On Succinct Non-Interactive Arguments in Relativized Worlds

Megan Chen
megchen@bu.edu
Boston University

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Nicholas Spooner
nicholas.spooner@warwick.ac.uk
University of Warwick

March 24, 2022

Abstract

Succinct non-interactive arguments of knowledge (SNARKs) are cryptographic proofs with strong efficiency properties. Applications of SNARKs often involve proving computations that include the SNARK verifier, a technique called recursive composition. Unfortunately, SNARKs with desirable features such as a transparent (public-coin) setup are known only in the random oracle model (ROM). In applications this oracle must be heuristically instantiated and used in a non-black-box way.

In this paper we identify a natural oracle model, the low-degree random oracle model, in which there exist transparent SNARKs for all NP computations *relative to this oracle*. Informally, letting $\mathcal{O}$ be a low-degree encoding of a random oracle, and assuming the existence of (standard-model) collision-resistant hash functions, there exist SNARKs relative to $\mathcal{O}$ for all languages in $\mathsf{NP}^{\mathcal{O}}$. Such a SNARK can directly prove a computation about its own verifier. This capability leads to proof-carrying data (PCD) in the oracle model $\mathcal{O}$ based solely on the existence of (standard-model) collision-resistant hash functions.

To analyze this model, we introduce a more general framework, the *linear code random oracle model* (LCROM). We show how to obtain SNARKs in the LCROM for computations that query the oracle, given an *accumulation scheme* for oracle queries in the LCROM. Then we construct such an accumulation scheme for the special case of a low degree random oracle.

Keywords: succinct non-interactive arguments; random oracle model; accumulation schemes

Contents

1 Introduction

Succinct non-interactive arguments (SNARGs) are short cryptographic proofs of NP computations. Many SNARG constructions also have the property of *succinct verification*: a SNARG proof can be verified faster than the original NP witness. An exciting application of this property, which motivates this paper, is building *incrementally-verifiable computation* (IVC) [Val08] and, more generally, *proof-carrying data* (PCD) [CT10].

IVC and PCD both address the problem of how untrusted provers can efficiently demonstrate the correctness of an *indefinite* (and, in the case of PCD, *distributed*) computation. The key efficiency requirement is that, after producing a proof for $t-1$ computation steps, the additional time required to prove t computation steps should be independent of t. Recursive proof composition is a general technique for "bootstrapping" a SNARG of knowledge (SNARK) to build IVC and PCD. The basic idea is relatively simple: to prove t steps of a computation, prove the NP statement "step t of the computation is correct, and there exists a proof of correctness for the previous $t - 1$ steps". Clearly this is an incremental proof: this statement depends only on a *single* step of the computation and the previous proof.

This technique is "recursive" in that it uses the SNARK to prove a statement *about its own verifier*. The succinct verification property ensures that the statement size can be bounded independently of t. This "non-black-box" use of the SNARK verifier leads to theoretical and practical issues, which we now discuss.

The first *efficient* approach to recursive composition [BCTV14] was based on a class of *pairing-based* SNARKs (which includes [Gro10; GGPR13; BCIOP13; Gro16; GKMMM18]) that are proven secure under knowledge assumptions or in the generic bilinear group model. This family of constructions yields extremely small proofs and highly efficient verification, but has two significant limitations.

First, all such constructions rely on a *secret setup*: sampling the structured reference string involves secret trapdoor values ("toxic waste") that can be used to attack the scheme. Hence the security of the system depends on these values being discarded — but this is difficult to ensure, even when accounting for the *cryptographic ceremonies* that researchers have designed to mitigate this sampling problem [BCGTV15; BGG17; BGM17; ABLSZ19]. Second, efficient recursion for these SNARKs relies on *pairing-friendly cycles* of elliptic curves. Only a single construction of such cycles is known, and that cycle's curves have undesirable algebraic properties that weaken their security.

A flurry of recent work has focused on developing new techniques that avoid both of these drawbacks [BGH19; COS20; BCMS20; BCLMS21; BDFG21; KST21]. However, all of the proposed schemes share an unfortunate detail: they rely on proving statements about *computations that query random oracles*.

To see how this arises, we consider as an example the construction of [COS20] (the other schemes work in different ways but the issue is the same in each case). This work presents a SNARK that is secure in the random oracle model, and then applies the recursive composition technique to obtain IVC and PCD. This entails producing a SNARK proof about a verifier that queries the random oracle. It was not previously known whether there exist SNARKs for such computations in the random oracle model. To avoid this issue, [COS20] perform a *heuristic step*: they assume that there exists a concrete hash function that, when used in place of the random oracle in their construction, yields a secure SNARK in the standard model.[1] SNARKs in the standard model can be (provably) recursively composed [BCCT12]. All of the cited constructions have similar heuristic steps.

This leads to a natural question: can we retain the benefits of these new constructions without this heuristic step? One approach would be to attempt to design better schemes in the standard or generic group models, but there has been little progress in this direction. In this work we propose a new approach: to build a SNARK in an oracle model that can prove statements *about its own oracle*. Such an oracle model admits a

[1]"Standard model" refers to the usual cryptographic setting without oracles (and parties may access a common reference string).

proof of security for recursive composition “within the model”, without any heuristic step.

On security in idealized models. Of course, the resulting system is proven secure only in an idealized model. Nonetheless, we argue that a black-box security proof in an oracle model has several advantages.

- *Flexibility.* The heuristic step in prior constructions requires instantiating the oracle via an efficient circuit. This rules out certain oracle instantiations, such as multi-party protocols or hardware tokens. In contrast, any instantiation of the oracle is possible if the oracle is used as a black box by the construction.
- *Efficiency.* The random oracle is typically instantiated in practice via a concrete “unpredictable” hash function such as SHA-3 or BLAKE. Unfortunately, producing SNARKs about these functions is expensive. This has motivated a line of work on hash functions designed to be more efficient for SNARKs [Alb+19a; Alb+19b; AABDS20; GKRRS21]. These new constructions have received far less scrutiny and cryptanalysis than standard hash functions. Our approach offers a possible alternative: if we can make only black-box use of the oracle, then we do not need to worry about an instantiation’s “SNARK-friendliness”.
- *Understanding security.* A proof in an idealized model can give more insight into the scheme’s security than one involving heuristic assumptions.
  - Heuristic assumptions make it harder to assess a scheme’s concrete security, while security proofs in idealized models (such as the random oracle model) often lead to precise security expressions.
  - Recursive composition in prior work involves a heuristic step in the “middle” of the security proof. Any change to the heuristic may require revisiting the security proof, or possibly even change the construction. In contrast, security proofs in idealized models provide “end-to-end” guarantees.
  - Choosing the correct heuristic analogue of a ROM security property is a balancing act between the requirements of the standard model proof and what can be justified by the idealized proof. There are *many* details, and the precise choice of assumption can affect the validity of the result; in the worst case, security flaws might be hidden in the heuristic step!

1.1 Our results

We identify a natural oracle model, the low-degree random oracle model, in which there exist SNARKs for all NP computations *relative to the same oracle*. That is, there is a natural distribution over oracles $\mathcal{O}$ such that, assuming the existence of (standard-model) collision-resistant hash functions, there exist SNARKs relative to $\mathcal{O}$ for all languages in $\mathsf{NP}^{\mathcal{O}}$. As a result, via recursive composition, we obtain *provably secure* IVC and PCD in the low-degree random oracle model, assuming the existence of collision-resistant hash functions.

We first introduce a more general model we call the *linear code random oracle model*, and investigate its structural and cryptographic properties. We then describe our construction of SNARKs in the low-degree random oracle model. This construction relies on an *accumulation scheme* for queries to the oracle, which we consider to be of independent interest. We now discuss these contributions in more detail.

(1) Linear code random oracles. We introduce the *linear code random oracle model* (LCROM) and study its structural and security properties. In the LCROM, all parties have oracle access to a codeword $\hat{\rho}$ sampled uniformly at random from a linear code $\mathcal{C} \subseteq (D \to \mathbb{F})$. We require that $\mathcal{C}$ have an associated efficient injective mapping $f \colon \{0,1\}^m \to D$ such that the restriction of $\hat{\rho}$ to $\mathrm{im}(f)$ is distributed as a uniformly random function $\mathrm{im}(f) \to \mathbb{F}$. This property ensures that we can query the random oracle by querying $\hat{\rho} \circ f$. We show that, while there exist exponential separations between the two models, all linear code random oracles are collision-resistant within $\mathrm{im}(f)$.

A *low-degree random oracle* is a linear code random oracle where $\mathcal{C}$ is a Reed-Muller code: the space of evaluation tables of multivariate polynomials of bounded (individual) degree. The polynomial structure of this code is important in several ways. First, these oracles can be efficiently simulated, which directly implies that standard model computational assumptions continue to hold in this model. Second, it allows for possible concrete instantiations via structured PRFs (see Section 2.1). Finally, our accumulation scheme (and hence also our SNARK) relies on polynomial interpolation to perform query reduction.

