Proof-Carrying Data from Accumulation Schemes

Benedikt Bünz
benedikt@cs.stanford.edu
Stanford University

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Pratyush Mishra
pratyush@berkeley.edu
UC Berkeley

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

April 30, 2020

Abstract

Recursive proof composition has been shown to lead to powerful primitives such as incrementally-verifiable computation (IVC) and proof-carrying data (PCD). All existing approaches to recursive composition take a succinct non-interactive argument of knowledge (SNARK) and use it to prove a statement about its own verifier. This technique requires that the verifier run in time sublinear in the size of the statement it is checking, a strong requirement that restricts the class of SNARKs from which PCD can be built. This in turn restricts the efficiency and security properties of the resulting scheme.

Bowe, Grigg, and Hopwood (ePrint 2019/1021) outlined a novel approach to recursive composition, and applied it to a particular SNARK construction which does *not* have a sublinear-time verifier. However, they omit details about this approach and do not prove that it satisfies any security property. Nonetheless, schemes based on their ideas have already been implemented in software.

In this work we present a collection of results that establish the theoretical foundations for a significant generalization of the above approach. We define an *accumulation scheme* for a non-interactive argument, and show that this suffices to construct PCD, even if the argument itself does not have a sublinear-time verifier. Moreover we give constructions of accumulation schemes for SNARKs, which yield PCD schemes with novel efficiency and security features.

Keywords: succinct arguments; proof-carrying data; recursive proof composition

Contents

1 Introduction

Proof-carrying data (PCD) [CT10] is a cryptographic primitive that enables mutually distrustful parties to perform distributed computations that run indefinitely, while ensuring that every intermediate state of the computation can be succinctly verified. PCD supports computations defined on (possibly infinite) directed acyclic graphs, with messages passed along directed edges. Verification is facilitated by attaching to each message a succinct proof of correctness. This is a generalization of the notion of *incrementally-verifiable computation* (IVC) due to [Val08], which can be viewed as PCD for the path graph (i.e., for automata). PCD has found applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], succinct blockchains [Co17; KB20; BMRS20], and others.

Recursive composition. Prior to this work, the only known method for constructing PCD was from *recursive composition* of succinct non-interactive arguments (SNARGs) [BCCT13; BCTV14; COS20]. This method informally works as follows. A proof that the computation was executed correctly for t steps consists of a proof of the claim "the t-th step of the computation was executed correctly, and there exists a proof that the computation was executed correctly for $t-1$ steps". The latter part of the claim is expressed using the SNARG verifier itself. This construction yields secure PCD (with IVC as a special case) provided the SNARG satisfies an adaptive knowledge soundness property (i.e., is a SNARK). The efficiency and security properties of the resulting PCD scheme correspond to those of a single invocation of the SNARK.

Limitations of recursion. Recursion as realized in prior work requires proving a statement that contains a description of the SNARK verifier. In particular, for efficiency, we must ensure that the statement we are proving (essentially) *does not grow* with the number of recursion steps t. For example, if the representation of the verifier were to grow even *linearly* with the statement it is verifying, then the size of the statement to be checked would grow *exponentially* in t. Therefore, prior works have achieved efficiency by focusing on SNARKs which admit sublinear-time verification: either SNARKs for machine computations [BCCT13] or preprocessing SNARKs for circuit computations [BCTV14; COS20]. This significantly restricts our choice of SNARK, which limits what we can achieve for PCD.

In addition to the above asymptotic considerations, recursion raises additional considerations concerning concrete efficiency. All SNARK constructions require that statements be encoded as instances of some particular (algebraic) NP-complete problem, and difficulties often arise when encoding the SNARK verifier itself as such an instance. The most well-known example of this is in recursive composition of pairing-based SNARKs, since the verifier performs operations over a finite field that is necessarily different from the field supported "natively" by the NP-complete problem [BCTV14]. This type of problem also appears when recursing SNARKs whose verifiers make heavy use of cryptographic hash functions [COS20].

A new technique. Bowe, Grigg, and Hopwood [BGH19] suggest an exciting novel approach to recursive composition that replaces the SNARK verifier in the circuit with a simpler algorithm. This algorithm does not itself verify the previous proof π_{t-1}. Instead, it adds the proof to an *accumulator* for verification at the end. The accumulator must not grow in size. A key contribution of [BGH19] is to sketch a mechanism by which this might be achieved for a particular SNARK construction. While they prove this SNARK construction secure, they do not include definitions or proofs of security for their recursive technique. Nonetheless, practitioners have already built software based on these ideas [Halo19; Pickles20].

1.1 Our contributions

In this work we provide a collection of results that establish the theoretical foundations for the above approach. We introduce the cryptographic object, an *accumulation scheme*, that enables this technique, and prove that it suffices for constructing PCD. We then provide generic tools for building accumulation schemes, as well as several concrete instantiations. Our framework establishes the security of schemes that are already being used by practitioners, and we believe that it will simplify and facilitate further research in this area.

Recursion from accumulation. We introduce the notion of an *accumulation scheme* for non-interactive arguments. This formalizes, and significantly generalizes, the idea presented in [BGH19]. We then show that any SNARK having a sufficiently efficient accumulation scheme can be used to build a proof-carrying data (PCD) scheme. This broadens the class of SNARKs from which PCD can be built. Similarly to [COS20], we show that if the SNARK and accumulation scheme are post-quantum secure, so is the PCD scheme. (Though it remains an open question whether there are non-trivial accumulation schemes for post-quantum SNARKs.)

Theorem 1 (informal)**.** *There is an efficient transformation that compiles any (preprocessing) SNARK with an accumulation scheme into a (preprocessing) PCD scheme, preserving zero knowledge. Also, if the SNARK and its accumulation scheme are post-quantum secure then the PCD scheme is also post-quantum secure.*

The above theorem holds in the standard model (where all parties have access to a common reference string, but no oracles). Since our construction makes non-black-box use of the accumulation scheme verifier, the theorem does not carry over to the random oracle model (ROM). It remains an intriguing open problem to determine whether or not SNARKs in the ROM imply PCD in the ROM.

Proof-carrying data is a powerful primitive: it implies IVC, and hence efficient SNARKs for machine computations. Hence, Theorem 1 may be viewed as an extension of the "bootstrapping" theorem of [BCCT13] to certain non-succinct-verifier SNARKs.

See Section 2.1 for a summary of the ideas behind Theorem 1, and Section 5 for technical details.

Accumulation from accumulation. Given the above, a natural question is: where do accumulation schemes for SNARKs come from? In [BGH19] it was informally observed that a specific SNARK construction, based on the hardness of the discrete logarithm problem, has an accumulation scheme. The verifier in the SNARK construction is succinct *except for* the evaluation of a certain predicate (checking an opening of a polynomial commitment [KZG10]), and [BGH19] outline a construction of an accumulation scheme for that predicate.

We prove that this idea is a special case of a general paradigm for building accumulation schemes for SNARKs.

Theorem 2 (informal)**.** *There is an efficient transformation that, given a SNARK whose verifier is succinct when given oracle access to a "simpler" predicate and an accumulation scheme for that predicate, constructs an accumulation scheme for the SNARK. Moreover, this transformation preserves zero knowledge and post-quantum security of the accumulation scheme.*

The construction underlying Theorem 2 is black-box. In particular, if both the SNARK and the accumulation scheme for the predicate are secure with respect to an oracle, then the resulting accumulation scheme for the SNARK is secure with respect to that oracle.

See Section 2.3 for a summary of the ideas behind Theorem 2, and Section 6 for technical details.

Accumulating polynomial commitments. Several works [MBKM19; GWC19; CHMMVW20] have constructed SNARKs whose verifiers are succinct relative to a specific predicate: checking the opening of a *polynomial commitment* [KZG10]. We prove that two natural polynomial commitment schemes possess accumulation schemes in the random oracle model: $\mathsf{PC}_{\mathsf{DL}}$, a scheme based on the security of discrete logarithms [BCCGP16; BBBPWM18; WTSTW18]; and $\mathsf{PC}_{\mathsf{AGM}}$, a scheme based on knowledge assumptions in bilinear groups [KZG10; CHMMVW20].

Theorem 3 (informal)**.** *In the random oracle model, there exist accumulation schemes for* $\mathsf{PC}_{\mathsf{DL}}$ *and* $\mathsf{PC}_{\mathsf{AGM}}$ *that achieve the efficiency outlined in the table below (*n *denotes the number of evaluation proofs, and* d *denotes the degree of committed polynomials).*

polynomial commitment	assumption	cost to check evaluation proofs	cost to check an accumulation step	cost to check final accumulator	accumulator size
$\mathsf{PC}_{\mathsf{DL}}$	DLOG + RO	$\Theta(nd)$ $\mathbb{G}$ mults.	$\Theta(n \log d)$ $\mathbb{G}$ mults.	$\Theta(d)$ $\mathbb{G}$ mults.	$\Theta(\log d)$ $\mathbb{G}$
$\mathsf{PC}_{\mathsf{AGM}}$	AGM + RO	$\Theta(n)$ pairings	$\Theta(n)$ $\mathbb{G}_1$ mults.	1 pairing	$2\ \mathbb{G}_1$

For both schemes the cost of checking that an accumulation step was performed correctly is *much less* than the cost of checking an evaluation proof. By plugging either of these accumulation schemes for polynomial commitments into the aforementioned predicate-efficient SNARKs, we can apply Theorem 2 to obtain concrete accumulation schemes for these SNARKs with the same efficiency benefits.

We remark that our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$ is a variation of a construction presented in [BGH19], and so our result establishes the security of a type of construction used by practitioners.

We sketch the constructions underlying Theorem 3 in Section 2.4, and provide details in Sections 7 and 8.

New constructions of PCD. By combining our results, we (heuristically) obtain constructions of PCD that achieve new properties. Namely, starting from either $\mathsf{PC}_{\mathsf{DL}}$ or $\mathsf{PC}_{\mathsf{AGM}}$, we can apply Theorem 2 to a suitable SNARK to obtain a SNARK with an accumulation scheme in the random oracle model. Then we can instantiate the random oracle, obtaining a SNARK and accumulation scheme with *heuristic* security in the standard (CRS) model, to which we apply Theorem 1 to obtain a corresponding PCD scheme. Depending on whether we started with $\mathsf{PC}_{\mathsf{DL}}$ or $\mathsf{PC}_{\mathsf{AGM}}$, we get a PCD scheme with different features, as summarized below.

- *From* $\mathsf{PC}_{\mathsf{DL}}$*: PCD based on discrete logarithms.* We obtain a PCD scheme in the *uniform reference string* model (i.e., without secret parameters) and small argument sizes. In contrast, prior PCD schemes require structured reference strings [BCTV14] or have larger argument sizes [COS20]. Moreover, our PCD scheme can be efficiently instantiated from any cycle of elliptic curves [SS11]. In contrast, prior PCD schemes with small argument size use cycles of pairing-friendly elliptic curves [BCTV14; CCW19], which are more expensive.

- *From* $\mathsf{PC}_{\mathsf{AGM}}$*: lightweight PCD based on bilinear groups.* The recursive statement inside this PCD scheme does not involve checking any pairing computations, because pairings are deferred to a verification that occurs *outside* the recursive statement. In contrast, the recursive statements in prior PCD schemes based on pairing-based SNARKs were more expensive because they checked pairing computations [BCTV14].

Note again that our constructions of PCD are *heuristic* as they involve instantiating the random oracle of certain SNARK constructions with an appropriate hash function. This is because Theorem 3 is proven in the random oracle model, but Theorem 1 is explicitly *not* (as is the case for all prior IVC/PCD constructions [Val08; BCCT13; BCTV14; COS20]). There is evidence that this limitation might be inherent [CL20].

1.2 Related work

Below we survey prior constructions of IVC/PCD.

PCD from SNARKs. Bitansky, Canetti, Chiesa, and Tromer [BCCT13] proved that recursive composition of SNARKs for machine computations implies PCD for constant-depth graphs, and that this in turn implies IVC for polynomial-time machine computations. From the perspective of concrete efficiency, however, one can achieve more efficient recursive composition by using *preprocessing* SNARKs for circuits rather than SNARKs for machines [BCTV14; COS20]; this observation has led to real-world applications [Co17;

BMRS20]. The features of the PCD scheme obtained from recursion depends on the features of the underlying preprocessing SNARK. Below we summarize the features of the two known constructions.

- *PCD from pairing-based SNARKs.* Ben-Sasson, Chiesa, Tromer, and Virza [BCTV14] used pairing-based SNARKs with a special algebraic property to achieve efficient recursive composition with very small argument sizes (linear in the security parameter λ). The use of pairing-based SNARKs has two main downsides. First, they require sampling a *structured reference string* involving secret values ("toxic waste") that, if revealed, compromise security. Second, the verifier performs operations over a finite field that is necessarily different from the field supported "natively" by the statement it is checking. To avoid expensive simulation of field arithmetic, the construction uses *pairing-friendly cycles of elliptic curves*, which severely restricts the choice of field in applications and requires a large base field for security.
- *PCD from IOP-based SNARKs.* Chiesa, Ojha, and Spooner [COS20] used a holographic IOP to construct a preprocessing SNARK that is unconditionally secure in the (quantum) random oracle model, which heuristically implies a post-quantum preprocessing SNARK in the *uniform reference string* model (i.e., without toxic waste). They then proved that any post-quantum SNARK leads to a post-quantum PCD scheme via recursive composition. The downside of this construction is that, given known holographic IOPs, the argument size is larger, currently at $O(\lambda^2 \log^2 N)$ bits for circuits of size N.

IVC from homomorphic encryption. Naor, Paneth, and Rothblum [NPR19] obtain a notion of IVC by using somewhat homomorphic encryption and an information-theoretic object called an "incremental PCP". The key feature of their scheme is that security holds under falsifiable assumptions.

There are two drawbacks, however, that restrict the use of the notion of IVC that their scheme achieves.

First, the computation to be verified must be *deterministic* (this appears necessary for schemes based on falsifiable assumptions given known impossibility results [GW11]). Second, and more subtly, completeness holds only in the case where intermediate proofs were honestly generated. This means that the following attack may be possible: an adversary provides an intermediate proof that verifies, but it is impossible for honest parties to generate new proofs for subsequent computations. Our construction of PCD achieves the stronger condition that completeness holds so long as intermediate proofs verify, ruling out this attack.

Both nondeterministic computation and the stronger completeness notion (achieved by all SNARK-based PCD schemes) are necessary for many of the applications of IVC/PCD.

2 Techniques

2.1 PCD from arguments with accumulation schemes

We summarize the main ideas behind Theorem 1, which obtains proof-carrying data (PCD) from any succinct non-interactive argument of knowledge (SNARK) that has an accumulation scheme. For the sake of exposition, in this section we focus on the special case of IVC, which can be viewed as repeated application of a circuit F. Specifically, we wish to check a claim of the form "$F^T(z_0) = z_T$" where F^T denotes F composed with itself T times.

Prior work: recursion from succinct verification. Recall that in previous approaches to efficient recursive composition [BCTV14; COS20], at each step i we prove a claim of the form "$z_i = F(z_{i-1})$, and there exists a proof π_{i-1} that attests to the correctness of z_{i-1}". This claim is expressed using a circuit R which is the conjunction of F with a circuit representing the SNARK verifier; in particular, the size of the claim is at least the size of the verifier circuit. If the size of the verifier circuit grows linearly (or more) with the size of the claim being checked, then verifying the final proof becomes more costly than the original computation.

For this reason, these works focus on SNARKs with *succinct verification*, where the verifier runs in time *sublinear* in the size of the claim. In this case, the size of the claim essentially *does not grow* with the number of recursive steps, and so checking the final proof costs roughly the same as checking a single step.

Succinct verification is a seemingly paradoxical requirement: the verifier does not even have time to *read* the circuit R. One way to sidestep this issue is *preprocessing*: one designs an algorithm that, at the beginning of the recursion, computes a small cryptographic digest of R, which the recursive verifier can use instead of reading R directly. Because this preprocessing need only be performed once for the given R in an offline phase, it has almost no effect on the performance of each recursive step (in the later online phase).

A new paradigm: IVC from accumulation. Even allowing for preprocessing, succinct verification remains a strong requirement, and there are many SNARKs that are not known to satisfy it (e.g., [BCCGP16; BBBPWM18; AHIV17; BCGGHJ17; BCRSVW19]). Bowe, Grigg, and Hopwood [BGH19] suggested a further relaxation of succinctness that appears to still suffice for recursive composition: a type of "post-processing". Their observation is as follows: if a SNARK is such that we can efficiently "defer" the verification of a claim in a way that does not grow in cost with the number of claims to be checked, then we can hope to achieve recursive composition by deferring the verification of all claims to the end.

In the remainder of this section, we will give an overview of the proof of Theorem 1, our construction of PCD from SNARKs that have this "post-processing" property. We note that this relaxation of requirements is useful because, as suggested in [BGH19], it leads to new constructions of PCD with desirable properties (see discussion at the end of Section 1.1). In fact, some of these efficiency features are already being exploited by practitioners working on recursing SNARKs [Halo19; Pickles20].

The specific property we require, which we discuss more formally in the next section, is that the SNARK has an *accumulation scheme*. This is a significant generalization of the idea described for a particular SNARK construction in [BGH19]. Informally, an accumulation scheme consists of three algorithms: an accumulation prover, an accumulation verifier, and a decider. The accumulation prover is tasked with taking an instance-proof pair (z, π) and a previous accumulator acc, and producing a new accumulator $\mathsf{acc}^\star$ that "includes" the new instance. The accumulation verifier, given $((z, \pi), \mathsf{acc}, \mathsf{acc}^\star)$, checks that $\mathsf{acc}^\star$ was computed correctly (i.e., that it accumulates (z, π)) into acc). Finally the decider, given a single accumulator acc, performs a single check that simultaneously ensures that *every* instance-proof pair accumulated in acc verifies.[1]

[1]We remark that the notion of an accumulation scheme is *distinct* from the notion of a cryptographic accumulator for a set (e.g., an RSA accumulator), which provides a succinct representation of a large set while supporting membership queries.

Given such an accumulation scheme, we can construct IVC as follows. Given a previous instance z_i, proof π_i, and accumulator acc_i, the IVC prover first accumulates (z_i, π_i) with acc_i to obtain a new accumulator acc_{i+1}. The IVC prover also generates a SNARK proof π_{i+1} of the claim: "$z_{i+1} = F(z_i)$, and there exist a proof π_i and an accumulator acc_i such that the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1})$", expressed as a circuit R. The final IVC proof then consists of (π_T, acc_T). The IVC verifier checks such a proof by running the SNARK verifier on π_T and the accumulation scheme decider on acc_T.

Why does this achieve IVC? Throughout the computation we maintain the invariant that if acc_i is a valid accumulator (according to the decider) and π_i is a valid proof, then the computation is correct up to the i-th step. Clearly if this holds at time T then the IVC verifier successfully checks the entire computation. Observe that if we were able to prove that "$z_{i+1} = F(z_i)$, π_i is a valid proof, and acc_i is a valid accumulator", by applying the invariant we would be able to conclude that the computation is correct up to step $i + 1$. Unfortunately we are not able to prove this directly, for two reasons: (i) proving that π_i is a valid proof requires proving a statement about the argument verifier, which may not be sublinear, and (ii) proving that acc_i is a valid accumulator requires proving a statement about the decider, which may not be sublinear.

Instead of proving this claim directly, we "defer" it by having the prover accumulate (z_i, π_i) into acc_i to obtain a new accumulator acc_{i+1}. The soundness property of the accumulation scheme ensures that if acc_{i+1} is valid and the accumulation verifier accepts $((z_i, \pi_i), \mathsf{acc}_i, \mathsf{acc}_{i+1})$, then π_i is a valid proof and acc_i is a valid accumulator. Thus all that remains to maintain the invariant is for the prover to prove that the accumulation verifier accepts; this is possible provided that the *accumulation verifier* is sublinear.

From sketch to proof. In Section 5, we give the formal details of our construction and a proof of correctness. In particular, we show how to construct PCD, a more general primitive than IVC. In the PCD setting, rather than each computation step having a single input z_i, it receives m inputs from different nodes. Proving correctness hence requires proving that *all* of these inputs were computed correctly. For our construction, this entails checking m proofs and m accumulators. To do this, we extend the definition of an accumulation scheme to allow accumulating multiple instance-proof pairs and multiple "old" accumulators.

We now informally discuss the properties of our PCD construction.

- *Efficiency requirements.* Observe that the statement to be proved includes only the *accumulation verifier*, and so the *only* efficiency requirement for obtaining PCD is that this algorithm run in time sublinear in the size of the circuit R. This implies, in particular, that an accumulator must be of size sublinear in the size of R, and hence must not grow with each accumulation step. The SNARK verifier and the decider algorithm need only be efficient in the usual sense (i.e., polynomial-time). See Section 5.2 for a detailed analysis.

- *Soundness.* We prove that the PCD scheme is sound provided that the SNARK is knowledge sound (i.e., is an adaptively-secure argument of knowledge) and the accumulation scheme is sound (see Section 2.2 for more on what this means). We stress that in both cases security should be in the standard (CRS) model, without any random oracles (as in prior PCD constructions). See Section 5.4 for details.

- *Zero knowledge.* We prove that the PCD scheme is zero knowledge, if the underlying SNARK and accumulation scheme are both zero knowledge (for this part we also formulate a suitable notion of zero knowledge for accumulation schemes). See Section 5.5 for details.

- *Post-quantum security.* We also prove that if both the SNARK and accumulation scheme are *post-quantum* secure, then so is the resulting PCD scheme. Here by post-quantum secure we mean that the relevant security properties continue to hold even against polynomial-size *quantum* circuits, as opposed to just polynomial-size *classical* circuits. See Section 5.6 for details.

2.2 Accumulation schemes

A significant contribution of this work is formulating a general notion of an accumulation scheme. An accumulation scheme for a non-interactive argument as described above is a particular instance of this definition; in subsequent sections we will apply the definition in other settings.

We first give an informal definition that captures the key features of an accumulation scheme. For clarity this is stated for the (minimal) case of a single predicate input q and a single "old" accumulator acc; we later extend this in the natural way to n predicate inputs and m "old" accumulators.

Definition 2.1 (informal)**.** *An* **accumulation scheme** *for a predicate $\Phi \colon X \to \{0,1\}$ consists of a triple of algorithms $(\mathrm{P}, \mathrm{V}, \mathrm{D})$, known as the prover, verifier, and decider, that satisfies the following properties.*

- Completeness: *For all accumulators acc and predicate inputs $\mathsf{q} \in X$, if $\mathrm{D}(\mathsf{acc}) = 1$ and $\Phi(\mathsf{q}) = 1$, then for $\mathsf{acc}^\star \leftarrow \mathrm{P}(\mathsf{acc}, \mathsf{q})$ it holds that $\mathrm{V}(\mathsf{acc}, \mathsf{q}, \mathsf{acc}^\star) = 1$ and $\mathrm{D}(\mathsf{acc}^\star) = 1$.*
- Soundness: *For all efficiently-generated accumulators $\mathsf{acc}, \mathsf{acc}^\star$ and predicate inputs $\mathsf{q} \in X$, if $\mathrm{D}(\mathsf{acc}^\star) = 1$ and $\mathrm{V}(\mathsf{acc}, \mathsf{q}, \mathsf{acc}^\star) = 1$ then, with all but negligible probability, $\Phi(\mathsf{q}) = 1$ and $\mathrm{D}(\mathsf{acc}) = 1$.*

An accumulation scheme for a SNARK is an accumulation scheme for the predicate induced by the argument verifier; in this case the predicate input q consists of an instance-proof pair $(\mathbb{x}, \pi)$. Note that the completeness requirement does not place any restriction on how the previous accumulator acc is generated; we require that completeness holds for any acc the decider D determines to be valid, and any q for which the predicate Φ holds. This is needed to obtain a similarly strong notion of completeness for PCD, required for applications where accumulation is done by multiple parties that do not trust one another.

The above informal definition omits many important details; we now highlight some of these.

Predicate specification. Suppose that, as required for IVC/PCD, we have some fixed circuit R for which we want to accumulate pairs $(\mathbb{x}_i, \pi_i)$, where π_i is a SNARK proof that there exists $\mathbb{w}_i$ such that $R(\mathbb{x}_i, \mathbb{w}_i) = 1$. In this case the predicate corresponding to the verifier depends not only on the pair $(\mathbb{x}_i, \pi_i)$, but also on the circuit R, as well as the public parameters of the argument scheme pp and (often) a random oracle ρ.

