\phi_{h,M,T}$ does not impose any bound on the depth (or size) of a compliant PCD distributed computation: given $\lambda, T \in \mathbb{N}$, $\phi_{h,M,T}$ allows a chain of computations of any length, independent of λ and T, to be aggregated together. In particular, given a batch of base cases, it is possible to combine them in any arbitrary way by hashing the outputs of their computations for some unknown number of times.

In sum, *no prespecified upper bound N would support this compliance predicate*.

Sketch of the application. We outline how the aggregator uses a PCD scheme $\mathsf{PCD} = (\mathbb{P}, \mathbb{V})$ relative to the compliance predicate $\phi_{h,M,T}$. Consider two computation requests (P_1, x_1, t_1) and (P_2, x_2, t_2), where P_i is a program, x_i is an input, and t_i is a time bound. The aggregator uses the PCD prover $\mathbb{P}$ to generate proofs π_1, π_2 attesting to the $\phi_{h,M,T}$-compliance of $(y_1, t_1), (y_2, t_2)$ respectively, where $y_1 := P(x_1), y_2 := P(x_2)$. Then, using these messages and proofs, the aggregator again uses the PCD $\mathbb{P}$ to create a proof $\mathbb{\Pi}$ attesting to the $\phi_{h,M,T}$-compliance of $(y, 0)$, where $y := h((y_1, y_2))$. More generally, the aggregator can generate a single compliance proof for a batch of computation requests $((P_i, x_i, t_i))_i$ as follows.

1. Compute the proofs for each user request (base case): For each i:
   - $y_i := M(P_i, x_i)$.
   - $\mathbb{\Pi}_i \leftarrow \mathbb{P}(\phi_{h,M,T}, (y_i, t_i), (P_i, x_i), \bot)$
2. Compute the proofs for the second layer: For each neighboring pair (i, j) of the computation requests:
   - $\mathbb{\Pi}_{i,j} \leftarrow \mathbb{P}(\phi_{h,M,T}, (h(y_i, y_j), 0), ((y_i, t_i), (y_j, t_j)), (\mathbb{\Pi}_i, \mathbb{\Pi}_j))$.
3. Keep aggregating proofs until there is only one node remaining. Output the last proof $\mathbb{\Pi}_{\mathrm{out}}$.

The above procedure always aggregates two proofs. In practice, users may submit computation requests in a streaming fashion. Hence, the above process can be generalized to handle streaming requests by greedily aggregating the requests submitted together (more than 2 requests can be handled at once). The resulting PCD transcript is not necessarily a binary tree.

This procedure is useful in multiple scenarios. One of them is when one or multiple users submit a series of computation requests: it is possible to construct *one* SNARK proof for all the computations so that the Ethereum smart contract can verify this proof and update the states all together. More specifically, the aggregator divides all requests into smaller batches, and compliance predicate $\phi_{h,M,T}$ is able to handle each batch in parallel and combine them together into one proof even when the computations are not the same.

Prior work vs. our result in this application. Prior security analyses of PCD constructions (both from non-black-box extractors and from straightline extractors as in [CT10; CCGOS23]) establish upper bounds on the PCD knowledge soundness error that depend on the transcript size and/or depth. Hence no security guarantees (for knowledge soundness) are provided for the compliance predicate $\phi_{h,M,T}$ deployed in [SW21] described above. Moreover, even if $\phi_{h,M,T}$ were modified to impose some pre-specified large transcript size/depth, the security loss depending on these parameters would have to be accounted for, which would cause a corresponding (and possibly large!) increase in the security parameters used in that system. (In other words, the underlying STARK would have to be much more secure to account for this loss.)

In contrast, our Theorem 1 establishes an upper bound on the PCD knowledge soundness error that applies to the compliance predicate $\phi_{h,M,T}$; indeed, in the ROM the adversary size does not matter, so the

upper bound does not depend on either transcript size or depth (see the end of Section 1.1.2). In turn, since the underlying recursive STARKs are a (heuristic) PCD construction obtained from SNARKs in the ROM, our discussion in Section 2.3, provides new insights for practitioners. Specifically, that discussion suggests the security achieved by a non-recursive one-shot STARK proof is inherited by the corresponding PCD scheme without any loss. This provides a heuristic justification for the current settings of parameters in that system.

2.5 Technical extension: a more general analysis

The notion of straightline extraction for SNARKs that we used so far imposes two requirements on the knowledge extractor $\mathcal{E}$ (see Definition 2.1): (i) $\mathcal{E}$ is deterministic; and (ii) $\mathcal{E}$ does not query the oracle. These requirements are typically fulfilled by straightline extractors for known relativized SNARKs. Under these requirements Theorem 1 yields the upper bound $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$.

We additionally ask: *how does the upper bound on κ_{PCD} change if we consider a notion of straightline extraction that relaxes either of these requirements?* Such relaxations can be useful; we give two examples.

- *Randomness.* The SNARK knowledge extractor $\mathcal{E}$ typically runs, as a subroutine, a knowledge extractor for an underlying probabilistic proof, which in turn may rely on a list-decoding algorithm for the error-correcting code used by the probabilistic proof. Randomness can be used to speed up list-decoding algorithms and, if $\mathcal{E}$ were required to be deterministic, those speedups would be ruled out.

- *Querying the oracle.* The SNARK knowledge extractor $\mathcal{E}$ may wish to query the oracle so to determine the decision bit of the SNARK verifier on the given instance and argument string (i.e., to compute $\mathcal{V}^f(\mathbb{x}, \pi)$).

In light of the above, we additionally give a general security analysis that upper bounds the PCD knowledge soundness error κ_{PCD} without assuming either of the requirements, broadening the applicability of our result. While this analysis results in slightly larger bounds, the bounds remain significant improvements over the prior state of the art.

Below, we elaborate on how we handle each relaxation individually; in the technical sections, we handle both relaxations simultaneously (in which case the different types of upper bounds combine in a single upper bound for both). Fix the security parameter λ. The error κ_{ARG} primarily depends on the number of queries made by the adversary and the size of the adversary, which is the focus of the reasoning below. (The error κ_{ARG} depends also on other values; we refer the reader to Section 7 for all technical details.)

- *Probabilistic extractors.* The basic analysis described in Section 2.1 that leads to the upper bound $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq N \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$ can be adapted to hold for probabilistic extractors. For the i-th invocation of the SNARK extractor $\mathcal{E}$, we consider a corresponding malicious argument prover $\widetilde{\mathcal{P}}$ that outputs the message-proof pair in the label of the unique outgoing edge of the i-th vertex considered by $\mathbb{E}$. The extraction error for each malicious argument prover is upper bounded by $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$ (as the argument prover $\widetilde{\mathcal{P}}$ runs the q-query s-size PCD prover $\widetilde{\mathbb{P}}$ to obtain the output of the PCD transcript and then invokes the argument extractor for at most N times along with some post-processing). Finally, the PCD knowledge soundness error follows from a union bound.

  This bound is essentially tight. The knowledge soundness error of $\mathbb{E}$ may come either from the choice of oracle or from $\mathcal{E}$'s randomness. The latter case is something that, intuitively, must be paid N times, once per extraction; and it might be that, say, half of the knowledge soundness error is due to $\mathcal{E}$'s randomness. (Each invocation of $\mathcal{E}$ has an independent error from other invocations of $\mathcal{E}$, so the errors accumulate.)

  More generally, we could consider a definition of straightline knowledge soundness for the SNARK that separates a global error κ_{ARG} due to the oracle and a local error ϵ due to $\mathcal{E}$'s randomness. In this case, the upper bound would be $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}))) + N \cdot \epsilon$.

- *Extractors with oracle queries.* Suppose that the SNARK extractor $\mathcal{E}$ makes q' queries to the oracle. In the PCD extractor $\mathbb{E}$, we need to account for a query-answer trace that grows with each invocation of the SNARK extractor $\mathcal{E}$. Indeed, as per the definition of straightline knowledge soundness of the SNARK, each invocation of the SNARK extractor $\mathcal{E}$ takes in the query-answer trace of the corresponding malicious SNARK prover, which in the security reduction is an algorithm that runs the malicious PCD prover *plus* prior executions of $\mathcal{E}$ (each of which contributes new queries).

  A basic analysis here would establish a soundness error of roughly $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \sum_{i=1}^{N} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q} + i \cdot \mathsf{q}', \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$. Indeed, after i extractions, the query-answer trace for the execution of the i-th argument adversary, which initially has length at most q due to the malicious PCD prover, increases by at most $i \cdot \mathsf{q}'$. A union bound over all vertices gives the aforementioned bound.

  A more careful analysis establishes a (tight) bound of roughly $\kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}, \mathsf{s}, N) \leq \sum_{i=1}^{N} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q} + d_i \cdot \mathsf{q}', \mathsf{s} + O(N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q})))$, where d_i is the depth of the vertex associated with the i-th extraction in the extracted PCD transcript T (which can be exponentially smaller than size). Indeed, an extracted PCD transcript T is a tree; when using the SNARK extractor $\mathcal{E}$ for a vertex v, the query-answer trace that "matters" for v only needs to include (the basic query-answer trace of the malicious PCD prover and) the queries and answers made by extractions *on the path from v to the root*. Hence, by giving each $\mathcal{E}$ only the query-answer traces it needs, the number of queries made by argument adversaries depends only on the depth of the PCD transcript.

3 Preliminaries

Definition 3.1. *An* **indexed relation** R *is a set of tuples* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *where* $\mathbb{i}$ *is the index,* $\mathbb{x}$ *the instance, and* $\mathbb{w}$ *the witness. The corresponding* **indexed language** $L(R)$ *is the set of pairs* $(\mathbb{i}, \mathbb{x})$ *for which there exists a witness* $\mathbb{w}$ *such that* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$.

Definition 3.2. *For a distribution over oracles* $\mathcal{U}$*, an* **oracle indexed relation** $R^{\mathcal{U}}$ *is a set of indexed relations* $\{R^f : f \in \mathcal{U}\}$.

Definition 3.3. *The* **query-answer trace** *of an algorithm* A *with oracle access to* $f \in \mathcal{U}(\lambda)$ *is a list* tr *of query-answer pairs that includes the queries made by* A *along with the corresponding answers by the oracle. We write* $z \xleftarrow{\mathsf{tr}} A^f$ *to mean that* A*, given oracle* f*, outputs* z *and has query-answer trace* tr.

3.1 Non-interactive arguments in oracle models

We provide notation and definitions for (preprocessing) non-interactive arguments as used in this paper. We do not describe the soundness property, as we always use a knowledge soundness property.

Definition 3.4. *A* **(preprocessing) non-interactive argument** *relative to an oracle distribution* $\mathcal{U}$ *for an oracle indexed relation* $R^{\mathcal{U}}$ *is a tuple of algorithms* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *that works as follows.*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: On input a security parameter* λ *(in unary), the generator* $\mathcal{G}$ *samples public parameters* pp.
- $\mathcal{I}^f(\mathsf{pp}, \mathbb{i}) \to (\mathsf{ipk}, \mathsf{ivk})$*: On input the public parameters* pp *and an index* $\mathbb{i}$ *for the relation* R^f*, the indexer* $\mathcal{I}$ *deterministically computes index-specific proving and verification keys* $(\mathsf{ipk}, \mathsf{ivk})$.
- $\mathcal{P}^f(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \to \pi$*: On input an index-specific proving key* ipk*, an instance* $\mathbb{x}$*, and a corresponding witness* $\mathbb{w}$*, the prover* $\mathcal{P}$ *computes an argument string* π *that attests to the claim that* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R^f$.
- $\mathcal{V}^f(\mathsf{ivk}, \mathbb{x}, \pi) \to b$*: On input an index-specific verification key* ivk*, and an instance* $\mathbb{x}$*, and a corresponding argument string* π*, the verifier* $\mathcal{V}$ *outputs a decision a bit* b.

Definition 3.5 (Perfect completeness)**.** *For every security parameter* $\lambda \in \mathbb{N}$ *and adversary* $\mathcal{A}$*,*

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

Definition 3.6 (Straightline knowledge soundness)**.** ARG *has* **(straightline) knowledge soundness error** κ_{ARG} **with extraction time** $\mathsf{t}_{\mathsf{ARG}}$ *if there exists a* **deterministic** *extractor* $\mathcal{E}$ *such that, for every security parameter* $\lambda \in \mathbb{N}$*, auxiliary input distribution* $\mathcal{D}$*, query bound* $\mathsf{q}_{\widetilde{\mathcal{P}}} \in \mathbb{N}$*, size bound* $\mathsf{s}_{\widetilde{\mathcal{P}}} \in \mathbb{N}$*,* $\mathsf{q}_{\widetilde{\mathcal{P}}}$*-query* $\mathsf{s}_{\widetilde{\mathcal{P}}}$*-size deterministic circuit* $\widetilde{\mathcal{P}}$*, index size bound* $n \in \mathbb{N}$*, and instance size bound* $k \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{c|r} & f \leftarrow \mathcal{U}(\lambda) \\ |\mathbb{i}| \le n & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \wedge\ |\mathbb{x}| \le k & \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin R^f & (\mathbb{i}, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}^f(\mathsf{pp}, \mathsf{ai}) \\ \wedge\ \mathcal{V}^f(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^f(\mathsf{pp}, \mathbb{i}) \\ & \mathbb{w} \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi, \mathsf{tr}) \end{array}\right] \le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathcal{P}}}, \mathsf{s}_{\widetilde{\mathcal{P}}}, n, k) \ ,$$

and $\mathcal{E}$ *runs in time* $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathcal{P}}}, n, k)$.

Remark 3.7. The auxiliary input distribution $\mathcal{D}$ can be the uniform random distribution. In that case the auxiliary input ai is a uniform random string, which enables the argument adversary $\widetilde{\mathcal{P}}$ to be randomized. In other words, $\widetilde{\mathcal{P}}$ is deterministic relative to the auxiliary input ai.

3.2 Proof-carrying data in oracle models

We provide notation and definitions for (preprocessing) proof-carrying data as used in this paper. This requires first introducing definitions for PCD transcripts and compliance.