(2) SNARKs in the low-degree random oracle model. We show that, if collision-resistant hash functions exist (in the standard model), there exist SNARKs in the low-degree random oracle model (LDROM) that can prove NP computations relative to the same oracle. Our construction is *transparent*: the only setup required is a uniform reference string (specifically, the choice of a collision-resistant hash function).

Theorem 1. *There exists a transparent (zero-knowledge) SNARK in the LDROM for computations in the LDROM, assuming the existence of collision-resistant hash functions in the standard model.*

As a corollary of the theorem, obtained by adapting known techniques in [BCCT13; COS20], is an analogous result about PCD. The PCD construction inherits the transparency property of the SNARK.

Theorem 2. *There exists transparent (zero-knowledge) PCD in the LDROM (for computations in the LDROM), assuming the existence of collision-resistant hash functions in the standard model.*

(3) Building blocks. The result is obtained by combining a SNARK in the LDROM for NP computations *without* oracles and an *accumulation scheme* for oracle queries in the LDROM. We consider each of these to be notable contributions in their own right, and obtaining them requires developing new tools for analyzing security in the LDROM.

- *SNARK in the LDROM.* We show that Micali's construction of a SNARK in the random oracle model [Mic00; Val08] is secure in the LDROM. The central difficulty is to show that the construction satisfies a *knowledge* property, because the straightline knowledge extractor in [Val08] fails in the LDROM.
- *Accumulation for the LDROM.* An accumulation scheme for an oracle is a primitive that allows a verifier, with the help of an untrusted prover, to "store" oracle queries in an *accumulator* that can be efficiently checked later by a *decider*. For non-triviality, the verifier cannot query the oracle, and the size of the accumulator and the query complexity of the decider cannot depend on the number of accumulated queries. This is a variation on the notion of accumulation schemes for *predicates* as introduced by [BCMS20].

  We build an accumulation scheme for the low-degree random oracle. To do so, we build on a prior interactive query reduction protocol [KR08; CFGS22], which reduces any number of queries to a low-degree polynomial to a single query via interpolation. To obtain an accumulation scheme, we use a variant of the Fiat-Shamir transformation to make (a variant of) the query reduction protocol non-interactive; this introduces an additional oracle query which must also be accumulated.

1.2 Related work

PCD from signature cards. PCD was first defined and constructed in [CT10], which achieved PCD in a model where all parties have access to a *signed random oracle*. Such an oracle, on input x, samples a random answer y, generates a signature σ on (x, y) under a secret signing key embedded in the oracle, and outputs (y, σ). They showed that there exist SNARKs for all NP computations relative to this oracle: to prove that querying the oracle at x yields (y, σ), one proves that σ is a valid signature on (x, y). This allows the SNARK verifier to avoid querying the oracle entirely: instead, it can simply verify the signature.

Our work can be viewed as trading the *cryptographic* structure of [CT10] for *algebraic* structure. Both constructions yield similar security guarantees when instantiated with hardware tokens. However, we believe that our oracle is likely to be more amenable to concrete heuristic instantiations.

Random low-degree oracles and zero knowledge. Random polynomial oracles have been used to achieve zero knowledge in the context of interactive PCPs [BCFGRS17; CFS17] and multi-prover interactive proofs with soundness against entangled strategies [CFGS22]. In these protocols, the oracle is provided by the prover(s), as opposed to being a part of the model. [BCFGRS17] introduced the *succinct constraint detection* technique, which enables the design of efficient zero knowledge simulators for protocols with random polynomial oracles. We use the same technique in the present work to prove computational soundness.

Probabilistic proofs in relativized worlds. Chiesa and Liu [CL20] give several impossibility results for probabilistic proofs in relativized worlds. They show that there do not exist nontrivial PCPs (or IOPs) for computations relative to a variety of types of oracle, including random oracles, generic group oracles, and, most relevant to us, random low-degree polynomial oracles. It is argued that these separations give evidence that SNARKs relative to these oracles do not exist. We view this work as a counterpoint: the LDROM does not admit efficient PCPs but *does* admit SNARKs (under a cryptographic assumption). This suggests that the relationship between PCPs and SNARKs in relativized worlds is more complex than previously thought.

Algebrization. Low-degree random oracles are reminiscent of the algebrization framework [AW09], introduced to understand the algebraic techniques used to prove non-relativizing results like $\mathsf{PSPACE} \subseteq \mathsf{IP}$. An example of an "algebrizing" inclusion is that for every oracle θ, $\mathsf{PSPACE}^{\theta} \subseteq \mathsf{IP}^{\hat{\theta}}$, where $\hat{\theta}$ is any low-degree extension of θ. Note that the left side of the containment is relative to the original oracle, whereas the right side is relative to $\hat{\theta}$. This is incomparable to our setting: on the one hand, we want the same low-degree oracle on "both sides"; on the other hand, our results hold only for specific choices of (distributions over) oracles. Nonetheless, some of our techniques for analyzing linear code random oracles are inspired by the study of algebraic query complexity in [AW09].

Generic group model. Shoup [Sho97] introduced the generic group model (GGM), which represents a prime-order group via two oracles: a random injection $L\colon \mathbb{Z}_p \to \{0,1\}^k$ and a mapping from $(L(x), L(y))$ to $L(x+y)$. Boneh and Boyen [BB04] introduced the generic bilinear group model (GBM), which augments the GGM with an oracle providing a bilinear map. Zhandry and Zhang [ZZ21] show that a generic group or generic bilinear group oracle can be used to construct a random oracle by taking $\rho(x)$ to be the first (say) $k/2$ bits of $L(x)$.[2] A natural question, which we leave to future work, is whether there exist SNARKs in the GGM/GBM (based on standard assumptions) that can prove GGM/GBM computations.

[2] This result is sensitive to definition of a generic group: it is not known to hold for the definition of the GGM in [Mau05].

2 Techniques

We overview the main ideas behind our results. In Section 2.1 we introduce linear code random oracles, as well as the special case of low-degree random oracles. In Section 2.2 we describe an accumulation scheme for queries to a low-degree random oracle. Then we discuss technical tools that we use to establish the security of the accumulation scheme: in Section 2.3 a forking lemma for algorithms that query any linear code random oracle; and in Section 2.4 the hardness of a certain zero-finding game for low-degree random oracles. In Section 2.5 we describe our SNARK construction that relativizes in the low-degree random oracle model. In Section 2.6 we describe how we then achieve PCD in the low-degree random oracle model.

2.1 Linear code random oracles

We informally introduce the model of *linear code random oracles*, in which the oracle is a uniformly random codeword of a linear code. Then, we explain why collision resistance holds for the linear code random oracle. Finally we discuss a notable special case for this paper, low-degree random oracles.

Definition. The (standard) random oracle model considers the setting where every party (honest or malicious) is granted oracle access to the same random function. In this paper we consider the setting where every party is granted oracle access to the same random codeword sampled from a given linear code $\mathcal{C}$.

Recall that a linear code $\mathcal{C}$ is a subspace of the vector space of functions from a domain D to a field $\mathbb{F}$. A *linear code random oracle* is a codeword $\hat{\rho}\colon D \to \mathbb{F}$ chosen uniformly at random from the code $\mathcal{C}$.

For cryptographic applications, it will be helpful to impose an additional requirement on the code $\mathcal{C}$, which we call the "full-rank" condition. Intuitively, the full-rank condition ensures that there is an efficient embedding of the standard random oracle ρ in $\hat{\rho}$. More precisely, we require that $\mathcal{C}$ have an associated efficiently-computable injection $f\colon \{0,1\}^m \to D$ for some $m \in \mathbb{N}$ known as the *arity* of $(\mathcal{C}, f)$. We require that the restriction of $\mathcal{C}$ to the image of f has dimension 2^m (equivalently, $\mathcal{C}$ is systematic on $\operatorname{im}(f)$). This ensures that the function $\hat{\rho} \circ f\colon \{0,1\}^m \to \mathbb{F}$ is uniformly random when $\hat{\rho}\colon D \to \mathbb{F}$ is uniformly random in $\mathcal{C}$. In fact, we can view $\hat{\rho}$ as a (randomized) *systematic encoding* of the random oracle using $\mathcal{C}$.

Definition 2.1 (informal)**.** *A* **linear code random oracle** *$\hat{\rho}$ is an oracle drawn uniformly at random from a full-rank linear code $\mathcal{C}$. A $\mathcal{C}$-oracle algorithm is an algorithm with oracle access to $c \in \mathcal{C}$. A systematic oracle algorithm is an oracle algorithm making queries in $\{0,1\}^m$.*

A systematic oracle algorithm is a $\mathcal{C}$-oracle algorithm for any full-rank $\mathcal{C}$ via the associated injection. If $\mathcal{C}$ is the space of all functions $\{0,1\}^m \to \mathbb{F}$ (and f is the identity) then we recover the standard random oracle.[3]

Query complexity. We wish to understand what additional power is granted to the adversary by giving it access to an *encoding* of the random oracle.