Moreover, each of these inputs has different security and efficiency considerations. The security of the SNARK (and the accumulation scheme) can only be guaranteed with high probability over public parameters drawn by the generator algorithm of the SNARK, and over the random oracle. The circuit R may be chosen adversarially, but cannot be part of the input q because it is too large; it must be fixed at the beginning.

These considerations lead us to define an accumulation scheme with respect to both a predicate $\Phi \colon \mathcal{U}(*) \times (\{0,1\}^*)^3 \to \{0,1\}$ and a *predicate-specification algorithm* $\mathcal{H}$. We then adapt Definition 2.1 to hold for the predicate $\Phi(\rho, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \cdot)$ where ρ is a random oracle, pp_Φ is output by $\mathcal{H}^\rho$, and i_Φ is chosen adversarially. In our SNARK example, $\mathcal{H}$ is equal to the SNARK generator, i_Φ is the circuit R, and $\Phi(\rho, \mathsf{pp}, R, (\mathbb{x}, \pi)) = \mathcal{V}^\rho(\mathsf{pp}, R, \mathbb{x}, \pi)$. For a more precise description, see Section 4.

Zero knowledge. For our PCD application, the notion of zero knowledge for an accumulation scheme that we use is the following: one can sample a "fake" accumulator that is indistinguishable from a real accumulator $\mathsf{acc}^\star$, *without knowing anything* about the old accumulator acc and predicate input q that were accumulated in $\mathsf{acc}^\star$. The existence of the accumulation verifier V complicates matters here: if the adversary knows acc and q, then it is easy to distinguish a real accumulator from a fake one by running V on different inputs. We resolve this issue by having the accumulation prover P produce a *verification proof* π_{V} in addition to the new accumulator $\mathsf{acc}^\star$. Then V uses π_{V} in verifying the real accumulator, but π_{V} is *not* required in subsequent accumulations. In particular, the simulator does *not* have to simulate π_{V}. This avoids the problem described: even if the adversary knows acc and q, unless he provides a correct proof π_{V}, V can simply reject, as it would for a simulated accumulator. For the precise definition, see Section 4.

Remark 2.2 (helped verification)**.** We compare accumulation schemes for SNARKs with the notion of "helped verification" [MBKM19]. In a SNARK with helped verification, an untrusted party known as the *helper* can, given n proofs, produce an auxiliary proof that enables checking the n proofs at lower cost than that of checking each proof individually. This batching capability can be viewed as a special case of accumulation, as it applies to n "fresh" proofs only; there is no notion of batching "old" accumulators. It is unclear whether the weaker notion of helped verification alone suffices to construct IVC/PCD schemes.

2.3 Constructing arguments with accumulation schemes

A key ingredient in our construction of PCD is a SNARK that has an accumulation scheme (see Section 2.1). Below we summarize the ideas behind Theorem 2, by explaining how to construct accumulation schemes for SNARKs whose verifier is succinct relative to an oracle predicate $\Phi_\circ$ that itself has an accumulation scheme.

Predicate-efficient SNARKs. We call a SNARK ARG for *predicate-efficient* with respect to a predicate $\Phi_\circ$ if its verifier $\mathcal{V}$ operates as follows: (i) run a fast "inner" verifier $\mathcal{V}_{\mathsf{pe}}$ to produce a bit b and query set Q; (ii) accept iff $b = 1$ and for all $\mathsf{q} \in Q$, $\Phi_\circ(\mathsf{q}) = 1$. In essence, $\mathcal{V}$ can be viewed as a circuit with "oracle gates" for $\Phi_\circ$.[2] The aim is for $\mathcal{V}_{\mathsf{pe}}$ to be significantly more efficient than $\mathcal{V}$; that is, the queries to $\Phi_\circ$ capture the "expensive" part of the computation of $\mathcal{V}$.

As noted in Section 1.1, one can view recent SNARK constructions [MBKM19; GWC19; CHMMVW20] as being predicate-efficient with respect to a "polynomial commitment" predicate. We discuss how to construct accumulation schemes for these predicates below in Section 2.4.

Accumulation scheme for predicate-efficient SNARKs. Let ARG be a SNARK that is predicate-efficient with respect to a predicate $\Phi_\circ$, and let $\mathsf{AS}_\circ$ be an accumulation scheme for $\Phi_\circ$. To check n proofs, instead of directly invoking the SNARK verifier $\mathcal{V}$, we can first run $\mathcal{V}_{\mathsf{pe}}$ n times to generate n query sets for $\Phi_\circ$, and then, instead of invoking $\Phi_\circ$ on each of these sets, we can accumulate these queries using $\mathsf{AS}_\circ$. Below we sketch the construction of an accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for ARG based on this idea.

To accumulate n instance-proof pairs $[(\mathbb{x}_i, \pi_i)]_{i=1}^n$ starting from an old accumulator acc, the accumulation prover $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{P}$ first invokes the inner verifier $\mathcal{V}_{\mathsf{pe}}$ on each $(\mathbb{x}_i, \pi_i)$ to generate a query set Q_i for $\Phi_\circ$, accumulates their union $Q = \cup_{i=1}^n Q_i$ into acc using $\mathsf{AS}_\circ.\mathrm{P}$, and finally outputs the resulting accumulator $\mathsf{acc}^\star$. To check that $\mathsf{acc}^\star$ indeed accumulates $[(\mathbb{x}_i, \pi_i)]_{i=1}^n$ into acc, the accumulation verifier $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ first checks, for each i, whether the inner verifier $\mathcal{V}_{\mathsf{pe}}$ accepts $(\mathbb{x}_i, \pi_i)$, and then invokes $\mathsf{AS}_\circ.\mathrm{V}$ to check whether $\mathsf{acc}^\star$ correctly accumulates the query set $Q = \cup_{i=1}^n Q_i$. Finally, to decide whether $\mathsf{acc}^\star$ is a valid accumulator, the accumulation scheme decider $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}$ simply invokes $\mathsf{AS}_\circ.\mathrm{D}$.

From sketch to proof. The foregoing sketch omits details required to construct a scheme that satisfies the "full" definition of accumulation schemes as stated in Section 4. For instance, as noted in Section 2.3, the predicate $\Phi_\circ$ may be be an oracle predicate, and could depend on the public parameters of the SNARK ARG. We handle this by requiring that the accumulation scheme for $\Phi_\circ$ uses the SNARK generator $\mathcal{G}$ as its predicate specification algorithm. We also show that zero knowledge and post-quantum security are preserved. See Section 6 for a formal treatment of these issues, along with security proofs.

From predicate-efficient SNARKs to PCD. In order to build an accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ that suffices for PCD, ARG and $\mathsf{AS}_\circ$ must satisfy certain efficiency properties. In particular, when verifying satisfiability for a circuit of size N, the running time of $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ must be sublinear in N, which means in turn that the running times of $\mathcal{V}_{\mathsf{pe}}$ and $\mathsf{AS}_\circ.\mathrm{V}$, as well as the size of the query set Q, must be sublinear in N. Crucially, however, $\mathsf{AS}_\circ.\mathrm{D}$ need only run in time polynomial in N. For further discussion, see Remark 6.3.

[2] This is not precisely the case, because the verifier is required to reject immediately if it ever makes a query q with $\Phi_\circ(\mathsf{q}) = 0$.

2.4 Accumulation schemes for polynomial commitments

As noted in Section 2.3, several SNARK constructions (e.g., [MBKM19; GWC19; CHMMVW20]) are predicate-efficient with respect to an underlying *polynomial commitment*, which means that constructing an accumulation scheme for the latter leads (via Theorem 2) to an accumulation scheme for the whole SNARK.

Informally, a polynomial commitment scheme (PC scheme) is a cryptographic primitive that enables one to produce a commitment C to a polynomial p, and then to prove that this committed polynomial evaluates to a claimed value v at a desired point z. (See Section 3.6 for a definition.) An accumulation scheme for a PC scheme thus accumulates claims of the form "C commits to p such that $p(z) = v$" for arbitrary polynomials p and evaluation points z.

In this section, we explain the ideas behind Theorem 3, by sketching how to construct accumulation schemes for two popular polynomial commitment schemes.

- In Section 2.4.1, we sketch our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$, a polynomial commitment scheme derived from [BCCGP16; BBBPWM18; WTSTW18] that is based on the hardness of discrete logarithms.
- In Section 2.4.2, we sketch our (zero knowledge) accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$, a (hiding) polynomial commitment scheme based on knowledge assumptions over bilinear groups [KZG10; CHMMVW20].

In each case, the running time of the accumulation verifier will be sublinear in the degree of the polynomial, and the accumulator itself will not grow with the number of accumulation steps. This allows the schemes to be used, in conjunction with a suitable predicate-efficient SNARK, to construct PCD.

We remark that each of our accumulation schemes is proved secure in the random oracle model by invoking a useful lemma about "zero-finding games" for committed polynomials (Lemma 3.3). Security also requires that the random oracle used for an accumulation scheme for a PC scheme is domain-separated from the random oracle used by the PC scheme itself.

2.4.1 Accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$

We sketch our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$. For univariate polynomials of degree less than d, $\mathsf{PC}_{\mathsf{DL}}$ achieves evaluation proofs of size $O(\lambda \log d)$ in the random oracle model, and assuming the hardness of the discrete logarithm problem in a prime order group $\mathbb{G}$. In particular, there are no secret parameters (so-called "toxic waste"). However, $\mathsf{PC}_{\mathsf{DL}}$ has poor verification complexity: checking an evaluation proof requires $\Omega(d)$ scalar multiplications in $\mathbb{G}$. Bowe, Grigg, and Hopwood [BGH19] suggested a way to amortize this cost across a batch of n proofs. Below we show that their idea leads to an accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$ with an accumulation verifier that uses only $O(n \log d)$ scalar multiplications instead of the naive $\Theta(n \cdot d)$, and with an accumulator of size $O(\log d)$ elements in $\mathbb{G}$.

Summary of $\mathsf{PC}_{\mathsf{DL}}$ (see Appendix A for details). The committer and receiver both sample (consistently via the random oracle) a list of group elements $\{G_0, G_1, \dots, G_d\} \in \mathbb{G}^{d+1}$ in a group $\mathbb{G}$ of prime order q (written additively). A commitment to a polynomial $p(X) = \sum_{i=0}^{d} a_i X^i \in \mathbb{F}_q^{\leq d}[X]$ is then given by $C := \sum_{i=0}^{d} a_i G_i$. To prove that the committed polynomial p evaluates to v at a given point $z \in \mathbb{F}_q$, it suffices to prove that the triple (C, z, v) satisfies the following NP statement:

$$\exists\, a_0, \dots, a_d \in \mathbb{F} \text{ s.t. } v = \textstyle\sum_{i=0}^{d} a_i z^i \text{ and } C = \sum_{i=0}^{d} a_i G_i \ .$$

This is a special case of an *inner product argument* (IPA), as defined in [BCCGP16], which proves the inner product of two committed vectors. The receiver simply verifies this inner product argument to check the evaluation. The fact that the vector $(1, z, \dots, z^d)$ is known to the verifier and has a certain structure is exploited in the accumulation scheme that we describe below.

Accumulation scheme for the IPA. Our accumulation scheme relies on a special structure of the IPA verifier: it generates $O(\log d)$ challenges using the random oracle, then performs cheap checks requiring $O(\log d)$ field and group operations, and finally performs an expensive check requiring $\Omega(d)$ scalar multiplications. This latter check asserts consistency between the challenges and a group element U contained in the proof. Hence, the IPA verifier is succinct *barring the expensive check*, and so constructing an accumulation scheme for the IPA reduces to the task of constructing an accumulation scheme for the expensive check involving U.

To do this, we rely on an idea of Bowe, Grigg, and Hopwood [BGH19], which itself builds on an observation in [BBBPWM18]. Namely, letting $(\xi_1, \ldots, \xi_{\log_2 d})$ be the protocol's challenges, U can be viewed as a commitment to the polynomial $h(X) := \prod_{i=0}^{\log_2(d)-1}(1 + \xi_{\log_2(d)-i} X^{2^i}) \in \mathbb{F}_q^{\leq d}[X]$. This polynomial has the special property that it can be evaluated at any point in just $O(\log d)$ field operations (exponentially smaller than its degree d). This allows transforming the expensive check on U into a check that is amenable to batching: instead of directly checking that U is a commitment to h, one can instead check that the polynomial committed inside U agrees with h at a challenge point z sampled via the random oracle.

We leverage this idea as follows. When accumulating evaluation claims about multiple polynomials $p_1, \ldots, p_n$, applying the foregoing transformation results in n checks of the form "check that the polynomial contained in U_i evaluates to $h_i(z)$ at the point z". Because these are all claims for the correct evaluation of the polynomials h_i at *the same point* z, we can accumulate them via standard homomorphic techniques. We now summarize how we apply this idea to construct our accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{DL}}$.

Accumulators in our accumulation scheme have the same form as the instances to be accumulated: they are tuples of the form (C, z, v, π) where π is an evaluation proof for the claim "$p(z) = v$" and p is the polynomial committed in C. For simplicity, below we consider the case of accumulating one old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and one instance (C_2, z_2, v_2, π_2) into a new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$.

Accumulation prover P*:* compute the new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$ from the old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and the instance (C_2, z_2, v_2, π_2) as follows.

- Compute U_1, U_2 from π_1, π_2 respectively. As described above, these elements can be viewed as commitments to polynomials h_1, h_2 defined by the challenges derived from π_1, π_2.
- Use the random oracle ρ to compute the random challenge $\alpha := \rho([(h_1, U_1), (h_2, U_2)])$.
- Compute $C := U_1 + \alpha U_2$, which is a polynomial commitment to $p(X) := h_1(X) + \alpha h_2(X)$.
- Compute the challenge point $z := \rho(C, p)$, where p is uniquely represented via the tuple $([h_1, h_2], \alpha)$.
- Construct an evaluation proof π for the claim "$p(z) = v$". (This step is the only expensive one.)
- Output the new accumulator $\mathsf{acc}^\star := (C, z, v, \pi)$.

Accumulation verifier V*:* to check that the new accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$ was correctly generated from the old accumulator $\mathsf{acc} = (C_1, z_1, v_1, \pi_1)$ and the instance (C_2, z_2, v_2, π_2), first compute the challenges α and z from the random oracle as above, and then check that (a) (C_1, z_1, v_1, π_1) and (C_2, z_2, v_2, π_2) pass the cheap checks of the IPA verifier, (b) $C = U_1 + \alpha U_2$, and (c) $h_1(z) + \alpha h_2(z) = v$.

Decider D*:* on input the (final) accumulator $\mathsf{acc}^\star = (C, z, v, \pi)$, check that π is a valid evaluation proof for the claim that the polynomial committed inside C evaluates to v at the point z.

This construction achieves the efficiency summarized in Theorem 3.

In Section 7, we show how to extend the above accumulation scheme to accumulate any number of old accumulators and instances. Our security proof for the resulting accumulation scheme relies on the hardness of zero-finding games (Lemma 3.3), and the security of $\mathsf{PC}_{\mathsf{DL}}$.

2.4.2 Accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$

We sketch our accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{AGM}}$. Checking an evaluation proof in $\mathsf{PC}_{\mathsf{AGM}}$ requires 1 pairing, and so checking n evaluation proofs requires n pairings. AS improves upon this as follows: the accumulation verifier V only performs $O(n)$ scalar multiplications in $\mathbb{G}_1$ in order to check the accumulation of n evaluation proofs, while the decider D performs only a single pairing in order to check the resulting accumulator. This is much cheaper: it reduces the number of pairings from n to 1, and also defers this single pairing to the end of the accumulation (the decider). In particular, when instantiating the PCD construction outlined in Section 2.1 with a $\mathsf{PC}_{\mathsf{AGM}}$-based SNARK and our accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$, we can eliminate *all* pairings from the circuit being verified in the PCD construction.

Below we explain how standard techniques for batching pairings using random linear combinations [CHMMVW20] allow us to realize an accumulation scheme for $\mathsf{PC}_{\mathsf{AGM}}$ with these desirable properties.

Summary of $\mathsf{PC}_{\mathsf{AGM}}$. The committer key ck and receiver key rk for a given maximum degree bound D are group elements from a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$: $\mathsf{ck} := \{G, \beta G, \dots, \beta^D G\} \in \mathbb{G}_1^{D+1}$ consists of group elements encoding powers of a random field element β, while $\mathsf{rk} := (G, H, \beta H) \in \mathbb{G}_1 \times \mathbb{G}_2^2$.

A commitment to a polynomial $p \in \mathbb{F}_q^{\leq D}[X]$ is the group element $C := p(\beta)G \in \mathbb{G}_1$. To prove that p evaluates to v at a given point $z \in \mathbb{F}_q$, the sender computes a "witness polynomial" $w(X) := (p(X) - v)/(X - z)$, and outputs the evaluation proof $\pi := w(\beta)G \in \mathbb{G}_1$. The receiver can check this proof by checking the pairing equation $e(C - vG, H) = e(\pi, \beta H - zH)$. This pairing equation is the focus of our accumulation scheme below. (This summary omits details about degree enforcement and about hiding.)

Accumulation scheme. We construct an accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for $\mathsf{PC}_{\mathsf{AGM}}$ by relying on standard techniques for batching pairing equations. Suppose that we wish to simultaneously check the validity of n instances $[(C_i, z_i, v_i, \pi_i)]_{i=1}^n$. First, rewrite the pairing check for the i-th instance as follows:

$$e(C_i - v_i G, H) = e(\pi_i, \beta H - z_i H) \iff e(C_i - v_i G + z_i \pi_i, H) = e(\pi_i, \beta H) \ . \tag{1}$$

After the rewrite, the $\mathbb{G}_2$ inputs to both pairings do not depend on the claim being checked. This allows batching the pairing checks by taking a random linear combination with respect to a random challenge $r := \rho([C_i, z_i, v_i, \pi_i]_{i=1}^n)$ computed from the random oracle, resulting in the following combined equation:

$$e(\textstyle\sum_{i=1}^n r^i (C_i - v_i G + z_i \pi_i), H) = e(\textstyle\sum_{i=1}^n r^i \pi_i, \beta H) \ . \tag{2}$$

We now have a pairing equation involving an "accumulated commitment" $C^\star := \sum_{i=1}^n r^i (C_i - v_i G + z_i \pi_i)$ and an "accumulated proof" $\pi^\star := \sum_{i=1}^n r^i \pi_i$. This observation leads to the accumulation scheme below.

An accumulator in AS consists of a commitment-proof pair $(C^\star, \pi^\star)$, which the decider D validates by checking that $e(C^\star, H) = e(\pi^\star, \beta H)$. Moreover, observe that by Eq. (1), checking the validity of a claimed evaluation (C, z, v, π) within $\mathsf{PC}_{\mathsf{AGM}}$ corresponds to checking that the "accumulator" $(C - vG + z\pi, \pi)$ is accepted by the decider D. Thus we can restrict our discussion to accumulating *accumulators*.

The accumulation prover P, on input a list of old accumulators $[\mathsf{acc}_i]_{i=1}^n = [(C_i^\star, \pi_i^\star)]_{i=1}^n$, computes a random challenge $r := \rho([\mathsf{acc}_i]_{i=1}^n)$, constructs $C^\star := \sum_{i=1}^n r^i C_i^\star$ and $\pi^\star := \sum_{i=1}^n r^i \pi_i^\star$, and outputs the new accumulator $\mathsf{acc}^\star := (C^\star, \pi^\star) \in \mathbb{G}_1^2$. To check that $\mathsf{acc}^\star$ accumulates $[\mathsf{acc}_i]_{i=1}^n$, the accumulation verifier V simply invokes P and checks that its output matches the claimed new accumulator $\mathsf{acc}^\star$.

This construction achieves the efficiency summarized in Theorem 3.

In Section 8, we show how to extend the above accumulation scheme to account for additional features of $\mathsf{PC}_{\mathsf{AGM}}$ (degree enforcement and hiding). Our security proof for the resulting accumulation scheme relies on the hardness of zero-finding games (Lemma 3.3). We also extend our construction to achieve zero knowledge by having the prover generate an extra accumulator corresponding to a "random" polynomial, which is provided to the verifier as π_{V}. This randomizes $\mathsf{acc}^\star$, statistically hiding the accumulated claims.

3 Preliminaries

Indexed relations. An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness; the corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathbb{i}$ is the description of a boolean circuit, $\mathbb{x}$ is a partial assignment to its input wires, and $\mathbb{w}$ is an assignment to the remaining wires that makes the circuit to output 0.

Security parameters. For simplicity of notation, we assume that all public parameters have length at least λ, so that algorithms which receive such parameters can run in time $\mathrm{poly}(\lambda)$.

Random oracles. We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. We denote by $\mathcal{U}(*)$ the set $\bigcup_{\lambda \in \mathbb{N}} \mathcal{U}(\lambda)$. A *random oracle* with security parameter λ is a function $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

3.1 Preprocessing non-interactive arguments in the ROM

A tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a *preprocessing non-interactive argument* in the random oracle model (ROM) for an indexed relation $\mathcal{R}$ if the following properties hold.

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

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

- **Soundness.** For every polynomial-size adversary $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \mathcal{V}^\rho(\mathrm{ivk}, \mathbb{x}, \pi) = 1 & (\mathbb{i}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^\rho(\mathsf{pp}) \\ & (\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

The above formulation of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on the random oracle ρ and public parameters pp, and the above formulation of soundness allows $(\mathbb{i}, \mathbb{x})$ to depend on the random oracle ρ and public parameters pp.

Our PCD construction makes use of the stronger property of *knowledge soundness*, and optionally also the property of (statistical) *zero knowledge*. We define both of these properties below. Note that this definition is stronger the standard definition of knowledge soundness; this is required to prove post-quantum security in Theorem 5.2. This stronger definition is similar to the notion of *witness-extended emulation* [Lin03].

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has *knowledge soundness* if for every polynomial-size adversary $\tilde{\mathcal{P}}$ and every polynomial-size auxiliary input distribution $\mathcal{D}$ there exists an efficient extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{c|r} \forall j \in [\ell],\ \Big(\mathcal{V}^\rho(\mathrm{ivk}_j, \mathbb{x}_j, \pi_j) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \Downarrow & z \leftarrow \mathcal{D}^\rho(\mathsf{pp}) \\ (\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \in \mathcal{R}\Big) & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}^\rho(\mathsf{pp}, z) \\ & \forall j\ ,\ (\mathrm{ipk}_j, \mathrm{ivk}_j) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i}_j) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda)$$

and, moreover, the following distributions are statistically close (as a function of λ)

$$\left\{ \begin{array}{c} (\rho, \mathsf{pp}, \vec{\mathbb{i}}, \\ \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ z \leftarrow \mathcal{D}^{\rho}(\mathsf{pp}) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^{\rho}(\mathsf{pp}, z) \end{array} \right\} \text{ and } \left\{ \begin{array}{c} (\rho, \mathsf{pp}, \vec{\mathbb{i}}, \\ \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ z \leftarrow \mathcal{D}^{\rho}(\mathsf{pp}) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}^{\rho}(\mathsf{pp}, z) \end{array} \right\} .$$

The above definition is polynomially related to the standard definition of adaptive knowledge soundness, which does not consider a vector of outputs or an auxiliary output by the prover.

Zero knowledge. We say that $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every polynomial-size honest adversary $\mathcal{A}$ the distributions below are computationally indistinguishable:

$$\left\{ (\rho, \mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi) \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ (\text{ipk}, \text{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}^{\rho}(\text{ipk}, \mathbb{x}, \mathbb{w}) \end{array} \right\} \text{ and } \left\{ (\rho[\mu], \mathsf{pp}, \mathbb{i}, \mathbb{x}, \pi) \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \tau) \leftarrow \mathcal{S}^{\rho}(\mathbb{i}, \mathbb{x}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ (\pi, \mu) \leftarrow \mathcal{S}^{\rho}(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \tau) \end{array} \right\} .$$

Above, $\rho[\mu]$ is the function that, on input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise. This definition uses explicitly-programmable random oracles [BR93]. (Non-interactive zero knowledge with non-programmable random oracles is impossible for non-trivial languages [Pas03; BCS16].)