Definition 3.8. *A* **(PCD) transcript** T *is a directed acyclic graph where each vertex* $u \in V(\mathrm{T})$ *is labeled by local data* $w_{\mathrm{loc}}^{(u)}$ *and each edge* $e \in E(\mathrm{T})$ *is labeled by a message* $z^{(e)} \neq \bot$*. The* **output** *of a transcript* T*, denoted* $\mathsf{out}(\mathrm{T})$*, is the message* $z^{(e)}$ *where* $e = (u, v)$ *is the lexicographically-first edge such that* v *is a sink.*

Definition 3.9. *A* **compliance predicate** ϕ *(with* M *input messages of size* l*) is an oracle boolean circuit that receives as input* 1 *output message of size at most* l*, some local data, and* M *input messages of size at most* l*, and outputs a decision bit. In particular,* ϕ *outputs* 0 *if more than* $M + 1$ *messages are given or any of the input messages are longer than* l *bits. We use* $|\phi|$ *to denote the size of the circuit* ϕ *and* $\mathsf{qnum}(\phi)$ *to denote the the number of queries by* ϕ *to the oracle function.*

Definition 3.10. *Let* $\mathcal{U}$ *be an oracle distribution and let* Φ *be a class of compliance predicates. Consider* $f \in \mathcal{U}$ *and* $\phi \in \Phi$*. Given a transcript* T*, a vertex* $u \in V(\mathrm{T})$ *is* (ϕ, f)**-compliant** *if the following holds for every outgoing edge* $e = (u, v) \in E(\mathrm{T})$ *from* u*:*

- (base case) *if* u *has no incoming edges,* $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(u)}, (\bot)) = 1$*;*
- (recursive case) *if* u *has incoming edges* $e_1, \ldots, e_M$*,* $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(u)}, (z^{(e_1)}, \ldots, z^{(e_M)})) = 1$.

The transcript T *is* (ϕ, f)**-compliant** *if* $E(\mathrm{T}) \neq \emptyset$ *and every vertex* $u \in V(\mathrm{T})$ *is* (ϕ, f)*-compliant.*

Definition 3.11. *We define the depth, size, and arity of a PCD transcript* T.

- *The* **depth** $\mathsf{depth}(\mathrm{T})$ *is the number of vertices of the longest path in* T.
- *The* **size** $\mathsf{size}(\mathrm{T})$ *is the number of non-sink vertices in* T.
- *The* **arity** $\mathsf{arity}(\mathrm{T})$ *is the maximum number of incoming edges of any vertex in* T.

Definition 3.12. *We define the transcript depth, transcript size, and transcript arity of a compliance predicate* ϕ*. Let* $\mathcal{U}$ *be an oracle distribution.*

- *The* **transcript depth** *is*
$$\mathsf{tdepth}(\phi) := \max_{\substack{f \in \mathcal{U} \\ \mathrm{T} \text{ is } (\phi, f)\text{-compliant}}} \mathsf{depth}(\mathrm{T}) \ ;$$
- *The* **transcript size** *is*
$$\mathsf{tsize}(\phi) := \max_{\substack{f \in \mathcal{U} \\ \mathrm{T} \text{ is } (\phi, f)\text{-compliant}}} \mathsf{size}(\mathrm{T}) \ ;$$
- *The* **transcript arity** *is*
$$\mathsf{tarity}(\phi) := \max_{\substack{f \in \mathcal{U} \\ \mathrm{T} \text{ is } (\phi, f)\text{-compliant}}} \mathsf{arity}(\mathrm{T}) \ .$$

Definition 3.13. *A compliance predicate* ϕ *is* (Φ, N, D, M, S, Q)**-compatible** *if:* $\phi \in \Phi$*;* $\mathsf{tsize}(\phi) \leq N$*;* $\mathsf{tdepth}(\phi) \leq D$*;* $\mathsf{tarity}(\phi) \leq M$*;* $|\phi| \leq S$*; and* $\mathsf{qnum}(\phi) \leq Q$.

Definition 3.14. *A* **proof-carrying data scheme (PCD scheme)** *for a class of compliance predicates* Φ *relative to an oracle distribution* $\mathcal{U}$ *is a tuple of algorithms* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *that works as follows.*

- $\mathbb{G}(1^\lambda) \to \mathbb{pp}$*: On input a security parameter* λ *(in unary), the generator* $\mathbb{G}$ *samples public parameters* $\mathbb{pp}$*.*

- $\mathbb{I}^f(\mathbb{pp}, \phi) \to (\mathbb{ipk}, \mathbb{ivk})$*: On input the public parameters* $\mathbb{pp}$ *and the compliance predicate* ϕ*, the indexer* $\mathbb{I}$ *deterministically computes proving and verification keys* $(\mathbb{ipk}, \mathbb{ivk})$*.*

- $\mathbb{P}^f(\mathbb{ipk}, z, w_{\mathrm{loc}}, ((z_i, \mathbb{\Pi}_i))_{i\in[M]}) \to \mathbb{\Pi}$*: On input the proving key* $\mathbb{ipk}$*, a message* z*, a local data* w_{loc}*, and a list of incoming messages and proofs* $((z_i, \mathbb{\Pi}_i))_{i\in[M]}$*, the prover* $\mathbb{P}$ *outputs a new proof* $\mathbb{\Pi}$ *for the outgoing message* z*.*

- $\mathbb{V}^f(\mathbb{ivk}, z, \mathbb{\Pi}) \to b$*: On input the verification key* $\mathbb{ivk}$*, a message* z*, and a corresponding proof* $\mathbb{\Pi}$*, the verifier* $\mathbb{V}$ *computes a decision bit* b*.*

Definition 3.15 (Perfect completeness)**.** *For every security parameter* $\lambda \in \mathbb{N}$ *and adversary* $\mathbb{A}$*,*

$$\Pr\left[\begin{array}{c|r} \left(\begin{array}{c} \phi \in \Phi \\ \wedge \left((\wedge_{i=1}^M z_i = \bot) \vee (\wedge_{i=1}^M \mathbb{V}^f(\mathbb{ivk}, z_i, \mathbb{\Pi}_i) = 1)\right) \\ \wedge\, \phi^f(z, w_{\mathrm{loc}}, (z_1, \ldots, z_M)) = 1 \end{array}\right) & f \leftarrow \mathcal{U}(\lambda) \\ & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ & (\phi, z, w_{\mathrm{loc}}, ((z_i, \mathbb{\Pi}_i))_{i=1}^M) \leftarrow \mathbb{A}^f(\mathbb{pp}) \\ \Downarrow & (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi) \\ \mathbb{V}^f(\mathbb{ivk}, z, \mathbb{\Pi}) = 1 & \mathbb{\Pi} \leftarrow \mathbb{P}^f(\mathbb{ipk}, z, w_{\mathrm{loc}}, ((z_i, \mathbb{\Pi}_i))_{i=1}^M) \end{array}\right] = 1 \ .$$

Definition 3.16 (Straightline knowledge soundness)**.** PCD *has* **(straightline) knowledge soundness error** κ_{PCD} **with extraction time** $\mathsf{t}_{\mathsf{PCD}}$ *if there exists a* **deterministic** *extractor* $\mathbb{E}$ *such that, for every security parameter* $\lambda \in \mathbb{N}$*, auxiliary input distribution* $\mathcal{D}$*, indexer query bound* $\mathsf{q}_{\mathbb{I}} \in \mathbb{N}$*, indexer size bound* $s_{\mathbb{I}}$*, adversary query bound* $\mathsf{q}_{\widetilde{\mathbb{P}}} \in \mathbb{N}$*, adversary size bound* $s_{\widetilde{\mathbb{P}}} \in \mathbb{N}$*,* $\mathsf{q}_{\widetilde{\mathbb{P}}}$*-query* $s_{\widetilde{\mathbb{P}}}$*-size deterministic circuit* $\widetilde{\mathbb{P}}$*, predicate size bound* $N \in \mathbb{N}$*, predicate depth bound* $D \in \mathbb{N}$*, predicate circuit size bound* $S \in \mathbb{N}$*, predicate query number bound* $Q \in \mathbb{N}$*, number of input edges bound* $M \in \mathbb{N}$*, and message size bound* $l \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{c|r} & f \leftarrow \mathcal{U}(\lambda) \\ \phi \textit{ is } (\Phi, N, D, M, S, Q)\textit{-compatible} & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \wedge\, |z_{\mathrm{out}}| \le l & \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ \wedge\, \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 & (\phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) \xleftarrow{\mathsf{tr}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai}) \\ \wedge\, (\mathrm{T} \textit{ is not } (\phi, f)\textit{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\mathrm{out}}) & (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi) \\ & \mathrm{T} \leftarrow \mathbb{E}(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}, \mathsf{tr}) \end{array}\right]$$
$$\le \kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\widetilde{\mathbb{P}}}, s_{\widetilde{\mathbb{P}}}, N, D, S, Q, M, l) \ ,$$

and $\mathbb{E}$ *runs in time* $\mathsf{t}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\widetilde{\mathbb{P}}}, N, D, S, Q, M, l)$*.*

4 From relativized ARG to PCD: construction

We describe how to construct a PCD scheme from a relativized non-interactive argument (Construction 4.2), and after that we describe how to construct a straightline PCD extractor from an underlying straightline non-interactive argument extractor (Construction 4.3).

These constructions are straightforward adaptations to the relativized case of prior constructions in the literature ([CT10; BCCT13; BCTV14; COS20]). Our main contribution is the security analysis of the PCD scheme (via this straightline PCD extractor), which we postpone to Section 5.

Let $\mathcal{U}$ be an oracle distribution. The definition below is a circuit used to realize the recursive composition.

Construction 4.1. Fix $\lambda \in \mathbb{N}$. Let $f \in \mathcal{U}(\lambda)$. Let $\mathcal{V}^{(\lambda,n,k)}$ be the circuit corresponding to the ARG verifier $\mathcal{V}$ with security parameter λ, checking indices of sizes at most n and instances of size at most k.

$[C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}]^f((\mathsf{ivk}, z_{\text{out}}), (w_{\text{loc}}, \vec{z}_{\text{in}}, \vec{\pi}_{\text{in}}))$:

1. Check that $\phi^f(z_{\text{out}}, w_{\text{loc}}, \vec{z}_{\text{in}}) = 1$.
2. If there exists i such that $(\vec{z}_{\text{in}}[i], \vec{\pi}_{\text{in}}[i]) \neq \bot$: check that $[\mathcal{V}^{(\lambda,n,k)}]^f(\mathsf{ivk}, (\mathsf{ivk}, \vec{z}_{\text{in}}[i]), \vec{\pi}_{\text{in}}[i]) = 1$ for every $i \in [M]$.

Construction 4.2 (PCD from ARG)**.** Let $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument for the oracle indexed relation $R_{\text{CSAT}}^{\mathcal{U}}$. We construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ as follows.

- $\mathbb{G}(1^\lambda)$:
  1. Sample public parameters $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$.
  2. Output $\mathbb{pp} := \mathsf{pp}$.
- $\mathbb{I}^f(\mathbb{pp}, \phi)$:
  1. Parse $\mathbb{pp}$ as pp.
  2. Construct the oracle recursion circuit $C := C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$.
  3. Compute the index key pair $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^f(\mathsf{pp}, C)$.
  4. Output $(\mathbb{ipk}, \mathbb{ivk}) := ((\mathsf{ipk}, \mathsf{ivk}), \mathsf{ivk})$.
- $\mathbb{P}^f(\mathbb{ipk}, z_{\text{out}}, (w_{\text{loc}}, \vec{z}_{\text{in}}, \vec{\mathbb{\pi}}_{\text{in}}))$:
  1. Parse the proving key $\mathbb{ipk}$ as $(\mathsf{ipk}, \mathsf{ivk})$.
  2. $\pi_{\text{out}} \leftarrow \mathcal{P}^f(\mathsf{ipk}, (\mathsf{ivk}, z_{\text{out}}), (w_{\text{loc}}, \vec{z}_{\text{in}}, \vec{\pi}_{\text{in}}))$.
  3. Output π_{out}.
- $\mathbb{V}^f(\mathbb{ivk}, z_{\text{out}}, \mathbb{\pi}_{\text{out}})$:
  1. Parse the verification key $\mathbb{ivk}$ as ivk.
  2. Parse the PCD proof $\mathbb{\pi}_{\text{out}}$ as an argument proof π_{out}.
  3. Check that $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z_{\text{out}}), \pi_{\text{out}}) = 1$.

Construction 4.3 (Knowledge extractor for PCD)**.** Let $\mathcal{E}$ be a straightline knowledge extractor for ARG. We construct a straightline knowledge extractor $\mathbb{E}$ for PCD as follows.

$\mathbb{E}(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\text{out}}, \mathbb{\pi}_{\text{out}}, \mathsf{tr})$:

1. Parse $\mathbb{pp}$ as pp.
2. Parse $\mathbb{\pi}_{\text{out}}$ as π_{out}.

3. Parse $\mathbb{ivk}$ as ivk.
4. Initialize graph $\mathrm{T} = (V, E)$ where $V = \{v_0, v_1\}$ and $E = \{(v_1, v_0)\}$.
5. Label the edge (v_1, v_0) by $(z_{\mathrm{out}}, \pi_{\mathrm{out}})$.
6. Initialize the extraction queue $\mathcal{L} \coloneqq (v_1)$.
7. Set $\mathbb{i} \coloneqq C^{(\lambda,M,n,k)}_{\mathcal{V},\phi}$.
8. While $\mathcal{L}$ is non-empty:
   (a) Let v be the first vertex in $\mathcal{L}$, remove the first vertex v from $\mathcal{L}$.
   (b) Let $z^{(e)}$ and $\pi^{(e)}$ be the message and proof in the label of the unique outgoing edge e from v.
   (c) Let $(\mathbb{i}_v, \mathbb{x}_v, \pi_v) \coloneqq (\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$.
   (d) Run the argument extractor $\mathbb{w}_v \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}_v, \mathbb{x}_v, \pi_v, \mathsf{tr})$.
   (e) Parse $\mathbb{w}_v$ as $(w^{(v)}_{\mathrm{loc}}, \vec{z}^{(v)}_{\mathrm{in}}, \vec{\mathbb{\Pi}}^{(v)}_{\mathrm{in}})$.
   (f) Label v in T by $w^{(v)}_{\mathrm{loc}}$.
   (g) For every j such that $\vec{z}^{(v)}_{\mathrm{in}}[j] \neq \perp$:
      i. Add a new vertex v' to V, and an edge (v', v) in E.
      ii. Parse $\vec{\mathbb{\Pi}}^{(v)}_{\mathrm{in}}[j]$ as $\pi^{(v',v)}$.
      iii. Add the label $(\vec{z}^{(v)}_{\mathrm{in}}[j], \pi^{(v',v)})$ to the edge (v', v) in T.
      iv. Add v' to $\mathcal{L}$.
9. Output the augmented transcript T.