A key difference between general linear code random oracles $\hat{\rho}$ and the standard random oracle ρ is that querying $\hat{\rho}$ outside of $\{0,1\}^m$ *can* yield information about evaluations inside $\{0,1\}^m$ that would otherwise be hard to obtain with a small number of queries. To illustrate this, consider the full-rank linear code $\mathcal{C} \subseteq \{0,1\}^m \cup \{\star\} \to \mathbb{F}$ (for a special symbol $\star$), consisting of all functions c such that $c(\star) = \sum_{a \in \{0,1\}^m} c(a)$. Clearly, with oracle access to $c \in \mathcal{C}$, one can determine $\sum_{a \in \{0,1\}^m} c(a)$ with a single query. On the other hand, it is not hard to show that no algorithm making fewer than 2^m queries to the *standard* random oracle can compute this quantity.

[3]Setting $\mathbb{F}$ to be the field of size 2^λ yields the more familiar definition of a random oracle mapping $\{0,1\}^m \to \{0,1\}^\lambda$; for generality we prefer not to fix the field choice.

Hence in general there is an exponential gap in query complexity between a linear code random oracle model and the standard random oracle model. What about for problems of cryptographic interest? We show that linear code random oracles are collision-resistant; in fact, having access to an encoding of the random oracle using a linear code provides *no advantage* in collision finding.

Lemma 2.2. *Given oracle access to $\hat{\rho} \leftarrow \mathcal{C}$, a t-query adversary finds $x, y \in \{0,1\}^m$ such that $\hat{\rho}(x) = \hat{\rho}(y)$ with probability at most $t^2/|\mathbb{F}|$.*

Note that for this lemma to hold it is crucial that $\mathcal{C}$ be full-rank and that x, y are restricted to $\{0,1\}^m$. Otherwise finding collisions may be very easy: consider the repetition code $\{(\alpha, \alpha) : \alpha \in \mathbb{F}\}$. We will make use of the collision-resistance property to prove security of our SNARK construction.

Low-degree random oracles. Our protocols rely on a special class of linear code random oracles that we call *low-degree random oracles*, obtained by choosing $\mathcal{C}$ to be the space $\mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ of m-variate polynomials over $\mathbb{F}$ of individual degree at most $d \in \mathbb{N}$, evaluated over the domain $\mathbb{F}^m$. This code is full-rank via the natural bijection between $\{0,1\}^m$ and $\{0_\mathbb{F}, 1_\mathbb{F}\}^m \subseteq \mathbb{F}^m$, because the latter is an interpolating set for the space of multilinear polynomials.

There is an efficient and perfect stateful simulation of all low-degree random oracles (for polynomial m, d) [BCFGRS17]. This implies that low-degree random oracles do not grant any additional computational power; in particular, it does not impact any "standard model" cryptography. We will use this fact in Section 2.2, where our accumulation scheme will require a standard model collision-resistant hash function.

We briefly discuss the possibility of instantiating this model. Given that the low-degree random oracle can be simulated efficiently, it can at least be implemented by a trusted party or hardware token. A candidate cryptographic instantiation is to obfuscate the algebraic pseudorandom functions of Benabbas, Gennaro and Vahlis [BGV11]. They construct a pseudorandom function $F \colon [d]^m \to \mathbb{G}$ (for group $\mathbb{G}$ of prime order p) that additionally allows, given the secret key, the efficient computation of group elements

$$P(x_1, \ldots, x_m) = \sum_{\vec{a} \in [d]^m} F(\vec{a}) \cdot x_1^{a_1} \cdots x_m^{a_m}$$

for $x_1, \ldots, x_m \in \mathbb{F}_p$. Note that if F is a random function then P is uniformly random in $\mathbb{F}_p^{\leq d}[X_1, \ldots, X_m]$.[4] Hence if F is pseudorandom, P is indistinguishable from a random polynomial.

Another natural instantiation strategy is to start with some "strong" hash function that we believe suffices to replace the random oracle in existing constructions, then arithmetize it to obtain a polynomial that extends the hash function. Of course, all of the security properties of the original hash function are maintained under this transformation. Unfortunately, directly arithmetizing a hash function yields a polynomial of quite high degree: approximately 2^D, where D is the circuit depth. The latter ranges from 25 to about 3000 for widely-used "strong" hash functions [Sma]. Since our SNARK construction involves proving a statement of size linear in the degree, this cost becomes prohibitive. While there exist techniques to reduce the degree of an arithmetization, these modify the function significantly so that it no longer behaves like a low-degree random oracle. We leave the question of instantiation via arithmetization to future work.

2.2 Accumulation scheme for low-degree random oracles

We describe our construction of an accumulation scheme for accumulating queries to the low-degree random oracle. First we review the notion of an accumulation scheme. Then we describe an interactive protocol based

[4]We view P as a polynomial over $\mathbb{F}_p$ via some isomorphism $\mathbb{G} \to \mathbb{F}_p$ (which need not be efficiently computable).

on the query-reduction technique for IPCPs in [KR08; CFGS22], and how this leads, via the Fiat–Shamir transformation, to our accumulation scheme. We conclude by discussing the challenges of proving security, which motivates developing the new tools that we introduce in subsequent sections.

Review: accumulation schemes. We review the notion of an accumulation scheme [BCMS20], stated for an arbitrary oracle distribution (rather than specifically for the random oracle model) and specialized to the accumulation of oracle queries rather than general predicates.

Definition 1. *An* **accumulation scheme** *for queries to an oracle θ (sampled according to some distribution) is a triple of algorithms* $(\mathrm{P}, \mathrm{V}, \mathrm{D})$*, known as the prover, verifier, and decider, that satisfies the following.*

- **Completeness:** *For all accumulators* acc *and query-answer pairs* (x, y)*, if* $\mathrm{D}^{\theta}(\mathsf{acc}) = 1$ *and* $\theta(x) = y$*, then for* $(\mathsf{acc}', \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\theta}(x, y, \mathsf{acc})$ *it holds that* $\mathrm{V}(x, y, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}}) = 1$ *and* $\mathrm{D}^{\theta}(\mathsf{acc}') = 1$.
- **Soundness:** *For efficiently generated accumulators* $\mathsf{acc}, \mathsf{acc}'$*, query-answer pairs* (x, y) *and accumulation proofs* π_{V}*, if* $\mathrm{D}^{\theta}(\mathsf{acc}') = 1$ *and* $\mathrm{V}(x, y, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}}) = 1$ *then* $\theta(x) = y$ *and* $\mathrm{D}^{\theta}(\mathsf{acc}) = 1$ *with all but negligible probability.*

The definition extends in a natural way to accumulate n query-answer pairs $[(x_i, y_i)]_{i=1}^{n}$ and ℓ old accumulators $[\mathsf{acc}_j]_{j=1}^{\ell}$ (see details in Section 3.4). For simplicity, below we present our accumulation scheme below for the case of general n and $\ell = 1$. Note that we do not grant the accumulation verifier V access to the oracle θ — this is a key requirement achieved by our construction and used in our applications.

Review: an interactive query reduction protocol. [KR08] describe an interactive query reduction protocol for interactive PCPs (IPCPs), a class of probabilistic proofs; subsequently, [CFGS22] adapted and simplified this protocol in their "low-degree" IPCP model. We recast this simplified protocol as an interactive query reduction protocol in the low-degree random oracle model.

Let $\hat{\rho} \colon \mathbb{F}^m \to \mathbb{F}$ be a polynomial of individual degree at most d (for now, $\hat{\rho}$ need not be random) and let $[\mathsf{q}_i]_{i=1}^{k} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a list of (alleged) query-answer pairs. The prover $\mathcal{P}$ wishes to convince the verifier $\mathcal{V}$ that $\hat{\rho}(x_i) = y_i$ for every $i \in [n]$, in a setting where both parties have oracle access to $\hat{\rho}$. While the verifier $\mathcal{V}$ can straightforwardly check this claim with n queries to $\hat{\rho}$ (and no help from the prover $\mathcal{P}$), the protocol below enables the verifier $\mathcal{V}$ to check this claim with a single query to $\hat{\rho}$ (up to some soundness error). Let $b_1, \ldots, b_n \in \mathbb{F}$ be a list of n distinct field elements, fixed in advance.

1. Both $\mathcal{P}$ and $\mathcal{V}$ compute the unique polynomial g of degree less than n such that $g(b_i) = x_i$ for all $i \in [n]$.
2. The prover $\mathcal{P}$ computes the composed polynomial $f := \hat{\rho} \circ g$, and sends $f \colon \mathbb{F} \to \mathbb{F}$ to the verifier.
3. The verifier $\mathcal{V}$ chooses a random $\beta \in \mathbb{F}$ and checks that $f(\beta) = \hat{\rho}(g(\beta))$ (by querying $\hat{\rho}$ at $g(\beta)$). Finally, the verifier $\mathcal{V}$ checks that $f(b_i) = y_i$ for every $i \in [n]$.

Observe that $\hat{\rho} \circ g$ is a univariate polynomial of degree less than nmd, and so the communication complexity of this protocol is $O(nmd \cdot \log |\mathbb{F}|)$ bits. If the prover is honest, then $\hat{\rho}(x_i) = \hat{\rho}(g(b_i)) = f(b_i) = y_i$ for every $i \in [n]$. On the other hand, if a cheating prover sends some polynomial $\tilde{f} \neq \hat{\rho} \circ g$, then $\tilde{f}(\beta) \neq \hat{\rho}(g(\beta))$ with probability $1 - \frac{nmd}{|\mathbb{F}|}$ over the choice of $\beta \in \mathbb{F}$, in which case the verifier rejects.