3.2 Preprocessing PCD

A triple of algorithms $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ is a *proof-carrying data scheme* (PCD scheme) for a class of compliance predicates F if the properties below hold.

Definition 3.1. *A* **transcript** T *is a directed acyclic graph where each vertex $u \in V(\mathsf{T})$ is labeled by local data $z_{\mathsf{loc}}^{(u)}$ and each edge $e \in E(\mathsf{T})$ is labeled by a message $z^{(e)} \neq \bot$. The* **output** *of a transcript T, denoted $\mathsf{o}(\mathsf{T})$, is $z^{(e)}$ where $e = (u, v)$ is the lexicographically-first edge such that v is a sink.*

Definition 3.2. *A vertex $u \in V(\mathsf{T})$ is* φ**-compliant** *for $\varphi \in \mathsf{F}$ if for all outgoing edges $e = (u, v) \in E(\mathsf{T})$:*
- *(base case) if u has no incoming edges, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \bot, \ldots, \bot)$ accepts;*
- *(recursive case) if u has incoming edges $e_1, \ldots, e_m$, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, \ldots, z^{(e_m)})$ accepts.*

We say that T is φ**-compliant** *if all of its vertices are φ-compliant.*

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

$$\Pr\left[\begin{array}{c} \Big(\varphi \in \mathsf{F} \wedge \big(\forall i, z_i = \bot \vee \forall i,\ \mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1\big) \wedge \\ \varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) \text{ accepts}\Big) \\ \Downarrow \\ \mathbb{V}(\mathsf{ivk}, z, \pi) = 1 \end{array} \middle| \begin{array}{r} \mathbb{pp} \leftarrow \mathbb{G}(1^{\lambda}) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^{m}) \leftarrow \mathcal{A}(\mathbb{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^{m}) \end{array} \right] = 1 .$$

Knowledge soundness. We say that $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has knowledge soundness if there exists some polynomial e such that for every polynomial-size adversary $\tilde{\mathbb{P}}$, there exists an extractor $\mathbb{E}$ of size at most

$e(|\tilde{\mathbb{P}}|)$ such that

$$\Pr\left[\begin{array}{c|r}\Big(\varphi \in \mathsf{F} \wedge \mathbb{V}(\mathsf{ivk}, \mathsf{o}(\mathsf{T}), \pi) = 1\Big) & \mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \Downarrow & (\varphi, \pi, \mathsf{T}) \leftarrow \mathbb{E}(\mathbb{pp}) \\ \mathsf{T} \text{ is } \varphi\text{-compliant} & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi)\end{array}\right] \geq 1 - \operatorname{negl}(\lambda) \ .$$

and, moreover, the following distributions are statistically close:

$$\left\{(\varphi, \mathsf{o}, \pi) \;\middle|\; \begin{array}{r}\mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, \mathsf{o}, \pi) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp})\end{array}\right\} \quad \text{and} \quad \left\{(\varphi, \mathsf{o}(\mathsf{T}), \pi) \;\middle|\; \begin{array}{r}\mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, \pi, \mathsf{T}) \leftarrow \mathbb{E}(\mathbb{pp})\end{array}\right\} \ .$$

Zero knowledge. We say that $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbb{S}$ such that for every polynomial-size *honest* adversary $\mathcal{A}$ the distributions below are computationally indistinguishable:

$$\left\{(\mathbb{pp}, \pi) \;\middle|\; \begin{array}{r}\mathbb{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathbb{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, \varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m)\end{array}\right\} \quad \text{and} \quad \left\{(\mathbb{pp}, \pi) \;\middle|\; \begin{array}{r}(\mathbb{pp}, \tau) \leftarrow \mathbb{S} \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp}) \\ \pi \leftarrow \mathbb{S}(\varphi, z, \tau)\end{array}\right\} \ .$$

In this case, $\mathcal{A}$ is honest if it outputs, with probability 1, $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m)$ such that $\varphi \in \mathsf{F}$, $\mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1$ for all i, and $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts.

Efficiency. The generator $\mathbb{G}$, prover $\mathbb{P}$, indexer $\mathbb{I}$ and verifier $\mathbb{V}$ run in polynomial time. A proof π has size $\operatorname{poly}(\lambda, |\varphi|)$; in particular, it is not permitted to grow with each application of $\mathbb{P}$.

3.3 Instantiating the random oracle

Almost all of the results in this paper are proved in the *random oracle model*, and so we give definitions which include random oracles. The single exception is our construction of proof-carrying data, in Section 5.1. We do not know how to build PCD schemes which are secure in the random oracle model from any standard assumption. Instead, we show that assuming the existence of a non-interactive argument with security in the standard (CRS) model, we obtain a PCD scheme which is also secure in the standard (CRS) model.

For this reason, the definition of PCD above is stated in the standard model (without oracles). We do not explicitly define non-interactive arguments in the standard model; the definition is easily obtained by removing the random oracle from the definition presented in Section 3.1.

3.4 Post-quantum security

The definitions of both non-interactive arguments (in the standard model) and proof-carrying data can be strengthened, in a straightforward way, to express post-quantum security. In particular, we replace "polynomial-size circuit" and "polynomial-time algorithm" with their quantum analogues. Since we do not prove post-quantum security of any construction in the random oracle model, we do not discuss the quantum random oracle model.

3.5 Commitment schemes

A commitment scheme $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ enables one to create binding commitments to messages.

- $\mathsf{CM.Setup}$, on input a *message format* L, outputs public parameters pp; this specifies a message universe $\mathcal{M}_{\mathsf{pp}}$ and a commitment universe $\mathcal{C}_{\mathsf{pp}}$.
- $\mathsf{CM.Trim}$, on input public parameters pp and a *trim specification* ℓ, outputs a commitment key ck containing a description of a message space $\mathcal{M}_{\mathsf{ck}} \subseteq \mathcal{M}_{\mathsf{pp}}$ (corresponding to ℓ).
- $\mathsf{CM.Commit}$, on input a commitment key ck, a message $m \in \mathcal{M}_{\mathsf{ck}}$ and randomness r, outputs a commitment $C \in \mathcal{C}_{\mathsf{pp}}$.

CM is binding if, for every message format L such that $|L| = \mathrm{poly}(\lambda)$, and for every efficient adversary $\mathcal{A}$, the following holds.

$$\Pr\left[\begin{array}{c|r} m_1 \in \mathcal{M}_{\mathsf{ck}_1},\, m_2 \in \mathcal{M}_{\mathsf{ck}_2},\, m_1 \neq m_2 & \mathsf{pp} \leftarrow \mathsf{CM.Setup}^{\rho}(1^\lambda, L) \\ \wedge & ((\ell_1, m_1, r_1), (\ell_2, m_2, r_2)) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ \mathsf{CM.Commit}(\mathsf{ck}_1, m_1; r_1) = \mathsf{CM.Commit}(\mathsf{ck}_2, m_2; r_2) & \mathsf{ck}_1 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_1) \\ & \mathsf{ck}_2 \leftarrow \mathsf{CM.Trim}^{\rho}(\mathsf{pp}, \ell_2) \end{array}\right] \leq \mathrm{negl}(\lambda) \ .$$

Note that $m_1 \neq m_2$ is well-defined since $\mathcal{M}_{\mathsf{ck}_1}, \mathcal{M}_{\mathsf{ck}_2} \subseteq \mathcal{M}_{\mathsf{pp}}$.

We now give a useful lemma, which bounds the probability that applying the random oracle to a commitment to a polynomial yields a zero of that polynomial. We refer to this as a *zero-finding game*.

Lemma 3.3. *Let $F\colon \mathbb{N} \to \mathbb{N}$, and $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ be a commitment scheme. Fix a number of variables $M \in \mathbb{N}$ and maximum degree $N \in \mathbb{N}$. Then for every family of (not necessarily efficient) functions $\{f_{\mathsf{pp}}\}_{\mathsf{pp}}$ and fields $\{\mathbb{F}_{\mathsf{pp}}\}_{\mathsf{pp}}$ where $f_{\mathsf{pp}}\colon \mathcal{M}_{\mathsf{pp}} \to \mathbb{F}_{\mathsf{pp}}^{\leq N}[X_1, \ldots, X_M]$ and $|\mathbb{F}_{\mathsf{pp}}| \geq F(\lambda)$; for every message format L and efficient t-query oracle algorithm $\mathcal{A}$, the following holds.*

$$\Pr\left[\begin{array}{c|r} & \rho \leftarrow \mathcal{U}(\lambda) \\ & \mathsf{pp} \leftarrow \mathsf{CM.Setup}(1^\lambda, L) \\ p \not\equiv 0 & (\ell, \mathfrak{p} \in \mathcal{M}_{\mathsf{ck}}, r) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}) \\ \wedge & \mathsf{ck} \leftarrow \mathsf{CM.Trim}(\mathsf{pp}, \ell) \\ p(\boldsymbol{z}) = 0 & C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, \mathfrak{p}; r) \\ & \boldsymbol{z} \in \mathbb{F}_{\mathsf{pp}}^{N} \leftarrow \rho(C) \\ & p \leftarrow f_{\mathsf{pp}}(\mathfrak{p}) \end{array}\right] \leq \sqrt{(t+1) \cdot \frac{MN}{F(\lambda)}} + \mathrm{negl}(\lambda) \ .$$

If CM is perfectly binding, then the above holds also for computationally-unbounded t-query adversaries $\mathcal{A}$.

Proof. Let $\mathcal{A}$ be an adversary which wins with probability δ in the above game; we construct an adversary $\mathcal{B}$ which breaks the binding of the commitment scheme with probability at least $\delta^2/(t+1) - \frac{MN}{F(\lambda)}$.

Note first that we may assume that $\mathcal{A}$ always queries $C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, \mathfrak{p}; r)$ for its output $(\mathfrak{p}, r)$, by increasing the query bound from t to $t+1$.

```
$\mathcal{B}(\mathsf{pp})$:
1. Run $(\ell, \mathfrak{p}, r) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp})$, simulating its queries to $\rho$.
2. Compute $\mathsf{ck} \leftarrow \mathsf{CM.Trim}(\mathsf{pp}, \ell)$.
3. Obtain $C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, \mathfrak{p}; r)$.
4. Rewind $\mathcal{A}$ to the query $\rho(C)$ and run to the end, drawing fresh randomness for this and subsequent
   oracle queries, to obtain $(\ell', \mathfrak{p}', r')$.
```

5. Output $((\ell, \mathfrak{p}, r), (\ell', \mathfrak{p}', r'))$.

Let $\mathsf{ck}' := \mathsf{CM.Trim}(\mathsf{pp}, \ell')$, $C' := \mathsf{CM.Commit}(\mathsf{ck}', \mathfrak{p}'; r')$, $\boldsymbol{z} := \rho(C)$, $\boldsymbol{z}' := \rho(C')$, $p := f_{\mathsf{pp}}(\mathfrak{p})$ and $p' := f_{\mathsf{pp}}(\mathfrak{p}')$. By the forking lemma, the probability that $p(\boldsymbol{z}) = p'(\boldsymbol{z}') = 0$ and $C = C'$ is at least $\delta^2/(t+1)$; call this event E. Then

$$\Pr[E] \leq \Pr[E \wedge (p = p')] + \Pr[E \wedge (p \neq p')] \leq MN/F(\lambda) + \Pr[E \wedge (\mathfrak{p} \neq \mathfrak{p}')] \ .$$

The lemma follows by noting that $E \wedge (\mathfrak{p} \neq \mathfrak{p}')$ implies that $\mathcal{B}$ breaks the binding property of CM. □

Remark 3.4. For Lemma 3.3 to hold, the algorithms of CM *must not* have access to the random oracle ρ used to generate the challenge point $\boldsymbol{z}$. The lemma is otherwise black-box with respect to CM, and so CM itself may use other oracles. The lemma continues to hold when $\mathcal{A}$ has access to these additional oracles. We use this fact later to justify the security of domain separation.

3.6 Polynomial commitments

A polynomial commitment scheme is a cryptographic primitive that enables a sender to commit to a polynomial p over a field $\mathbb{F}$ and then later prove the correct evaluation of the polynomial at a desired point. In more detail, a polynomial commitment scheme PC is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ with the following syntax and properties:

- $\mathsf{PC.Setup}^{\rho}(1^{\lambda}, D) \to \mathsf{pp}$. On input a security parameter λ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC.Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{PC}}$. The parameters contain the description of a finite field $\mathbb{F}$ (which has size that is super-polynomial in λ).

- $\mathsf{PC.Trim}^{\rho}(\mathsf{pp}, [d_i]_{i=1}^{n}) \to (\mathsf{ck}, \mathsf{rk})$. On input public parameters $\mathsf{pp}_{\mathsf{PC}}$, and degree bounds $[d_i]_{i=1}^{n}$, $\mathsf{PC.Trim}$ deterministically computes a key pair $(\mathsf{ck}, \mathsf{rk})$ that is specialized to $[d_i]_{i=1}^{n}$.

- $\mathsf{PC.Commit}^{\rho}(\mathsf{ck}, p, d; \omega) \to C$. On input ck, a univariate polynomial p over the field $\mathbb{F}$, and a degree bound d such that $\deg(p) \leq d \in [d_i]_{i=1}^{n}$, $\mathsf{PC.Commit}$ outputs a commitment C to the polynomial p. The randomness ω is used if the commitment C is hiding.

- $\mathsf{PC.Open}^{\rho}(\mathsf{ck}, p, C, d, z; \omega) \to \pi$. On input the commitment key ck, a univariate polynomial p over the field $\mathbb{F}$, a commitment C to p, a degree bound d, an evaluation point z, and commitment randomness ω, $\mathsf{PC.Open}$ outputs an evaluation proof π.

- $\mathsf{PC.Check}^{\rho}(\mathsf{rk}, C, d, z, v, \pi) \to b$. On input the receiver key rk, a commitment C, a degree bound d, an evaluation point z, a claimed evaluation v, and an evaluation proof π, $\mathsf{PC.Check}$ checks that the degree bound $d \in [d_i]_{i=1}^{n}$, and outputs 1 if π attests that the polynomial p committed in C has degree at most d and evaluates to v at z.

A polynomial commitment scheme PC must be such that $(\mathsf{PC.Setup}, \mathsf{PC.Trim}, \mathsf{PC.Commit})$ is a (binding) commitment scheme for bounded-degree polynomials over a field. The message format L is equal to the maximum degree bound D; the message universe is the set of polynomials over some field $\mathbb{F}$ of degree at most D. The trim specification ℓ is equal to the list of degree bounds $[d_i]_{i=1}^{n}$; the corresponding message space is the set of polynomials over $\mathbb{F}$ of degree at most $\max_i d_i$.

A polynomial commitment scheme must also satisfy the following additional properties.

Completeness. For every maximum degree bound $D = \mathrm{poly}(\lambda) \in \mathbb{N}$ and every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} d \in [d_i]_{i=1}^n & \mathsf{pp} \leftarrow \mathsf{PC.Setup}^\rho(1^\lambda, D) \\ \deg(p) \leq d \leq D & ([d_i]_{i=1}^n, p, d, z, \omega) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ \Downarrow & (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^\rho(\mathsf{pp}, [d_i]_{i=1}^n) \\ \mathsf{PC.Check}^\rho(\mathsf{rk}, C, d, z, v, \pi) = 1 & C \leftarrow \mathsf{PC.Commit}^\rho(\mathsf{ck}, p, d; \omega) \\ & v \leftarrow p(z) \\ & \pi \leftarrow \mathsf{PC.Open}^\rho(\mathsf{ck}, p, C, d, z; \omega) \end{array}\right] = 1 \ .$$

Extractability. For every maximum degree bound $D = \mathrm{poly}(\lambda) \in \mathbb{N}$ and polynomial-size adversary $\mathcal{A}$ there exists an efficient extractor $\mathcal{E}$ such that the following holds.

$$\Pr\left[\begin{array}{c|r} \mathsf{PC.Check}^\rho(\mathsf{rk}, C, d, z, v, \pi) = 1 & \\ \Downarrow & \rho \leftarrow \mathcal{U}(\lambda) \\ C = \mathsf{PC.Commit}^\rho(\mathsf{ck}, p, d; \omega) & \mathsf{pp} \leftarrow \mathsf{PC.Setup}^\rho(1^\lambda, D) \\ v = p(z) & ([d_i]_{i=1}^n, (C, d, z, v), \pi) \leftarrow \mathcal{A}^\rho(\mathsf{pp}) \\ d \in [d_i]_{i=1}^n & (p, \omega) \leftarrow \mathcal{E}^\rho(\mathsf{pp}) \\ \deg(p) \leq d \leq D & (\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^\rho(\mathsf{pp}, [d_i]_{i=1}^n) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \ .$$

4 Accumulation schemes

In Section 4.1 we formally define an accumulation scheme. In Section 4.2 we define accumulation schemes for predicates related to the cryptographic primitives used in this paper.

4.1 Definition

Let $\Phi \colon \mathcal{U}(*) \times (\{0,1\}^*)^3 \to \{0,1\}$ be a predicate (for clarity we write $\Phi^\rho(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q})$ for $\Phi(\rho, \mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q})$). Let $\mathcal{H}$ be a randomized algorithm with access to a (random) oracle, which outputs predicate parameters pp_Φ.

An **accumulation scheme for** $(\Phi, \mathcal{H})$ is a tuple of algorithms $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ all of which have access to the same random oracle ρ. The algorithms have the following syntax and properties.

Syntax. The algorithms comprising AS have the following syntax:

- *Generator:* On input a security parameter λ (in unary), G samples and outputs public parameters pp.
- *Indexer:* On input public parameters pp, predicate parameters pp_Φ (generated by $\mathcal{H}$), and a predicate index i_Φ, I deterministically computes and outputs a triple $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk})$ consisting of an accumulator proving key apk, an accumulator verification key avk, and a decision key dk.[3]
- *Accumulation prover:* On input the accumulator proving key apk, inputs $[\mathsf{q}_i]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j]_{j=1}^m$, P outputs a new accumulator acc and a proof π_V for the accumulation verifier.
- *Accumulation verifier:* On input the accumulator verification key avk, inputs $[\mathsf{q}_i]_{i=1}^n$, accumulator instances $[\mathsf{acc}_j]_{j=1}^m$, a new accumulator instance acc, and a proof π_V, V outputs a bit indicating whether $\mathbb{a}$ correctly accumulates $[\mathsf{q}_i]_{i=1}^n$ and $[\mathsf{acc}_j]_{j=1}^m$.
- *Decider:* On input the decision key dk, and an accumulator acc, D outputs a bit indicating whether acc is a valid accumulator.

These algorithms must satisfy two properties, *completeness* and *soundness*, defined below. We additionally define a notion of zero knowledge that we will rely on to achieve zero knowledge PCD (see Section 5).

Completeness. For all (unbounded) adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} \forall j \in [m],\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}_j) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \forall i \in [n],\ \Phi^\rho(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q}_i) = 1 & \mathsf{pp} \leftarrow \mathrm{G}^\rho(1^\lambda) \\ \Downarrow & \mathsf{pp}_\Phi \leftarrow \mathcal{H}^\rho(1^\lambda) \\ \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V}) = 1 & (\mathsf{i}_\Phi, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \leftarrow \mathcal{A}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi) \\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}) = 1 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ & (\mathsf{acc}, \pi_\mathrm{V}) \leftarrow \mathrm{P}^\rho(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m) \end{array}\right] = 1 \ .$$

Note that for $m = n = 0$, the precondition on the left-hand side holds vacuously; this is required for the completeness condition to be non-trivial.

Soundness. For every polynomial-size adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V}) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp} \leftarrow \mathrm{G}^\rho(1^\lambda) \\ \Downarrow & \mathsf{pp}_\Phi \leftarrow \mathcal{H}^\rho(1^\lambda) \\ \forall j \in [m],\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}_j) = 1 & \begin{pmatrix} \mathsf{i}_\Phi & [\mathsf{q}_i]_{i=1}^n & [\mathsf{acc}_j]_{j=1}^m \\ & \mathsf{acc} & \pi_\mathrm{V} \end{pmatrix} \leftarrow \mathcal{A}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi) \\ \forall i \in [n],\ \Phi^\rho(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q}_i) = 1 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \ .$$

[3] We remark that in some schemes it is important, for the sake of efficiency, for the indexer I to have oracle access to the predicate parameters pp_Φ and predicate index i_Φ, rather than reading them in full. All of our constructions and statements extend, in a straightforward way, to this case.

Zero knowledge. There exists a polynomial-time simulator S such that for every polynomial-size "honest" adversary $\mathcal{A}$ (see below) the following distributions are computationally indistinguishable:

$$\left\{ (\rho, \mathsf{pp}, \mathsf{acc}) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{G}^{\rho}(1^{\lambda}) \\ \mathsf{pp}_{\Phi} \leftarrow \mathcal{H}^{\rho}(1^{\lambda}) \\ (\mathsf{i}_{\Phi}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\Phi}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}) \\ (\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\rho}(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \end{array} \right\}$$

and

$$\left\{ (\rho[\mu], \mathsf{pp}, \mathsf{acc}) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{pp}, \tau) \leftarrow \mathrm{S}^{\rho}(1^{\lambda}) \\ \mathsf{pp}_{\Phi} \leftarrow \mathcal{H}^{\rho}(1^{\lambda}) \\ (\mathsf{i}_{\Phi}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\Phi}) \\ (\mathsf{acc}, \mu) \leftarrow \mathrm{S}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}, \tau) \end{array} \right\} .$$

Here $\mathcal{A}$ is *honest* if it outputs, with probability 1, a tuple $(\mathsf{i}_{\Phi}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m})$ such that $\Phi^{\rho}(\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}, \mathsf{q}_i) = 1$ and $\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}_j) = 1$ for all $i \in [n]$ and $j \in [m]$. Note that the simulator S is *not* required to simulate the accumulation verifier proof π_{V}.

Security in the standard model. The corresponding security definitions in the standard (CRS) model are obtained from the above by removing the random oracle ρ wherever it appears.

Post-quantum security. The post-quantum analogues of the above definitions are obtained by modifying the soundness and zero knowledge guarantees to quantify over polynomial-size *quantum* adversaries $\mathcal{A}$; we also strengthen the zero knowledge guarantee to require quantum computational indistinguishability (although we do not permit S to be quantum). In the random oracle variant of the definitions, we should also allow the adversary superposition access to the random oracle (i.e., require security in the *quantum* random oracle model [BDFLSZ11]). Note, however, that this latter issue is not relevant to the present work: our only result on post-quantum security is Theorem 5.2, which is in the standard (CRS) model.

4.2 Accumulation schemes for certain predicates

We conclude by specializing the definition of an accumulation scheme to the case of predicates induced by the verifier in a non-interactive argument (Definition 4.1) and in a polynomial commitment scheme (Definition 4.2).

Definition 4.1 (accumulation for ARG)**.** *A non-interactive argument system* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *has an accumulation scheme if the pair* $(\Phi_{\mathcal{V}}, \mathcal{G})$ *has an accumulation scheme, where* $\Phi_{\mathcal{V}}$ *is defined as follows:*

```
$\Phi_{\mathcal{V}}^{\rho}(\mathsf{i}_{\Phi} = (\mathsf{pp}, \mathbb{i}), \mathsf{q} = (\mathbb{x}, \pi))$:
1. $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i})$.
2. Output $\mathcal{V}^{\rho}(\mathrm{ivk}, \mathbb{x}, \pi)$.
```

Definition 4.2 (accumulation for PC)**.** *A polynomial commitment scheme* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ *has an accumulation scheme if, for every* $D(\lambda) = \mathrm{poly}(\lambda)$*, the pair* $(\Phi_{\mathsf{PC}}, \mathcal{H}_{\mathsf{PC},D})$ *defined below has an accumulation scheme.*

```
$\Phi_{\mathsf{PC}}^{\rho}(\mathsf{i}_{\Phi} = (\mathsf{pp}_{\mathsf{PC}}, [d_i]_{i=1}^{n}), q = ((C, d, z, v), \pi))$:
1. $(\mathsf{ck}, \mathsf{rk}) \leftarrow \mathsf{PC.Trim}^{\rho}(\mathsf{pp}_{\mathsf{PC}}, [d_i]_{i=1}^{n})$.
2. Output $\mathsf{PC.Check}^{\rho}(\mathsf{rk}, C, d, z, v, \pi)$.
```

```
$\mathcal{H}_{\mathsf{PC},D}^{\rho}(1^{\lambda})$:
    Output $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{PC.Setup}^{\rho}(1^{\lambda}, D(\lambda))$.
```

5 Proof-carrying data from accumulation schemes

We formally restate and then prove Theorem 1, which provides a construction of proof-carrying data (PCD) from any SNARK that has an accumulation scheme with certain efficiency properties.