5 From relativized ARG to PCD: security reduction

In Section 4, we described how to construct a PCD scheme from a relativized non-interactive argument, and how to construct a straightline PCD extractor from an underlying straightline non-interactive argument extractor. In this section, we give our security analysis of the PCD scheme, via this straightline PCD extractor.

Let $\mathcal{U}$ be an oracle distribution. Suppose that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a non-interactive argument for the oracle CSAT relation $R_{\mathrm{CSAT}}^{\mathcal{U}}$ with straightline knowledge soundness error $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$ and extraction time $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$. Recall that $\lambda \in \mathbb{N}$ denotes the security parameter, $S \in \mathbb{N}$ the bound on the predicate circuit size, $M \in \mathbb{N}$ the bound on the arity of the PCD transcript, and l the bound on the size of each message. We define an index size bound n and instance size bound k:

- $n := \mathsf{nsize}(\lambda, S, M, l)$, where $\mathsf{nsize}(\cdot)$ is carefully defined in Lemma 5.9; and
- $k := |\mathsf{ivk}| + l$.

Theorem 5.1. *The PCD scheme* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *constructed from* ARG *using Construction 4.2 has straightline knowledge soundness error* $\kappa_{\mathsf{PCD}} = \kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, s_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$ *and extraction time* $\mathsf{t}_{\mathsf{PCD}} = \mathsf{t}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$ *such that*

$$\kappa_{\mathsf{PCD}} \le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k) \ ,$$
$$\mathsf{t}_{\mathsf{PCD}} \le N \cdot (\mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n,k)) + \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}, n, k)) \ ,$$

where

- $\mathsf{q}_{\tilde{\mathcal{P}}} := \mathsf{q}_{\tilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}} + N \cdot Q$;
- $\mathsf{s}_{\tilde{\mathcal{P}}} := s_{\tilde{\mathbb{P}}} + s_{\mathbb{I}} + N \cdot S + N \cdot \mathsf{vsize}(\lambda, n, k) + \mathsf{t}_{\mathsf{PCD}}$, *where* $\mathsf{vsize}(\lambda, n, k)$ *is the size of the argument verifier* $\mathcal{V}$ *when invoked with security parameter* λ*, index of size* n*, and instance of size* k*.*
- $\mathsf{arglen}(\lambda, n, k)$ *is the size of the argument proof* π *outputted by the argument prover when invoked with security parameter* λ*, index of size* n*, and instance of size* k*.*

We analyze the knowledge soundness error in Section 5.1 and the extraction time in Section 5.2.

Remark 5.2 (the information-theoretic setting)**.** We discuss a special case of Theorem 5.1 that yields an even stronger result in a notable setting. Suppose that ARG has straightline knowledge soundness error κ_{ARG} that does not depend on the size of the adversary $\mathsf{s}_{\tilde{\mathcal{P}}}$; for example, this is the case in the "pure" random oracle setting discussed in Section 6.3, where adversaries may be computationally unbounded (and are limited only in the number of queries to the random oracle). Suppose further that the compliance predicate ϕ does not query the oracle ($Q = 0$); this is a common case as typically the oracle appears only in the argument verifier (not part of the compliance predicate). In this case the knowledge error is as follows:

$$\kappa_{\mathsf{PCD}} \le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}, n, k) \ .$$

5.1 Knowledge soundness error

The analysis below is stated for non-interactive arguments with split verification (Definition 5.3), a property that holds essentially without loss of generality (Remark 5.4).

Definition 5.3. $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *has* **split verification** *if the argument string contains a list of query-answer pairs* tr *that the verifier* $\mathcal{V}$ *checks during verification. More precisely,* $\mathcal{V}$ *can be written as follows:*

$\mathcal{V}^f(\mathsf{ivk}, \mathbb{x}, \pi)$*:*

1. *Check that* $\mathsf{VerifyProof}(\mathsf{ivk}, \mathbb{x}, \pi) = 1$.
2. *Check that* $\mathsf{VerifyTrace}^f(\pi) = 1$.

The above subroutines are defined as follows:

- $\mathsf{VerifyProof}(\mathsf{ivk}, \mathbb{x}, \pi)$*: Parse* π *as* (π', tr) *and check that* $\mathcal{V}^{\mathsf{tr}}(\mathsf{ivk}, \mathbb{x}, \pi') = 1$. *(Output* 0 *if* $\mathcal{V}$ *makes a query* q *that is not contained in* tr.
- $\mathsf{VerifyTrace}^f(\pi)$*: Parse* π *as* (π', tr), *and check that, for every* $(q, a) \in \mathsf{tr}$, $f(q) = a$.

Remark 5.4. Any non-interactive argument system can be modified to satisfy Definition 5.3, by augmenting the argument string with the list of query-answer pairs to be made by the argument verifier. In particular, the examples in Section 6 can have split verifiers. We assume split verification throughout this proof.

Let $\widetilde{\mathbb{P}}$ be a $\mathsf{q}_{\widetilde{\mathbb{P}}}$-query PCD prover. Our goal is to upper bound the following expression:

$$\Pr\left[\begin{array}{c} \phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \leq l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\mathrm{out}}) \end{array} \middle| \begin{array}{r} f \leftarrow \mathcal{U}(\lambda) \\ \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ (\phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) \xleftarrow{\mathsf{tr}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai}) \\ (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi) \\ \mathrm{T} \leftarrow \mathbb{E}(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}, \mathsf{tr}) \end{array}\right] . \quad (1)$$

As explained in Section 2.1, we construct an argument prover $\widetilde{\mathcal{P}}$.

$\widetilde{\mathcal{P}}^f(\mathsf{pp}, \mathsf{ai})$:

1. Set $\mathbb{pp} := \mathsf{pp}$.
2. Run $(\phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) \xleftarrow{\mathsf{tr}_{\widetilde{\mathbb{P}}}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai})$.
3. Run $(\mathbb{ipk}, \mathbb{ivk}) \xleftarrow{\mathsf{tr}_{\mathbb{I}}} \mathbb{I}^f(\phi)$.
4. Set $\mathsf{tr} := \mathsf{tr}_{\widetilde{\mathbb{P}}} \parallel \mathsf{tr}_{\mathbb{I}}$.
5. Parse the PCD proof $\mathbb{\Pi}_{\mathrm{out}}$ as an argument string π_{out}.
6. Parse the PCD verification key $\mathbb{ivk}$ as an argument verification key ivk.
7. Initialize a graph $\mathrm{T} = (V, E)$ where $V = \{v_0, v_1\}$ and $E = \{(v_1, v_0)\}$.
8. Label the edge (v_1, v_0) by $(z_{\mathrm{out}}, \pi_{\mathrm{out}})$.
9. Initialize the extraction queue as $\mathcal{L} := (v_1)$.
10. Set $\mathbb{i} := C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$.
11. While the extraction queue $\mathcal{L}$ is non-empty:
    (a) Let v be the first vertex in $\mathcal{L}$, remove the first vertex v from $\mathcal{L}$.
    (b) Let $z^{(e)}$ and $\pi^{(e)}$ be the message and proof in the label of the unique outgoing edge e from v.
    (c) Let $(\mathbb{i}_v, \mathbb{x}_v, \pi_v) := (\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$.
    (d) Run the argument extractor $\mathbb{w}_v \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}_v, \mathbb{x}_v, \pi_v, \mathsf{tr})$.
    (e) Parse $\mathbb{w}_v$ as $(w_{\mathrm{loc}}^{(v)}, \vec{z}_{\mathrm{in}}^{(v)}, \vec{\mathbb{\Pi}}_{\mathrm{in}}^{(v)})$.
    (f) Label the vertex v in T by $w_{\mathrm{loc}}^{(v)}$.
    (g) For every j such that $\vec{z}_{\mathrm{in}}^{(v)}[j] \neq \bot$:
        i. Add a new vertex v' to V, and an edge (v', v) in E.
        ii. Parse $\vec{\mathbb{\Pi}}_{\mathrm{in}}^{(v)}[j]$ as $\pi^{(v',v)}$.
        iii. Add the label $(\vec{z}_{\mathrm{in}}^{(v)}[j], \pi^{(v',v)})$ to the edge (v', v) in T.
        iv. Add the vertex v' to the extraction queue $\mathcal{L}$.

(h) Let $(v_1, \ldots, v_M)$ be the child-vertices just added for v (maybe there are less than M child-vertices of v in T, but the exact number does not matter as long as it is upper-bounded by M).

(i) Let e be the outgoing edge of v, check if at least one of the following is true:

i. $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(v)}, (z^{(v_1,v)}, \ldots, z^{(v_M,v)})) \neq 1$.

ii. There exists $i \in [M]$ such that $\mathsf{VerifyProof}(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_i,v)}), \pi^{(v_i,v)}) \neq 1$, where $\mathsf{VerifyProof}$ is as defined in Definition 5.3.

If the check passes, then output $(\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$.

12. Output $(\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$ where e is the topologically first edge in E. (This is a default output.)

Consider an example execution of the above prover. Suppose the PCD extractor $\mathbb{E}$ produces a transcript with depth 3 and 7 vertices as shown in Fig. 2, the red numbering on the edges indicate the extraction order of the vertices. Suppose that $v_{2,1}$ in Fig. 2 is the first problematic vertex (in the extraction order by $\mathbb{E}$). The malicious prover $\widetilde{\mathcal{P}}$ defined above would output the labels corresponding to $v_{2,1}$, as shown in Fig. 3. Notice that $\widetilde{\mathcal{P}}$ does not call the argument extractor $\mathcal{E}$ on vertices after $v_{2,1}$.

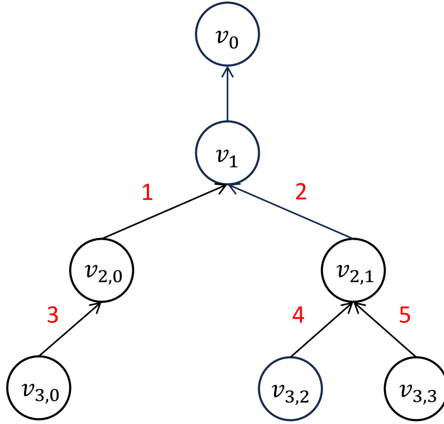

Figure 2: An example transcript extracted by $\mathbb{E}$ with the extraction order.

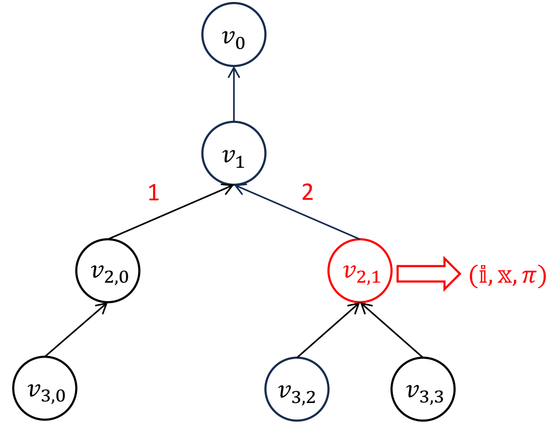

Figure 3: The behavior of the malicious prover $\widetilde{\mathcal{P}}$ for the transcript in Fig. 5.

The argument prover $\widetilde{\mathcal{P}}$ queries f when $\widetilde{\mathbb{P}}$, $\mathbb{I}$, and (each time it runs) ϕ. Hence the query complexity of $\widetilde{\mathcal{P}}$ can be upper bounded as follows:

$$\mathsf{q}_{\widetilde{\mathcal{P}}} \leq \mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}} + N \cdot Q \ .$$

Similarly, the size of $\widetilde{\mathcal{P}}$ can be upper bounded as follows:

$$\mathsf{s}_{\widetilde{\mathcal{P}}} \leq s_{\widetilde{\mathbb{P}}} + s_{\mathbb{I}} + N \cdot S + N \cdot \mathsf{vsize}(\lambda, n, |\mathsf{ivk}| + l) + \mathsf{t}_{\mathsf{PCD}} \ .$$

For the rest of the discussion, we consider the following experiment:

Experiment 1.

$$\left[\begin{array}{r} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ \mathbb{pp} \coloneqq \mathsf{pp} \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ (\phi, z_{\mathrm{out}}, \mathbb{\pi}_{\mathrm{out}}) \xleftarrow{\mathsf{tr}_{\widetilde{\mathbb{P}}}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai}) \\ (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\phi) \\ \mathrm{T} \leftarrow \mathbb{E}(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\mathrm{out}}, \mathbb{\pi}_{\mathrm{out}}, \mathsf{tr}_{\widetilde{\mathbb{P}}}) \\ (\mathbb{i}, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}_{\widetilde{\mathcal{P}}}} \widetilde{\mathcal{P}}^f(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{ivk}, \mathsf{ipk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i}) \\ \mathbb{w} \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi, \mathsf{tr}_{\widetilde{\mathcal{P}}}) \end{array}\right] .$$

To bound the probability in Eq. (1), we note that the condition $\mathsf{out}(\mathrm{T}) = z_{\mathrm{out}}$ always holds by Construction 4.3, so we focus on the probability that T is not (ϕ, f)-compliant. Intuitively, our goal is to reduce the probability of T being not (ϕ, f)-compliant to the probability that the argument prover $\widetilde{\mathcal{P}}$ constructed above successfully outputs an argument string that fools the argument verifier.

Towards this, we use a notion called *strong* (ϕ, f)*-compliance*, which requires all vertices to be (ϕ, f)-compliant and also every index-instance-proof tuple associated with an edge to be accepted by the argument verifier. If a transcript T is strongly (ϕ, f)-compliant then, in particular, it is (ϕ, f)-compliant.