Our accumulation scheme. We construct an accumulation scheme for accumulating queries to the low-degree random oracle, based on the above query reduction protocol. At a high level, the accumulation prover and verifier engage in the above query reduction protocol, except that rather than directly checking that $\tilde{f}(\beta) = \hat{\rho}(g(\beta))$, the prover outputs the new accumulator $\mathsf{acc}' := (g(\beta), \tilde{f}(\beta))$ containing the query $g(\beta)$ and claimed answer $\tilde{f}(\beta)$. The decider can check that acc' contains a valid query-answer pair with a single query to the oracle $\hat{\rho}$. Notice that because an accumulator consists of a query-answer pair, we can simply include the old accumulator in the input to the query reduction protocol as an extra pair (x_{n+1}, y_{n+1}).

The challenge now is that an accumulation scheme is a non-interactive protocol while the above query-reduction protocol is interactive. Superficially, achieving non-interactivity appears to be a standard application of the Fiat–Shamir transform [FS86] because the interactive query-reduction protocol is public-coin. That is, since a low-degree random oracle $\hat{\rho}$ embeds a (standard) random oracle, the accumulation prover can use that random oracle to generate the verifier's random challenge β from the composed polynomial f as $\beta := \hat{\rho}(x_1, \ldots, x_{n+1}, f)$ (the embedding from binary strings into the domain of $\hat{\rho}$ is implicit). Note that we include $x_1, \ldots, x_{n+1}$ as input to $\hat{\rho}$ to achieve adaptive security. However, this setting is quite different from the familiar one for Fiat–Shamir: (i) the original interactive protocol already involves the oracle; (ii) the oracle is a random low-degree oracle rather than a standard random oracle; and (iii) the accumulation verifier cannot query $\hat{\rho}$! We discuss the latter point in more detail next, since resolving it requires modifying the construction; the other two points will be addressed later when we discuss the security proof.

Since the accumulation verifier is not allowed to query the oracle (this is the point of designing an accumulation scheme for oracle queries), it cannot check the query-answer pair $((x_1, \ldots, x_{n+1}, f), \beta)$ for correctness. The natural approach is to store the pair $((x_1, \ldots, x_{n+1}, f), \beta)$ in the accumulator, so that an accumulator contains an additional query-answer pair (x_{n+2}, y_{n+2}). Unfortunately, this results in the length of the Fiat–Shamir query, and hence the accumulator, increasing without bound (it will simply contain all accumulated queries). To address this, we rely on a succinct commitment scheme Commit, and derive the challenge as $\beta := \hat{\rho}(\mathsf{cm})$ for $\mathsf{cm} := \mathsf{Commit}(x_1, \ldots, x_{n+2}, f)$ instead. This ensures that the query to derive the challenge β does not grow in size with each accumulation.

These considerations lead us to design the following accumulation scheme.

- *Accumulation prover:* P receives as input an old accumulator acc and a list of query-answer pairs $[(x_i, y_i)]_{i=1}^{n}$, and outputs a new accumulator acc' and accumulation proof π_{V} computed as follows.
  1. *Query-answer list.* The old accumulator acc consists of two query-answer pairs, which we denote by (x_{n+1}, y_{n+1}) and (x_{n+2}, y_{n+2}). Set the query-answer list $Q := [(x_i, y_i)]_{i=1}^{n+2}$.
  2. *Interpolate queries.* Compute the unique polynomial $g \colon \mathbb{F} \to \mathbb{F}^m$ of degree less than $n+2$ such that $g(b_i) = x_i$ for all $i \in [n+2]$.
  3. *Compose polynomials.* Compute the polynomial $f := \hat{\rho} \circ g$.
  4. *Commit to polynomials.* Compute the commitment $\mathsf{cm} := \mathsf{Commit}(x_1, \ldots, x_{n+2}, f)$.
  5. *Fiat–Shamir challenge.* Compute the challenge $\beta := \hat{\rho}(\mathsf{cm}) \in \mathbb{F}$.
  6. *Output.* Output the new accumulator $\mathsf{acc}' := \big((\mathsf{cm}, \beta), (g(\beta), f(\beta))\big)$ and accumulation proof $\pi_{\mathrm{V}} := f$.
- *Accumulation verifier:* V receives as input $([\mathsf{q}_i]_{i=1}^{k}, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}})$, where $\mathsf{acc}' = \big((\mathsf{cm}', \beta), (x, y)\big)$, and works as follows. Compute the query-answer list Q as in Step 1 of the prover, the polynomial g as in Step 2 of the prover, and the commitment cm as in Step 4 of the prover. Then check that $\mathsf{cm}' = \mathsf{cm}$, $f(b_i) = y_i$ for every $i \in [n+2]$, $x = g(\beta)$, and $y = f(\beta)$.
- *Decider:* D checks that the input accumulator $\mathsf{acc} = \big((\mathsf{cm}, \beta), (x, y)\big)$ satisfies $y = \hat{\rho}(x)$ and $\beta = \hat{\rho}(\mathsf{cm})$.

Intuitively, the accumulation scheme is secure against efficient attackers because the binding property of the commitment scheme ensures that setting the challenge β to $\hat{\rho}(\mathsf{cm})$ is as good as $\hat{\rho}(x_1, \ldots, x_{n+2}, f)$, which gives a random challenge based on the prover's first message of the interactive query-reduction protocol.

We remark that the commitment scheme is the *only* part of the construction that uses cryptography outside of the oracle (i.e., is not information theoretic).

Zero-knowledge. Zero-knowledge for an accumulation scheme means that there exists a simulator that can sample a new accumulator $\mathsf{acc}_i = \big((\mathsf{cm}, \beta), (x, y)\big)$, without access to inputs $[\mathsf{q}_i]_{i=1}^{k}$ or an old accumulator acc_{i-1}. The accumulation scheme described above is *not* zero knowledge, because acc_i includes the value

$g(\beta)$, which depends on $[\mathsf{q}_i]_{i=1}^k$ and acc_{i-1}. To remedy this, we modify the accumulation scheme so that the prover P additionally accumulates a random query $x_{n+3} \in \mathbb{F}^m$. This ensures that $g(\beta)$ is uniformly random in $\mathbb{F}^m$.[5] We also require Commit to be a hiding commitment, and include the commitment randomness in π_V.

Finally, we note that if zero-knowledge is *not* required, then the commitment scheme used by the prover to obtain cm does not need to be hiding; in this case a collision-resistant hash function suffices.

How to prove security? Proving the security of the above accumulation scheme is not straightforward, despite the intuition that underlies its design. The main difficulty is that existing tools for establishing security work for standard random oracles rather than low-degree random oracles (e.g., security analyses of the Fiat–Shamir transform). We develop new tools to overcome the above problems. We formulate a *zero-finding game lemma for low-degree random oracles*, which shows that it is computationally hard for an adversary to find $f \not\equiv \hat{\rho} \circ g$ such that $f(z) = (\hat{\rho} \circ g)(z)$ for $z := \hat{\rho}(\mathsf{Commit}(x_1, \ldots, x_{n+2}, f))$.[6] To prove this lemma, we additionally prove a *forking lemma for algorithms that query any linear code random oracle*. We discuss these next: first our forking lemma in Section 2.3, and then our zero-finding game lemma in Section 2.4.

2.3 A forking lemma for linear code random oracles

We review a forking lemma for the standard random oracle, and then describe a new forking lemma for any linear code random oracle.

Review: a forking lemma for random oracles. In cryptography a forking lemma relates the probability of an adversary winning some game in multiple related executions, as a function of the adversary's winning probability in a single execution. Below we describe a forking lemma (based on [BN06]) that considers the setting of non-interactive protocols with forks of size 2 via algorithms that run in strict time.[7]

Let p be a predicate that captures the winning condition. Consider an adversary $\mathcal{A}$ that queries the random oracle t times and produces an output (x, o) such that $\mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1$ with probability δ, where $\mathsf{tr} = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ are the query-answer pairs of $\mathcal{A}$. We think of x as the adversary's "chosen" query point, and o as some additional input to the predicate p.

Now suppose that we additionally run $\mathcal{A}$ in a *forked* execution. Let $i := \mathsf{FP}_\mathcal{C}(\mathsf{tr}, x) \in [t]$ be the location of the query x in its query-answer list tr (abort if x does not appear in tr). Consider the *forking algorithm* Fork that, given access to $\mathcal{A}$ and input (tr, i), works as follows: (i) run $\mathcal{A}$ answering the first $i-1$ queries to the random oracle according to tr; (ii) answer subsequent queries uniformly at random; (iii) output the output (x', o') of $\mathcal{A}$ and the query transcript tr' induced by this execution of $\mathcal{A}$.