First, we provide some notation for these properties.

Definition 5.1. *Let* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *be an accumulation scheme for a non-interactive argument (see Section 6). We denote by* $\mathrm{V}^{(\lambda,m,N,k)}$ *the circuit corresponding to the computation of the accumulation verifier* V*, for security parameter* λ*, when checking the accumulation of* m *instance-proof pairs and accumulators, on an index of size at most* N*, where each instance is of size at most* k*.*

We denote by $\mathsf{v}(\lambda, m, N, k)$ *the size of the circuit* $\mathrm{V}^{(\lambda,m,N,k)}$*, by* $|\mathsf{avk}(\lambda, m, N)|$ *the size of the accumulator verification key* avk*, and by* $|\mathsf{acc}(\lambda, m, N)|$ *the size of an accumulator.*

Note that here we have specified that the size of acc is bounded by a function of λ, m, N; in particular, it *may not* depend on the number of instances accumulated.

When we invoke the accumulation verifier in our construction of PCD, an instance will consist of an accumulator verification key, an accumulator, and some additional data of size ℓ. Thus the size of the accumulation verifier circuit used in the scheme is given by

$$\mathsf{v}^*(\lambda, m, N, \ell) := \mathsf{v}(\lambda, m, N, |\mathsf{avk}(\lambda, m, N)| + |\mathsf{acc}(\lambda, m, N)| + \ell) \ .$$

The notion of "sublinear verification" which is important here is that v^* is sublinear in N. The following theorem shows that when this is the case, this accumulation scheme can be used to construct PCD.

Theorem 5.2. *There exists a polynomial-time transformation* T *such that if* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a preprocessing SNARK for circuit satisfiability and* AS *is an accumulation scheme for* ARG *then* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathrm{T}(\mathsf{ARG}, \mathsf{AS})$ *is a PCD scheme for constant-depth compliance predicates, provided*

$$\exists\, \epsilon \in (0, 1) \text{ and a polynomial } \alpha \text{ s.t. } \mathsf{v}^*(\lambda, m, N, \ell) = O(N^{1-\epsilon} \cdot \alpha(\lambda, m, \ell)) \ .$$

Moreover:

- *If* ARG *and* AS *are secure against quantum adversaries, then* PCD *is secure against quantum adversaries.*
- *If* ARG *and* AS *are (post-quantum) zero knowledge, then* PCD *is (post-quantum) zero knowledge.*
- *If the size of the predicate* $\varphi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ *is* $f = \omega(\alpha(\lambda, m, \ell)^{1/\epsilon})$ *then:*
  - *the cost of running* $\mathbb{I}$ *is equal to the cost of running both* $\mathcal{I}$ *and* I *on an index of size* $f + o(f)$*;*
  - *the cost of running* $\mathbb{P}$ *is equal to the cost of accumulating* m *instance-proof pairs using* P*, and running* $\mathcal{P}$*, on an index of size* $f + o(f)$ *and instance of size* $o(f)$*;*
  - *the cost of running* $\mathbb{V}$ *is equal to the cost of running both* $\mathcal{V}$ *and* D *on an index of size* $f + o(f)$ *and an instance of size* $o(f)$*.*

This last point gives the conditions for a *sublinear additive* recursive overhead; i.e., when the *additional* cost of proving that φ is satisfied recursively is asymptotically smaller than the cost of proving that φ is satisfied locally. Note that the smaller the compliance predicate φ, the more efficient the accumulation scheme has to be in order to achieve this.

Our PCD construction and its proof of security follow those given in [COS20], except for several important differences.

- In [COS20], the circuit on which the SNARK prover is invoked contains the SNARK verifier circuit. In our setting, this is not possible in general since the verifier may not be succinct. Instead, we invoke the SNARK prover on a circuit containing the *accumulation verifier circuit*.
- The PCD proof consists of both a SNARK proof π and an accumulator acc; verifying the computation requires running the SNARK verifier on π and the accumulation scheme decider on acc.
- Since the security of the accumulation scheme is proved separately to the security of the SNARK itself, we require that the SNARK remain secure with respect to the auxiliary input distribution induced by the public parameters of the accumulation scheme.

The rest of this section is dedicated to proving Theorem 5.2: in Section 5.1 we construct the PCD scheme; in Section 5.2 we prove that accumulation verifiers that are sublinear suffice for PCD; in Section 5.3 we prove completeness; in Section 5.4 we prove knowledge soundness; in Section 5.5 we discuss zero knowledge; and in Section 5.6 we discuss post-quantum security.

Remark 5.3. Theorem 5.2 yields PCD that is secure for constant-depth compliance predicates. The depth restriction is necessary because of the recursive invocation of the extractor: if the extractor has size $e(n) = n^c$, then recursively applying the extractor d times (for a depth-d predicate) yields a circuit of size n^{c^d}, which for $d = \omega(1)$ is superpolynomial. Under a subexponential knowledge assumption one can increase d to $o(\log n)$.

5.1 Construction

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a non-interactive argument for circuit satisfiability, and let $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ be an accumulation scheme for ARG. Below we construct a PCD scheme $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$.

Given a compliance predicate $\varphi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$, the circuit that realizes the recursion is as follows.

```
$R^{(\lambda,N,k)}_{\mathrm{V},\varphi}\big((\mathsf{avk}, z, \mathsf{acc}), (z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m, \pi_{\mathrm{V}})\big)$:
  1. Check that the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, \dots, z_m)$ accepts.
  2. If there exists $i \in [m]$ such that $z_i \neq \bot$, check that the SNARK accumulation verifier accepts:
       $\mathrm{V}^{(\lambda,m,N,k)}(\mathsf{avk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$ .
  3. If the above checks hold, output 1; otherwise, output 0.
```

Above, $\mathrm{V}^{(\lambda,m,N,k)}$ refers to the circuit representation of V with input size appropriate for security parameter λ, number of instance-proof pairs and accumulators m, index size N, and instance size k.

Next we describe the generator $\mathbb{G}$, indexer $\mathbb{I}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$ of the PCD scheme.

- $\mathbb{G}(1^\lambda)$: Sample $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$, and output $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$.
- $\mathbb{I}(\mathbb{pp}, \varphi)$:
  1. Compute the integer $N := N(\lambda, |\varphi|, m, \ell)$, where N is defined in Lemma 5.4 below.
  2. Construct the circuit $R := R^{(\lambda,N,k)}_{\mathrm{V},\varphi}$ where $k := |\mathsf{avk}(\lambda, N)| + \ell + |\mathsf{acc}(\lambda, N)|$.
  3. Compute the index key pair $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R)$ for the circuit R for the SNARK.
  4. Compute the index key triple $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{i}_\Phi = (\mathsf{pp}, R))$ for the accumulator.
  5. Output $\mathsf{ipk} := (\mathrm{ipk}, \mathsf{apk})$ and $\mathsf{ivk} := (\mathrm{ivk}, \mathsf{dk}, \mathsf{avk})$.
- $\mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, (\pi_i, \mathsf{acc}_i)]_{i=1}^m)$:
  1. If $z_i = \bot$ for all $i \in [m]$ then set $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}(\mathsf{apk}, \bot)$.
  2. If $z_i \neq \bot$ for some $i \in [m]$ then compute $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}(\mathsf{apk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
  3. Compute $\pi \leftarrow \mathcal{P}\big(\mathrm{ipk}, (\mathsf{avk}, z, \mathsf{acc}), (z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m, \pi_{\mathrm{V}})\big)$.

4. Output (π, acc).

- $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc}))$: Accept if both $\mathcal{V}(\mathsf{ivk}, (\mathsf{avk}, z, \mathsf{acc}), \pi)$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc})$ accept.

5.2 Efficiency

Denote by f the size of the compliance predicate φ as an R1CS instance. In the above construction, the explicit input consists of the accumulator verification key avk, whose size depends on N and λ, a message z whose size is ℓ (independent of N), and an accumulator acc whose size depends on N and λ. The security parameter λ is independent of N. The circuit $R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ on which we wish to invoke V is of size

$$S(\lambda, f, m, \ell, N) = f + S_0(m, \ell) + \mathsf{v}^*(\lambda, m, N, \ell) \quad \text{for some} \quad S_0(m, \ell) = O(m\ell) \ ,$$

recalling that $\mathsf{v}^*(\lambda, m, N, \ell) = \mathsf{v}(\lambda, m, N, |\mathsf{avk}(\lambda, m, N)| + |\mathsf{acc}(\lambda, m, N)| + \ell)$.

The goal of this section to find the (asymptotically) smallest index size bound function N such that $S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell)$. This ensures that the circuit for checking an index of size N is of size at most N, which permits recursion.

Lemma 5.4. *Suppose that for every security parameter $\lambda \in \mathbb{N}$, arity m, and message size $\ell \in \mathbb{N}$ the ratio of accumulation verifier circuit size to index size $\mathsf{v}^*(\lambda, m, N, \ell)/N$ is monotone decreasing in N. Then there exists a size function $N(\lambda, f, m, \ell)$ such that*

$$\forall\, \lambda, f, m, \ell \in \mathbb{N} \quad S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell) \ .$$

Moreover if for some $\epsilon > 0$ and some increasing function α it holds that, for all N, λ, m, ℓ sufficiently large,

$$\mathsf{v}^*(\lambda, m, N, \ell) \leq N^{1-\epsilon} \alpha(\lambda, m, \ell)$$

then, for all λ, m, ℓ sufficiently large,

$$N(\lambda, f, m, \ell) \leq O(f + \alpha(\lambda, m, \ell)^{1/\epsilon}) \ .$$

Proof. Let $N_0 := N_0(\lambda, m, \ell)$ be the smallest integer such that $\mathsf{v}^*(\lambda, m, N_0, \ell)/N_0 < 1/2$; this exists because of the monotone decreasing condition. Let $N(\lambda, f, m, \ell) := \max(N_0(\lambda, m, \ell), 2(f + S_0(m, \ell)))$. Then for $N := N(\lambda, f, m, \ell)$ it holds that

$$S(\lambda, f, m, \ell, N) = f + S_0(m, \ell) + N \cdot \mathsf{v}^*(\lambda, m, N, \ell)/N < N/2 + N/2 = N \ .$$

Clearly $f + S_0(m, \ell) = O(f)$. Now suppose that $\mathsf{v}^*(\lambda, m, N, \ell) \leq N^{1-\epsilon}\alpha(\lambda, m, \ell)$ for all sufficiently large N, λ, m, ℓ. Let $N'(\lambda, m, \ell) := (2 \cdot \alpha(\lambda, m, \ell))^{1/\epsilon}$. Then since α is increasing, for sufficiently large λ, m, ℓ, for $N' := N'(\lambda, m, \ell)$,

$$\mathsf{v}^*(\lambda, m, N', \ell)/N' < \alpha(\lambda, m, \ell) \cdot (2\alpha(\lambda, m, \ell))^{-1} = 1/2 \ .$$

Hence $N_0 \leq N' = (2 \cdot \alpha(\lambda, m, \ell))^{1/\epsilon}$, for sufficiently large λ, m, ℓ, and so $N(\lambda, f, m, \ell) \leq O(f + \alpha(\lambda, m, \ell)^{1/\epsilon})$. □

We can now bound the size of the recursive circuit.

Corollary 5.5. *For the function N above, $S(\lambda, f, m, \ell, N) = f + O(f^{1-\epsilon} \cdot \alpha(\lambda, m, \ell) + \alpha(\lambda, m, \ell)^{1/\epsilon})$.*

Proof. Using the expression for S above, and the bound on N,

$$\begin{aligned} S(\lambda, f, m, \ell, N) &= f + O(m\ell) + \mathsf{v}^*(\lambda, m, N, \ell) \\ &= f + O(N^{1-\epsilon}\alpha(\lambda, m, \ell)) \\ &= f + O(f^{1-\epsilon} \cdot \alpha(\lambda, m, \ell) + \alpha(\lambda, m, \ell)^{1/\epsilon}) \ . \end{aligned} \qquad \square$$

In particular if $f = \omega(\alpha(\lambda, m, \ell)^{1/\epsilon})$ then this is $f + o(f)$, and so the stated efficiency bounds hold.

5.3 Completeness

Let $\mathcal{A}$ be any adversary that causes the completeness condition of PCD to be satisfied with probability p. We construct an adversary $\mathcal{B}$, as follows, that causes the completeness condition of AS to be satisfied with probability at most p.

```
$\mathcal{B}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$:
1. Set $\mathbb{pp} := (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$ and compute $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathbb{pp})$.
2. Run $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)})$.
3. Output $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m)$.
```

Suppose that $\mathcal{A}$ outputs $(\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m)$ such that the completeness precondition is satisfied, but $\mathbb{V}(\mathsf{ivk}, z, (\pi, \mathsf{acc})) = 0$. Then, by construction of $\mathbb{V}$, it holds that either $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}), \pi) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$. If $z_i = \bot$ for all i, then by perfect completeness of ARG both of these algorithms output 1; hence there exists i such that $z_i \neq \bot$. Hence it holds that for all i, $\mathbb{V}(\mathsf{ivk}, z_i, (\pi_i, \mathsf{acc}_i)) = 1$, whence for all i, $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i) = \Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$.

If $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z, \mathsf{acc}), \pi) = 0$, then, by perfect completeness of ARG, we know that $R_{\mathrm{V},\varphi}^{(\lambda,N,k)}$ rejects $\big((\mathsf{avk}, z, \mathsf{acc}), (z_{\mathsf{loc}}, [z_i, \pi_i, \mathsf{acc}_i]_{i=1}^m), \pi_{\mathrm{V}}\big)$, and so $\mathrm{V}(\mathsf{avk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m, \mathsf{acc}) = 0$. Otherwise, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$.

Now consider the completeness experiment for AS with adversary $\mathcal{B}$. Since $\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}$ are drawn identically to the PCD experiment, the distribution of the output of $\mathcal{A}$ is identical. Hence in particular it holds that for all i, $\Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i) = 1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_i) = 1$. By the above, it holds that either $\mathrm{V}(\mathsf{avk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m, \mathsf{acc}) = 0$ or $\mathrm{D}(\mathsf{dk}, \mathsf{acc}) = 0$, and so $\mathcal{B}_1, \mathcal{B}_2$ cause the completeness condition for AS to be satisfied with probability at most p.

5.4 Knowledge soundness

Since the extracted transcript T will be a tree, we find it convenient to associate the label $z^{(u,v)}$ of the unique outgoing edge of a node u with the node u itself, so that the node u is labelled with $(z^{(u)}, z_{\mathsf{loc}}^{(u)})$. For the purposes of the proof we also associate with each node u a SNARK proof $\pi^{(u)}$ and an accumulator $\mathsf{acc}^{(u)}$, so that the full label for a node is $(z^{(u)}, z_{\mathsf{loc}}^{(u)}, \pi^{(u)}, \mathsf{acc}^{(u)})$. It is straightforward to transform such a transcript into one that satisfies Definition 3.1.

Given a malicious prover $\tilde{\mathbb{P}}$, we will define an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ that satisfies knowledge soundness. In the process we construct a sequence of extractors $\mathbb{E}_1, \ldots, \mathbb{E}_d$ for $d := d(\varphi)$ (the depth of φ); $\mathbb{E}_j$ outputs a tree of depth $j + 1$. Let $\mathbb{E}_0(\mathbb{pp})$ run $(\varphi, \mathsf{o}, \pi, \mathsf{acc}) \leftarrow \tilde{\mathbb{P}}(\mathbb{pp})$ and output (φ, T_0), where T_0 is a single node labeled with $(\mathsf{o}, \pi, \mathsf{acc})$. Let $l_{\mathsf{T}}(j)$ denote the vertices of T at depth j; $l_{\mathsf{T}}(0) := \emptyset$ and $l_{\mathsf{T}}(1)$ is the singleton containing the root.

Now we define the extractor $\mathbb{E}_j$ inductively for each $j \in [d]$. Suppose we have already constructed $\mathbb{E}_{j-1}$. We construct a SNARK prover $\tilde{\mathcal{P}}_j$ as follows:

$\tilde{\mathcal{P}}_j(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$:

1. Compute $(\varphi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$.
2. For each vertex $v \in l_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$.
3. Run the argument indexer $(\mathrm{ipk}, \mathrm{ivk}) \leftarrow \mathcal{I}(\mathsf{pp}, R^{(\lambda,N,k)}_{\mathrm{V},\varphi})$. Run the accumulator indexer $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R^{(\lambda,N,k)}_{\mathrm{V},\varphi})$.
4. Output

$$(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) := \Big(\vec{R}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\varphi, \mathsf{T}_{j-1})\Big)$$

where $\vec{R}$ is the vector $(R^{(\lambda,N,k)}_{\mathrm{V},\varphi}, \ldots, R^{(\lambda,N,k)}_{\mathrm{V},\varphi})$ of the appropriate length.

Next let $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor that corresponds to $\tilde{\mathcal{P}}_j$, via the knowledge soundness of the non-interactive argument ARG. Finally the extractor $\mathbb{E}_j$ is defined as follows:

$\mathbb{E}_j(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}))$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}})$.
2. Parse the auxiliary output aux as (φ, T').
3. If T' is not a transcript of depth j, abort.
4. Output (φ, T_j) where T_j is the transcript constructed from T' by doing the following for each vertex $v \in l_{\mathsf{T}'}(j)$:
   - obtain the local data $z^{(v)}_{\mathsf{loc}}$ and input messages $\big(z_i, \pi_i, \mathsf{acc}_i\big)_{i \in [m]}$ from $\mathbb{w}^{(v)}$;
   - append $z^{(v)}_{\mathsf{loc}}$ to the label of v, and if there exists any z_i with $z_i \neq \bot$, attach m children to v where the i-th child is labeled with $(z_i, \pi_i, \mathsf{acc}_i)$.

The extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ runs $(\varphi, \mathsf{T}_d) \leftarrow \mathbb{E}_d(\mathsf{urs})$ and outputs $(\varphi, (\pi, \mathsf{acc}), \mathsf{T}_d)$, where $(z, z_{\mathsf{loc}}, \pi, \mathsf{acc})$ labels the root node.

We now show that $\mathbb{E}_{\tilde{\mathbb{P}}}$ has polynomial size and that it outputs a transcript that is φ-compliant.

Size of the extractor. $\tilde{\mathcal{P}}_j$ is a circuit of size $|\mathbb{E}_{j-1}| + |\mathrm{I}| + O(2^j)$, so $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ is a circuit of size $e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + O(2^j))$. Then $|\mathbb{E}_j| \leq e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + c \cdot 2^j)$ for some $c \in \mathbb{N}$.

A solution to this recurrence (for $e(n) \geq n$) is $|\mathbb{E}_j| \leq e^{(j)}(|\tilde{\mathbb{P}}| + j \cdot |\mathcal{I}| + 2c \cdot 2^j)$, where $e^{(j)}$ is the function e iterated j times. Hence in particular if $d(\varphi)$ is a constant, $\mathbb{E}_{\tilde{\mathbb{P}}}$ is a circuit of polynomial size.

Correctness of the extractor. Suppose that $\tilde{\mathbb{P}}$ causes $\mathbb{V}$ to accept with probability μ. We show by induction that, for all $j \in \{0, \ldots, d\}$, the transcript T_j output by $\mathbb{E}_j$ is φ-compliant up to depth j, and that for all $v \in \mathsf{T}$, both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept, with probability $\mu - \mathrm{negl}(\lambda)$.

For $j = 0$ the statement is implied by $\mathbb{V}$ accepting with probability μ.

Now suppose that $\mathsf{T}_{j-1} \leftarrow \mathbb{E}_{j-1}$ is φ-compliant up to depth $j-1$, and that both $\mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}), \pi^{(v)})$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accept for all $v \in \mathsf{T}_{j-1}$, with probability $\mu - \mathrm{negl}(\lambda)$.

Let $(\vec{\mathbb{i}}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)})_v, (\pi^{(v)})_v, (\varphi, \mathsf{T}'), \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}$.

By knowledge soundness of ARG, with probability $\mu - \mathrm{negl}(\lambda)$:

- for $v \in l_{\mathsf{T}'}(j)$, v is labeled with $(z^{(v)}, \pi^{(v)}, \mathsf{acc}^{(v)})$ and $\mathsf{avk}_v = \mathsf{avk}$ where $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, \Phi)$,
- for every vertex $v \in l_{\mathsf{T}'}(j)$, $(R^{(\lambda,N,k)}_{\mathrm{V},\varphi}, (\mathsf{avk}, z^{(v)}, \mathsf{acc}^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, and

- by induction T' is φ-compliant up to depth $j-1$ and $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ accepts for all $v \in \mathsf{T}'$.

Here we use the auxiliary output in the knowledge soundness definition of ARG to ensure consistency between the values $z^{(v)}$ and T', and to ensure that T' is φ-compliant.

Consider some $v \in l_{\mathsf{T}'}(j)$. Since $(R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}_v, z^{(v)}, \mathsf{acc}^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, we obtain from $\mathbb{w}^{(v)}$ either

- local data z_{loc}, input messages $\left(z_i, \pi_i, \mathsf{acc}_i\right)_{i\in[m]}$ and proof π_{V} such that $\varphi(z^{(v)}, z_{\mathsf{loc}}, z_1, \ldots, z_m)$ accepts and the accumulation verifier $\mathrm{V}^{(\lambda,N,k)}(\mathsf{avk}, [(\mathsf{avk}, z_i, \mathsf{acc}_i), \pi_i]_{i=1}^m, [\mathsf{acc}_i]_{i=1}^m, \mathsf{acc}^{(v)}, \pi_{\mathrm{V}})$ accepts; or
- local data z_{loc} such that $\varphi(z^{(v)}, z_{\mathsf{loc}}, \bot, \ldots, \bot)$ accepts.

In both cases we append $z_{\mathsf{loc}}^{(v)} := z_{\mathsf{loc}}$ to the label of v. In the latter case, v has no children and so is φ-compliant by the base case condition. In the former case we label the children of v with $(z_i, \pi_i, \mathsf{acc}_i)$, and so v is φ-compliant. Moreover, by the soundness of the accumulation scheme, since $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(v)})$ and the accumulation verifier accept, it holds that for all descendants w, $\mathrm{D}(\mathsf{dk}, \mathsf{acc}^{(w)})$ accepts and $\Phi_{\mathcal{V}}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z_{\mathsf{in}}^{(w)}, \mathsf{acc}^{(w)}), \pi_{\mathsf{in}}^{(w)}) = \mathcal{V}(\mathrm{ivk}, (\mathsf{avk}, z_{\mathsf{in}}^{(w)}, \mathsf{acc}^{(w)}), \pi_{\mathsf{in}}^{(w)})$ accepts.

Hence by induction, $(\varphi, \pi, \mathsf{T}) \leftarrow \mathbb{E}$ has φ-compliant T.

Since $(\varphi, \mathsf{o}(\mathsf{T}), \pi)$ are "passed up" from $\tilde{\mathbb{P}}$ via a series of d extractors, the distribution output by $\mathbb{E}$ is statistically close to the output of $\tilde{\mathbb{P}}$ by the knowledge soundness of ARG.

5.5 Zero knowledge

The simulator $\mathbb{S}$ operates as follows.

$\mathbb{S}(1^\lambda)$:
1. Generate $(\mathsf{pp}_{\mathsf{AS}}, \tau_{\mathsf{AS}}) \leftarrow \mathrm{S}(1^\lambda)$, and $(\mathsf{pp}, \tau) \leftarrow \mathcal{S}(1^\lambda)$.
2. Output $(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), (\tau, \tau_{\mathsf{AS}}))$.

$\mathbb{S}(\mathbb{pp} = (\mathsf{pp}, \mathsf{pp}_{\mathsf{AS}}), \varphi, z, (\tau, \tau_{\mathsf{AS}}))$:
1. Compute $\mathsf{acc} \leftarrow \mathrm{S}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, \tau_{\mathsf{AS}})$.
2. Compute $(\mathsf{apk}, \mathsf{dk}, \mathsf{avk}) \leftarrow \mathrm{I}(\mathsf{pp}_{\mathsf{AS}}, (\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}))$.
3. Compute $\pi \leftarrow \mathcal{S}(\mathsf{pp}, R_{\mathrm{V},\varphi}^{(\lambda,N,k)}, (\mathsf{avk}, z, \mathsf{acc}), \tau)$.
4. Output (π, acc).