Definition 5.5. *A transcript* $\mathrm{T} = (V, E)$ *is* **strongly** (ϕ, f)**-compliant** *if the following holds:*
- *For every vertex* $u \in V$*,* u *is* (ϕ, f)*-compliant.*
- *For every edge* $e \in E$*,* $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)}) = 1$.

Remark 5.6. The transcript T output by the PCD extractor $\mathbb{E}$ in Construction 4.3 is a tree, so every vertex $v \in V(\mathrm{T})$ has a unique outgoing edge. Definitions 3.10 and 5.5 can be simplified accordingly for the purpose of this discussion.

If the transcript T in Experiment 1 is not (ϕ, f)-compliant, then it must be the case that T is not strongly (ϕ, f)-compliant, which implies that there exists $v \in V(\mathrm{T})$ such that at least one of the following is true:

1. $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(v)}, (z^{(v_1,v)}, \ldots, z^{(v_M,v)})) \neq 1$,
2. $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_i,v)}), \pi^{(v_i,v)}) \neq 1$ for some $i \in [M]$,

where $e = (v, v')$ is the unique outgoing edge of v and $v_1, \ldots, v_M$ are child-vertices of v. Hence, by definition of $C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$ (Construction 4.1), $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \coloneqq (C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)}) \notin R_{\mathrm{CSAT}}^f$.

Let v be the first such vertex (in the order that $\mathbb{E}$ extracts). Let i be the iteration in which $\mathbb{E}$ extracts v. Since $\mathcal{E}$ is **deterministic**, we know that $\widetilde{\mathcal{P}}$ also extracts v in the i-th iteration, and the corresponding index-instance-proof tuple $(\mathbb{i}_v, \mathbb{x}_v, \pi_v) \coloneqq (\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$ for $e = (v, v')$ is the same as the one in $\mathbb{E}$. Hence, $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \notin R_{\mathrm{CSAT}}$ by the argument above. Moreover, since v is the first such vertex, we can deduce that $\mathcal{V}^f(\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)}) = 1$. If $(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$ is the tuple output by $\widetilde{\mathcal{P}}$, then from Definition 3.6,

$$\Pr\left[\begin{array}{c} \phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \le l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\pi}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\mathrm{out}}) \end{array}\right] \le \Pr\left[\begin{array}{c} |\mathbb{i}| \le n \\ \wedge\ |\mathbb{x}| \le k \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin R \\ \wedge\ \mathcal{V}^f(\mathsf{ivk}, \mathbb{x}, \pi) = 1 \end{array}\right]$$
$$\le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathcal{P}}}, \mathsf{s}_{\widetilde{\mathcal{P}}}, n, k) \ ,$$

where $n := \mathsf{nsize}(\lambda, S, M, l)$ ($\mathsf{nsize}(\cdot)$ is the circuit size defined in Lemma 5.9) and $k := |\mathsf{ivk}| + l$.

We are left to show that $\widetilde{\mathcal{P}}$ outputs $(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$. We proceed in two steps.

- Fix any $j < i$. We argue that $\widetilde{\mathcal{P}}$ does not output at iteration j. Let v_j be the vertex extracted at iteration j, and let $(\mathbb{i}_j, \mathbb{x}_j, \pi_j, \mathbb{w}_j)$ be the corresponding index, instance, proof, and witness of v_j. Let e_j be the unique outgoing edge of v_j, and let $v_{j,1}, \ldots, v_{j,M}$ be the child-vertices of v_j. Since we assume that v is the first vertex that "breaks" the strong compliance of T, it must be the case that
  - $\phi^f(z^{(e_j)}, w_{\mathrm{loc}}^{(e_j)}, (z^{(v_{j,1},v_j)}, \ldots, z^{(v_{j,M},v_j)})) = 1$; and
  - $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_{j,k},v_j)}), \pi^{(v_{j,k},v_j)}) = 1$ for all $k \in [M]$.

  Thus, for all $k \in [M]$ we know that $\mathsf{VerifyProof}(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_{j,k},v_j)}), \pi^{(v_{j,k},v_j)}) = 1$, which follows since $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_{j,k},v_j)}), \pi^{(v_{j,k},v_j)}) = 1$. Therefore, the checks in Item 11i cannot pass, and $\widetilde{\mathcal{P}}$ does not output at iteration j.

- We argue that $\widetilde{\mathcal{P}}$ outputs $(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$ at iteration i. Recall that v is the first vertex that "breaks" the strong compliance of T. Therefore, according to Definition 5.5, we distinguish between following two cases:
  - $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(v)}, (z^{(v_1,v)}, \ldots, z^{(v_M,v)})) \neq 1$: In this case, the first check in Item 11i passes, and $\widetilde{\mathcal{P}}$ outputs $(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$ as desired.
  - $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_k,v)}), \pi^{(v_k,v)}) \neq 1$ for some $k \in [M]$: We have either $\mathsf{VerifyProof}(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_i,v)}), \pi^{(v_i,v)}) \neq 1$, which makes the second check in Item 11i pass as desired; or $\mathsf{VerifyTrace}^f(\pi^{(v_k,v)}) \neq 1$, which cannot happen if the PCD verifier $\mathbb{V}$ accepts $(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}})$.

5.2 Extraction time bound

We prove the upper bound on extraction time claimed in Theorem 5.1.

Proof. For every depth d and i-th vertex at depth d, we construct an argument prover $\widetilde{\mathcal{P}}_{d,i}$ for the invocation of the argument extractor $\mathcal{E}$ for the i-th vertex at depth d. (We consider v_1 to be the 0-th vertex at depth 1.)

Construction 5.7. The argument prover $\widetilde{\mathcal{P}}_{1,0}$ corresponds to the first invocation of $\mathcal{E}$ in $\mathbb{E}$.

$\widetilde{\mathcal{P}}_{1,0}^f(\mathsf{pp}, \mathsf{ai})$:
1. Set $\mathbb{pp} := \mathsf{pp}$.
2. Run $(\phi, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) \leftarrow \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai})$.
3. Run $(\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi)$.
4. Parse $\mathbb{ivk}$ as ivk.
5. Parse $\mathbb{\Pi}_{\mathrm{out}}$ as π_{out}.
6. Set $\mathbb{i} := C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$.
7. Output $(\mathbb{i}, (\mathsf{ivk}, z_{\mathrm{out}}), \pi_{\mathrm{out}})$.

Construction 5.8. The (recursively defined) argument prover $\widetilde{\mathcal{P}}_{d,i}$ corresponds to the invocation of $\mathcal{E}$ for the i-th vertex at depth d for $d > 1$ and $0 \leq i < M^{d-1}$ in $\mathbb{E}$.

$\widetilde{\mathcal{P}}_{d,i}^f(\mathsf{pp}, \mathsf{ai})$:
1. Let $(\mathrm{parent}, \mathrm{pos}) := (\lfloor i/M \rfloor, i \bmod M)$.
2. Run $(\mathbb{i}, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}_{d-1,\mathrm{parent}}^f(\mathsf{pp}, \mathsf{ai})$.

3. If $(\mathbb{i}, \mathbb{x}, \pi) = \bot$, halt and output $\bot$.
4. Run the argument extractor $\mathbb{w} \leftarrow \mathcal{E}(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi, \mathsf{tr})$.
5. Parse $\mathbb{w}$ as $(w_{\mathrm{loc}}, \vec{z}_{\mathrm{in}}, \vec{\mathbb{m}}_{\mathrm{in}})$.
6. Parse $\mathbb{x}$ as (ivk, z).
7. If $\vec{z}_{\mathrm{in}}[\mathrm{pos}] = \bot$, output $\bot$.
8. Otherwise, output $(\mathbb{i}, (\mathsf{ivk}, \vec{z}_{\mathrm{in}}[\mathrm{pos}]), \vec{\mathbb{m}}_{\mathrm{in}}[\mathrm{pos}])$.

The running time of $\mathbb{E}$ can be upper bounded in terms of the running time of $\mathcal{E}$ and extra processing time for the outputs of $\mathcal{E}$. Specifically, for every d and i, let $\mathsf{q}_{d,i}$ be the number of queries made by $\widetilde{\mathcal{P}}_{d,i}$, $n_{d,i}$ and $k_{d,i}$ be the sizes of the index and instance output by $\widetilde{\mathcal{P}}_{d,i}$. The running time of the argument extractor $\mathcal{E}$ when invoked for the i-th vertex v at depth d is, by definition, at most

$$\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{d,i}, n_{d,i}, k_{d,i}) \ .$$

We now upper bound $\mathsf{q}_{d,i}, n_{d,i}, k_{d,i}$.

Index size and instance size. Every $\widetilde{\mathcal{P}}_{d,i}$ outputs an instance of the form $\mathbb{x} = (\mathsf{ivk}, z)$, so $k_{d,i} \leq k = |\mathsf{ivk}(\lambda, n)| + l$. Every $\widetilde{\mathcal{P}}_{d,i}$ outputs the index $\mathbb{i} = C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$ (the recursive circuit), whose size is at most

$$\mathsf{csize}\,(\lambda, S, M, l, n) := S + O(M \cdot l) + M \cdot \mathsf{vsize}(\lambda, n, |\mathsf{ivk}(\lambda, n)| + l) \ .$$

Above:

- S is an upper bound on the predicate circuit size $|\phi|$;
- $O(M \cdot l)$ bounds the cost of going over all incoming messages;
- $\mathsf{vsize}(\lambda, n, k)$ is the size of $\mathcal{V}^{(\lambda,n,k)}$.

The below lemma gives a bound on the above expression, showing that $n_{d,i} \leq n = \mathsf{nsize}\,(\lambda, S, M, l)$.

Lemma 5.9 ([COS20, Lemma 11.8]). *Suppose that for every security parameter $\lambda \in \mathbb{N}$ and message size bound $l \in \mathbb{N}$, the ratio of verifier circuit size to index size $\frac{\mathsf{vsize}(\lambda,n,|\mathsf{ivk}(\lambda,n)|+l)}{n}$ is monotonically decreasing in n. Then there exists a size function $\mathsf{nsize}\,(\lambda, S, M, l)$ such that*

$$\forall \lambda, S, M, l \in \mathbb{N}, \mathsf{csize}\,(\lambda, S, M, l, \mathsf{nsize}\,(\lambda, S, M, l)) \leq \mathsf{nsize}\,(\lambda, S, M, l) \ .$$

Query bound on the argument adversary. The malicious PCD prover $\widetilde{\mathbb{P}}$ makes $\mathsf{q}_{\widetilde{\mathbb{P}}}$ queries to the oracle, and the PCD indexer $\mathbb{I}$ makes $\mathsf{q}_{\mathbb{I}}$ queries to the oracle. Therefore, the argument prover $\widetilde{\mathcal{P}}_{1,0}$ makes $\mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$ queries to the oracle; in fact, every argument prover $\widetilde{\mathcal{P}}_{d,i}$ makes $\mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$ queries to the oracle.

Therefore, the cost of running the argument extractor $\mathcal{E}$ across all its invocations in $\mathbb{E}$ is at most

$$N \cdot \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{d,i}, n_{d,i}, k_{d,i}) \ .$$

Additionally, during the execution of $\mathbb{E}$, for every vertex in the transcript, the extractor $\mathbb{E}$ reads the input messages and attaches the augmented label to the vertex, which takes at most $\mathsf{poly}(l, \mathsf{arglen}(n_{d,i}, k_{d,i}))$ bit operations, where $\mathsf{arglen}(n_{d,i}, k_{d,i})$ is the length of the argument proof in the label of the vertex. Also, the basic pop and push operations for a queue take at most $O(\log N)$ steps since there are at most $\mathsf{tsize}(\phi) \leq N$ extracted vertices. Therefore, the overhead is at most $N \cdot \mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n_{d,i}, k_{d,i}))$.

In conclusion, $\mathbb{E}$ runs in time

$$N \cdot (\mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n, k)) + \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}, \mathsf{psize}(\mathsf{depth}(\mathrm{T}, v)), n, k)) \ . \tag{2}$$

□

Remark 5.10. In the proof above, we construct $\sum_{i=0}^{D} M^i$ argument provers (Construction 5.7 and Construction 5.8). However, the number of vertices in a PCD transcript is at most N, which can be much smaller than $\sum_{i=0}^{D} M^i$. Why do we define argument provers that we do not use? The reason is that the structure of the transcript extracted by $\mathbb{E}$ is unknown before the end of the extraction. Hence, we have to consider all possible configurations of the PCD transcript. As an example, consider a compliance predicate whose transcript depth is 3. The above definition induces 7 malicious argument provers as shown in Fig. 4. Suppose that the PCD

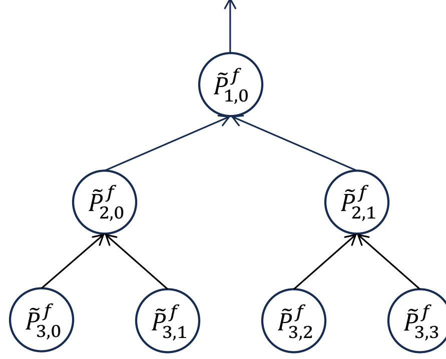

Figure 4: The malicious provers constructed as in Construction 5.7 and Construction 5.8 when transcript depth is at most 3.

extractor $\mathbb{E}$ extracts a transcript as described in Fig. 5: there are only four vertices $(v_{3,2}, v_{2,1}, v_1, v_0)$ in the transcript. The malicious argument provers corresponding to the dashed (nonexistent) vertices output $\perp$ according to the construction (see Fig. 6).

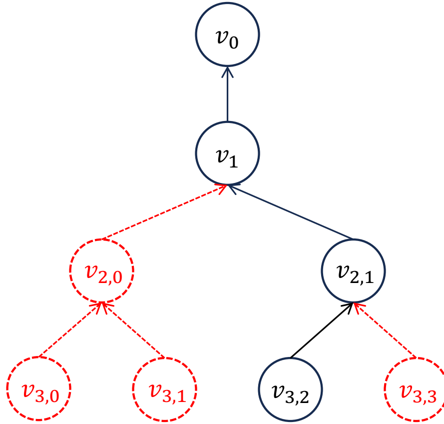

Figure 5: An example transcript extracted by $\mathbb{E}$ with 4 vertices. The vertices circled with red dashed lines are illustrations of missing vertices.