The forking lemma below gives a lower bound on the probability that: (1) $\mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1$ ($\mathcal{A}$ wins the original game); (2) $\mathsf{p}(x', \mathsf{o}', \mathsf{tr}') = 1$ ($\mathcal{A}$ wins the game in the forked execution); (3) $x = x'$ (the queries output by $\mathcal{A}$ in the two related executions are equal).[8]

Lemma 2.3. *Suppose that $\mathcal{A}$ is a t-query random oracle algorithm such that*

$$\delta := \Pr_{\rho}\left[\ \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1 \ \middle|\ (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^\rho\ \right] .$$

[5] Both [KR08] and [CFGS22] also add a random point to the curve. In contrast to our construction, in both cases this point is added by the *verifier* to ensure soundness: the query-reduction protocol is composed with a low-degree test, whose proximity guarantee holds only with respect to a uniform query. In our setting, the oracle is guaranteed to be low degree.

[6] Equivalently, the commitment can be written as $\mathsf{Commit}(g, f)$ since g is an equivalent representation of $x_1, \ldots, x_{n+2}$.

[7] The cryptography literature contains several types of forking lemmas, depending on aspects such as: (i) they apply to interactive protocols (without any oracles) or non-interactive protocols (in the random oracle model); (ii) they have a fork of size 2, or any size; (iii) the forking algorithm runs in strict time or expected time. The specific setting that we study is motivated by the present application, though we expect that the ideas for linear code random oracles that we introduce will extend to other settings as well.

[8] The lower bound that appears in [BN06] is $\frac{\delta^2}{t} - \mathrm{negl}(\lambda)$. The negligible term arises from an additional condition: $\mathsf{tr}(x) \neq \mathsf{tr}'(x)$. Our applications of the forking lemma refer directly to the distribution of $\mathsf{tr}'(x)$ and so we do not need this explicit condition.

Then

$$\Pr_{\rho}\left[\begin{array}{c|r} x = x' & (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\rho} \\ \wedge\ \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1 & i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) \\ \wedge\ \mathsf{p}(x', \mathsf{o}', \mathsf{tr}') = 1 & (x', \mathsf{o}', \mathsf{tr}') \leftarrow \mathsf{Fork}^{\mathcal{A}}(\mathsf{tr}, i) \end{array}\right] \geq \delta^2/t\ .$$

Extending the forking lemma to any linear code random oracle. When extending Lemma 2.3 to the linear code random oracle setting, a key difficulty is choosing the *fork point* i. The role of the fork point is to ensure that the answer to query x is resampled in the fork, while all prior queries stay the same. For the standard random oracle case, this point is easy to find: it is simply the index of the query x.

In contrast, for linear code random oracles, recall from Section 2.1 that $\mathcal{A}$ may learn the evaluation of unqueried points due to the structure of the code (e.g., if the code is locally decodable). This makes it unclear at which query $\mathcal{A}$ learns the value of $\hat{\rho}(x)$, or even if a single such query exists at all! We show that the linear structure of the code $\mathcal{C}$ implies that there does exist a query x_i at which $\mathcal{A}$ first learns the value of $\hat{\rho}$ at x (and before that $\mathcal{A}$ knows nothing about it); we define $\mathsf{FP}_{\mathcal{C}}(x, \mathsf{tr}) := i$.

Can $\mathsf{FP}_{\mathcal{C}}$ be efficiently computed? Note that $\mathcal{A}$ may try to obfuscate the fork point by trying to learn $\hat{\rho}(x)$ in some complicated way via the structure of the code. The problem of finding the fork point can be solved via *constraint detection* [BCFGRS17]: the fork point is the first query i at which there is a linear constraint over the set $\{\hat{\rho}(x_1), \dots, \hat{\rho}(x_i), \hat{\rho}(x)\}$. For low-degree random oracles, we can implement $\mathsf{FP}_{\mathcal{C}}$ efficiently using the efficient constraint detection algorithm for Reed–Muller codes of [BCFGRS17]. For many interesting linear codes, efficient constraint detection is an open problem.

To state the forking lemma requires one additional consideration. Strictly speaking, Lemma 2.3 holds only for adversaries that do not repeat queries (otherwise the adversary can distinguish a fork from the original execution), which is without loss of generality. In the LCROM, this restriction is not enough: the structure of the code might allow an adversary to learn a single query point in a variety of ways. Instead we must restrict our adversary further, so that it does not make any query *whose answer it can already infer*. Any adversary can be converted into a non-redundant adversary via constraint detection; hence we assume non-redundancy for efficient adversaries if $\mathcal{C}$ has efficient constraint detection. Due to this complication we prefer to state the non-redundancy condition explicitly in our forking lemma below.

Lemma 2.4. *Let $\hat{\rho} \leftarrow \mathcal{C}$ be a linear code random oracle. Suppose that $\mathcal{A}$ is a t-query non-redundant $\mathcal{C}$-oracle algorithm such that*

$$\delta := \Pr_{\hat{\rho}}\left[\ \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1\ \middle|\ (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}}\ \right]\ .$$

Then

$$\Pr_{\hat{\rho}}\left[\begin{array}{c|r} x = x' \in \{0,1\}^m & (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}} \\ \wedge\ \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1 & i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) \\ \wedge\ \mathsf{p}(x', \mathsf{o}', \mathsf{tr}') = 1 & (x', \mathsf{o}', \mathsf{tr}') \leftarrow \mathsf{Fork}(\mathsf{tr}, i) \end{array}\right] \geq \delta^2/t\ .$$

Note that the Fork algorithm here is *the same* as in Lemma 2.3, except that we choose the random answers from the alphabet $\mathbb{F}$ of $\mathcal{C}$.

2.4 A zero-finding game for low-degree random oracles

We review the zero-finding game for *standard* random oracles in [BCMS20], and then we describe our variant for *low-degree* random oracles.

Review: a zero-finding game for the standard random oracle. Bünz, Chiesa, Mishra, and Spooner [BCMS20] consider a *zero-finding game* where an efficient adversary $\mathcal{A}$ with query access to a random oracle

ρ is challenged to output a commitment cm and a non-zero polynomial $f \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ such that cm is a commitment to f and $f(\rho(\mathsf{cm})) = 0$. Intuitively, the binding property of the commitment scheme implies that $\mathcal{A}$ is unlikely to win the game because the polynomial f is "fixed" before $\mathcal{A}$ learns the value of $\rho(\mathsf{cm})$, and so the probability that this value is a zero of f is small by the Schwartz–Zippel lemma. Indeed, [BCMS20] shows that every efficient adversary $\mathcal{A}$ that makes at most t queries to ρ wins with probability at most negligibly more than $\sqrt{(t+1)md/|\mathbb{F}|}$ (note that md is the total degree of f).

Their proof is based on the forking lemma for the standard random oracle (Lemma 2.3), as we now sketch. We define the forking lemma predicate $\mathsf{p}(x, f, \mathsf{tr})$ to be satisfied if and only if: (i) $f \not\equiv 0$; (ii) $f(\mathsf{tr}(x)) \neq 0$; and (iii) x is a commitment to f. By increasing the query complexity of $\mathcal{A}$ to $t+1$ we can ensure that x is in the support of tr. It follows that if $\mathcal{A}$ wins the zero-finding game with probability δ, then it satisfies the premise of Lemma 2.3, and so the forking experiment succeeds with probability at least $\delta^2/(t+1)$.

Suppose that the forking experiment succeeds: we obtain $(x, f, f', \mathsf{tr}, \mathsf{tr}')$ such that f and f' are non-zero polynomials that are both valid openings of the same commitment x, and it holds that $f(\mathsf{tr}(x)) \neq 0$ and $f'(\mathsf{tr}'(x)) \neq 0$. In the forked execution, $\mathsf{tr}'(x)$ is sampled uniformly at random independently of f, and so $\Pr[f(\mathsf{tr}'(x)) = 0] \leq md/|\mathbb{F}|$. If $f = f'$ then the probability that $\mathsf{p}(x, f', \mathsf{tr}')$ is satisfied in this case is at most $md/|\mathbb{F}|$; if instead $f \neq f'$, then we break the commitment scheme, which can occur with at most negligible probability. Hence $\delta^2/(t+1) \leq md/|\mathbb{F}| + \operatorname{negl}(\lambda)$; rearranging yields the bound.

A zero-finding game for low-degree random oracles. We require a variant of the zero-finding game to prove security of our accumulation scheme from Section 2.2. In more detail, we want to bound the probability that an efficient adversary with query access to a low-degree random oracle $\hat{\rho}$ outputs a commitment cm and two polynomials f, g such that: (1) cm is a commitment to (f, g); (2) $f \not\equiv \hat{\rho} \circ g$; and (3) the oracle answer $z := \hat{\rho}(\mathsf{cm})$ satisfies $f(z) = (\hat{\rho} \circ g)(z)$. For this we prove the following lemma.