We consider the following sequence of hybrids.

- $\mathbf{H}_0$: The original experiment.
- $\mathbf{H}_1$: As $\mathbf{H}_0$, but the public parameters pp and proof π are generated by the simulator $\mathcal{S}$ for ARG.
- $\mathbf{H}_2$: As $\mathbf{H}_1$, but the public parameters $\mathsf{pp}_{\mathsf{AS}}$ and accumulator acc is generated by the simulator S for AS.

Observe that if $\mathbf{H}_0$ and $\mathbf{H}_2$ are indistinguishable, then $\mathbb{S}$ witnesses the zero knowledge property for PCD.

Since $\mathcal{A}$ is honest (for PCD), by completeness of AS it induces an honest adversary for ARG, whence $\mathbf{H}_0$ and $\mathbf{H}_1$ are indistinguishable by the zero knowledge property of ARG. Note that since they are part of the witness, the input and accumulator lists $[\mathsf{q}_i]_{i=1}^n$, $[\mathsf{acc}_j]_{j=1}^m$ and verifier proof π_{V} are not used in $\mathbf{H}_1$. Hence, since $\mathcal{A}$ induces an honest adversary for AS, $\mathbf{H}_1, \mathbf{H}_2$ are indistinguishable by the zero knowledge property of AS. This establishes that $\mathbf{H}_0, \mathbf{H}_2$ are indistinguishable.

5.6 Post-quantum security

We consider post-quantum knowledge soundness and zero knowledge.

Knowledge soundness. In the quantum setting, $\tilde{\mathbb{P}}$ is taken to be a polynomial-size *quantum* circuit; hence also $\tilde{\mathcal{P}}_j, \mathcal{E}_{\tilde{\mathcal{P}}_j}, \mathbb{E}_j$ are quantum circuits for all j, as is the final extractor $\mathbb{E}$. Our definition of knowledge soundness is such that this proof then generalizes immediately to show security against quantum adversaries. In particular, the only difficulty arising from quantum adversaries is that they can generate their own randomness, whereas in the classical case we can force an adversary to behave deterministically by fixing its randomness. This is accounted for by the distributional requirement placed on the extractor of the argument system ARG.

Zero knowledge. From the argument in the preceding section it is clear that, by modifying the definitions of zero knowledge as appropriate for the quantum setting, if ARG and AS both achieve post-quantum zero knowledge, then so does PCD.

6 Accumulation schemes for non-interactive arguments

We formally restate and then prove Theorem 2, which provides a way to "lift" an accumulation scheme for a predicate into an accumulation scheme for any non-interactive argument whose verifier is succinct when given oracle access to that predicate. Below we define the notion of a predicate-efficient non-interactive argument and then state the theorem that we prove.

Definition 6.1. *Let* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *be a non-interactive argument,* $\Phi_\circ \colon \mathcal{U}(*) \times (\{0,1\}^*)^3 \to \{0,1\}$ *a predicate, and* $T, \mathsf{t} \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$*. We say that* ARG *is* (T, t)***-predicate-efficient*** *with respect to* $\Phi_\circ$ *if the following conditions hold.*

- *The index verification key* ivk *has the form* $(\mathsf{i}_\circ, \mathrm{ivk}_{\mathsf{pe}})$.
- *The verifier* $\mathcal{V}$ *is equivalent to the following decision procedure, for some oracle algorithm* $\mathcal{V}_{\mathsf{pe}}$*: compute* $(b, Q_i) \leftarrow \mathcal{V}_{\mathsf{pe}}^{\rho}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}, \pi)$*; output* 1 *if and only if (a)* $b = 1$*, and (b) for each* $\mathsf{q} \in Q_i$*,* $\Phi_\circ^{\rho}(\mathsf{pp}, \mathsf{i}_\circ, \mathsf{q}_i) = 1$.[4]
- $\mathcal{V}_{\mathsf{pe}}$ *runs in time* $T(N, k)$*, and the number of queries* t *to* $\Phi_\circ$ *equals* $\mathsf{t}(N, k)$ *for indices* $\mathbb{i}$ *of size* N *and instances* $\mathbb{x}$ *of size* k.

The following theorem shows that an accumulation scheme for $(\Phi_\circ, \mathcal{G})$ implies an accumulation scheme for $(\Phi_{\mathcal{V}}, \mathcal{G})$, where $\Phi_{\mathcal{V}}$ is as specified in Definition 4.1.

Theorem 6.2. *Let* ARG *be a non-interactive argument that is* (T, t)*-predicate-efficient with respect to* $\Phi_\circ$*. If* $(\Phi_\circ, \mathcal{G})$ *has an accumulation scheme* $\mathsf{AS}_\circ$*, then* ARG *has an accumulation scheme* $\mathsf{AS}_{\mathsf{ARG}}$ *with the efficiency properties below.*

- Generator: $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{G}^{\rho}(1^{\lambda})$ *takes time equal to* $\mathsf{AS}_\circ.\mathrm{G}^{\rho}(1^{\lambda})$.
- Indexer: $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i})$ *takes time equal to the time to run* $\mathsf{AS}_\circ.\mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{i}_\circ)$.
- Accumulation prover: $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{P}^{\rho}(\mathsf{apk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m})$ *runs in time* $\sum_{i=1}^{n} T(N, |\mathbb{x}_i|)$ *plus the time taken to run* $\mathsf{AS}_\circ.\mathrm{P}^{\rho}(\mathsf{apk}_\circ, Q, [\mathsf{acc}_j]_{j=1}^{m})$*, where* $|Q| = \sum_{i=1}^{n} \mathsf{t}(N, |\mathbb{x}_i|)$.
- Accumulation verifier: $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}^{\rho}(\mathsf{avk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$ *runs in time* $\sum_{i=1}^{n} T(N, |\mathbb{x}_i|)$ *plus the time taken to run* $\mathsf{AS}_\circ.\mathrm{V}^{\rho}(\mathsf{avk}_\circ, Q := \cup_{i=1}^{n} Q_i, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$.
- Decider: $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc})$ *takes time equal to* $\mathsf{AS}_\circ.\mathrm{D}^{\rho}(\mathsf{dk}_\circ, \mathsf{acc})$.

Moreover, if $\mathsf{AS}_\circ$ *is post-quantum secure, then so is* $\mathsf{AS}_{\mathsf{ARG}}$*, and if* $\mathsf{AS}_\circ$ *is zero knowledge, so is* $\mathsf{AS}_{\mathsf{ARG}}$.

The rest of this section is dedicated to proving Theorem 6.2: in Section 6.1 we describe the construction of an accumulation scheme for the argument scheme, from which the stated efficiency properties are clear; in Section 6.2 we argue completeness; in Section 6.3 we argue soundness; in Section 6.4 we argue zero knowledge.

Remark 6.3 (efficiency for PCD)**.** We briefly discuss the properties required of ARG and $\mathsf{AS}_\circ$ so that, applying Theorem 6.2, we obtain an accumulation scheme suitable for the PCD construction in Section 5. Recall that Theorem 5.2 states that a PCD scheme can be obtained from any SNARK for circuit satisfiability with an accumulation scheme whose accumulation verifier runs in time that is sub-linear in the size N of the circuit. Hence ARG must be (T, t)-predicate-efficient for $T, \mathsf{t} = o(N)$, and $\mathsf{AS}_\circ.\mathrm{V}$ must run in time $o(N)$. In particular, since it is an input to $\mathsf{AS}_\circ.\mathrm{V}$, the size of an accumulator acc for $\mathsf{AS}_\circ$ must be $o(N)$. The number of inputs n and accumulators m are both equal to the arity of the compliance predicate and may be regarded as constant. There is no restriction on the running time of $\mathsf{AS}_\circ.\mathrm{D}$.

Remark 6.4. Definition 6.1 (and hence Theorem 6.2) restricts $\mathcal{V}$'s access to $\Phi_\circ$: $\mathcal{V}$ must reject if any of its queries to $\Phi_\circ$ are answered with 0. In particular, the queries of $\mathcal{V}$ to $\Phi_\circ$ are non-adaptive. This is necessary so that the accumulator does not grow with the number of accumulated queries.

[4]Here we assume for simplicity that the public parameters for $\Phi_\circ$ are the same as those for $\mathcal{V}$.

6.1 Construction

We construct the accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for $(\Phi_{\mathcal{V}}, \mathcal{H}_{\mathsf{ARG}})$ (specified in Definition 4.1). The efficiency properties stated in Theorem 6.2 follow from the construction below.

Data structures. An accumulator acc for $\mathsf{AS}_{\mathsf{ARG}}$ is the accumulator for the underlying accumulation scheme $\mathsf{AS}_{\circ}$.

Generator. On input the security parameter λ (in unary), $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{G}$ samples public parameters $\mathsf{pp} \leftarrow \mathsf{AS}_{\circ}.\mathrm{G}^{\rho}(1^{\lambda})$ for the accumulation scheme for $(\Phi_{\circ}, \mathcal{H}_{\circ})$, and outputs pp.

Indexer. On input public parameters $\mathsf{pp}_{\mathsf{AS}}$, succinct argument parameters pp and index $\mathbb{i}$, $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{I}$ computes index keys $(\mathrm{ipk}, \mathrm{ivk} = (\mathsf{i}_{\circ}, \mathrm{ivk}_{\mathsf{pe}})) \leftarrow \mathcal{I}(\mathsf{pp}, \mathbb{i})$ and accumulation keys $(\mathsf{apk}_{\circ}, \mathsf{avk}_{\circ}, \mathsf{dk}_{\circ}) \leftarrow \mathsf{AS}_{\circ}.\mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{i}_{\circ})$, and outputs the accumulation keys $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \big((\mathsf{apk}_{\circ}, \mathrm{ivk}_{\mathsf{pe}}), (\mathsf{avk}_{\circ}, \mathrm{ivk}_{\mathsf{pe}}), \mathsf{dk}_{\circ}\big)$.

Accumulation prover. On input the accumulator proving key $\mathsf{apk} = (\mathsf{apk}_{\circ}, \mathrm{ivk}_{\mathsf{pe}})$, instance-proof pairs $[(\mathbb{x}_i, \pi_i)]_{i=1}^{n}$, and old accumulators $[\mathsf{acc}_j]_{j=1}^{m}$, $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{P}$ computes a new accumulator acc and proof π_{V} as follows: for each $i \in [n]$, compute $(b_i, Q_i) \leftarrow \mathcal{V}^{\rho}_{\mathsf{pe}}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}_i, \pi_i)$; then output the new accumulator and proof $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathsf{AS}_{\circ}.\mathrm{P}^{\rho}(\mathsf{apk}_{\circ}, \cup_{i=1}^{n} Q_i, [\mathsf{acc}_j]_{j=1}^{m})$.

Accumulation verifier. On input the accumulation verification key $\mathsf{avk} = (\mathsf{avk}_{\circ}, \mathrm{ivk}_{\mathsf{pe}})$, instance-proof pairs $[(\mathbb{x}_i, \pi_i)]_{i=1}^{n}$, old accumulators $[\mathsf{acc}_j]_{j=1}^{m}$, the new accumulator acc, and a proof π_{V}, $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ works as follows. For each $i \in [n]$, compute $(b_i, Q_i) \leftarrow \mathcal{V}^{\rho}_{\mathsf{pe}}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}_i, \pi_i)$, and check that $b_i = 1$, and then accept if the new accumulator is valid, i.e., $\mathsf{AS}_{\circ}.\mathrm{V}^{\rho}(\mathsf{avk}_{\circ}, Q := \cup_{i=1}^{n} Q_i, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$.

Decider. On input decision key $\mathsf{dk} = \mathsf{dk}_{\circ}$ and accumulator acc, $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}$ accepts if $\mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc}) = 1$.

6.2 Completeness

Let $\mathcal{A}$ be a (possibly inefficient) adversary that breaks the completeness of the accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for $(\Phi_{\mathcal{V}}, \mathcal{G})$ in Section 6.1. That is, the following probability is non-zero:

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \forall j \in [m],\ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}_j) = 1 \\ \forall i \in [n],\ \Phi^{\rho}_{\mathcal{V}}(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_i, \pi_i)) = 1 \\ \not\Downarrow \\ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}^{\rho}\begin{pmatrix} \mathsf{avk} & [(\mathbb{x}_i, \pi_i)]_{i=1}^{n} & [\mathsf{acc}_j]_{j=1}^{m} \\ & \mathsf{acc} \quad \pi_{\mathrm{V}} & \end{pmatrix} = 1 \\ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathsf{AS}_{\mathsf{ARG}}.\mathrm{G}^{\rho}(1^{\lambda}) \\ \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ (\mathbb{i}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}) \\ (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathsf{AS}_{\mathsf{ARG}}.\mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \\ (\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathsf{AS}_{\mathsf{ARG}}.\mathrm{P}^{\rho}\begin{pmatrix} \mathsf{apk} & [(\mathbb{x}_i, \pi_i)]_{i=1}^{n} \\ & [\mathsf{acc}_j]_{j=1}^{m} \end{pmatrix} \end{array} \end{array}\right]. \quad (3)$$

We will use $\mathcal{A}$ to construct adversary $\mathcal{B}$ that breaks the completeness of the accumulation scheme $\mathsf{AS}_{\circ}$ for $(\Phi_{\circ}, \mathcal{G})$. That is, $\mathcal{B}$ makes the following probability nonzero:

$$\Pr\left[\begin{array}{c|r} \begin{array}{c} \forall j \in [m],\ \mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{pp}_{\circ}, \mathsf{dk}_{\circ}, \mathsf{acc}_j) = 1 \\ \forall i \in [n],\ \Phi^{\rho}_{\circ}(\mathsf{pp}, \mathsf{i}_{\circ}, \mathsf{q}_i) = 1 \\ \not\Downarrow \\ \mathsf{AS}_{\circ}.\mathrm{V}^{\rho}\begin{pmatrix} \mathsf{avk}_{\circ} & [\mathsf{q}_i]_{i=1}^{n} & [\mathsf{acc}_j]_{j=1}^{m} \\ & \mathsf{acc} \quad \pi_{\mathrm{V}} & \end{pmatrix} = 1 \\ \mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc}) = 1 \end{array} & \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{pp}_{\circ} \leftarrow \mathsf{AS}_{\circ}.\mathrm{G}^{\rho}(1^{\lambda}) \\ \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ (\mathsf{i}_{\circ}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \leftarrow \mathcal{B}^{\rho}(\mathsf{pp}_{\circ}, \mathsf{pp}) \\ (\mathsf{apk}_{\circ}, \mathsf{avk}_{\circ}, \mathsf{dk}_{\circ}) \leftarrow \mathsf{AS}_{\circ}.\mathrm{I}^{\rho}(\mathsf{pp}_{\circ}, \mathsf{pp}, \mathsf{i}_{\circ}) \\ (\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathsf{AS}_{\circ}.\mathrm{P}^{\rho}\begin{pmatrix} \mathsf{apk}_{\circ} & [\mathsf{q}_i]_{i=1}^{n} \\ & [\mathsf{acc}_j]_{j=1}^{m} \end{pmatrix} \end{array} \end{array}\right]. \quad (4)$$

We define the adversary $\mathcal{B}$ to operate as follows.

```
$\mathcal{B}^{\rho}(\mathsf{pp}_{\circ}, \mathsf{pp})$:
1. Compute $(\mathbb{i}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}_{\mathsf{AS}} = \mathsf{pp}_{\circ}, \mathsf{pp})$.
2. Compute $(\mathrm{ipk}, \mathrm{ivk} = (\mathsf{i}_{\circ}, \mathrm{ivk}_{\mathsf{pe}})) \leftarrow \mathcal{I}^{\rho}(\mathsf{pp}, \mathbb{i})$.
3. For each $i \in [n]$, compute $(b_i, Q_i) \leftarrow \mathcal{V}_{\mathsf{pe}}^{\rho}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}_i, \pi_i)$.
4. Set $Q := \cup_{i=1}^{n} Q_i$ and output $(\mathsf{i}_{\circ}, Q, [\mathsf{acc}_j]_{j=1}^{m})$.
```

By construction, the distribution of ρ, $\mathsf{pp}_{\mathsf{AS}}$ and pp are identical in both experiments, and hence so is the output of $\mathcal{A}$. It remains to show that for every fixed choice of these variables, the implication in Eq. (4) does not hold. We first use Eq. (3) to argue that the premises of the implication in Eq. (4) are satisfied:

- For each $j \in [m]$, we know that $\mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc}_j) = 1$ because $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}_j) = 1$, and so the corresponding condition in Eq. (4) is satisfied.
- For each $i \in [n]$, $\Phi_{\mathcal{V}}^{\rho}\big(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_i, \pi_i)\big) = 1$. This means that $\mathcal{V}^{\rho}(\mathrm{ivk}, \mathbb{x}_i, \pi_i) = 1$, which in turn implies that $\mathcal{V}_{\mathsf{pe}}^{\rho}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}_i, \pi_i) = (1, Q_i)$ and, for each query $\mathsf{q} \in Q_i$, the predicate $\Phi_{\circ}^{\rho}(\mathsf{pp}, \mathsf{i}_{\circ}, \mathsf{q}) = 1$. Hence, for each $\mathsf{q} \in Q = \cup_{i=1}^{n} Q_i$, $\Phi_{\circ}^{\rho}(\mathsf{pp}, \mathsf{i}_{\circ}, \mathsf{q}) = 1$.

We are now left to show that at least one of $\mathsf{AS}_{\circ}.\mathrm{V}^{\rho}(\mathsf{avk}_{\circ}, \cup_{i=1}^{n} Q_i, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$ or $\mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc})$ rejects. We do this by considering each case.

- By construction, $\mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc})$ rejects if and only if $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc})$ rejects.
- By construction, $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}^{\rho}(\mathsf{avk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$ rejects if at least one of the following conditions is satisfied:
  - For some $i \in [n]$, $\mathcal{V}_{\mathsf{pe}}$ rejects: $\mathcal{V}_{\mathsf{pe}}^{\rho}(\mathrm{ivk}_{\mathsf{pe}}, \mathbb{x}_i, \pi_i) = (0, Q_i)$.
  - $\mathsf{AS}_{\circ}.\mathrm{V}$ rejects: $\mathsf{AS}_{\circ}.\mathrm{V}^{\rho}(\mathsf{avk}_{\circ}, Q = \cup_{i=1}^{n} Q_i, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}}) = 0$.

  However, because $\Phi_{\mathcal{V}}^{\rho}\big(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_i, \pi_i)\big) = 1$, we know that $\mathcal{V}_{\mathsf{pe}}$ accepts. Hence, if $\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}$ rejects, it must be because $\mathsf{AS}_{\circ}.\mathrm{V}$ also rejects.

Together, these cases imply that:

$$\begin{gathered}\mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}^{\rho}(\mathsf{avk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}}) = 0 \ \vee \ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}) = 0 \\ \Downarrow \\ \mathsf{AS}_{\circ}.\mathrm{V}^{\rho}(\mathsf{avk}_{\circ}, Q, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}}) = 0 \ \vee \ \mathsf{AS}_{\circ}.\mathrm{D}^{\rho}(\mathsf{dk}_{\circ}, \mathsf{acc}) = 0 \ .\end{gathered}$$

Thus if $\mathcal{A}$ breaks completeness of the accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for $(\Phi_{\mathcal{V}}, \mathcal{H}_{\mathsf{ARG}})$ then $\mathcal{B}$ breaks completeness of the accumulation scheme $\mathsf{AS}_{\circ}$ for $(\Phi_{\circ}, \mathcal{H}_{\circ})$ (a contradiction).

6.3 Soundness

Let $\mathcal{A}$ be an efficient adversary that breaks the soundness of the accumulation scheme $\mathsf{AS}_{\mathsf{ARG}}$ for $(\Phi_{\mathcal{V}}, \mathcal{G})$. This means that the following probability is non-negligible:

$$\Pr\left[\begin{array}{c|r} \mathsf{AS}_{\mathsf{ARG}}.\mathrm{V}^{\rho}\begin{pmatrix}\mathsf{avk} & [(\mathbb{x}_i, \pi_i)]_{i=1}^{n} & [\mathsf{acc}_j]_{j=1}^{m} \\ & \mathsf{acc} \quad \pi_{\mathrm{V}} & \end{pmatrix} = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp}_{\mathsf{AS}} \leftarrow \mathsf{AS}_{\mathsf{ARG}}.\mathrm{G}^{\rho}(1^{\lambda}) \\ \not\Downarrow & \mathsf{pp} \leftarrow \mathcal{G}^{\rho}(1^{\lambda}) \\ \forall j \in [m],\ \mathsf{AS}_{\mathsf{ARG}}.\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}_j) = 1 & \begin{pmatrix}\mathbb{i} & [(\mathbb{x}_i, \pi_i)]_{i=1}^{n} & [\mathsf{acc}_j]_{j=1}^{m} \\ & \mathsf{acc} \quad \pi_{\mathrm{V}} & \end{pmatrix} \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}) \\ \forall i \in [n],\ \Phi_{\mathcal{V}}^{\rho}\big(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_i, \pi_i)\big) = 1 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathsf{AS}_{\mathsf{ARG}}.\mathrm{I}^{\rho}(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}, \mathbb{i}) \end{array}\right] . \quad (5)$$

We will use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the soundness of the accumulation scheme $\mathsf{AS}_\circ$ for $(\Phi_\circ, \mathcal{G})$. That is, $\mathcal{B}$ makes the following probability non-negligible:

$$\Pr\left[\begin{array}{c|r} \mathsf{AS}_\circ.\mathrm{V}^\rho(\mathsf{avk}_\circ, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V}) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}) = 1 & \mathsf{pp}_\circ \leftarrow \mathsf{AS}_\circ.\mathrm{G}^\rho(1^\lambda) \\ \not\Downarrow & \mathsf{pp} \leftarrow \mathcal{G}^\rho(1^\lambda) \\ \forall\, j \in [m],\ \mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}_j) = 1 & \begin{pmatrix} \mathsf{i}_\circ & [\mathsf{q}_i]_{i=1}^n & [\mathsf{acc}_j]_{j=1}^m \\ & \mathsf{acc} & \pi_\mathrm{V} \end{pmatrix} \leftarrow \mathcal{B}^\rho(\mathsf{pp}_\circ, \mathsf{pp}) \\ \forall\, i \in [n],\ \Phi^\rho(\mathsf{pp}, \mathsf{i}_\circ, \mathsf{q}_i) = 1 & (\mathsf{apk}_\circ, \mathsf{avk}_\circ, \mathsf{dk}_\circ) \leftarrow \mathsf{AS}_\circ.\mathrm{I}^\rho(\mathsf{pp}_\circ, \mathsf{i}_\circ) \end{array}\right] . \quad (6)$$

We define the adversary $\mathcal{B}$ to operate as follows.

```
$\mathcal{B}^\rho(\mathsf{pp}_\circ, \mathsf{pp})$:
1. Compute $(\mathbb{i}, [(\mathbb{x}_i, \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V}) \leftarrow \mathcal{A}^\rho(\mathsf{pp}_\mathsf{AS} = \mathsf{pp}_\circ, \mathsf{pp})$.
2. For each $i \in [n]$, compute $(b_i, Q_i) \leftarrow \mathcal{V}_\mathsf{pe}^\rho(\mathrm{ivk}_\mathsf{pe}, \mathbb{x}_i, \pi_i)$.
3. Set $Q := \cup_{i=1}^n Q_i$ and output $(Q, [\mathsf{acc}_j]_{j=1}^m, \mathsf{i}_\circ, \mathsf{acc}, \pi_\mathrm{V})$.
```

By construction, the distribution of ρ, pp_AS, and pp are identical in both experiments, and hence so is the output of $\mathcal{A}$. It remains to show that for every fixed choice of these variables, the implication in Eq. (6) does not hold. We start by using Eq. (5) to argue that the premises of the implication in Eq. (6) are satisfied:

- By construction, $\mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}) = \mathsf{AS}_\mathsf{ARG}.\mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc})$.
- By construction of V, we know that if $\mathsf{AS}_\mathsf{ARG}.\mathrm{V}^\rho(\mathsf{avk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V})$ accepts, then $\mathsf{AS}_\circ.\mathrm{V}^\rho(\mathsf{apk}_\circ, Q := \cup_{i=1}^n Q_i, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V})$ also accepts.