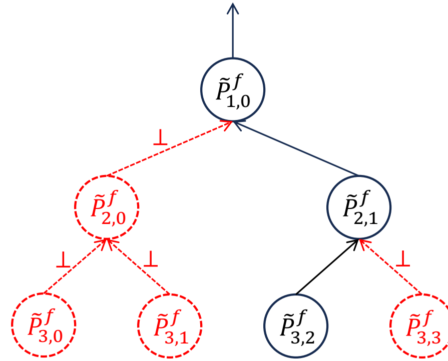

Figure 6: The behavior of the malicious provers for the transcript in Fig. 5. The provers circled with red dashed lines output $\perp$.

6 Relativized non-interactive arguments with straightline extraction

We describe three examples of relativized non-interactive arguments with straightline extraction in different oracle models: the arithmetized random oracle model [CCGOS23]; the signed random oracle model [CT10]; and a weak construction in any oracle model (which includes the random oracle model). For concreteness, we state results for the oracle CSAT relation.

Definition 6.1 (CSAT relation)**.** *The relation R_{CSAT} is defined as*

$$R_{\mathrm{CSAT}} \coloneqq \{(C, \mathbb{x}, \mathbb{w}) \colon C(\mathbb{x}, \mathbb{w}) = 1\} \ .$$

Definition 6.2 (CSAT oracle relation)**.** *For any $\lambda \in \mathbb{N}$, and any $f \in \mathcal{U}(\lambda)$, the relation R^f_{CSAT} is defined as*

$$R^f_{\mathrm{CSAT}} \coloneqq \{(C, \mathbb{x}, \mathbb{w}) \colon C^f(\mathbb{x}, \mathbb{w}) = 1\} \ .$$

6.1 The arithmetized random oracle model

In [CCGOS23], the authors construct a relativized SNARK with straightline extraction in the *arithmetized random oracle model* (AROM). Briefly, they show that any SNARK in the ROM can be transformed into a relativized SNARK in the AROM, while preserving straightline extraction; then applying this transformation to Micali's SNARK [Mic00] yields the desired relativized SNARK.

Theorem 6.3 ([CCGOS23])**.** *Let $\mathcal{U}$ be the distribution over functions corresponding to the arithmetized random oracle model. Assume the existence of collision-resistant hash functions in the standard model. Then there exists a (succinct) non-interactive argument relative to the oracle distribution $\mathcal{U}$ for the relation $R^{\mathcal{U}}_{\mathrm{CSAT}}$ with (straightline) knowledge soundness error $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$ that is negligible (if the input parameters are polynomially bounded in the security parameter λ).*

6.2 The signed random oracle model

In [CT10], the authors introduce the notion of PCD, and construct PCDs in the *signed random oracle model* (SROM). Implicit in their construction is a relativized SNARK in the SROM with straightline extraction, directly obtained by modifying Micali's SNARK where all oracle calls are replaced with sub-computations that verify signatures on the relevant messages.

Theorem 6.4 ([CT10])**.** *Let $\mathcal{U}$ be the distribution over functions corresponding to the signed random oracle model (realized with a secure signature scheme). There exists a (succinct) non-interactive argument relative to the oracle distribution $\mathcal{U}$ for the relation $R^{\mathcal{U}}_{\mathrm{CSAT}}$ with (straightline) knowledge soundness error $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$ that is negligible (if the input parameters are polynomially bounded in the security parameter λ).*

6.3 The random oracle model (and any other oracle model)

For any given oracle distribution, we describe how to transform any non-interactive argument into a relativized non-interactive argument at the expense of succinctness. The transformation preserves straightline extraction.

Theorem 6.5. *Let $\mathcal{U}$ be an oracle distribution. Let $\mathsf{ARG}_1 = (\mathcal{G}_1, \mathcal{I}_1, \mathcal{P}_1, \mathcal{V}_1)$ be a non-interactive argument for R_{CSAT} with straightline knowledge soundness error $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$ and extraction time*

$\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$. *Then* $\mathsf{ARG}_2 = (\mathcal{G}_2, \mathcal{I}_2, \mathcal{P}_2, \mathcal{V}_2)$ *in Construction 6.8 is a non-interactive argument for* $R^{\mathcal{U}}_{\mathrm{CSAT}}$ *with straightline knowledge soundness error* κ'_{ARG} *and extraction time* $\mathsf{t}'_{\mathsf{ARG}}$ *such that*

$$\kappa'_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k) \le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}} + O(n + \mathsf{q}_C), O(n + \mathsf{q}_C), k + \mathsf{q}_C) \ ,$$
$$\mathsf{t}'_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k) \le \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, O(n + \mathsf{q}_C), k + \mathsf{q}_C) + O(n + \mathsf{q}_C) \ ,$$

where q_C *is the number of queries made by the oracle circuit whose satisfiability is proved.*

The construction of ARG_2 from ARG_1 is in Construction 6.8 below. Before that, we define a subroutine for checking the correctness of query-answer traces (Definition 6.6) and how to modify a circuit to replace oracle gates with lookups into an augmented instance (Definition 6.7).

Definition 6.6. *Fix an oracle* $f \in \mathcal{U}(\lambda)$. *The algorithm* $\mathsf{TraceCheck}$, *given oracle access to* f *and given as input a query-answer trace* tr *(Definition 3.3), checks that* tr *is consistent with* f*: parse* tr *as a list of query-answer pairs* $((q_i, a_i))_i$ *and, for every* i, *check that* $f(q_i) = a_i$.

Definition 6.7. *Let* C *be an oracle circuit where oracle gates are for functions in* $\mathcal{U}(\lambda)$ *that receives as input an instance* $\mathbb{x}$ *and witness* $\mathbb{w}$, *and outputs a decision bit. The circuit* C' *takes in input an augmented instance* $(\mathbb{x}, \mathsf{tr})$ *and witness* $\mathbb{w}$, *and outputs a decision bit.*

> $C'((\mathbb{x}, \mathsf{tr}), \mathbb{w})$*: Run* $C(\mathbb{x}, \mathbb{w})$, *answering each oracle query* q *with* $\mathsf{tr}(q)$. *(Abort and output* 0 *if* tr *does not contain the query* q.*) Output the output of* $C(\mathbb{x}, \mathbb{w})$.

Note that $C^f(\mathbb{x}, \mathbb{w}) = 1$ *if and only if* $C'((\mathbb{x}, \mathsf{tr}), \mathbb{w})$ *outputs* 1 *and* tr *is such that* $\mathsf{TraceCheck}^f(\mathsf{tr}) = 1$.

Construction 6.8. Let $\mathsf{ARG}_1 = (\mathcal{G}_1, \mathcal{I}_1, \mathcal{P}_1, \mathcal{V}_1)$ be a non-interactive argument system for R_{CSAT}. We construct a non-interactive argument system $\mathsf{ARG}_2 = (\mathcal{G}_2, \mathcal{I}_2, \mathcal{P}_2, \mathcal{V}_2)$ for $R^{\mathcal{U}}_{\mathrm{CSAT}}$.

- $\mathcal{G}_2(1^\lambda)$: Sample public parameters $\mathsf{pp} \leftarrow \mathcal{G}_1(1^\lambda)$.
- $\mathcal{I}_2^f(\mathsf{pp}, \mathbb{i})$:
  1. Parse the index $\mathbb{i}$ as an oracle circuit C.
  2. Construct the (non-oracle) circuit C' from C as in Definition 6.7.
  3. Obtain the proving and verification key for the new circuit C': $(\mathsf{ipk}', \mathsf{ivk}') \leftarrow \mathcal{I}_1^f(\mathsf{pp}, C')$.
  4. Output $(\mathsf{ipk}, \mathsf{ivk}) := ((\mathsf{ipk}', C), \mathsf{ivk}')$.
- $\mathcal{P}_2^f(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$:
  1. Parse the proving key ipk as (ipk', C).
  2. Get the query-answer trace of the original circuit C given the instance and the witness: $(\cdot) \xleftarrow{\mathsf{tr}} C^f(\mathbb{x}, \mathbb{w})$.
  3. Compute the argument string $\pi' \leftarrow \mathcal{P}_1^f(\mathsf{ipk}', (\mathbb{x}, \mathsf{tr}), \mathbb{w})$.
  4. Output (π', tr).
- $\mathcal{V}_2^f(\mathsf{ivk}, \mathbb{x}, \pi)$:
  1. Parse π as (π', tr).
  2. Check that $\mathsf{TraceCheck}^f(\mathsf{tr}) = 1$.
  3. Check that $\mathcal{V}_1^f(\mathsf{ivk}, (\mathbb{x}, \mathsf{tr}), \pi') = 1$.

Perfect completeness of ARG_2 follows from perfect completeness of ARG_1: if $(C, \mathbb{x}, \mathbb{w}) \in R^f_{\mathrm{CSAT}}$ then $(C', (\mathbb{x}, \mathsf{tr}), \mathbb{w}) \in R_{\mathrm{CSAT}}$ where tr is the query-answer trace of $C^f(\mathbb{x}, \mathbb{w})$; hence $\mathsf{TraceCheck}$ accepts tr and $\mathcal{P}_1$ produces an argument string that is accepted by $\mathcal{V}_1$. We are left to argue knowledge soundness.

Knowledge soundness. Let $\widetilde{\mathcal{P}}_2$ be a $\mathsf{q}_{\widetilde{\mathcal{P}}}$-query $\mathsf{s}_{\widetilde{\mathcal{P}}}$-size adversary against ARG_2. We construct an adversary $\widetilde{\mathcal{P}}_1$ against ARG_1.

$\widetilde{\mathcal{P}}_1^f(\mathsf{pp}, \mathsf{ai})$:
1. Run $(C, \mathbb{x}, (\pi, \mathsf{tr})) \leftarrow \widetilde{\mathcal{P}}_2^f(\mathsf{pp}, \mathsf{ai})$.
2. Construct C' as in Definition 6.7.
3. Output $(C', (\mathbb{x}, \mathsf{tr}), \pi)$.

The number of queries made by $\widetilde{\mathcal{P}}_1$ is the same as that made by $\widetilde{\mathcal{P}}_2$, which is $\mathsf{q}_{\widetilde{\mathcal{P}}}$. The size of $\widetilde{\mathcal{P}}_1$ is at most $\mathsf{s}_{\widetilde{\mathcal{P}}} + O(n + \mathsf{q}_C)$ (where q_C is the query complexity of C), since constructing C' takes time $O(n + \mathsf{q}_C)$.

Let $\mathcal{E}_1$ be the (universal) straightline extractor for ARG_1. We construct a (universal) straightline extractor $\mathcal{E}_2$ for ARG_2.

$\mathcal{E}_2(\mathsf{pp}, C, \mathbb{x}, (\pi, \mathsf{tr}_C), \mathsf{tr})$:
1. Construct the oracle circuit C' from C according to Definition 6.7.
2. Compute $\mathbb{w} \leftarrow \mathcal{E}_1(\mathsf{pp}, C', (\mathbb{x}, \mathsf{tr}_C), \pi, \mathsf{tr})$.
3. Output $\mathbb{w}$.

Therefore,

$$\Pr\left[\begin{array}{c|r} |C| \le n & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, |\mathbb{x}| \le k & \mathsf{pp} \leftarrow \mathcal{G}_2(1^\lambda) \\ \wedge\, (C, \mathbb{x}, \mathbb{w}) \notin R^f & \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ \wedge\, \mathcal{V}_2^f(\mathsf{ivk}, \mathbb{x}, (\pi, \mathsf{tr}_C)) = 1 & (C, \mathbb{x}, (\pi, \mathsf{tr}_C)) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}_2^f(\mathsf{pp}, \mathsf{ai}) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}_2^f(\mathsf{pp}, C) \\ & \mathbb{w} \leftarrow \mathcal{E}_2(\mathsf{pp}, C, \mathbb{x}, (\pi, \mathsf{tr}_C), \mathsf{tr}) \end{array}\right]$$
$$\le \Pr\left[\begin{array}{c|r} |C'| \le n' & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, |(\mathbb{x}, \mathsf{tr}_C)| \le k' & \mathsf{pp} \leftarrow \mathcal{G}_1(1^\lambda) \\ \wedge\, (C', (\mathbb{x}, \mathsf{tr}_C), \mathbb{w}) \notin R & \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ \wedge\, \mathcal{V}_1^f(\mathsf{ivk}', (\mathbb{x}, \mathsf{tr}_C), \pi) = 1 & (C', (\mathbb{x}, \mathsf{tr}_C), \pi) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}_1^f(\mathsf{pp}, \mathsf{ai}) \\ & (\mathsf{ipk}', \mathsf{ivk}') \leftarrow \mathcal{I}_1^f(\mathsf{pp}, C') \\ & \mathbb{w} \leftarrow \mathcal{E}_1(\mathsf{pp}, C', (\mathbb{x}, \mathsf{tr}_C), \pi, \mathsf{tr}) \end{array}\right]$$
$$\le \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathcal{P}}}, \mathsf{s}_{\widetilde{\mathcal{P}}} + O(n + \mathsf{q}_C), n', k') \enspace ,$$

where $n' := |C'| = O(n + \mathsf{q}_C)$ and $k' := |\mathbb{x}| + |\mathsf{tr}_C| \le k + \mathsf{q}_C$. The first inequality follows from the construction of $\mathcal{V}_2$, and the second inequality follows from the knowledge soundness error of ARG_1 (Definition 3.6), which depends on the security parameter λ, the number of queries $\mathsf{q}_{\widetilde{\mathcal{P}}}$ made by the adversary $\widetilde{\mathcal{P}}_1$, the size $\mathsf{s}_{\widetilde{\mathcal{P}}} + O(n + \mathsf{q}_C)$ of $\widetilde{\mathcal{P}}_1$, the new index size n', and the new instance size k'.

Finally, the running time of $\mathcal{E}_2$ is the sum of the running time of $\mathcal{E}_1$, which is upper-bounded by $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\widetilde{\mathcal{P}}}, n' = O(n + \mathsf{q}_C), k' = k + \mathsf{q}_C)$ according to Definition 3.6, and the time to construct C' from C, which can be upper-bounded by $O(n + \mathsf{q}_C)$.