Lemma 2.5 (informal)**.** *Let* Commit *be a binding commitment scheme, and let* $\hat{\rho}$ *be a low-degree random oracle defined over a field* $\mathbb{F}$. *Then for every efficient* t*-query oracle algorithm* $\mathcal{A}$*,*

$$\Pr_{\hat{\rho}}\left[\begin{array}{c} \mathsf{cm} = \mathsf{Commit}(f, g; \omega) \\ \wedge\ f(X) \not\equiv \hat{\rho}(g(X)) \\ \wedge\ f(\hat{\rho}(\mathsf{cm})) = \hat{\rho}(g(\hat{\rho}(\mathsf{cm}))) \end{array} \middle|\ (\mathsf{cm}, f, g, \omega) \leftarrow \mathcal{A}^{\hat{\rho}} \right] = O\left(\sqrt{t \cdot \frac{m \cdot \deg(\hat{\rho}) \cdot \deg(g)}{|\mathbb{F}|}}\right) + \operatorname{negl}(\lambda) \ .$$

A key difference of our zero-finding game compared to the one in [BCMS20] (besides the different oracle models) is in the polynomial equation: the polynomial equation not only involves a random evaluation at a point $\hat{\rho}(\mathsf{cm})$ determined by the oracle $\hat{\rho}$ (analogous to $\rho(\mathsf{cm})$ before) but it also involves the low-degree oracle $\hat{\rho}$ itself as a polynomial. This causes significant complications to the security proof, as we discuss next.

A natural approach to prove Lemma 2.5 would be to adapt the proof of the zero-finding game in [BCMS20], using our forking lemma for linear code random oracles (Lemma 2.4) rather than the standard forking lemma. Recall that the non-redundancy requirement is satisfied without loss of generality because low-degree random oracles have efficient constraint detection. To invoke the lemma we must choose a forking predicate $\mathsf{p}(\mathsf{cm}, (f, g, \omega), \mathsf{tr})$ that captures the three conditions on the left. Similarly to before, one condition is "$\mathsf{cm} = \mathsf{Commit}(f, g; \omega)$". However, translating the other two conditions to be compatible with the forking lemma is much more involved, and requires overcoming several challenges.

Since the predicate p cannot query the oracle $\hat{\rho}$, these conditions must be converted into a statement about the query transcript tr. In the [BCMS20] proof, the corresponding condition becomes $f(\mathsf{tr}(\mathsf{cm})) = 0$. This is justified in their setting because adversaries that do not query the oracle at cm cannot win the game. This is *not* true for low-degree oracles! An adversary can, for example, learn $\hat{\rho}(\mathsf{cm})$ by querying points on a curve that passes through cm. To handle this issue, we use the structure of the linear code to define a partial function $\overline{\mathsf{tr}}$ which captures all of the information that the adversary knows about $\hat{\rho}$ given the points it has queried. We show that if $\overline{\mathsf{tr}}(x) = \bot$, no adversary can determine $\hat{\rho}(x)$ from tr better than guessing.

We summarize the above discussion by describing the forking lemma predicate explicitly. On input query point cm, auxiliary input (f, g, ω), and query transcript tr, p accepts if the following three conditions hold:

(1) $\mathsf{cm} = \mathsf{Commit}(f, g; \omega)$,
(2) $f \not\equiv \overline{\mathsf{tr}} \circ g$, and
(3) $f(\overline{\mathsf{tr}}(\mathsf{cm})) = \overline{\mathsf{tr}}(g(\overline{\mathsf{tr}}(\mathsf{cm})))$.

Note that since $\overline{\mathsf{tr}}$ is a partial function, so is $\overline{\mathsf{tr}} \circ g$, in general. If $\overline{\mathsf{tr}} \circ g$ is not total then condition (2) holds immediately, since f is total.

We now continue following the template of [BCMS20]. Let $\mathcal{A}$ be an adversary winning the Lemma 2.5 game with probability δ. Invoking our forking lemma (Lemma 2.4), we obtain, with probability δ^2/t, $(\mathsf{cm}, (f, g, \omega), (f', g', \omega'), \mathsf{tr}, \mathsf{tr}')$, where p holds for $(\mathsf{cm}, (f, g, \omega), \mathsf{tr})$ and $(\mathsf{cm}, (f', g', \omega'), \mathsf{tr}')$. Similarly to before, by the binding property of Commit we can focus on the case where $(f, g) = (f', g')$. The analogous next step would be to conclude by applying Schwartz–Zippel to the polynomial $f - \overline{\mathsf{tr}} \circ g$. Here, however, we run into another issue: since this polynomial depends on tr, it may differ between the two forks!

To resolve this, we use a slightly different polynomial. Denote by $\mathsf{tr}|_{i-1}$ the truncation of tr to the first $i - 1$ entries, where i is the fork point. By construction, $\mathsf{tr}'|_{i-1} = \mathsf{tr}|_{i-1}$. Hence $\overline{\mathsf{tr}|_{i-1}} \circ g$ is the same function in both forks. Of course, it may be that $\overline{\mathsf{tr}|_{i-1}} \circ g$ is not total (even if $\overline{\mathsf{tr}} \circ g$ is). We show that in this case the adversary is very unlikely to win: in particular, it can be shown that $\overline{\mathsf{tr}|_{i-1}} \circ g$ is not total only if $\overline{\mathsf{tr}|_{i-1}}(g(\alpha)) = \bot$ for all but $m \cdot \deg(\hat{\rho}) \cdot \deg(g)$ choices of $\alpha \in \mathbb{F}$. But then since $\overline{\mathsf{tr}'}(\mathsf{cm})$ is chosen uniformly at random independently of $\mathsf{tr}|_{i-1}$, in this case condition (3) holds with probability at most $m \cdot \deg(\hat{\rho}) \cdot \deg(g)/|\mathbb{F}|$.

In the case that $\overline{\mathsf{tr}|_{i-1}} \circ g$ is total, we can now apply Schwartz–Zippel to the polynomial $f - \overline{\mathsf{tr}|_{i-1}} \circ g$, which concludes the proof by rearranging to bound δ as before.

2.5 SNARKs for oracle computations

We outline the proof of Theorem 1, which provides a transparent (zero-knowledge) SNARK in the LDROM for computations in the LDROM, assuming the existence of collision-resistant hash functions in the standard model. Technical details about this result are in Section 8.

The proof is in two steps: first we generically construct a SNARK relative to a given oracle model from a SNARK for non-oracle computations and an accumulation scheme for queries to oracles in that model; then we instantiate this generic construction specifically for the low-degree random oracle model.

2.5.1 Step 1: a generic construction

Let $\theta \leftarrow \mathcal{O}$ be any oracle (for now not necessarily low-degree). Informally, we can view a computation that has oracle access to θ as made of two parts that share a common (untrusted) witness of query-answer pairs: (i) a non-oracle computation where oracle answers are read from the witness; and (ii) a computation that checks all query-answer pairs for consistency with the oracle θ. We prove the former using a SNARK $\mathsf{ARG}_{\mathsf{in}} = (\mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ for non-oracle computations (whose security holds in a model where parties have access to θ), and the latter via an accumulation scheme $\mathsf{AS} = (\mathrm{P}, \mathrm{V}, \mathrm{D})$ for queries to θ. We then combine these two components to obtain a SNARK for computations that query θ in a model where parties have access to θ.

In more detail, let $\mathcal{R}^\theta := \{(\mathbb{x}, \mathbb{w}) \colon M^\theta(\mathbb{x}, \mathbb{w}) = 1\} \in \mathsf{NP}^\theta$ be the target oracle relation, described by a polynomial-time oracle Turing machine M. We define the following associated non-oracle relation:

$$\mathcal{R}_{\mathsf{in}} := \Big\{ \big((\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})\big) \colon M_\circ(\mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1 \wedge \mathrm{V}(\mathsf{tr}, \bot, \mathsf{acc}, \pi_{\mathrm{V}}) = 1 \Big\} \in \mathsf{NP} \enspace ,$$

where $M_\circ$ works like M except that its oracle queries are answered using the query-answer transcript tr, V is the accumulation verifier, $\bot$ denotes an empty (old) accumulator, and π_{V} is the accumulation scheme's verification proof. Intuitively, if $(\mathbb{x}, \mathsf{acc}) \in \mathcal{L}(\mathcal{R}_{\mathsf{in}})$ and acc is a valid accumulator, then for tr consistent with θ and some witness $\mathbb{w}$, $M_\circ(\mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1$. This implies that $M^\theta(\mathbb{x}, \mathbb{w}) = 1$, and so $\mathbb{x} \in \mathcal{L}(\mathcal{R}^\theta)$.

We construct the SNARK $\mathsf{ARG}_{\mathsf{out}} = (\mathcal{P}_{\mathsf{out}}, \mathcal{V}_{\mathsf{out}})$ for $\mathcal{R}^\theta$ from a SNARK $\mathsf{ARG}_{\mathsf{in}}$ for $\mathcal{R}_{\mathsf{in}}$ and an accumulation scheme AS for queries to θ, as follows.