We are now left to show that at least one of the following occurs: (a) for some $j \in [m]$, $\mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}_j)$ rejects, or (b) for some $\mathsf{q} \in Q$, $\Phi_\circ^\rho(\mathsf{pp}, \mathsf{i}_\circ, \mathsf{q}) = 0$.

- By construction, $\mathsf{AS}_\mathsf{ARG}.\mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}_j) = 0$ for some $j \in [m]$ implies that $\mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}_j) = 0$.
- We know that $\mathsf{AS}_\mathsf{ARG}.\mathrm{V}^\rho(\mathsf{avk}, [(\mathbb{x}_i, \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}, \pi_\mathrm{V}) = 1$. Hence, for each $i \in [n]$, $\mathcal{V}_\mathsf{pe}^\rho(\mathrm{ivk}_\mathsf{pe}, \mathbb{x}_i, \pi_i)$ accepts and outputs a query set Q_i. This in turn implies that if there does exist $k \in [n]$ such that $\Phi_\mathcal{V}^\rho(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_k, \pi_k))$ rejects, then there exists $\mathsf{q} \in Q_k \subseteq Q$ such that $\Phi_\circ^\rho(\mathsf{pp}, \mathsf{i}_\circ, \mathsf{q}) = 0$.

Together, these cases imply that

$$\begin{gathered} \exists\, j \in [m] \text{ s.t. } \mathsf{AS}_\mathsf{ARG}.\mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}_j) = 0 \ \vee\ \exists\, i \in [n] \text{ s.t. } \Phi_\mathcal{V}^\rho\big(\mathsf{pp}, \mathbb{i}, (\mathbb{x}_i, \pi_i)\big) = 0 \\ \Downarrow \\ \exists\, j \in [m] \text{ s.t. } \mathsf{AS}_\circ.\mathrm{D}^\rho(\mathsf{dk}_\circ, \mathsf{acc}_j) = 0 \ \vee\ \exists\, \mathsf{q} \in Q \text{ s.t. } \Phi_\circ^\rho(\mathsf{pp}, \mathsf{i}_\circ, \mathsf{q}) = 0 \end{gathered}$$

Hence $\mathcal{B}$ break the soundness of $\mathsf{AS}_\circ$ whenever $\mathcal{A}$ breaks the soundness of AS_ARG.

Post-quantum security. Note that the soundness argument applies equally to quantum adversaries $\mathcal{A}$, and so if $\mathsf{AS}_\circ$ is post-quantum secure, then so is AS_ARG.

6.4 Zero knowledge

We describe the simulator S_ARG for AS_ARG, which is constructed using the simulator $\mathrm{S}_\circ$ for $\mathsf{AS}_\circ$. The first stage of the simulator is identical to $\mathrm{S}_\circ$: $\mathrm{S}_\mathsf{ARG}(1^\lambda) = \mathrm{S}_\circ(1^\lambda)$. The second stage is as follows.

```
$\mathrm{S}_\mathsf{ARG}(\mathsf{pp}_\mathsf{AS}, \mathsf{pp}, \mathbb{i}, \tau)$:
1. Run $(\mathrm{ipk}, \mathrm{ivk} = (\mathsf{i}_\circ, \mathrm{ivk}_\mathsf{pe})) \leftarrow \mathcal{I}^\rho(\mathsf{pp}, \mathbb{i})$.
2. Output $(\mathsf{acc}, \mu) \leftarrow \mathrm{S}_\circ(\mathsf{pp}_\circ = \mathsf{pp}_\mathsf{AS}, \mathsf{pp}, \mathsf{i}_\circ, \tau)$.
```

For an “honest” adversary $\mathcal{A}$ for $\mathsf{AS}_{\mathsf{ARG}}$, the adversary $\mathcal{B}$ described in Section 6.2 is an “honest” adversary for $\mathsf{AS}_{\circ}$. Moreover, the “view” of $\mathrm{S}_{\circ}$ is the same when $\mathrm{S}_{\mathsf{ARG}}$ is interacting with $\mathcal{A}$ as when $\mathrm{S}_{\circ}$ is interacting directly with $\mathcal{B}$. Hence, the zero knowledge property of $\mathrm{S}_{\circ}$ ensures that $\mathrm{S}_{\mathsf{ARG}}$ achieves zero knowledge.

Post-quantum security. We simply observe that this argument also implies that quantum computational indistinguishability is preserved, and continues to hold when $\mathcal{A}$ is a quantum circuit.

7 Accumulating polynomial commitments based on discrete logarithms

We construct an accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$, a polynomial commitment scheme that is inspired from several prior works [BCCGP16; BBBPWM18; WTSTW18], and which we describe in Appendix A.

The scheme $\mathsf{PC}_{\mathsf{DL}}$ is secure in the random oracle model assuming hardness of the discrete logarithm problem, and has the attractive feature that evaluation proofs are $O(\log d)$ elements in $\mathbb{G}$, where d is the degree of the committed polynomial. However, $\mathsf{PC}_{\mathsf{DL}}$ has the drawback that checking an evaluation proof requires $O(d)$ scalar multiplications in $\mathbb{G}$.

Our accumulation scheme for $\mathsf{PC}_{\mathsf{DL}}$, which is based on the batching ideas of [BGH19], enables deferring the expensive check to the decider: the accumulation verifier V only requires $O(\log d)$ scalar multiplications per accumulation, while the decider D requires $O(d)$ scalar multiplications. Hence, checking n evaluation proofs requires a total of $O(n \log d + d)$ scalar multiplications, as opposed to $\Omega(n \cdot d)$ for the naive approach.

Theorem 7.1. *If* $\mathsf{PC}_{\mathsf{DL}}$ *described in Appendix A is a polynomial commitment scheme then the tuple* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *constructed in Section 7.1 is an accumulation scheme in the random oracle model for* $\mathsf{PC}_{\mathsf{DL}}$. AS *achieves the following efficiency:*

- Generator: $\mathrm{G}(1^\lambda)$ *runs in time* $O(\lambda)$.
- Indexer: $\mathrm{I}(\mathsf{pp}, \mathsf{i}_\Phi)$ *runs in time* $O_\lambda(d)$.
- Accumulation prover: *The time of* $\mathrm{P}^\rho(\mathsf{apk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ *is dominated by the time to perform* $O(n+m+d)$ *scalar multiplications in* $\mathbb{G}$.
- Accumulator size: *The output accumulator is a polynomial commitment consisting of one element in* $\mathbb{G}$ *plus an evaluation proof consisting of* $O(\log d)$ *elements in* $\mathbb{G}$.
- Accumulation verifier: *The time of* $\mathrm{V}^\rho(\mathsf{avk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc})$ *is dominated by the time to perform* $O((n+m) \cdot \log d)$ *scalar multiplications in* $\mathbb{G}$.
- Decider: *The time of* $\mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc})$ *is dominated by the time to perform* $O(d)$ *scalar multiplications in* $\mathbb{G}$.

Recall that obtaining an accumulation scheme for a polynomial commitment scheme entails obtaining an accumulation scheme for the pair $(\Phi_{\mathsf{PC}}, \mathcal{H}_{\mathsf{PC},D})$ specified in Definition 4.2.

We remark that Theorem 7.1 considers the special case of accumulating claims about a single degree bound (i.e., when the input to $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}$ is a singleton). We leave the case of multiple degree bounds to future work (and believe that the degree enforcement techniques in [CHMMVW20] would work in this setting).

Finally, we note that our construction of AS for $\mathsf{PC}_{\mathsf{DL}}$ does not use the accumulation proof π_{V}, and so we omitted it from the interfaces in Theorem 7.1 and the construction below.

7.1 Construction

We present our accumulation scheme AS for $\mathsf{PC}_{\mathsf{DL}}$. Recall that the definition of an accumulation scheme requires simultaneous accumulation of previous accumulators and of new instances. In AS below, accumulators and instances are both evaluation proofs for openings of committed polynomials. That is, an accumulator acc and an instance q are both of the form $((C, d, z, v), \pi)$. Also, the predicate Φ_{PC} (from Definition 4.2) and the accumulation decider D are identical, and so there is no distinction between new instances and previous accumulators. Below, we exploit this to simplify exposition by accumulating only instances.

Our construction below uses algorithms from $\mathsf{PC}_{\mathsf{DL}}$ defined in Appendix A. In particular, we use the subroutine $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}$ (see Figure 2) which is the "succinct" part of $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}$.

Note that the algorithms of both AS and $\mathsf{PC}_{\mathsf{DL}}$ use random oracles. For security, we must ensure that the random oracle used by the algorithms of $\mathsf{PC}_{\mathsf{DL}}$ differs from that used by the algorithms of AS. We do so by relying on domain separation: we derive two different random oracles $\rho_0(\cdot) := \rho(0\|\cdot)$ (for $\mathsf{PC}_{\mathsf{DL}}$) and $\rho_1(\cdot) := \rho(1\|\cdot)$ (for AS) from the common random oracle ρ.

Generator. On input a security parameter 1^λ, the generator G outputs 1^λ.

Indexer. On input the accumulator parameters $\mathsf{pp}_{\mathsf{AS}}$, the $\mathsf{PC}_{\mathsf{DL}}$ public parameters $\mathsf{pp}_{\mathsf{PC}}$, and a predicate index $\mathsf{i}_\Phi = D$, the indexer I proceeds as follows. Compute the committer and receiver keys $(\mathsf{ck}_{\mathsf{PC}}, \mathsf{rk}_{\mathsf{PC}}) := \mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}(\mathsf{pp}_{\mathsf{PC}}, D)$. Parse $\mathsf{ck}_{\mathsf{PC}}$ as a tuple (ck, H), and ck as $(\langle\mathsf{group}\rangle, \mathsf{hk})$. Output the accumulator proving key $\mathsf{apk} := \mathsf{ck}_{\mathsf{PC}}$, accumulator verification key $\mathsf{avk} := (\langle\mathsf{group}\rangle, H, D)$, and decision key $\mathsf{dk} := \mathsf{rk}_{\mathsf{PC}}$.

Common subroutine. The accumulation prover and verifier share a common subroutine, described below.

```
$T^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n)$:
1. Parse avk as $(\langle\mathsf{group}\rangle = (\mathbb{G}, q, G), H, D)$.
2. For each $i \in [n]$:
    (a) Parse $\mathsf{q}_i$ as a tuple $((C_i \in \mathbb{G}, d_i \in \mathbb{N}, z_i \in \mathbb{F}_q, v_i \in \mathbb{F}_q), \pi_i)$.
    (b) Compute $(h_i(X), U_i) := \mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}^{\rho_0}(\mathsf{avk}, C_i, z_i, v_i, \pi_i)$ (see Figure 2).
3. For each $i$ in $[n]$, check that $d_i = D$. (We accumulate only the degree bound $D$.)
4. Compute the challenge $\alpha := \rho_1([h_i, U_i]_{i=1}^n)$.
5. Set the polynomial $h(X) := \sum_{i=1}^n \alpha^{i-1} h_i(X)$.
6. Compute the accumulated commitment $C := \sum_{i=1}^n \alpha^{i-1} \cdot U_i$.
7. Compute the challenge $z := \rho_1(C, h)$.
8. Output $(C, d, z, h(X))$.
```

Accumulation prover. On input the accumulator proving key $\mathsf{apk} = \mathsf{ck}_{\mathsf{PC}}$ and new inputs $[\mathsf{q}_i]_{i=1}^n$, the accumulation prover P proceeds as follows. Construct avk from apk, and use it to compute the tuple $(C, d, z, h(X)) := T^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n)$. Compute the evaluation $v := h(z)$, and generate the evaluation proof $\pi := \mathsf{PC}_{\mathsf{DL}}.\mathsf{Open}^{\rho_0}(\mathsf{ck}_{\mathsf{PC}}, h(X), C, d, z)$. Finally, output the accumulator $\mathsf{acc} = ((C, d, z, v), \pi)$.

Accumulation verifier. On input the accumulator verification key avk, new inputs $[\mathsf{q}_i]_{i=1}^n$, and a new accumulator $\mathsf{acc} = ((C, d, z, v), \pi)$, the accumulation verifier V computes $(C', d', z', h(X)) := T^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n)$, and checks that $C' = C$, $d' = d$, $z' = z$, and $h(z) = v$.

Decider. On input $\mathsf{dk} = \mathsf{rk}_{\mathsf{PC}}$ and $\mathsf{acc} = ((C, d, z, v), \pi)$, the decider D outputs $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}^{\rho_0}(\mathsf{rk}_{\mathsf{PC}}, C, d, z, v, \pi)$.

7.2 Proof of Theorem 7.1

Recall that $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}$ is a *deterministic* function of ck and the committed polynomial. Below we will write "A is a commitment to p" when $A = \mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}(\mathsf{ck}, p)$ (and "A is not a commitment to p" when $A \neq \mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}(\mathsf{ck}, p)$); the value of ck will be clear from context (and is also equal to rk).

Completeness. Note that both instances and accumulators have the same form: they consist of claims about the correct evaluation of a committed polynomial. Note also that both the predicate Φ_{PC} and the decider D check the same condition: that the claim of correct evaluation holds. Hence, without loss of generality, we can consider the following simplified definition of completeness that only considers the accumulation of instances, and omits old accumulators. For all (unbounded) adversaries $\mathcal{A}_1, \mathcal{A}_2$, the following holds:

$$\Pr\left[\begin{array}{c|r} \forall\, i \in [n],\ \Phi^{\rho}_{\mathsf{PC}}(\mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi, \mathsf{q}_i) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \Downarrow & \mathsf{pp} \leftarrow \mathrm{G}^{\rho}(1^\lambda) \\ & \mathsf{pp}_{\mathsf{PC}} \leftarrow \mathcal{H}^{\rho}_{\mathsf{PC},D}(1^\lambda) \\ \mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, \mathsf{acc}) = 1 & (\mathsf{i}_\Phi, [\mathsf{q}_i]_{i=1}^n) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}) \\ \mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}) = 1 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi) \\ & \mathsf{acc} \leftarrow \mathrm{P}^{\rho}(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^n) \end{array}\right] = 1\ .$$

We prove this directly.

First, since for each $i \in [n]$ we have $\Phi^{\rho}_{\mathsf{PC}}(\mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi, \mathsf{q}_i) = 1$, we know that $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}^{\rho_0}(\mathsf{rk}, C_i, d_i, z_i, v_i, \pi_i) = 1$, where $((C_i, d_i, z_i, v_i), \pi_i) = \mathsf{q}_i$. This in turn means that:

- $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}^{\rho_0}(\langle\mathsf{group}\rangle, C, d, z, v, \pi)$ accepts and outputs (h_i, U_i); and that
- U_i is a commitment to the polynomial h_i.

Next, since the new accumulator acc is generated honestly (i.e., $\mathsf{acc} = ((C, d, z, v), \pi) = \mathrm{P}^{\rho}(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^n)$), the following statements hold:

- The commitment C equals the linear combination $\sum_{i=1}^n \alpha^{i-1} U_i$, so C is a commitment to $h = \sum_{i=1}^n \alpha^{i-1} h_i$ (as each U_i is a commitment to the polynomial h_i, and $\mathsf{PC}_{\mathsf{DL}}$ is a homomorphic commitment scheme).
- The evaluation of h at z equals v (i.e., $v = h(z)$).
- $\pi = \mathsf{PC}_{\mathsf{DL}}.\mathsf{Open}^{\rho_0}(\mathsf{ck}, h(X), C, d, z)$ is an honestly-generated proof of a true statement.

Together, these points imply that:

1. *The accumulation verifier accepts.* Recall that the accumulation verifier V: (a) runs the same subroutine T that the prover P runs, and checks that the output matches the one in the claimed (honest) accumulator; and (b) checks that $v = h(z)$. Both of these checks pass because the accumulation prover is honest.
2. *The decider accepts.* We know that $v = h(z)$, C is a commitment to h, and that π is an honestly-generated proof that C commits to a polynomial which evaluates to v at z. By completeness of $\mathsf{PC}_{\mathsf{DL}}$, D accepts.

We conclude that the accumulation scheme AS constructed in Section 7.1 is complete.

Soundness. Similarly to the completeness case, we modify the definition of soundness in Section 4 to only consider the accumulation of new instances, and hence to omit old accumulators. The resulting definition requires that the following probability is negligible for every polynomial-size adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{c|r} \mathrm{V}^{\rho}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, \mathsf{acc}) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp} \leftarrow \mathrm{G}^{\rho}(1^\lambda) \\ \wedge & \mathsf{pp}_{\mathsf{PC}} \leftarrow \mathcal{H}^{\rho}_{\mathsf{PC},D}(1^\lambda) \\ & (\mathsf{i}_\Phi, [\mathsf{q}_i]_{i=1}^n, \mathsf{acc}) \leftarrow \mathcal{A}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}) \\ \exists\, i \in [n],\ \Phi^{\rho}_{\mathsf{PC}}(\mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi, \mathsf{q}_i) = 0 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi) \end{array}\right]\ . \tag{7}$$

Fix a polynomial-size adversary $\mathcal{A}$ and degree bound D, and denote by δ the above probability for these choices. We will construct an adversary for the zero-finding game in Lemma 3.3 that wins with probability $\delta/2 - \mathrm{negl}(\lambda)$, from which it follows that δ is negligible (since q is superpolynomial in λ).

We first describe the commitment schemes $\mathsf{CM}_1, \mathsf{CM}_2$ used in the zero-finding games. Both schemes have common setup and trim algorithms, and public parameters pp equal to the public parameters of $\mathsf{PC}_{\mathsf{DL}}$ with maximum degree L. The message space $\mathcal{M}_{\mathsf{pp}}$ for CM_1 consists of tuples (p, h), where p and h are univariate polynomials of degree at most L. Note that h is uniquely represent by $([h_i]_{i=1}^n, \alpha)$, where each h_i is a univariate polynomial of degree L, and $\alpha \in \mathbb{F}_q$. The message space $\mathcal{M}_{\mathsf{pp}}$ for CM_2 consists of lists of pairs $[(h_i, U_i)]_{i=1}^n$, where each h_i is a univariate polynomial of degree at most L, and each U_i is a group element.

```
$\mathsf{CM}_j.\mathsf{Setup}^{\rho_0}(1^\lambda, L)$: Output $\mathsf{pp} \leftarrow \mathsf{PC}_{\mathsf{DL}}.\mathsf{Setup}^{\rho_0}(1^\lambda, L)$.

$\mathsf{CM}_j.\mathsf{Trim}^{\rho_0}(\mathsf{pp}, n, N)$:
1. Compute $(\mathsf{ck}_0, \mathsf{rk}_0) \leftarrow \mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}^{\rho_0}(\mathsf{pp}, N)$.
2. Output $\mathsf{ck} := (\mathsf{ck}_0, n)$.

$\mathsf{CM}_1.\mathsf{Commit}(\mathsf{ck} = (\mathsf{ck}_0, n), \mathfrak{p} = (p, h); r)$:
1. Commit to $p$: $C \leftarrow \mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}(\mathsf{ck}_0, p)$.
2. Output $(C, h)$.

$\mathsf{CM}_2.\mathsf{Commit}(\mathsf{ck}, \mathfrak{p} = ([(h_i, U_i)]_{i=1}^n); r)$: Output $\mathfrak{p}$.
```

Both commitment schemes are binding. It remains to specify the families of functions $\{f_{\mathsf{pp}}^{(1)}\}_{\mathsf{pp}}$,$\{f_{\mathsf{pp}}^{(2)}\}_{\mathsf{pp}}$ that we use in the respective zero-finding games. We define $f_{\mathsf{pp}}^{(1)}(p, h = ([h_i]_{i=1}^n, \alpha)) := p - \sum_{i=1}^n \alpha^{i-1} h_i$, and

```
$f_{\mathsf{pp}}^{(2)}(\mathfrak{p} = ([(h_i, U_i)]_{i=1}^n))$:
1. Construct the key pair $(\mathsf{ck}_0, \mathsf{rk}_0) := \mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}(\mathsf{pp}_{\mathsf{PC}}, \deg(h_1))$.
2. For each $i \in [n]$, construct a $\mathsf{PC}_{\mathsf{DL}}$ commitment to $h_i$: $B_i \leftarrow \mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}(\mathsf{ck}_0, h_i; \perp)$.
3. For each $i \in [n]$, compute $a_i \in \mathbb{F}_q$ such that $a_i G = U_i - B_i$.
4. Output the polynomial $a(Z) := \sum_{i=1}^n a_i Z^{i-1}$.
```

We next describe an adversary $\mathcal{C}$ against $\mathsf{PC}_{\mathsf{DL}}$, which simply runs the soundness experiment for the accumulation scheme and outputs acc as output by $\mathcal{A}$. For convenience we also have $\mathcal{C}$ output $[\mathsf{q}_i]_{i=1}^n$; this will be ignored by the extractor.

```
$\mathcal{C}^\rho(\mathsf{pp}_{\mathsf{PC}})$:
1. Set AS public parameters $\mathsf{pp}_{\mathsf{AS}} := 1^\lambda$.
2. Compute $(\mathsf{i}_\Phi, [\mathsf{q}_i]_{i=1}^n, \mathsf{acc}) \leftarrow \mathcal{A}^\rho(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{PC}})$.
3. Parse $\mathsf{i}_\Phi$ as the degree bound $N$.
4. Output $(N, \mathsf{acc} = ((C, d, z, v), \pi); [\mathsf{q}_i]_{i=1}^n)$.
```

We use the extractor $\mathcal{E}_{\mathcal{C}}$ corresponding to $\mathcal{C}$ to construct adversaries $\mathcal{B}_1, \mathcal{B}_2$ for zero-finding games against $(\mathsf{CM}_1, \{f_{\mathsf{pp}}^{(1)}\}_{\mathsf{pp}}), (\mathsf{CM}_2, \{f_{\mathsf{pp}}^{(2)}\}_{\mathsf{pp}})$ respectively, with $L = D$ where $D = \mathrm{poly}(\lambda)$ is the maximum degree parameter as in the soundness experiment for the accumulation scheme.

```
$\mathcal{B}_j^\rho(\mathsf{pp})$:
1. Compute $(N, \mathsf{acc}, [\mathsf{q}_i]_{i=1}^n) \leftarrow \mathcal{C}^\rho(\mathsf{pp})$.
2. Parse $[\mathsf{q}_i]_{i=1}^n$ as $[((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n$.
3. Compute $p \leftarrow \mathcal{E}_{\mathcal{C}}^\rho(\mathsf{pp})$.
4. For each $i \in [n]$, obtain $h_i$ and $U_i$ from $\pi_i$.
5. Compute $\alpha := \rho_1([(h_i, U_i)]_{i=1}^n)$.
6. If $j = 1$, output $((n, N), (p, h := ([h_i]_{i=1}^n, \alpha)))$. If $j = 2$, output $((n, N), ([(h_i, U_i)]_{i=1}^n))$.
```

We show that either $\mathcal{B}_1$ or $\mathcal{B}_2$ wins its respective zero-finding game with probability at least $\delta/2 - \mathrm{negl}(\lambda)$.

Since D accepts with probability δ, and by the extraction property of $\mathsf{PC}_{\mathsf{DL}}$, the following holds with probability at least $\delta - \mathrm{negl}(\lambda)$: $\mathcal{E}_{\mathcal{C}}$ outputs a polynomial p whose commitment is C, $p(z) = v$, and $\deg(p) \leq d$; and, moreover, $(\mathsf{acc}, [\mathsf{q}_i]_{i=1}^n)$ satisfies the left-hand side of Eq. (7). This latter point implies that, parsing q_i as $((C_i, d_i, z_i, v_i), \pi_i)$ and letting $(h_i, U_i) := \mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}^{\rho_0}(\langle\mathsf{group}\rangle, C_i, d_i, z_i, v_i, \pi_i)$:

- Since $\mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, \mathsf{acc})$ accepts, it holds that (a) for each i in $[n]$, $b_i = 1$; and (b) parsing acc as $((C, d, z, v), \pi)$ and setting $\alpha := \rho_1([h_i, U_i]_{i=1}^n)$, we have that $z = \rho_1(C, [h_i]_{i=1}^n, \alpha)$, $C = \sum_{i=1}^n \alpha^{i-1} U_i$, and $v = \sum_{i=1}^n \alpha^{i-1} h_i(z)$.