7 Technical extension: probabilistic extractors with oracle access

We have discussed the security of PCD obtained from relativized non-interactive arguments with straightline extraction: in Section 5 we provide the security analysis for the PCD construction in Section 4. The notion of straightline extraction considered in that analysis involves knowledge extractors that are *deterministic* and *do not have query access to the oracle*. While known constructions of relativized non-interactive arguments satisfy these constraints (see Section 6), in this section we provide, as an interesting technical extension, a security analysis for a relaxed notion of straightline extraction, which considers straightline knowledge extractors that *are probabilistic and have oracle access*.

Definition 7.1. ARG *has* **probabilistic (straightline) knowledge soundness error** $\kappa_{\mathsf{ARG}} = \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$ **with extraction time** $\mathsf{t}_{\mathsf{ARG}} = \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$ **and extraction query bound** $\mathsf{q}_{\mathsf{ARG}} = \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$ *if Definition 3.6 holds with a* **probabilistic** *extractor $\mathcal{E}$ that has query access to the oracle function and makes at most $\mathsf{q}_{\mathsf{ARG}}$ queries. Similarly,* PCD *has* **probabilistic (straightline) knowledge soundness error** $\kappa_{\mathsf{PCD}} = \kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, s_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$ **with extraction time** $\mathsf{t}_{\mathsf{PCD}} = \mathsf{t}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$ **and extraction query bound** $\mathsf{q}_{\mathsf{PCD}} = \mathsf{q}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$ *if Definition 3.16 holds with a* **probabilistic** *extractor $\mathbb{E}$ that has query access to the oracle function and makes at most $\mathsf{q}_{\mathsf{PCD}}$ queries.*

Remark 7.2. There is a simple, but inefficient, reduction from probabilistic extraction to deterministic extraction: run the extractor for every possible choice of randomness for the probabilistic extractor. The knowledge soundness error is preserved after the reduction, but the running time of the extractor (and its query complexity) increases with a multiplicative factor of all possible randomness.

We extend our results in Section 5 to the setting of Definition 7.1.

Definition 7.3. *$\mathcal{T}^{(N,D,M)}$ is the set of all M-ary trees T with size at most N and depth at most D such that the root vertex v_0 of T has a single child-vertex v_1.*

Definition 7.4. *The function* depth *receives as input a tree* T *and vertex v and outputs the depth d of v in* T.

Definition 7.5. *The function* parent *takes as input a tree $\mathrm{T} \in \mathcal{T}^{(N,D,M)}$ and vertex v, and outputs an integer i such that the i-th vertex at depth $d-1$ is the parent-vertex of v. (The i-th vertex at depth $d-1$ is with respect to the complete M-ary tree of depth D. In other words, vertices are labeled as if there are M^d vertices at depth d regardless of the structure of T.)*

Definition 7.6. *We define the recurrence functions* psize *and* pquery *to bound the size of the argument adversaries and the number of queries made by argument adversaries, respectively.*

- $\mathsf{pquery}(1) := \mathsf{q}_{\tilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$,
- $\mathsf{psize}(1) := s_{\tilde{\mathbb{P}}} + s_{\mathbb{I}} + n + k$,
- $\mathsf{pquery}(d) := \mathsf{pquery}(d-1) + \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1), \mathsf{psize}(d-1), n, k)$ *for* $d > 1$,
- $\mathsf{psize}(d) := \mathsf{psize}(d-1) + \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1), \mathsf{psize}(d-1), n, k) + n + k$ *for* $d > 1$.

Theorem 7.7 (Generalization of Theorem 5.1)**.** *Let $\mathcal{U}$ be an oracle distribution. Suppose that* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a non-interactive argument for oracle CSAT relation $R^{\mathcal{U}}_{\mathrm{CSAT}}$ with* **probabilistic oracle** *straightline knowledge soundness error $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, \mathsf{s}_{\tilde{\mathcal{P}}}, n, k)$, extraction time $\mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$, and extractor query bound $\mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{\tilde{\mathcal{P}}}, n, k)$.*

Then we can construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ *from* ARG *with straightline knowledge soundness error $\kappa_{\mathsf{PCD}} = \kappa_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, s_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$, extraction time $\mathsf{t}_{\mathsf{PCD}} = \mathsf{t}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\tilde{\mathbb{P}}}, N, D, S, Q, M, l)$,*

and extractor query bound $\mathsf{q}_{\mathsf{PCD}} = \mathsf{q}_{\mathsf{PCD}}(\lambda, \mathsf{q}_{\mathbb{I}}, s_{\mathbb{I}}, \mathsf{q}_{\widetilde{\mathbb{P}}}, N, D, S, Q, M, l)$ *such that*

$$\kappa_{\mathsf{PCD}} \leq \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), \mathsf{psize}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ ,$$

$$\mathsf{q}_{\mathsf{PCD}} \leq \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ ,$$

$$\mathsf{t}_{\mathsf{PCD}} \leq N \cdot \mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n,k)) + \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ ,$$

where

- $n := \mathsf{nsize}(\lambda, S, M, l)$, *where* $\mathsf{nsize}(\cdot)$ *is defined in Lemma 5.9; and*
- $k := |\mathsf{ivk}| + l$.
- $\mathsf{q}_{\widetilde{\mathcal{P}}} := \mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}} + N \cdot Q + \mathsf{q}_{\mathsf{PCD}}$.
- $\mathsf{s}_{\widetilde{\mathcal{P}}} := s_{\widetilde{\mathbb{P}}} + s_{\mathbb{I}} + N \cdot S + N \cdot \mathsf{vsize}(\lambda, n, k) + \mathsf{t}_{\mathsf{PCD}}$, *where* $\mathsf{vsize}(\lambda, n, k)$ *is the size of the argument verifier* $\mathcal{V}$ *when invoked with security parameter* λ*, index of size* n*, and instance of size* k*.*
- $\mathsf{arglen}(\lambda, n, k)$ *is the size of the argument proof* π *outputted by the argument prover when invoked with security parameter* λ*, index of size* n*, and instance of size* k*.*

Corollary 7.8 (From depth to size)**.** *The upper bound on the knowledge soundness error, the extractor query bound and the extraction time in Theorem 7.7 can be relaxed to the following:*

$$\kappa_{\mathsf{PCD}} \leq \sum_{i=1}^{N} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(i), \mathsf{psize}(i), n, k) \ ,$$

$$\mathsf{q}_{\mathsf{PCD}} \leq \sum_{i=1}^{N} \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(i), n, k) \ ,$$

$$\mathsf{t}_{\mathsf{PCD}} \leq N \cdot \mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n,k)) + \sum_{i=1}^{N} \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(i), n, k) \ .$$

Proof. Theorem 7.7 computes upper bounds by relying on the depth of each vertex in the transcript. Here, we provide an upper bound by relying on the breath-first-search (BFS) numbering of each vertex (the BFS numbering of the vertex is the number of vertices before itself in the BFS ordering of the transcript). Roughly this corresponds to keeping track of transcript size rather than transcript depth.

Let T be an arbitrary transcript in $\mathcal{T}^{(N,D,M)}$. Let v be an arbitrary vertex in $V(\mathrm{T})$. Let i be the BFS number of v. Observe that $i \geq \mathsf{depth}(\mathrm{T}, v)$ by definition of BFS and depth of a vertex. Therefore

$$\mathsf{psize}(i) \geq \mathsf{psize}(\mathsf{depth}(\mathrm{T}, v_i)) \text{ and } \mathsf{pquery}(i) \geq \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v_i)) \ .$$

Since κ_{ARG}, $\mathsf{q}_{\mathsf{ARG}}$, and $\mathsf{t}_{\mathsf{ARG}}$ are non-decreasing functions, we conclude the proof. □

Lemma 7.9 (Maximum PCD transcript)**.** *There exists a algorithm that takes* $N, D, M \in \mathbb{N}$ *as input, runs in time* $\mathsf{poly}(N, D, M)$*, and outputs a tree* T^* *in* $\mathcal{T}^{(N,D,M)}$ *that maximizes the expressions below:*

$$\max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), \mathsf{psize}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ ,$$

$$\max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ ,$$

$$\max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ .$$

Proof. We show that the maximum value over the set of all transcripts $\mathcal{T}^{(N,D,M)}$ for $\kappa_{\mathsf{PCD}}, \mathsf{q}_{\mathsf{PCD}}$, and $\mathsf{t}_{\mathsf{PCD}}$ (which appears in Theorem 7.7) has a clear resolution: the transcript T^* output by the algorithm GetMaxTree below.

Define $\mathrm{T}_{D,M}$ to be the complete M-ary tree of depth D. Let v_1 be the root of $\mathrm{T}_{D,M}$, add v_0 to $V(\mathrm{T}_{D,M})$ and (v_1, v_0) to $E(\mathrm{T}_{D,M})$. We construct GetMaxTree as follows:

GetMaxTree(N, D, M):
1. Abort if $N > \sum_{i=0}^{d} M^i$.
2. Initialize T^* as an empty graph.
3. While $\mathsf{size}(\mathrm{T}^*) < N$:
   (a) Let v be the next vertex (in the depth-first-search order) in $V(\mathrm{T}_{D,M})$ that is not in $V(T^*)$.
   (b) Add v and its unique outgoing edge to $V(T^*)$ and $E(T^*)$, respectively.
4. Output T^*.

The above algorithm clearly runs in time $\mathsf{poly}(N, D, M)$.

Now we show that the tree T^* output by GetMaxTree(N, D, M) attains the maximum. Let T be an arbitrary tree in $\mathcal{T}^{(N,D,M)}$. Sort $V(\mathrm{T}^*) = \{v_0^*, \ldots, v_N^*\}$ and $V(\mathrm{T}) = \{v_0, \ldots, v_N\}$ with respect to the depth of the vertices in non-increasing order. Note that depth-first-search always gives the deepest possible tree. In other words, for every $i \in [N]$, $\mathsf{depth}(\mathrm{T}^*, v_i^*) \geq \mathsf{depth}(\mathrm{T}, v_i)$. Since $\kappa_{\mathsf{ARG}}, \mathsf{q}_{\mathsf{ARG}}$, and $\mathsf{t}_{\mathsf{ARG}}$ are non-decreasing functions in the size of the argument adversary $\widetilde{\mathcal{P}}$, T^* maximizes the target expression. □

In the rest of this section we discuss the proof of Theorem 7.7.

7.1 Probabilistic oracle extractor for PCD

We first construct the knowledge extractor for the PCD constructed in Construction 4.2. Note that we need a different extractor from the one in Construction 4.3 because now the extractor for the underlying argument has oracle access to the oracle function. In particular, we separately consider an extraction queue for each vertex in the transcript since the query-answer traces induced by the argument extractor $\mathcal{E}$ need to be considered (see Fig. 7 and Fig. 8 for a concrete example of how $\mathbb{E}$ defined below works).

Construction 7.10. The knowledge extractor for the PCD scheme works as follows.

$\mathbb{E}^f(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}, \mathsf{tr})$:
1. Parse $\mathbb{pp}$ as pp.
2. Parse $\mathbb{\Pi}_{\text{out}}$ as π_{out}.
3. Parse $\mathbb{ivk}$ as ivk.
4. Initialize graph $\mathrm{T} = (V, E)$ where $V = \{v_0, v_1\}$ and $E = \{(v_1, v_0)\}$.
5. Label the edge (v_1, v_0) by $(z_{\text{out}}, \pi_{\text{out}})$.
6. Initialize the extraction queue for v_0 as $\mathcal{L}_{v_0} := (v_1)$.
7. Set $\mathsf{tr}_{v_0} := \mathsf{tr}$.
8. Initialize the queue of extraction queues $\mathcal{F}$ with $(\mathsf{tr}_{v_0}, \mathcal{L}_{v_0})$ as the first element.
9. Set $\mathbb{i} := C_{\mathcal{V},\phi}^{(\lambda,M,n,k)}$.
10. While $\mathcal{F}$ is non-empty:

(a) Let $(\mathsf{tr}, \mathcal{L})$ be the first tuple in $\mathcal{F}$, remove the first tuple $(\mathsf{tr}, \mathcal{L})$ from $\mathcal{F}$.

(b) While $\mathcal{L}$ is non-empty:

i. Let v be the first vertex in $\mathcal{L}$, remove the first vertex v from $\mathcal{L}$.

ii. Let $z^{(e)}$ and $\pi^{(e)}$ be the message and proof in the label of the unique outgoing edge e from v.

iii. Let $(\mathbb{i}_v, \mathbb{x}_v, \pi_v) := (\mathbb{i}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)})$.

iv. Run the argument extractor $\mathbb{w}_v \xleftarrow{\mathsf{tr}_{\mathcal{E}}} \mathcal{E}^f(\mathsf{pp}, \mathbb{i}_v, \mathbb{x}_v, \pi_v, \mathsf{tr})$.

v. Parse $\mathbb{w}_v$ as $(w_{\text{loc}}^{(v)}, \vec{z}_{\text{in}}^{(v)}, \vec{\mathbb{III}}_{\text{in}}^{(v)})$.

vi. Label v in T by $w_{\text{loc}}^{(v)}$.

vii. Initialize $\mathcal{L}_v$ to be an empty queue.

viii. For every j such that $\vec{z}_{\text{in}}^{(v)}[j] \neq \bot$:

A. Add a new vertex v' to V, and an edge (v', v) in E.

B. Parse $\vec{\mathbb{III}}_{\text{in}}^{(v)}[j]$ as $\pi^{(v',v)}$.

C. Add the label $(\vec{z}_{\text{in}}^{(v)}[j], \pi^{(v',v)})$ to the edge (v', v) in T.

D. Add v' to $\mathcal{L}_v$.

ix. Add $(\mathsf{tr} \parallel \mathsf{tr}_{\mathcal{E}}, \mathcal{L}_v)$ to $\mathcal{F}$.