- $\mathcal{P}^\theta_{\mathsf{out}}(\mathbb{x}, \mathbb{w})$:
  1. Simulate $M^\theta(\mathbb{x}, \mathbb{w})$ and record the query-answer transcript tr.
  2. Accumulate the queries in tr using the accumulation prover: $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^\theta(\mathsf{tr}, \bot)$.
  3. Run the SNARK prover to obtain a proof that $(\mathbb{x}, \mathsf{acc}) \in \mathcal{R}_{\mathsf{in}}$: $\pi_{\mathsf{in}} \leftarrow \mathcal{P}^\theta_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}}))$.
  4. Output $\pi_{\mathsf{out}} := (\mathsf{acc}, \pi_{\mathsf{in}})$.
- $\mathcal{V}^\theta_{\mathsf{out}}(\mathbb{x}, \pi_{\mathsf{out}} = (\mathsf{acc}, \pi_{\mathsf{in}}))$:
  1. Check that the accumulation decider accepts: $\mathrm{D}^\theta(\mathsf{acc}) = 1$.
  2. Check that the SNARK verifier accepts: $\mathcal{V}^\theta_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$.

For simplicity, we have omitted public parameters from the above description. If parameter generation is transparent for $\mathsf{ARG}_{\mathsf{in}}$ and AS (as in our instantiation in Section 8), then parameter generation is transparent for $\mathsf{ARG}_{\mathsf{out}}$ as well.

We briefly comment on the knowledge soundness and zero knowledge of $\mathsf{ARG}_{\mathsf{out}}$.

Knowledge soundness. Consider a malicious prover $\tilde{\mathcal{P}}^\theta_{\mathsf{out}}$ that outputs an instance $\mathbb{x}$ and a proof $(\mathsf{acc}, \pi_{\mathsf{in}})$ that $\mathcal{V}^\theta_{\mathsf{out}}$ accepts (which means that $\mathcal{V}^\theta_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$ and $\mathrm{D}^\theta(\mathsf{acc}) = 1$). By the knowledge guarantee of the SNARK $\mathsf{ARG}_{\mathsf{in}}$, we can extract a witness $(\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})$ for the instance $(\mathbb{x}, \mathsf{acc})$. If tr is not consistent with θ, then the soundness of the accumulation scheme implies that $\mathrm{D}^\theta(\mathsf{acc}) = 1$ with at most negligible probability. Hence tr is consistent with θ, which implies that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}^\theta$ by definition of $\mathcal{R}_{\mathsf{in}}$.[9]

Zero knowledge. We prove that $\mathsf{ARG}_{\mathsf{out}}$ is zero knowledge, assuming $\mathsf{ARG}_{\mathsf{in}}$ and AS are both zero knowledge. This requires a careful consideration of zero knowledge definitions for both $\mathsf{ARG}_{\mathsf{in}}$ and AS. In particular, zero knowledge simulators for zkSNARKs often need to *program* the oracle; this is the case, for example, in Micali's zkSNARK construction. This causes problems when composing with an accumulation scheme for the oracle, because whether the statement "$\theta(x) = y$" is actually *true* may be affected by programming. As such, our definition of zero knowledge for SNARKs allows the simulator to program the oracle *only after the statement has been chosen*. Implicitly this prevents the simulator from changing the truth value of a statement, because this would be easily detectable by a distinguisher. We show that Micali's SNARK (which we discuss next in Section 2.5.2) satisfies this zero knowledge definition.

2.5.2 Step 2: instantiating the building blocks

We instantiate the above generic construction of $\mathsf{ARG}_{\mathsf{out}}$ in the case where θ is a low-degree random oracle.

[9]Interestingly, we do not know how to merely establish soundness of $\mathsf{ARG}_{\mathsf{out}}$ while only relying on soundness of $\mathsf{ARG}_{\mathsf{in}}$. The difficulty is that, if we cannot efficiently extract a witness $(\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})$ for the instance $(\mathbb{x}, \mathsf{acc})$, then we cannot invoke AS's soundness to detect when the adversary uses a tr that is inconsistent with θ. We leave investigating this implication as an open question.

- We set the accumulation scheme AS to be our construction for low-degree oracle queries from Section 2.2.
- We set the SNARK $\mathsf{ARG}_{\mathsf{in}}$ to be the Micali construction [Mic00], brought over to the LDROM.

In this instantiation, $\mathsf{ARG}_{\mathsf{in}}$ has no public parameters and AS's public parameters are those of the underlying collision resistant hash function. In particular, parameter generation for $\mathsf{ARG}_{\mathsf{out}}$ can be transparent.

The main technical challenge here is *proving that the Micali construction is a SNARK in the low-degree random oracle model*. We describe the problem that arises and how we solve it.

Problem. The Micali construction, when instantiated with a PCP of knowledge, is a SNARG of knowledge in the random oracle model via a (straightline) extractor [Val08]. From a syntactic perspective, the Micali construction naturally carries over to the low-degree random oracle model. However the proof of knowledge soundness does not because the extractor in [Val08] fails in the LDROM. Briefly, the extractor in [Val08] needs all random oracle entries that the adversary knows, which in the ROM coincide with the adversary's query-answer pairs. However, in the LDROM, the adversary may learn the answers to unqueried entries (e.g., via interpolation), so the extractor in [Val08] does not work as-is. In fact, we do not know how to efficiently recover the set of "learned" answers, given the adversary's query-answer transcript; thus we cannot implement the [Val08] extractor in the LDROM.

Solution. We leverage a *rewinding* extractor that uses a forking lemma, and is inspired by the extractor for Kilian's protocol [Kil92; BG08]. The extractor forks the malicious prover's oracle transcript polynomially-many times, in order to obtain a large number of (accepting) PCP verifier views; to achieve this we extend our 2-fork forking lemma in the LCROM (Lemma 2.4) to support polynomially-many forks. Next, the extractor constructs a PCP string by combining these views, aborting if any of them are inconsistent. Finally, it gives this constructed PCP string to the PCP's knowledge extractor to recover the witness.

This rewinding extractor runs in *expected* polynomial time, as the forking algorithm samples oracle transcripts until a sufficient number satisfy the forking predicate. If the extractor obtains inconsistent views, this can be used to obtain a collision in the oracle, and so by collision-resistance in the LCROM the extractor aborts with only negligible probability.

To prove knowledge soundness, we argue that the extracted PCP string convinces the PCP verifier with high-enough probability for the PCP extractor to succeed. Suppose that the malicious prover causes the argument verifier to accept with probability at least ε, and fix any "bad" PCP string $\tilde{\Pi}$ (i.e., it causes the PCP verifier to accept with probability less than $\varepsilon/3$). If the extractor outputs $\tilde{\Pi}$, it must be that the malicious prover answers consistently with $\tilde{\Pi}$; in particular, the probability that the argument verifier accepts is less than $\varepsilon/3$. Thus the running time of the extractor when it outputs $\tilde{\Pi}$ is more than, say, $2N/\varepsilon$ with overwhelming probability for N large enough. By a union bound over all possible PCP strings, the running time of the extractor when it outputs *any* bad PCP string is more than $2N/\varepsilon$ with high probability. On the other hand, the argument verifier accepts with probability at least ε, so the running time of the extractor is less than $2N/\varepsilon$ with high probability. Thus the probability that the extractor outputs a bad PCP string is exponentially small.

Beyond the LDROM. Our proof of security for Micali's SNARK works in every linear code random oracle with an efficient constraint detection algorithm; the LDROM is one example. Indeed, all linear code random oracles are collision-resistant (Lemma 2.2), and our forking lemma supports every linear code random oracle with efficient constraint detection (Lemma 2.4). It follows that, in order to build SNARKs for other linear code oracle computations, it suffices to design an accumulation scheme for oracle queries and an efficient constraint detection algorithm for the code.

Zero knowledge. In the ROM, Micali's construction can be adapted to achieve (statistical) zero knowledge when the underlying PCP is honest-verifier zero knowledge [BCS16]. This remains the case, via essentially the same adaptation, in any LCROM with efficient constraint detection and, in particular, in the LDROM. The simulator relies on constraint detection to "program" the oracle (after the adversary has chosen the statement, as discussed in Section 2.5.1); this is more complex than in the ROM case because of the code structure.

2.6 Proof-carrying data from relativized SNARKs

We outline the proof of Theorem 2, which provides transparent PCD in the LDROM (for computations in the LDROM), assuming the existence of collision-resistant hash functions in the standard model. Technical details about this result are in Section 9.

The theorem follows directly by applying a generic transformation, which we now describe, to the SNARK in the LDROM for computations in the LDROM described in Section 2.5 (and provided in Theorem 1).

The generic transformation is a direct adaptation to the relativized setting of the PCD construction based on recursive proof composition of SNARKs [BCCT13; COS20]. Informally, we transform a SNARK relative to $\mathcal{O}$ for $\mathsf{NP}^{\mathcal{O}}$ into a PCD scheme relative to $\mathcal{O}$ for NP (in fact, even for $\mathsf{NP}^{\mathcal{O}}$). If the SNARK is zero-knowledge, then the PCD scheme is also zero knowledge.

The PCD construction is as in prior work except that we give all PCD algorithms access to the oracle (due to the underlying SNARK) and allow oracle calls in the recursively-proved statement (so to express the SNARK verifier's computation). In particular, we do not need to heuristically instantiate any oracles for the recursive composition, because the SNARK is capable of proving oracle computations.