- For some $i \in [n]$, $\Phi_{\mathsf{PC}}^\rho(\mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi, \mathsf{q}_i) = \mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}^{\rho_0}(\mathsf{rk}, (C_i, d_i, z_i, v_i), \pi_i) = 0$. By construction (see Appendix A.2), this implies that either $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}$ rejects, or the group element U_i is not a commitment to h_i.

The above tells us that there exists some $i \in [n]$ such that U_i is not a commitment to h_i. In other words, if we define $B_i := \mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}(\mathsf{ck}, h_i)$, then there exists i such that $U_i \neq B_i$. Letting $a_i \in \mathbb{F}_q$ be such that $a_i G = U_i - B_i$, we deduce that the polynomial $a(Z) = \sum_{i=1}^n a_i Z^{i-1}$ is not identically zero.

There are then two cases.

1. $C \neq \sum_{i=1}^n \alpha^{i-1} B_i$. Then since C is a commitment to p, $p(X) - h(X)$ is not identically zero, but $p(z) = v = h(z)$. Hence $\mathcal{B}_1$ wins the zero-finding game against $(\mathsf{CM}_1, \{f_{\mathsf{pp}}^{(1)}\}_{\mathsf{pp}})$.

2. $C = \sum_{i=1}^n \alpha^{i-1} B_i$. Then since $C = \sum_{i=1}^n \alpha^{i-1} U_i$, α is a zero of the polynomial $a(Z)$. Hence $\mathcal{B}_2$ wins the zero-finding game against $(\mathsf{CM}_2, \{f_{\mathsf{pp}}^{(2)}\}_{\mathsf{pp}})$.

Since at least one of these two cases occurs with probability at least $\delta/2 - \mathrm{negl}(\lambda)$, the claim follows.

Efficiency. We now analyze the efficiency of our accumulation scheme.

- *Generator:* $\mathrm{G}^\rho(1^\lambda)$ outputs 1^λ, and hence takes $O(\lambda)$ time.

- *Indexer:* $\mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi)$ runs $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}$ and outputs $(\mathsf{apk}, \mathsf{dk}) := (\mathsf{ck}, \mathsf{rk})$, which have size $O_\lambda(d)$, and also avk, which has size $O_\lambda(1)$. This takes time $O_\lambda(d)$.

- *Accumulation prover:* The time of $\mathrm{P}^\rho(\mathsf{apk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m)$ is dominated by running $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Open}$, which uses $3d$ scalar multiplications for each i, for a total of $3d \cdot n$ scalar multiplications.

- *Accumulator size:* The accumulator acc consists of an evaluation claim and a proof. The total size is $2\log_2(d) + 1$ elements in $\mathbb{G}$ ($2\log_2(d)$ group elements for the proof and 1 for the commitment) and a constant number of field elements.

- *Accumulation verifier:* $\mathrm{V}^\rho(\mathsf{avk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc})$ runs $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}$ as a subroutine, which requires $(2\log_2(d) + 2)$ scalar multiplications per iteration among n iterations. (Note that h is succinctly represented, and can be evaluated in $O(\log d)$ field operations).

- *Decider:* $\mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc})$ runs $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}$, which requires uses $O(d)$ scalar multiplications.

8 Accumulating polynomial commitments based on bilinear groups

We construct an accumulation scheme for the pairing-based polynomial commitment scheme $\mathsf{PC}_{\mathsf{AGM}}$ [KZG10; CHMMVW20]. The main feature of our accumulation scheme is that, while invoking $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Check}$ to check n evaluation proofs requires $O(n)$ pairings, verifying an accumulation of these proofs is much cheaper: it requires only $O(n)$ scalar multiplications in $\mathbb{G}_1$ to run the accumulation scheme verifier V, plus one pairing to run the decider D. In more detail, we prove the following theorem.

Theorem 8.1. *The tuple* $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *constructed in Section 8.1 is a zero knowledge accumulation scheme in the random oracle model for the polynomial commitment scheme* $\mathsf{PC}_{\mathsf{AGM}}$ *described in Fig. 1.* AS *achieves the following efficiency:*

- Generator: $\mathrm{G}^{\rho}(1^{\lambda})$ *runs in* $\mathrm{poly}(\lambda)$ *time.*
- Indexer: $\mathrm{I}^{\rho}(\mathsf{pp}, \mathsf{i}_{\Phi})$ *runs in* $\mathrm{poly}(\lambda)$ *time.*
- Accumulation prover: *The time of* $\mathrm{P}^{\rho}(\mathsf{apk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m})$ *is dominated by the time to perform* $O(n+m)$ *scalar multiplications in* $\mathbb{G}_1$.
- Accumulator size: *The accumulator* acc *consists of two elements in* $\mathbb{G}_1$.
- Accumulation proof size: *The accumulation proof* π_{V} *consists of two elements in* $\mathbb{G}_1$.
- Accumulation verifier: *The time of* $\mathrm{V}^{\rho}(\mathsf{avk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$ *is dominated by the time to perform* $O(n+m)$ *scalar multiplications in* $\mathbb{G}_1$.
- Decider: *The time of* $\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc})$ *is dominated by the time to perform* 1 *pairing.*

We proceed as follows: in Section 8.1 we describe our accumulation scheme; and then in Section 8.2 we prove that it fulfills Theorem 8.1. Recall that obtaining an accumulation scheme for a polynomial commitment scheme involves obtaining an accumulation scheme for the pair $(\Phi_{\mathsf{PC}}, \mathcal{H}_{\mathsf{PC},D})$ specified in Definition 4.2.

Throughout, we highlight in blue the parts of $\mathsf{PC}_{\mathsf{AGM}}$ used for the hiding property, and the corresponding parts of our accumulation scheme dealing with these; they can be dropped if no hiding is used for $\mathsf{PC}_{\mathsf{AGM}}$.

8.1 Construction

The algorithms of $\mathsf{PC}_{\mathsf{AGM}}$ and of AS use random oracles, but for security they have to be distinct. For this we use domain separation: we derive two different random oracles $\rho_0(\cdot) := \rho(0\|\cdot)$ (for $\mathsf{PC}_{\mathsf{AGM}}$) and $\rho_1(\cdot) := \rho(1\|\cdot)$ (for AS) from the common random oracle ρ that all algorithms have access to.

Generator. On input a security parameter λ (written in unary), G outputs 1^{λ}.

Indexer. On input the accumulator parameters $\mathsf{pp}_{\mathsf{AS}}$, the $\mathsf{PC}_{\mathsf{AGM}}$ public parameters $\mathsf{pp}_{\mathsf{PC}}$, and a predicate index $\mathsf{i}_{\Phi} = [d_i]_{i=1}^{n}$, the indexer I computes the receiver key rk the same way as $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Trim}$, and outputs $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := (\mathsf{rk}, \mathsf{rk}, \mathsf{rk})$.

The accumulation prover and the accumulation verifier both rely on the following common subroutine:

```
$\mathsf{ComputeAcc}^{\rho}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \pi_{\mathrm{V}})$:
1. Parse $[\mathsf{q}_i]_{i=1}^{n}$ as $[((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^{n}$.
2. Parse $\pi_{\mathrm{V}}$ as $(s\beta G, sG)$.
3. Obtain the supported degree bounds $\boldsymbol{d}$ from $\mathsf{apk} = \mathsf{rk}$.
4. For each $i \in [n]$,
    (a) Check that $d_i \in \boldsymbol{d}$.
    (b) Parse the commitment $C_i$ as a tuple $(U_i, S_i) \in \mathbb{G}_1^2$.
    (c) Generate the $i$-th opening challenge $\xi_i := \rho_0(\mathsf{rk}, d_i, C_i, z_i, v_i) \in \mathbb{F}_q$.
```

   (d) Parse the proof π_i as $(W_i, \bar{v}_i)$.
5. For each $j \in [m]$, parse the j-th old accumulator acc_j as a tuple $(C^\star_j, \pi^\star_j) \in \mathbb{G}_1^2$.
6. Compute the new accumulation challenge $r \in \mathbb{F}_q$ as

$$r := \rho_1(\mathsf{rk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \pi_{\mathrm{V}}) \ .$$

7. Accumulate commitments and proofs as follows:
   (a) Compute $C := \sum_{i=1}^n r^{i-1}((U_i - v_i G) + \xi_i(S_i - v_i \beta^{D-d_i} G) - \bar{v}_i \gamma G + z_i W_i)$.
   (b) Accumulate old accumulated commitments: $C^\star := C + \sum_{j=1}^m r^{n+j-1} C^\star_j + r^{n+m} \cdot s\beta G$.
   (c) Accumulate all old and new proofs: $\pi^\star := \sum_{i=1}^n r^{i-1} W_i + \sum_{j=1}^m r^{n+j-1} \pi^\star_j + r^{n+m} \cdot sG$.
8. Output the new accumulator $\mathsf{acc} := (C^\star, \pi^\star)$.

Accumulation prover. On input the accumulator proving key apk, new inputs $[\mathsf{q}_i]_{i=1}^n$, and old accumulators $[\mathsf{acc}_j]_{j=1}^m$, P computes a new accumulator acc as follows. Sample a random scalar $s \in \mathbb{F}_q$, set the accumulation proof $\pi_{\mathrm{V}} := (s\beta G, sG)$, compute the new accumulator $\mathsf{acc} := \mathsf{ComputeAcc}^\rho(\mathsf{apk} = \mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \pi_{\mathrm{V}})$, and output $(\mathsf{acc}, \pi_{\mathrm{V}})$.

Accumulation verifier. On input the accumulator verification key avk, new instances $[\mathsf{q}_i]_{i=1}^n$, old accumulators $[\mathsf{acc}_j]_{j=1}^m$, and a new accumulator acc, V checks that acc accumulates $[\mathsf{q}_i]_{i=1}^n$ and $[\mathsf{acc}_j]_{j=1}^m$ simply by running the common subroutine $\mathsf{ComputeAcc}$, i.e. by checking that

$$\mathsf{acc} = \mathsf{ComputeAcc}^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \pi_{\mathrm{V}}) \ .$$

Decider. On input the decision key dk and an accumulator acc, D checks the validity of acc by parsing acc as a tuple $(C^\star, \pi^\star) \in \mathbb{G}_1^2$ and checking that $e(C^\star, H) = e(\pi^\star, \beta H)$.

8.2 Proof of Theorem 8.1

Note that in the foregoing construction, one can view the accumulation proof $\pi_{\mathrm{V}} = (s\beta G, sG)$ as an accumulator because it passes the decider's pairing check: $e(s\beta G, H) = e(sG, \beta H)$. We adopt this viewpoint below, and omit explicit discussion of π_{V} in both the completeness and soundness proofs. This is a valid change because it never weakens the adversary: in the completeness proof, the honest prover simply adds another honest old accumulator to the list sampled by the adversary $\mathcal{A}$, and in the soundness proof the adversary $\mathcal{A}$ already samples old accumulators.

Completeness. We argue completeness directly. First, we follow [CHMMVW20] and rewrite the pairing equation in $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Check}$ as follows:

$$\begin{aligned} e(C' - vG - \bar{v}\gamma G, H) &= e(W, \beta H - zH) \\ &= e(W, \beta H) - e(W, zH) \\ &= e(W, \beta H) - e(zW, H) \\ \implies e(C' - vG - \bar{v}\gamma G + zW, H) &= e(W, \beta H) \ . \end{aligned}$$

Now, we want to show that AS.P, when given accepting inputs $[((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n$ and old accumulators $[\mathsf{acc}_j]_{j=1}^m$ that are accepted by AS.D, outputs a new accumulator acc such that AS.V successfully verifies that acc is the accumulation of $[((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n$ and $[\mathsf{acc}_j]_{j=1}^m$, and that AS.D accepts acc.

First, since for each $i \in [n]$, $\Phi_{\mathsf{PC}}(\mathsf{i}_\Phi, (C_i, z_i, d_i, v_i), \pi_i) = 1$, we know that $e(C'_i - v_i G - \bar{v}_i \gamma G, H) = e(W_i, \beta H - z_i H)$, which, by the foregoing manipulation, implies that $e(C' - vG - \bar{v}_i \gamma G + z_i W_i, H) = e(W_i, \beta H)$. Hence, for any choice of $r \in \mathbb{F}_q$, it holds that

$$e(\textstyle\sum_{i=1}^n r^{i-1}(C'_i - v_i G - \bar{v}_i \gamma G + zW_i), H) = e(\textstyle\sum_{i=1}^n r^{i-1} W_i, \beta H) \ . \quad (8)$$

Next, since for each $j \in [m]$, acc_j is accepted by D, we know that $e(C_j^\star, H) = e(\pi_j^\star, \beta H)$. Hence, for any choice of $r \in \mathbb{F}_q$, it holds that

$$e(\textstyle\sum_{j=1}^m r^{n+j-1} C_j^\star, H) = e(\sum_{j=1}^m r^{n+j-1}\pi_j^\star, \beta H) \ . \tag{9}$$

Adding Equations (8) and (9), we obtain that $e(C^\star, H) = e(\pi^\star, \beta H)$, and hence $\mathsf{AS}.\mathrm{D}(\mathsf{dk}, \mathsf{acc} = (C^\star, \pi^\star)) = 1$, as required. Furthermore, V accepts as it just runs the common subroutine P. Hence AS achieves completeness.

Soundness. Recall that we must show that the following probability is negligible for all polynomial-size adversaries $\mathcal{A}_1, \mathcal{A}_2$:

$$\Pr\left[\begin{array}{c|r} \mathrm{V}^\rho(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}) = 1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp} \leftarrow \mathrm{G}^\rho(1^\lambda) \\ \wedge & (\mathsf{i}_\Phi, \mathsf{aux}) \leftarrow \mathcal{H}_{\mathsf{PC},D}^{\rho,\mathcal{A}_1}(\mathsf{pp}) \\ \big(\exists\, j \in [m],\ \mathrm{D}^\rho(\mathsf{dk}, \mathsf{acc}_j) = 0 \vee & ([\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}) \leftarrow \mathcal{A}_2^\rho(\mathsf{pp}, \mathsf{i}_\Phi, \mathsf{aux}) \\ \exists\, i \in [n],\ \Phi_{\mathsf{PC}}^\rho(\mathsf{i}_\Phi, \mathsf{q}_i, \pi_i) = 0\big) & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^\rho(\mathsf{pp}, \mathsf{i}_\Phi) \end{array}\right] . \tag{10}$$

First, we rewrite the pairing equation check in the decider D as follows:

$$e(C^\star, H) = e(\pi^\star, \beta H)$$
$$\Updownarrow$$
$$e(\textstyle\sum_i r^{i-1}(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i), H) + e(\sum_j r^{n+j-1} C_j^\star, H) = e(\sum_i r^{i-1} W_i, \beta H) + e(\sum_j r_1^{n+j-1} \pi_j^\star, \beta H)$$
$$\Updownarrow$$
$$e(\textstyle\sum_i r^{i-1}(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i), H) - e(\sum_i r^{i-1} W_i, \beta H) = e(\sum_j r^{n+j-1} \pi_j^\star, \beta H) - e(\sum_j r^{n+j-1} C_j^\star, H)$$

Writing $\mathbb{G}_T$ operations additively, this is equivalent to:

$$\sum_{i=1}^n r^{i-1}(e(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i, H) - e(W_i, \beta H)) = \sum_{j=1}^m r^{n+j-1}(e(\pi_j^\star, \beta H) - e(C_j^\star, H))$$

We define a function $s \colon \mathbb{F}_q \to \mathbb{G}_T$:

$$s(X) := \sum_{i=1}^n X^{i-1}(e(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i, H) - e(W_i, \beta H)) - \sum_{j=1}^m X^{n+j-1}(e(\pi_j^\star, \beta H) - e(C_j^\star, H)) \ ,$$

along with an associated polynomial $\hat{s}(X) = \sum_{i=1}^{m+n} a_i X^{i-1} \in \mathbb{F}_q[X]$, where $a_i \in \mathbb{F}_q$ is such that the coefficient of X^{i-1} in the above expression is equal to $a_i G_T$ for some fixed generator G_T of $\mathbb{G}_T$. Note that, for all $r \in \mathbb{F}_q$, $\hat{s}(r) = 0$ if and only if $s(r) = 0$.

If the pairing equation $e(C^\star, H) = e(\pi^\star, \beta H)$ holds, then $s(r) = 0$, whence $\hat{s}(r) = 0$. Furthermore, observe that $\hat{s}$ is identically zero if and only if:

- For each i in $[n]$, the coefficient of X^{i-1} is zero. That is, each $e(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i, H) - e(\pi_i, \beta H) = 0$, which in turn means that the predicate $\Phi_{\mathsf{PC}}(\mathsf{i}_\Phi, \mathsf{q}_i, \pi_i)$ accepts.
- For each j in $[m]$, the coefficient of X^{n+j-1} is zero. That is, each $e(C_j^\star, H) - e(\pi_j^\star, \beta H) = 0$, which in turn means that the decider $\mathrm{D}(\mathsf{dk}, \mathsf{acc}_j)$ accepts.

Together, this means that the implication in Eq. (10) is equivalent to the condition that $\hat{s} \not\equiv 0$ but $\hat{s}(r) = 0$.

To show that this occurs with negligible probability, we define a zero-finding game and apply Lemma 3.3. We define a commitment scheme $\mathsf{CM}' = (\mathsf{Setup}, \mathsf{Commit})$ and associated family of mapping functions $\{f_{\mathsf{pp}}\}_{\mathsf{pp}}$. The message space $\mathcal{M}_{\mathsf{pp}}$ for CM' is $\mathbb{G}_1^{n+m}$.

```
$\mathsf{CM}'.\mathsf{Setup}^{\rho_0}(1^\lambda, L = (m, n))$:
1. Sample a bilinear group
   $\langle\mathsf{group}\rangle = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e) \leftarrow \mathsf{SampleGrp}^{\rho_0}(1^\lambda)$.
2. Sample $\beta, \gamma \in \mathbb{F}_q$.
3. Set the public parameters $\mathsf{pp} = (\langle\mathsf{group}\rangle, \beta, \gamma, m, n)$.
4. Output $\mathsf{pp}$.

$\mathsf{CM}'.\mathsf{Commit}(\mathsf{pp}, \mathfrak{p} \in \mathcal{M}_{\mathsf{pp}}; r)$:    Output $\mathfrak{p}$.
```

```
$f_{\mathsf{pp}}(\mathfrak{p} \in \mathcal{M}_{\mathsf{pp}}) \to p$:
1. Parse $\mathfrak{p}$ as $(A_1, \ldots A_{n+m})$.
2. Let $G_T = e(G, H)$ be a generator of $\mathbb{G}_T$.
3. Compute $a_i \in \mathbb{F}_q$ such that $A_i = a_i G_T$.
4. Output the bivariate polynomial $\hat{s}(X) := \sum_{i=1}^{n+m} a_i X^{i-1}$.
```

Fix a choice of $D \in \mathbb{N}$ (inside $\mathcal{H}_{\mathsf{PC},D}$ from Definition 4.2) that maximizes the success probability of the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ against Lemma 3.3 for CM' and $\{f_{\mathsf{pp}}\}_{\mathsf{pp}}$:

```
$\mathcal{B}^\rho(\mathsf{pp}_{\mathsf{CM}'})$:
1. Parse $\mathsf{pp}_{\mathsf{CM}'}$ as $(\langle\mathsf{group}\rangle, \beta, \gamma, m, n)$.
2. Compute multiples of $G$ as follows: $\mathbf{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \ldots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \ldots & \gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{2D+2}$ .
3. Set $\mathsf{PC}_{\mathsf{AGM}}$ public parameters $\mathsf{pp}_{\mathsf{PC}} := (\langle\mathsf{group}\rangle, \mathbf{\Sigma}, \beta H)$.
4. Set AS public parameters $\mathsf{pp}_{\mathsf{AS}} := 1^\lambda$.
5. Compute the predicate index $\mathsf{i}_\Phi$ as follows:
   (a) Compute $(\mathbb{i}, \mathsf{aux}) \leftarrow \mathcal{A}_1^\rho(\mathsf{pp}_{\mathsf{PC}}, \mathsf{pp}_{\mathsf{AS}})$.
   (b) Set $\mathsf{i}_\Phi := (\mathbb{i}, \mathsf{pp}_{\mathsf{PC}})$.
6. Compute $([\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^m, \mathsf{acc}) \leftarrow \mathcal{A}_2^\rho(\mathsf{pp}_{\mathsf{AS}}, \mathsf{i}_\Phi, \mathsf{aux})$.
7. Parse $[\mathsf{q}_i]_{i=1}^n$ as $[((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^n$, and $[\mathsf{acc}_j]_{j=1}^m$ as $[(C_j^\star, \pi_j^\star)]_{j=1}^m$.
8. For each $i \in [n]$, set $A_i := e(C_i' - v_i G - \bar{v}_i \gamma G + z_i W_i, H) - e(W_i, \beta H)$.
9. For each $j \in [m]$, set $A_{n+j} := e(\pi_j^\star, \beta H) - e(C_j^\star, H)$.
10. Output $\mathfrak{p} := ([A_i]_{i=1}^{n+m})$.
```

Notice that if $\mathcal{A}$ succeeds, then $f(\mathfrak{p}) = \hat{s} \not\equiv 0$, but $\hat{s}(r) = 0$. This is exactly the winning condition for Lemma 3.3, and so $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does. This means that if $\mathcal{A}$ succeeds with probability δ, then $\mathcal{B}$ succeeds in the zero-finding game with probability δ, and so $\delta \leq \mathrm{negl}(\lambda)$ (since q is superpolynomial).

Zero knowledge. We demonstrate that AS is zero knowledge by constructing an efficient simulator S. During the setup phase, $\mathrm{S}^\rho(1^\lambda)$ computes $\mathsf{pp} \leftarrow \mathrm{G}^\rho(1^\lambda)$, and outputs $(\mathsf{pp}, \tau = \bot)$. Then, during the proving phase, $\mathrm{S}^\rho(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi, \tau)$ samples a random scalar $s \in \mathbb{F}_q$, sets $(C^\star, \pi^\star) := (s\beta G, sG)$, and outputs the accumulator $\mathsf{acc} := (C^\star, \pi^\star)$. Note that the simulator does not program the random oracle. This implies that AS is zero knowledge, because the simulated public parameters are identical to the honest ones, and the accumulators are identically distributed: in both cases, the accumulator consists of a pair of random group elements.

Efficiency. We now analyze the efficiency of our accumulation scheme.

- *Generator:* $\mathrm{G}^\rho(1^\lambda)$ outputs 1^λ, and hence takes $\mathrm{poly}(\lambda)$ time.

- *Indexer:* $\mathrm{I}^\rho(\mathsf{pp}, \mathsf{pp}_{\mathsf{PC}}, \mathsf{i}_\Phi)$ takes time $\mathrm{poly}(\lambda)$.

- *Accumulation prover:* $\mathrm{P}^{\rho}(\mathsf{apk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m})$ invokes the subroutine $\mathsf{ComputeAcc}$, which in turn computes $O(1)$ linear combinations of $O(n+m)$ elements in $\mathbb{G}_1$, and so requires time equal to $O(n+m)$ scalar multiplications in $\mathbb{G}_1$.
- *Accumulator size:* The accumulator acc consists of a pair $(C^{\star}, \pi^{\star})$ of elements in $\mathbb{G}_1$.
- *Accumulation proof size:* The accumulation proof π_{V} consists of two elements in $\mathbb{G}_1$.
- *Accumulation verifier:* $\mathrm{V}^{\rho}(\mathsf{avk}, [((C_i, d_i, z_i, v_i), \pi_i)]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{m}, \mathsf{acc}, \pi_{\mathrm{V}})$ simply invokes the common subroutine $\mathsf{ComputeAcc}$, and so runs in the same time as P.
- *Decider:* $\mathrm{D}^{\rho}(\mathsf{dk}, \mathsf{acc})$ performs one pairing, and so its running time is as claimed.

All algorithms below have access to the same random oracle ρ_0.

Setup. On input a security parameter λ (in unary) and a maximum degree D, $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Setup}$ samples public parameters $\mathsf{pp}_{\mathsf{PC}}$ as follows. Sample a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e) \leftarrow \mathsf{SampleGrp}^{\rho_0}(1^\lambda)$, and then sample a field element $\beta \in \mathbb{F}_q$. Compute

$$\mathbf{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{2D+2} .$$

Output $\mathsf{pp}_{\mathsf{PC}} := (\langle\mathsf{group}\rangle, \mathbf{\Sigma}, \beta H)$.