11. Output the augmented transcript T.

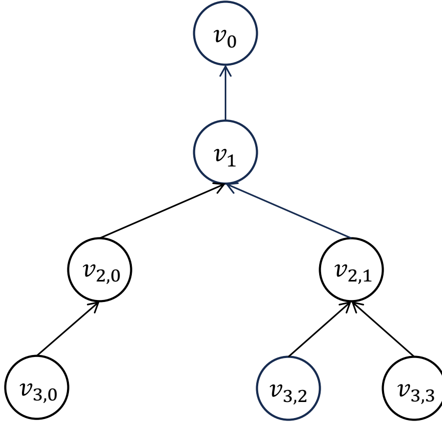

Figure 7: An example PCD transcript extracted by $\mathbb{E}$.

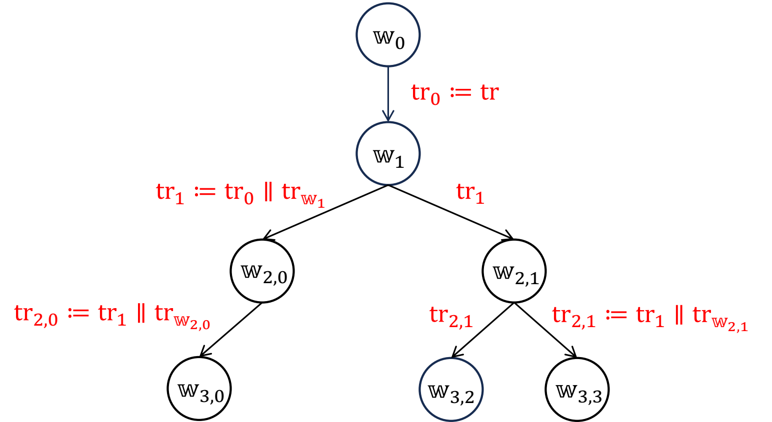

Figure 8: The extraction order of $\mathbb{E}$ for the PCD transcript in Fig. 7 with corresponding query-answer traces.

7.2 Knowledge soundness error

Our goal is to upper bound the following expression

$$\Pr\left[\begin{array}{c|r} \phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\ |z_{\text{out}}| \leq l & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\text{out}}, \mathbb{III}_{\text{out}}) = 1 & \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\text{out}}) & (\phi, z_{\text{out}}, \mathbb{III}_{\text{out}}) \xleftarrow{\mathsf{tr}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai}) \\ & (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi) \\ & \mathrm{T} \leftarrow \mathbb{E}^f(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\text{out}}, \mathbb{III}_{\text{out}}, \mathsf{tr}) \end{array}\right] . \quad (3)$$

Recall that in Section 5.1, the malicious argument prover $\widetilde{\mathcal{P}}$ is defined following the structure of the PCD extractor in Construction 4.3. Therefore, we can similarly define a malicious argument prover $\widetilde{\mathcal{P}}$ following Construction 7.10 if the argument extractor $\mathcal{E}$ is deterministic but with oracle access. The same bound and the same analysis for the deterministic knowledge soundness error generalize to this setting directly.

We consider the set of malicious argument provers defined in a similar way as in Section 5.2 (the only difference is when invoking $\mathcal{E}$, it has oracle access to f):

- $\widetilde{\mathcal{P}}^f_{1,0}(\mathsf{pp}, \mathsf{ai})$:
  1. Set $\mathbb{pp} := \mathsf{pp}$.
  2. Run $(\phi, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}) \leftarrow \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai})$.
  3. Run $(\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi)$.
  4. Parse $\mathbb{ivk}$ as ivk.
  5. Parse $\mathbb{\Pi}_{\text{out}}$ as π_{out}.
  6. Set $\mathbb{i} := C^{(\lambda, M, n, k)}_{\mathcal{V}, \phi}$.
  7. Output $(\mathbb{i}, (\mathsf{ivk}, z_{\text{out}}), \pi_{\text{out}})$.
- $\widetilde{\mathcal{P}}^f_{d,i}(\mathsf{pp}, \mathsf{ai})$:
  1. Let $(\text{parent}, \text{pos}) := (\lfloor i/M \rfloor, i \bmod M)$.
  2. Run $(\mathbb{i}, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}^f_{d-1,\text{parent}}(\mathsf{pp}, \mathsf{ai})$.
  3. If $(\mathbb{i}, \mathbb{x}, \pi) = \bot$, output $\bot$.
  4. Otherwise, run the argument extractor $\mathbb{w} \leftarrow \mathcal{E}^f(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi, \mathsf{tr})$.
  5. Parse $\mathbb{w}$ as $(w_{\text{loc}}, \vec{z}_{\text{in}}, \vec{\mathbb{\Pi}}_{\text{in}})$.
  6. Parse $\mathbb{x}$ as (ivk, z).
  7. If $\vec{z}_{\text{in}}[\text{pos}] = \bot$, output $\bot$.
  8. Otherwise, output $(\mathbb{i}, (\mathsf{ivk}, \vec{z}_{\text{in}}[\text{pos}]), \vec{\mathbb{\Pi}}_{\text{in}}[\text{pos}])$.

For every d and i, let $\mathsf{q}_{d,i}$ be the number of queries made by $\widetilde{\mathcal{P}}_{d,i}$ and $\mathsf{s}_{d,i}$ be the size of $\widetilde{\mathcal{P}}_{d,i}$.

For the rest of this section, we consider the following experiment:

Experiment 2.

$$\left[\begin{array}{r} f \leftarrow \mathcal{U}(\lambda) \\ \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ (\phi, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}) \xleftarrow{\mathsf{tr}} \widetilde{\mathbb{P}}^f(\mathbb{pp}, \mathsf{ai}) \\ (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^f(\mathbb{pp}, \phi) \\ \mathrm{T} \leftarrow \mathbb{E}^f(\mathbb{pp}, \mathbb{ivk}, \phi, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}, \mathsf{tr}) \end{array}\right].$$

Let T_i be the partial transcript produced by $\mathbb{E}$ at the end of the i-th iteration. We say that T_i ***is consistent with a complete transcript*** T if T_i is the same as the subtree of T induced by the first i-vertices (in the breadth-first-search order, which is the order that $\mathbb{E}$ extracts) of T. Similar to the argument in Section 5.1, we define a stronger notion of compliance to connect non-compliant transcript with falsely accepting argument verifier.

Definition 7.11. *A partial transcript $\mathrm{T}_i = (V_i, E_i)$ is* **partially** (ϕ, f)**-compliant** *if the following holds:*
- *For every non-source vertex $u \in V$:*
  - *Let $(v_1, \ldots, v_M)$ be the child-vertices of u.*

– *For every outgoing edge e of u, $\phi^f(z^{(e)}, w_{\mathrm{loc}}^{(u)}, (z^{(v_1,u)}, \ldots, z^{(v_M,u)})) = 1$.*
- *For every edge $e \in E$, $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(e)}), \pi^{(e)}) = 1$.*

Note that we only require compliance for non-source vertices above because the source vertices for partial transcripts cannot be checked: they may have child-vertices that are not extracted yet.

From the law of total probability, we have

$$
\begin{aligned}
&\Pr\left[\begin{array}{c}\phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \le l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \ne z_{\mathrm{out}})\end{array}\right] \\
&= \sum_{\mathrm{T}' \in \mathcal{T}^{(N,D,M)}} \Pr\left[\begin{array}{c}\phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \le l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \ne z_{\mathrm{out}})\end{array} \middle| \mathrm{T} = \mathrm{T}'\right] \Pr\left[\mathrm{T} = \mathrm{T}'\right] \\
&\le \max_{\mathrm{T}' \in \mathcal{T}^{(N,D,M)}} \Pr\left[\begin{array}{c}\phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \le l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T} \text{ is not } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \ne z_{\mathrm{out}})\end{array} \middle| \mathrm{T} = \mathrm{T}'\right] .
\end{aligned}
$$

Observe that $\mathsf{out}(\mathrm{T}) = z_{\mathrm{out}}$ always, we focus on the event that T being not (ϕ, f)-compliant, which implies that either

- T_N is not partially (ϕ, f)-compliant for T_N consistent with T; or
- there exists a source vertex v in T_N that is not (ϕ, f)-compliant.

Thus, the following claim gives an upper bound of Eq. (3).

Claim 7.12. *Fix some $\mathrm{T}' \in \mathcal{T}^{(N,D,M)}$ where $v_1 \in \mathrm{T}'$ is written as $v_{1,0}$. We have,*

$$
\begin{aligned}
&\Pr\left[\begin{array}{c}\phi \textit{ is } (\Phi, N, D, M, S, Q)\textit{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \le l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{\Pi}_{\mathrm{out}}) = 1 \\ \wedge \left(\begin{array}{c}\mathrm{T}_N \textit{ is not partially } (\phi, f)\textit{-compliant} \\ \vee\, \exists\, \textit{source vertex } v \in V(\mathrm{T}_N) : v \textit{ is not } (\phi, f)\textit{-compliant} \\ \vee\, \mathsf{out}(\mathrm{T}) \ne z_{\mathrm{out}}\end{array}\right)\end{array} \middle| \mathrm{T}_N \textit{ is consistent with } \mathrm{T}'\right] \\
&\le \sum_{v_{d,i} \in \mathrm{T}' \setminus \{v_0\}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{d,i}, \mathsf{s}_{d,i}, n, k) \ .
\end{aligned}
$$

We can conclude that

$$
\kappa_{\mathsf{PCD}} \le \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v_{d,i} \in \mathrm{T} \setminus \{v_0\}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}_{d,i}, \mathsf{s}_{d,i}, n, k) \ ,
$$

where $n := \mathsf{nsize}(\lambda, S, M, l)$ ($\mathsf{nsize}(\cdot)$ is the circuit size defined in Lemma 5.9) and $k := |\mathsf{ivk}| + l$.

To get the knowledge soundness stated in Theorem 7.7, we are left to compute $\mathsf{q}_{d,i}$ and $\mathsf{s}_{d,i}$.

Argument adversary query and size bound. We know that $\widetilde{\mathbb{P}}$ makes $\mathsf{q}_{\widetilde{\mathbb{P}}}$ queries to the oracle, and $\mathbb{I}$ makes $\mathsf{q}_{\mathbb{I}}$ queries to the oracle. Therefore, $\widetilde{\mathcal{P}}_{1,0}$ makes $\mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$ queries to f. Moreover, the size of $\widetilde{\mathcal{P}}_{1,0}$ is dominated by the size of $\widetilde{\mathbb{P}}$, the size of $\mathbb{I}$, and the size of the output, which is at most $s_{\widetilde{\mathbb{P}}} + s_{\mathbb{I}} + n + k$.

We use a similar analysis to bound the number of queries made by $\widetilde{\mathcal{P}}_{d,i}$. Let $\mathsf{pquery}(d,i)$ denote the number of queries made by $\widetilde{\mathcal{P}}_{d,i}$. According to the construction, the following recurrence holds:

- $\mathsf{pquery}(1,0) \coloneqq \mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$,
- $\mathsf{psize}(1,0) \coloneqq s_{\widetilde{\mathbb{P}}} + s_{\mathbb{I}} + n + k$,
- $\mathsf{pquery}(d,i) \coloneqq \mathsf{pquery}(d-1,\lfloor i/M \rfloor) + \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1,\lfloor i/M \rfloor), \mathsf{psize}(d-1,\lfloor i/M \rfloor), n, k)$ for $d > 1$,
- $\mathsf{psize}(d,i) \coloneqq \mathsf{psize}(d-1,\lfloor i/M \rfloor) + \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1,\lfloor i/M \rfloor), \mathsf{psize}(d-1,\lfloor i/M \rfloor), n, k) + n + k$ for $d > 1$.

We can simplify the recurrences. We argue, by induction on d, that $\mathsf{psize}(d,i) = \mathsf{psize}(d,j)$ and $\mathsf{pquery}(d,i) = \mathsf{pquery}(d,j)$ for every i and j.

- *Base case ($d = 1$).* There is a single vertex v_1 at depth 1. The claim is vacuously true.
- *Inductive step.* Assume the claim for all depths smaller than d. If $\lfloor i/M \rfloor = \lfloor j/M \rfloor$ (i.e., i and j have the same parent) then the claim holds by definition of psize. So suppose that $\lfloor i/M \rfloor \neq \lfloor j/M \rfloor$. By the inductive hypothesis, $\mathsf{psize}(d-1,\lfloor i/M \rfloor) = \mathsf{psize}(d-1,\lfloor j/M \rfloor)$ and $\mathsf{pquery}(d-1,\lfloor i/M \rfloor) = \mathsf{pquery}(d-1,\lfloor j/M \rfloor)$, and again deduce the claim by definition of psize.

Therefore, we can simplify the recurrence as follows (which matches the function in Definition 7.6):

- $\mathsf{pquery}(1) \coloneqq \mathsf{q}_{\widetilde{\mathbb{P}}} + \mathsf{q}_{\mathbb{I}}$,
- $\mathsf{psize}(1) \coloneqq s_{\widetilde{\mathbb{P}}} + s_{\mathbb{I}} + n + k$,
- $\mathsf{pquery}(d) \coloneqq \mathsf{pquery}(d-1) + \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1), \mathsf{psize}(d-1), n, k)$ for $d > 1$,
- $\mathsf{psize}(d) \coloneqq \mathsf{psize}(d-1) + \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d-1), \mathsf{psize}(d-1), n, k) + n + k$ for $d > 1$.

Hence, $\mathsf{q}_{d,i} = \mathsf{pquery}(d)$ and $\mathsf{s}_{d,i} = \mathsf{psize}(d)$ and we have the following bound:

$$\kappa_{\mathsf{PCD}} \leq \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T},v)), \mathsf{psize}(\mathsf{depth}(\mathrm{T},v)), n, k) \enspace .$$

Proof of Claim 7.12. We first show by induction that for every $\mathrm{T}' \in \mathcal{T}^{(N,D,M)}$ and $i \in [N]$,

$$\begin{aligned} &\Pr\left[\begin{array}{c} \phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\mathrm{out}}| \leq l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\mathrm{out}}, \mathbb{II}_{\mathrm{out}}) = 1 \\ \wedge\ (\mathrm{T}_i \text{ is not partially } (\phi,f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\mathrm{out}}) \end{array} \middle|\ \mathrm{T}_i \text{ is consistent with } \mathrm{T}' \right] \\ \leq &\sum_{\substack{v \in \mathrm{T}_i \setminus \{v_0\} \\ v \text{ is not a source vertex}}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}_i,v)), \mathsf{psize}(\mathsf{depth}(\mathrm{T}_i,v)), n, k) \enspace . \end{aligned} \tag{4}$$

- (*Base case, $i = 0$.*) According to Construction 7.10, T_0 is a tree where $V(\mathrm{T}_0) = \{v_0, v_1\}$ and $E(\mathrm{T}_0) = \{(v_1, v_0)\}$. Hence, T_0 is partially (ϕ,f)-compliant vacuously. Moreover, the edge (v_1, v_0) is labeled by $(z_{\mathrm{out}}, \mathbb{II}_{\mathrm{out}})$, which implies that $\mathsf{out}(\mathrm{T}_0) = z_{\mathrm{out}}$. Thus, the probability that T_0 is not partially (ϕ,f)-compliant or $\mathsf{out}(\mathrm{T}_0) \neq z_{\mathrm{out}}$ is 0, and Eq. (4) is true for $i = 0$.

- (*Inductive step.*) We assume that Eq. (4) holds for all values smaller than i. Assume that T_i is consistent with T. We again have that $\mathsf{out}(\mathrm{T}_0) = z_{\mathrm{out}}$, so we only need to focus on the probability that T_i is not partially (ϕ,f)-compliant. Note that if T_i is not partially (ϕ,f)-compliant, then either T_{i-1} is not

partially (ϕ, f)-compliant, or T_{i-1} is partially (ϕ, f)-compliant but T_i is not. Therefore, we know that the probability expression in Eq. (4) can be upper-bounded by the sum of

$$\Pr\left[\begin{array}{c} \phi \text{ is } (\Phi, N, D, M, S, Q)\text{-compatible} \\ \wedge\ |z_{\text{out}}| \leq l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}) = 1 \\ \wedge\ (\mathrm{T}_{i-1} \text{ is not partially } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\text{out}}) \end{array} \middle|\ \mathrm{T}_i \text{ is consistent with } \mathrm{T}' \right] \tag{5}$$

and

$$\Pr\left[\begin{array}{c} \phi \text{ is } (\Phi, N, S, D, M, Q)\text{-compatible} \\ \wedge\ |z_{\text{out}}| \leq l \\ \wedge\ \mathbb{V}^f(\mathbb{ivk}, z_{\text{out}}, \mathbb{\Pi}_{\text{out}}) = 1 \\ \wedge\ \mathrm{T}_{i-1} \text{ is partially } (\phi, f)\text{-compliant} \\ \wedge\ (\mathrm{T}_{i} \text{ is not partially } (\phi, f)\text{-compliant} \vee \mathsf{out}(\mathrm{T}) \neq z_{\text{out}}) \end{array} \middle|\ \mathrm{T}_i \text{ is consistent with } \mathrm{T}' \right] . \tag{6}$$

For the first case, let v_{i-1} be the $(i-1)$-th vertex (in the breadth-first-search order) in T. From the inductive hypothesis, we can upper-bound Eq. (5) by

$$\sum_{\substack{v \in \mathrm{T}_{i-1} \setminus \{v_0\} \\ v \text{ is not a source vertex}}} \kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}_{i-1}, v_{i-1})), \mathsf{psize}(\mathsf{depth}(\mathrm{T}_{i-1}, v_{i-1})), n, \mathbb{x}) \ .$$

We now discuss the second case. Let v be the i-th vertex (in the breadth-first-search order) in T. By Construction 7.10, v is the vertex extracted by $\mathbb{E}$ in the i-th iteration. Let $(d, k) := (\mathsf{depth}(\mathrm{T}, v), \mathsf{parent}(\mathrm{T}, v))$. By definition of depth, we know that $\widetilde{\mathcal{P}}_{d,k}$ is an argument adversary that outputs the index-instance-proof tuple $(\mathbb{i}_v, \mathbb{x}_v, \pi_v)$ that corresponds to vertex v. Let e be the unique outgoing edge of v, let $v_1, \ldots, v_M$ be the child-vertices of v in T_i (maybe there are less than M child-vertices of v in T_i, but the exact number would not affect the analysis as long as it is upper-bounded by M). According to Definition 7.11, if T_{i-1} is partially (ϕ, f)-compliant but T_i is not, then at least one of the following is true:

- $\phi^f(z^{(e)}, w_{\text{loc}}^{(v)}, (z^{(v_\ell, v)})_{\ell \in [M]}) = 0$;
- $\mathcal{V}^f(\mathsf{ivk}, (\mathsf{ivk}, z^{(v_\ell, v)}), \pi^{(v_\ell, v)}) = 0$ for some $\ell \in [M]$.

Let $\mathbb{w}_v$ be the witness outputted by $\mathcal{E}$ when extracting v. By Construction 4.1, we know that $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \notin R_{\text{CSAT}}^f$. On the other hand, since T_{i-1} is partially (ϕ, f)-compliant, we know that $\mathcal{V}^f(\mathsf{ivk}, \mathbb{x}_v, \pi_v) = 1$. Therefore, T_{i-1} being partially (ϕ, f)-compliant but T_i being not partially (ϕ, f)-compliant implies that $(\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \notin R_{\text{CSAT}}^f$ and $\mathcal{V}^f(\mathsf{ivk}, \mathbb{x}_v, \pi_i) = 1$. We then can conclude that Eq. (6) can be upper-bounded by

$$\Pr\left[\begin{array}{c} |\mathbb{i}_v| \leq n \\ \wedge\ |\mathbb{x}_v| \leq k \\ \wedge\ (\mathbb{i}_v, \mathbb{x}_v, \mathbb{w}_v) \notin R \\ \wedge\ \mathcal{V}^f(\mathsf{ivk}, \mathbb{x}_v, \pi_v) = 1 \end{array} \middle|\ \begin{array}{r} \mathsf{pp} := \mathbb{pp} \\ (\mathbb{i}_v, \mathbb{x}_v, \pi_v) \xleftarrow{\mathsf{tr}} \widetilde{\mathcal{P}}_{d,k}^f(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^f(\mathsf{pp}, \mathbb{i}_v) \\ \mathbb{w}_v \leftarrow \mathcal{E}^f(\mathsf{pp}, \mathbb{i}_v, \mathbb{x}_v, \pi_v, \mathsf{tr}) \end{array} \right] .$$

From Definition 7.1, the above expression is upper-bounded by

$$\kappa_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(d), \mathsf{psize}(d), n, k) \ .$$

Therefore, by union bound, Eq. (4) holds for i.

Applying the same (inductive) analysis to the source vertices in T_N concludes Claim 7.12. □

7.3 Extraction query bound

From Construction 7.10, the only place that $\mathbb{E}$ makes query to f is when invoking $\mathcal{E}$ in Item 10(b)iv. Fix $\mathrm{T} \in \mathcal{T}^{(N,D,M)}$. Let v be an arbitrary vertex in T. According to Definition 7.1, the number of queries made by $\mathcal{E}$ when extracting v is

$$\mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ .$$

Therefore, we can conclude that the total number of queries made by $\mathbb{E}$ is

$$\max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{q}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ .$$

7.4 Extraction time bound

The extraction time can be analyzed in the same way as in Section 5.2:

$$\begin{aligned} \mathsf{t}_{\mathsf{PCD}} &\leq N \cdot \mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n_{d,i}, k_{d,i}))) + \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{q}_{d,i}, n_{d,i}, k_{d,i}) \\ &\leq N \cdot \mathsf{poly}(\log N, \log M, l, \mathsf{arglen}(n, k)) + \max_{\mathrm{T} \in \mathcal{T}^{(N,D,M)}} \sum_{v \in \mathrm{T} \setminus \{v_0\}} \mathsf{t}_{\mathsf{ARG}}(\lambda, \mathsf{pquery}(\mathsf{depth}(\mathrm{T}, v)), n, k) \ . \end{aligned}$$

Acknowledgments

We thank Sarah Bordage for valuable discussions in early stages of this work. We thank Giacomo Fenzi, Christian Knabenhans, and Giorgio Seguini for valuable feedback and comments on earlier drafts of this paper. Ziyi Guan is partially supported by the Ethereum Foundation. Eylon Yogev is supported by an Alon Young Faculty Fellowship, by the Israel Science Foundation (Grant No. 2302/22), and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

References

[BBHMR19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. "On the (In)security of Kilian-Based SNARGs". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. 2019, pp. 522–551.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 733–764.

[BC23] Benedikt Bünz and Binyi Chen. "Protostar: Generic Efficient Accumulation/Folding for Special-SoundProtocols". In: *Proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '23. 2023, pp. 77–110.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 111–120.

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. *Relativized Succinct Arguments in the ROM Do Not Exist*. Cryptology ePrint Archive, Paper 2024/728. 2024. URL: `https://eprint.iacr.org/2024/728`.

[BCLMS21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 681–710.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. "Proof-Carrying Data from Accumulation Schemes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020, pp. 1–18.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. Extended version at `http://eprint.iacr.org/2014/595`. 2014, pp. 276–294.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. "Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 649–680.

[BF23] Josh Beal and Ben Fisch. *Derecho: Privacy Pools with Proof-Carrying Disclosures*. Cryptology ePrint Archive, Paper 2023/273. 2023. URL: `https://eprint.iacr.org/2023/273`.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. *Coda: Decentralized Cryptocurrency at Scale*. IACR Cryptology ePrint Archive, Report 2020/352. 2020.

[BR93] Mihir Bellare and Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS ’93. 1993, pp. 62–73.

[CCDW20] Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. *Reducing Participation Costs via Incremental Verification for Ledger Systems*. Cryptology ePrint Archive, Report 2020/1522. 2020.

[CCGOS23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O’Connor, and Nicholas Spooner. “Proof-Carrying Data From Arithmetized Random Oracles”. In: *Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’23. 2023, pp. 379–404.

[CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. “On Succinct Non-interactive Arguments in Relativized Worlds”. In: *Proceedings of the 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’22. 2022.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. “The random oracle methodology, revisited”. In: *Journal of the ACM* 51.4 (2004), pp. 557–594.

[CL20] Alessandro Chiesa and Siqi Liu. “On the Impossibility of Probabilistic Proofs in Relativized Worlds”. In: *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*. ITCS ’20. 2020.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. “Fractal: Post-Quantum and Transparent Recursive Proofs from Holography”. In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’20. 2020, pp. 769–793.

[CT10] Alessandro Chiesa and Eran Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards”. In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS ’10. 2010, pp. 310–331.

[CTV13] Stephen Chong, Eran Tromer, and Jeffrey A. Vaughan. *Enforcing Language Semantics Using Proof-Carrying Data*. Cryptology ePrint Archive, Report 2013/513. 2013.

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. “Cluster Computing in Zero Knowledge”. In: *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT ’15. 2015, pp. 371–403.

[CY21a] Alessandro Chiesa and Eylon Yogev. “Subquadratic SNARGs in the Random Oracle Model”. In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO ’21. 2021, pp. 711–741.

[CY21b] Alessandro Chiesa and Eylon Yogev. “Tight Security Bounds for Micali’s SNARGs”. In: *Proceedings of the 19th Theory of Cryptography Conference*. TCC ’21. 2021, pp. 401–434.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[E23] Ethereum. *Zero-Knowledge Rollups*. https://ethereum.org/en/developers/docs/scaling/zk-rollups/. 2023.

[FN16] Dario Fiore and Anca Nitulescu. “On the (In)Security of SNARKs in the Presence of Oracles”. In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC ’16-B. 2016, pp. 108–138.

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. *Cairo: a Turing-complete STARK-friendly CPU architecture*. IACR Cryptology ePrint Archive, Report 2021/1063. 2021.

[GW11] Craig Gentry and Daniel Wichs. “Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions”. In: *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*. STOC ’11. 2011, pp. 99–108.

[HN23] Mathias Hall-Andersen and Jesper Buus Nielsen. "On Valiant's Conjecture: Impossibility of Incrementally Verifiable Computation from Random Oracles". In: *Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '23. 2023.

[KB23] Assimakis Kattis and Joseph Bonneau. "Proof of Necessary Work: Succinct State Verification with Fairness Guarantees". In: *Proceedings of the 27th Financial Cryptography and Data Security*. FC '23. 2023.

[KS22] Abhiram Kothapalli and Srinath Setty. *SuperNova: Proving universal machine executions without universal circuits*. Cryptology ePrint Archive, Paper 2022/1758. 2022.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *Proceedings of the 42nd Annual International Cryptology Conference*. CRYPTO '22. 2022, pp. 359–388.

[ML20] Matter Labs. *zkSync v1.1 "Reddit Edition": Recursion*. `https://blog.matter-labs.io/zksync-v1-1-reddit-edition-recursion-up-to-3-000-tps-subscriptions-and-more-fea668b5b0ff`. 2020.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.

[Mina] O(1) Labs. *Mina Cryptocurrency*. `https://minaprotocol.com/`. 2017.

[NT16] Assa Naveh and Eran Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *Proceedings of the 37th IEEE Symposium on Security and Privacy*. S&P '16. 2016, pp. 255–271.

[P23] Polygon. *The Go Fast Machine: Adding Recursion to Polygon zkEVM*. `https://polygon.technology/blog/the-go-fast-machine-adding-recursion-to-polygon-zkevm`. 2023.

[PL22] Polymer Labs. *A Tutorial on Writing proofs with Plonky2*. `https://polymerlabs.medium.com/a-tutorial-on-writing-zk-proofs-with-plonky2-part-i-be5812f6b798`. 2022.

[PP22] Omer Paneth and Rafael Pass. "Incrementally Verifiable Computation via Rate-1 Batch Arguments". In: *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science*. FOCS '22. 2022, pp. 1045–1056.

[SW21] StarkWare Industries. *Starkware: SHARP Verifier*. `https://etherscan.io/address/0x47312450b3ac8b5b8e247a6bb6d523e7605bdb60`. 2021.

[SW22] StarkWare Industries. *Recursive STARKs*. `https://medium.com/@starkware/recursive-starks-78f8dd401025`. 2022.

[TFZBT22] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. "VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries". In: *Proceedings of the 29th ACM Conference on Computer and Communications Security*. CCS '22. 2022, pp. 2793–2807.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 1–18.