Trim. Given oracle access to public parameters $\mathsf{pp}_{\mathsf{PC}}$, and on input a list of degree bounds $[d_i]_{i=1}^n$, $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Trim}$ specializes the public parameters $\mathsf{pp}_{\mathsf{PC}}$ to the degree bounds $[d_i]_{i=1}^n$ as follows. Compute $d := \max_{i\in[n]}(d_1, \dots, d_n)$. From the powers $\mathbf{\Sigma}$ in $\mathsf{pp}_{\mathsf{PC}}$, select $\mathbf{\Sigma}_{\mathsf{ck}}$ as follows:

$$\mathbf{\Sigma}_{\mathsf{ck}} := \begin{pmatrix} G & \beta G & \dots & \beta^d G & \beta^{D-d} G & \beta^{D-d+1} G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \dots & \gamma\beta^d G & & & & \end{pmatrix} \in \mathbb{G}_1^{3d+3} .$$

Select $\mathbf{\Sigma}_{\mathsf{rk}} := \{\beta^{D-d_i} G\}_{i\in[n]}$. Set the commitment key $\mathsf{ck} := (\langle\mathsf{group}\rangle, D, \mathbf{\Sigma}_{\mathsf{ck}})$ and receiver key $\mathsf{rk} := (D, \langle\mathsf{group}\rangle, \mathbf{\Sigma}_{\mathsf{rk}}, \gamma G, \beta H, [d_i]_{i=1}^n)$. Output $(\mathsf{ck}, \mathsf{rk})$.

Commit. On input the commitment key ck, a univariate polynomial p over the field $\mathbb{F}_q$, a degree bound d, and commitment randomness $(\omega, \omega^\star)$, $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Commit}$ computes a commitment to p as follows. Obtain the supported degree bounds $[d_i]_{i=1}^n$ from ck. If $\deg(p) > d$ or $d \notin [d_i]_{i=1}^n$, abort. If ω and $\omega^\star$ are not $\perp$, then obtain from them random univariate polynomials $\bar{p}$ and $\bar{p}^\star$ of degree $\deg(p)$; otherwise, set $\bar{p}$ and $\bar{p}^\star$ to be the zero polynomial. Compute an "unshifted" commitment $U := p(\beta)G + \bar{p}(\beta)\gamma G$ and a "shifted" commitment $S := \beta^{D-d_i} p(\beta)G + \bar{p}^\star(\beta)\gamma G$. Finally, output $C := (U, S)$. Note that the U and S can be computed as linear combinations of terms in ck.

Open. On input the commitment key ck, a univariate polynomial p over the field $\mathbb{F}_q$, a commitment C to p, a degree bound d, an evaluation point z, and commitment randomness $(\omega, \omega^\star)$, $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Open}$ computes an opening proof π as follows.
Obtain the supported degree bounds $[d_i]_{i=1}^n$ from ck. If ω and $\omega^\star$ are not $\perp$, then obtain from them random univariate polynomials $\bar{p}$ and $\bar{p}^\star$ of degree $\deg(p)$; otherwise, set $\bar{p}$ and $\bar{p}^\star$ to be the zero polynomial. If $\deg(p) > d$ or $d \notin [d_i]_{i=1}^n$, abort. Compute the evaluation $v := p(z)$ and the opening challenge $\xi := \rho_0(\mathsf{rk}, d, C, z, v) \in \mathbb{F}_q$. Then, define the polynomial $p^\star(X) := X^{D-d}p(X) - X^{D-d}p(z)$, and compute a witness polynomial $w(X) := \frac{p(X)-p(z)}{X-z}$ for p, and a witness polynomial $w^\star(X) := X^{D-d}w(X)$ for $p^\star$. Combine these into $w' := w + \xi w^\star$
Compute witness polynomials $\bar{w}(X) := \frac{\bar{p}(X)-\bar{p}(z)}{X-z}$ for $\bar{p}$ and $\bar{w}^\star(X) := \frac{\bar{p}^\star(X)-\bar{p}^\star(z)}{X-z}$ for $\bar{p}^\star$. Combine these into $\bar{w}' := \bar{w} + \xi\bar{w}^\star$. Compute the evaluation $\bar{v} := \bar{w}'(z)$.
Compute $W := w'(\beta)G + \bar{w}'(\beta)\gamma G$, and output the evaluation proof $\pi := (W, \bar{v})$.

Check. On input the receiver key rk, a commitment C, a degree bound d, an evaluation point z, a claimed evaluation v, a evaluation proof π, $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Check}$ proceeds as follows.
If $d \notin \mathsf{rk}$, abort. Parse the commitment C as a tuple $(U, S) \in \mathbb{G}_1^2$. Parse the proof π as $(W, \bar{v}) \in \mathbb{G}_1 \times \mathbb{F}_q$. Compute the opening challenge $\xi := \rho_0(\mathsf{rk}, C, z, d, v)$. Compute the combined commitment $C' := U + \xi S$, and check the evaluation proof via the equality $e(C' - vG - v\beta^{D-d}G - \bar{v}\gamma G, H) = e(W, \beta H - zH)$.

Figure 1: The polynomial commitment scheme $\mathsf{PC}_{\mathsf{AGM}}$. Note that this scheme differs from the one described in [CHMMVW20] in that above $\mathsf{PC}_{\mathsf{AGM}}.\mathsf{Check}$ take as input the commitment, and generates an opening challenge by evaluating a random oracle on the commitment, whereas in [CHMMVW20], the opening challenge is provided as an explicit external input.

A Construction of $\mathsf{PC}_{\mathsf{DL}}$

We describe $\mathsf{PC}_{\mathsf{DL}}$, a polynomial commitment scheme based on the discrete logarithm problem that is inspired by several prior works [BCCGP16; BBBPWM18; WTSTW18]. We state the scheme in terms of the abstraction of *doubly-homomorphic commitments*, implicitly used in prior works. This section is organized as follows: in Appendix A.1 we define doubly-homomorphic commitments; in Appendix A.2 we provide the construction of $\mathsf{PC}_{\mathsf{DL}}$; and in Appendix A.3 we discuss the security of $\mathsf{PC}_{\mathsf{DL}}$.

A.1 Doubly-homomorphic commitments

A doubly-homomorphic commitment is a binding commitment on vectors of field elements that is linearly homomorphic with respect to both the commitment key and the committed elements.

In more detail, a *doubly-homomorphic commitment scheme* is a binding commitment scheme $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ that satisfies a particular bilinearity property.

- $\mathsf{CM.Setup}$, on input a security parameter λ (written in unary) and a message format $L \in \mathbb{N}$, samples public parameters pp of the form $\mathsf{pp} = (\langle\mathsf{group}\rangle, \mathbf{\Sigma})$ where $\langle\mathsf{group}\rangle = (\mathbb{G}, q, G)$ is a description of a group $\mathbb{G}$ of prime order q with generator G, and $\mathbf{\Sigma} = (G_1, \dots, G_L)$ is a list of generators for $\mathbb{G}$. The commitment space is $\mathcal{C}_{\mathsf{pp}} := \mathbb{G}$.
- $\mathsf{CM.Trim}$, on input public parameters pp and a trim specification $\ell \in \mathbb{N}$, selects $\mathsf{hk} := (G_1, \dots, G_\ell) \in \mathbb{G}^\ell$ from $\mathbf{\Sigma}$ and outputs $\mathsf{ck} := (\langle\mathsf{group}\rangle, \mathsf{hk})$. The message space is $\mathcal{M} := \mathbb{F}_q^\ell$.
- $\mathsf{CM.Commit}$, on input a commitment key ck and a message $m \in \mathcal{M}$, outputs a commitment $C \in \mathcal{C}_{\mathsf{pp}}$.

$\mathsf{CM.Commit}$ satisfies the following bilinearity property: if $\mathsf{ck} = (\langle\mathsf{group}\rangle, \mathsf{hk})$ and $\mathsf{ck}' = (\langle\mathsf{group}\rangle, \mathsf{hk}')$ are commitment keys with the same group description and $|\mathsf{hk}| = |\mathsf{hk}'|$ then

$$\mathsf{Commit}(\mathsf{ck}, \vec{a}) + \mathsf{Commit}(\mathsf{ck}, \vec{b}) = \mathsf{Commit}(\mathsf{ck}, \vec{a} + \vec{b}) \quad \text{and}$$
$$\mathsf{Commit}(\mathsf{ck}, \vec{a}) + \mathsf{Commit}(\mathsf{ck}', \vec{a}) = \mathsf{Commit}(\mathsf{ck} + \mathsf{ck}', \vec{a}) \ ,$$

where $\mathsf{ck} + \mathsf{ck}' := (\langle\mathsf{group}\rangle, \mathsf{hk} + \mathsf{hk}')$, and $\mathsf{hk} + \mathsf{hk}'$ denotes addition in the group $\mathbb{G}^\ell$.

Sometimes we use $\mathsf{CM.Commit}_{\mathsf{hk}}(\cdot)$ to denote $\mathsf{CM.Commit}(\mathsf{ck}, \cdot)$ when $\mathsf{ck} = (\langle\mathsf{group}\rangle, \mathsf{hk})$. Note that the foregoing bilinearity property implies that $\mathsf{CM.Commit}_{\mathsf{hk}}(\vec{a}) = \mathsf{CM.Commit}_{x^{-1}\cdot\mathsf{hk}}(x \cdot \vec{a})$ for all $x \in \mathbb{F}_q^*$. Note also that the commit algorithm $\mathsf{CM.Commit}$ is deterministic.

DL instantiation. Pedersen commitments [Ped92] are an example of a doubly-homomorphic commitment scheme. Below we describe only the Setup and Commit algorithms, as Trim is unchanged from above.

- $\mathsf{CM.Setup}$, on input a security parameter λ (written in unary) and a message format $L \in \mathbb{N}$, samples public parameters $\mathsf{pp} = (\langle\mathsf{group}\rangle, \mathbf{\Sigma})$ where $\langle\mathsf{group}\rangle = (\mathbb{G}, q, G) \leftarrow \mathsf{SampleGrp}^\rho(1^\lambda)$ is a description of a group of prime order, and $\mathbf{\Sigma} := (G_1, \dots, G_L) \leftarrow \rho(\langle\mathsf{group}\rangle, L)$ is a list of generators for $\mathbb{G}$.
- $\mathsf{CM.Commit}$, on input a commitment key $\mathsf{ck} = (\langle\mathsf{group}\rangle, \mathsf{hk})$ and a message $m \in \mathcal{M} = \mathbb{F}_q^\ell$, computes and outputs the commitment $C := \sum_{i=1}^{\ell} m_i G_i \in \mathbb{G}$.

The Pedersen commitment scheme is binding provided the discrete logarithm problem is hard for $\mathsf{SampleGrp}$.

A.2 Construction

The construction below uses a doubly-homomorphic commitment scheme $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$. All algorithms have oracle access to the same random oracle ρ_0. Below we assume without loss of generality that expressions of the form $d+1$ and $D+1$ are powers of 2. For a vector $\vec{a} \in S^n$ for any set S, below we use the notation $\mathbf{l}(\vec{a}) := (a_1, \ldots, a_{n/2})$ and $\mathbf{r}(\vec{a}) := (a_{n/2+1}, \ldots, a_n)$ to denote the left and right halves of $\vec{a}$.

Setup. On input security parameter λ (written in unary) and a maximum supported degree D, $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Setup}$ samples public parameters $\mathsf{pp} = (\langle\mathsf{group}\rangle = (\mathbb{G}, q, G), \mathbf{\Sigma}) \leftarrow \mathsf{CM}.\mathsf{Setup}^{\rho_0}(1^\lambda, D+1)$, samples a random generator $H \leftarrow \rho_0(\mathsf{pp})$ in the group $\mathbb{G}$, and outputs the public parameters $\mathsf{pp}_{\mathsf{PC}} := (\mathsf{pp}, H)$.

Trim. Given oracle access to the public parameters $\mathsf{pp}_{\mathsf{PC}}$ and a single degree bound d that is at most D, $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}$ specializes the public parameters for the degree bounds as follows: parse $\mathsf{pp}_{\mathsf{PC}}$ as (pp, H), compute $\mathsf{ck} \leftarrow \mathsf{CM}.\mathsf{Trim}(\mathsf{pp}, d+1)$, and output $(\mathsf{ck}_{\mathsf{PC}}, \mathsf{rk}_{\mathsf{PC}}) := \big((\mathsf{ck}, H), (\mathsf{ck}, H)\big)$. (We note that we consider the case where $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Trim}$ only receives a single degree bound d rather than a vector of degrees $[d_i]_{i=1}^n$. We omit details for this more general case.)

Commit. On input the commitment key $\mathsf{ck}_{\mathsf{PC}} = (\mathsf{ck}, H)$, a univariate polynomial p over the field $\mathbb{F}_q$, a degree bound d, and commitment randomness ω, $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Commit}$ outputs $C := \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, \vec{c})$, where $\vec{c} = (c_0, \ldots, c_d)$ are the coefficients of p.

Open. On input the commitment key $\mathsf{ck}_{\mathsf{PC}} = (\mathsf{ck}, H)$, a univariate polynomial $p(X) := \sum_{i=0}^{d} c_i X^i \in \mathbb{F}_q[X]$, a commitment C to p, a degree bound d, and an evaluation point z, $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Open}$ computes an evaluation proof π by using (a variant of) the inner product argument in [BCCGP16; BBBPWM18] as follows.

Compute the evaluation $v := p(z)$. Compute the 0-th challenge field element $\xi_0 := \rho_0(C, z, v) \in \mathbb{F}_q$, and use it to compute the group element $H' := \xi_0 H \in \mathbb{G}$. Initialize the following vectors:

$$\vec{c}_0 := (c_0, c_1, \ldots, c_d) \in \mathbb{F}_q^{d+1} \quad \text{and} \quad \vec{z}_0 := (1, z, \ldots, z^d) \in \mathbb{F}_q^{d+1} \quad \text{and} \quad \vec{G}_0 := (G_0, G_1, \ldots, G_d) \in \mathbb{G}^{d+1} \quad .$$

Next, for each $i \in [\log{(d+1)}]$, perform the following steps:

1. Setting $\mathbf{\Sigma}_{\mathrm{L}} := \mathbf{l}(\vec{G}_{i-1}) \,||\, H'$, compute the left commitment $L_i := \mathsf{CM}.\mathsf{Commit}_{\mathbf{\Sigma}_{\mathrm{L}}}\big(\mathbf{r}(\vec{c}_{i-1}) || \langle \mathbf{r}(\vec{c}_{i-1}), \mathbf{l}(\vec{z}_{i-1})\rangle\big)$.
2. Setting $\mathbf{\Sigma}_{\mathrm{R}} := \mathbf{r}(\vec{G}_{i-1}) \,||\, H'$, compute the right commitment $R_i := \mathsf{CM}.\mathsf{Commit}_{\mathbf{\Sigma}_{\mathrm{R}}}\big(\mathbf{l}(\vec{c}_{i-1}) || \langle \mathbf{l}(\vec{c}_{i-1}), \mathbf{r}(\vec{z}_{i-1})\rangle\big)$.
3. Generate the i-th challenge $\xi_i := \rho_0(\xi_{i-1}, L_i, R_i)$.
4. Construct the commitment key for the next round: $\vec{G}_i := \mathbf{l}(\vec{G}_{i-1}) + \xi_i \cdot \mathbf{r}(\vec{G}_{i-1})$.
5. Construct commitment inputs for the next round: $\vec{c}_i := \mathbf{l}(\vec{c}_{i-1}) + \xi_i^{-1} \cdot \mathbf{r}(\vec{c}_{i-1})$ and $\vec{z}_i := \mathbf{l}(\vec{z}_{i-1}) + \xi_i \cdot \mathbf{r}(\vec{z}_{i-1})$.

Finally, set $U := G_{\log{(d+1)}}$, $c := c_{\log{(d+1)}}$, and output the evaluation proof $\pi := (\vec{L}, \vec{R}, U, c)$.

Check. On input the receiver key $\mathsf{rk}_{\mathsf{PC}} = (\mathsf{ck}, H)$, a commitment C, a degree bound d, an evaluation point z, a claimed evaluation v, and an evaluation proof π, $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}$ verifies the evaluation proof by invoking the verifier of the inner product argument as follows.

1. Parse ck as $(\langle\mathsf{group}\rangle, \mathsf{hk})$.
2. Set $d' := |\mathsf{hk}| - 1$.
3. Set $\mathsf{rk}' := (\langle\mathsf{group}\rangle, H, d')$.
4. Check that $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}^{\rho_0}(\mathsf{rk}', C, d, z, v, \pi)$ accepts and outputs (h, U). (See Figure 2).
5. Check that $U = \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, \vec{h})$, where $\vec{h}$ is the coefficient vector of the polynomial h.

```
$\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}^{\rho_0}(\mathsf{rk}', C, d, z, v, \pi)$:
1. Parse $\mathsf{rk}'$ as $(\langle\mathsf{group}\rangle, H, d')$, and $\pi$ as $(\vec{L}, \vec{R}, U, c)$.
2. Check that $d = d'$.
3. Compute the 0-th challenge $\xi_0 := \rho_0(C, z, v)$, and set $H' := \xi_0 H$.
4. Compute the group element $C_0 := C + vH' \in \mathbb{G}$.
5. For each $i \in [\log(d+1)]$:
    (a) Generate the $i$-th challenge: $\xi_i := \rho_0(\xi_{i-1}, L_i, R_i)$.
    (b) Compute the $i$-th commitment: $C_i := \xi_i^{-1} L_i + C_{i-1} + \xi_i R_i$.
6. Define the univariate polynomial $h(X) := \prod_{i=0}^{\log(d+1)-1}(1 + \xi_{\log(d+1)-i} X^{2^i})$.
7. Compute the evaluation $v' := c \cdot h(z) \in \mathbb{F}_q$.
8. Check that $C_{\log(d+1)} = \mathsf{CM}.\mathsf{Commit}_{\mathbf{\Sigma}}(c \,||\, v')$, where $\mathbf{\Sigma} = (U \,||\, H')$.
9. Output $(h, U)$.
```

Figure 2: The subroutine $\mathsf{PC}_{\mathsf{DL}}.\mathsf{SuccinctCheck}$ that is invoked by $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}$ and by our accumulation scheme.

A.3 Security

It is conjectured that if $\mathsf{CM} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit})$ is a doubly-homomorphic commitment scheme, then $\mathsf{PC}_{\mathsf{DL}}$ constructed in Appendix A.2 is a polynomial commitment scheme in the random oracle model (Section 3.6). Below we summarize what is currently known about this conjecture.

Recall that in $\mathsf{PC}_{\mathsf{DL}}$, the algorithms $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Open}$ and $\mathsf{PC}_{\mathsf{DL}}.\mathsf{Check}$ are obtained by applying the Fiat–Shamir transformation [FS86] to a public-coin "inner-product" argument system. We first discuss the security of this interactive argument, and then discuss the security (in the random oracle model) of the non-interactive argument obtained after applying the Fiat–Shamir transformation.

Security of the interactive argument. A sequence of works [BCCGP16; BBBPWM18; WTSTW18; BMMV19] have shown how to construct various inner-product arguments of knowledge from any doubly-homomorphic commitment scheme by explicitly constructing an efficient rewinding extractor. The inner-product argument used in $\mathsf{PC}_{\mathsf{DL}}$ is a special case of an inner-product argument in [BMMV19], and thus inherits its knowledge soundness guarantees.

Security of the resulting non-interactive argument. It is known from folklore that applying the Fiat–Shamir transformation to a public-coin k-round interactive argument of knowledge with negligible soundness error yields a non-interactive argument of knowledge in the random-oracle model where the extractor $\mathcal{E}$ runs in time exponential in k. In more detail, to extract from an adversary that makes t queries to the random oracle, $\mathcal{E}$ runs in time $\mathsf{t}^{O(k)}$. In our setting, the inner-product argument has $k = O(\log d)$ rounds, which means that if we apply this folklore result, we would obtain an extractor that runs in superpolynomial (but subexponential) time $\mathsf{t}^{O(\log d)} = 2^{O(\log(\lambda)^2)}$. It remains an interesting open problem to construct an extractor that runs in polynomial time.

Acknowledgements

The authors thank William Lin for pointing out an error in a prior version of the construction of $\mathsf{PC}_{\mathsf{DL}}$. This research was supported in part by: the Berkeley Haas Blockchain Initiative and a donation from the Ethereum Foundation. Benedikt Bünz performed part of the work while visiting the Simons Institute for the Theory of Computing.

References

[AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104.

[BBBPWM18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BCCGP16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 327–357.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. 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.

[BCGGHJ17] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. K. Jakobsen. "Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability". In: *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security*. ASIACRYPT '17. 2017, pp. 336–365.

[BCRSVW19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. Full version available at `https://eprint.iacr.org/2018/828`. 2019, pp. 103–128.

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

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. 2014, pp. 276–294.

[BDFLSZ11] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. "Random Oracles in a Quantum World". In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '11. 2011, pp. 41–69.

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

[BMMV19] B. Bünz, M. Maller, P. Mishra, and N. Vesely. *Proofs for Inner Pairing Products and Applications*. Cryptology ePrint Archive, Report 2019/1177. 2019.

[BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. *Coda: Decentralized Cryptocurrency at Scale*. Cryptology ePrint Archive, Report 2020/352. 2020.

[BR93] M. Bellare and P. 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.

[CCW19] A. Chiesa, L. Chua, and M. Weidner. “On Cycles of Pairing-Friendly Elliptic Curves”. In: *SIAM Journal on Applied Algebra and Geometry* 3.2 (2019), pp. 175–192.

[CHMMVW20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. “Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS”. In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’20. 2020.

[CL20] A. Chiesa and S. Liu. “On the Impossibility of Probabilistic Proofs in Relativized Worlds”. In: *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*. ITCS ’20. 2020, 57:1–57:30.

[Co17] O(1) Labs. *Coda Cryptocurrency*. `https://codaprotocol.com/`. 2017.

[COS20] A. Chiesa, D. Ojha, and N. 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.

[CT10] A. Chiesa and E. 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] S. Chong, E. Tromer, and J. A. Vaughan. *Enforcing Language Semantics Using Proof-Carrying Data*. Cryptology ePrint Archive, Report 2013/513. 2013.

[CTV15] A. Chiesa, E. Tromer, and M. 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.

[FS86] A. Fiat and A. Shamir. “How to prove yourself: practical solutions to identification and signature problems”. In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO ’86. 1986, pp. 186–194.

[GW11] C. Gentry and D. 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.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019.

[Halo19] S. Bowe, J. Grigg, and D. Hopwood. *Halo*. 2019. URL: `https://github.com/ebfull/halo`.

[KB20] A. Kattis and J. Bonneau. *Proof of Necessary Work: Succinct State Verification with Fairness Guarantees*. Cryptology ePrint Archive, Report 2020/190. 2020.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’10. 2010, pp. 177–194.

[Lin03] Y. Lindell. “Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation”. In: *Journal of Cryptology* 16.3 (2003), pp. 143–184.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings”. In: *Proceedings of the 26th ACM Conference on Computer and Communications Security*. CCS ’19. 2019.

[NPR19] M. Naor, O. Paneth, and G. N. Rothblum. “Incrementally Verifiable Computation via Incremental PCPs”. In: *Proceedings of the 17th International Conference on the Theory of Cryptography*. TCC ’19. 2019, pp. 552–576.

[NT16] A. Naveh and E. 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.

[Pas03] R. Pass. “On Deniability in the Common Reference String and Random Oracle Model”. In: *Proceedings of the 23rd Annual International Cryptology Conference*. CRYPTO ’03. 2003, pp. 316–337.

[Ped92] T. P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *Proceedings of the 11th Annual International Cryptology Conference*. CRYPTO ’91. 1992, pp. 129–140.

[Pickles20] O(1) Labs. *Pickles*. URL: `https://github.com/o1-labs/marlin`.

[SS11] J. H. Silverman and K. E. Stange. “Amicable Pairs and Aliquot Cycles for Elliptic Curves”. In: *Experimental Mathematics* 20.3 (2011), pp. 329–357.

[Val08] P. 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.

[WTSTW18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P ’18. 2018, pp. 926–943.