The security analysis involves some delicate differences to prior work, because the knowledge property that we obtain is slightly different from those that were previously hypothesized from knowledge assumptions or heuristics. In prior work the SNARK's knowledge extractor is *assumed* to run in (non-uniform) strict polynomial time, which in turn means that the resulting PCD knowledge extractor also runs in (non-uniform) strict polynomial time. In contrast, the SNARK knowledge extractor that we construct in this work runs in expected polynomial time (see Section 2.5), and in turn so does the PCD knowledge extractor.

In light of the above considerations, in Section 9, we explicitly describe a PCD construction and provide a full security analysis for our setting. In particular, unlike in prior work, we are able to provide a concrete expression for the (expected) running time of the PCD extractor, which determines the tradeoff between recursion depth and security.

Note that there are two complementary PCD constructions based on recursive composition of SNARKs:[10] fully-succinct PCD from fully-succinct SNARKs [BCCT13]; and preprocessing PCD from preprocessing SNARKs [COS20]. For concreteness, in the technical section we choose to describe the preprocessing case. The description can be straightforwardly ported to the fully-succinct case.

[10]There is no need to consider accumulation-based constructions of PCD [BGH19; BCMS20; BCLMS21; BDFG21; KST21] to establish our results because we obtain a SNARK whose efficiency properties suffice for "plain" recursive composition.

3 Preliminaries

3.1 Notations

We define $[n] := \{1, \dots, n\}$. We use $\mathbb{F}^{\leq d}[X_1, \dots, X_m]$ to denote to the set of m-variate polynomials of individual degree at most d with coefficients in $\mathbb{F}$; we write $\deg(\cdot)$ to denote individual degree.

Functions. We use $(X \to Y)$ to denote the set of all functions $\{f : X \to Y\}$. We denote by $f : X \rightharpoonup Y$ a partial function from X to Y. We use $\mathrm{im}(f)$ to denote the image of a function f. For a set $S \subseteq X$ we denote by $f|_S : S \to Y$ the restriction of f to S.

Distributions. For finite set X, we write $x \leftarrow X$ to denote that x is drawn uniformly at random from X. We use $\mathrm{supp}(X)$ to denote the support of the distribution X.

Random oracles. A *random oracle* is typically defined as a function ρ sampled uniformly at random from $(\{0,1\}^m \to \{0,1\}^n)$ for some $m, n \in \mathbb{N}$. However, since in this paper we work with oracles whose outputs are in finite fields, a random oracle in this paper is a function ρ sampled uniformly at random from $(\{0,1\}^m \to \mathbb{F})$ for a given finite field $\mathbb{F}$.

Oracle algorithms. For a function $\theta : X \to Y$, we write A^θ for an algorithm with oracle access to θ. We say that A is t-query if A makes at most t queries to θ.

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 index-instance 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 output 0.

Oracle relations. For a distribution over oracles $\mathcal{O}$, we write $\mathcal{R}^\mathcal{O}$ to denote the set of indexed relations $\{\mathcal{R}^\theta : \theta \in \mathrm{supp}(\mathcal{O})\}$. We define $\mathcal{R}^\mathcal{O} \in \mathsf{NP}^\mathcal{O}$ if and only if there exists a polynomial-time oracle Turing machine M such that, for every $\theta \in \mathrm{supp}(\mathcal{O})$, $\mathcal{R}^\theta = \{(\mathbb{i}, \mathbb{x}, \mathbb{w}) : M^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 1\}$.

Security parameters. We assume for simplicity that all public parameters have length at least λ, so that efficient algorithms which receive such parameters can run in time (at least) polynomial in λ.

Adversaries. An adversary (or extractor) is *polynomial-size* if it can be expressed as a circuit of polynomial size. We also consider a relaxed definition: an adversary (or extractor) running in *(non-uniform) expected polynomial-time* is a Turing machine provided with a *polynomial-size* non-uniform advice string and access to an infinite random tape, whose expected running time for all choices of advice is polynomial.

An adversary $\mathcal{A}$ with expected running time t and success probability p can be converted into a circuit of size $O(t/\epsilon)$ with success probability $p - \epsilon$ as follows: first truncate the execution of $\mathcal{A}$ at running time t/ϵ; then choose as advice the randomness that maximizes the success probability of the truncated $\mathcal{A}$.

3.2 Non-interactive arguments in oracle models

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

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

- $\mathcal{V}^\theta(\mathsf{ivk}, \mathbb{x}, \pi) \to b$. On input an index-specific verification key ivk, an instance $\mathbb{x}$, and a corresponding proof π, the verifier $\mathcal{V}$ computes a bit indicating whether π is a valid proof.

We require ARG to satisfy the following completeness and soundness properties.

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

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

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

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

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

We additionally consider properties of knowledge soundness and zero knowledge for ARG.

Knowledge soundness. ARG has *knowledge soundness* (with respect to auxiliary input distribution $\mathcal{D}$) if there exists an (non-uniform) expected polynomial-time extractor $\mathcal{E}$ such that for every (non-uniform) expected polynomial-time adversary $\tilde{\mathcal{P}}$,

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

Above, we write expected polynomial-time to mean that the expectation is taken over the random variables $\theta \leftarrow \mathcal{O}(\lambda)$, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, and $\mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp})$.

Zero knowledge. ARG has statistical zero knowledge if there exists a probabilistic polynomial-time stateful simulator $\mathcal{S}$ such that for every polynomial-size honest stateful adversary $\mathcal{A}$, the following distributions are $\operatorname{negl}(\lambda)$-close in statistical distance:

$$\left\{\mathcal{A}^\theta(\pi) \,\middle|\, \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\theta(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^\theta(\mathsf{pp}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}^\theta(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right\} \quad \text{and} \quad \left\{\mathcal{A}^{\mathcal{S}^\theta}(\pi) \,\middle|\, \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{S}(1^\lambda) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}; \mathsf{tr}) \leftarrow \mathcal{A}^\theta(\mathsf{pp}) \\ \pi \leftarrow \mathcal{S}^\theta(\mathbb{i}, \mathbb{x}, \mathsf{tr}) \end{array}\right\} \ . \qquad (1)$$

An adversary $\mathcal{A}$ is *honest* if it outputs $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^\theta$ with probability 1. Above, the notation $\mathcal{A}^{\mathcal{S}^\theta}$ indicates that the simulator $\mathcal{S}$ (with oracle access to θ) answers the oracle queries of $\mathcal{A}$.

Succinctness. In this work, we say that a non-interactive argument system ARG for $\mathcal{R}^{\mathcal{O}}$ is *succinct* if there is a fixed polynomial p such that *both* the length of the proof and the running time of the argument verifier are bounded by $p(\lambda, |\mathbb{x}|)$. In this case we refer to ARG as a SNARG; if ARG also has knowledge soundness then it is a SNARK.

3.3 Proof-carrying data

A triple of algorithms $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ is a (preprocessing) *proof-carrying data scheme* (PCD scheme) for a class of compliance predicates F relative to an oracle distribution $\mathcal{O}$ 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 $\Phi \in \mathsf{F}$ if for all outgoing edges $e = (u, v) \in E(\mathsf{T})$:*
- *(base case) if u has no incoming edges, $\Phi^{\theta}(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \bot, \ldots, \bot) = 1$;*
- *(recursive case) if u has incoming edges $e_1, \ldots, e_m$, $\Phi^{\theta}(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, \ldots, z^{(e_m)}) = 1$.*

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

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

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

Knowledge soundness. $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has knowledge soundness (with respect to auxiliary input distribution $\mathcal{D}$) if there exists an (non-uniform) expected polynomial-time extractor $\mathbb{E}^{\tilde{\mathbb{P}}}$ such that for every (non-uniform) expected polynomial-time adversary $\tilde{\mathbb{P}}$,

$$\Pr\left[\begin{array}{l|r} & \theta \leftarrow \mathcal{O}(\lambda) \\ \Phi \in \mathsf{F} & \mathbb{pp} \leftarrow \mathbb{G}(1^{\lambda}) \\ \wedge\ \mathbb{V}(\mathbb{ivk}, \mathsf{o}, \pi) = 1 & \mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp}) \\ \wedge \left(\mathsf{T} \text{ is not } \Phi\text{-compliant} \vee \mathsf{o}(\mathsf{T}) \neq \mathsf{o}\right) & (\Phi, \mathsf{o}, \pi, \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}^{\theta}(\mathbb{pp}, \mathsf{ai}) \\ & (\mathbb{ipk}, \mathbb{ivk}) \leftarrow \mathbb{I}^{\theta}(\mathbb{pp}, \Phi) \\ & \mathsf{T} \leftarrow \mathbb{E}^{\theta, \tilde{\mathbb{P}}}(\mathbb{pp}, \mathsf{ai}) \end{array}\right] \leq \operatorname{negl}(\lambda) \ .$$

Above, we write expected polynomial-time to mean that the expectation is taken over the random variables $\theta \leftarrow \mathcal{O}(\lambda)$, $\mathbb{pp} \leftarrow \mathbb{G}(1^{\lambda})$, and $\mathsf{ai} \leftarrow \mathcal{D}(\mathbb{pp})$.

Zero knowledge. PCD has statistical zero knowledge if there exists a probabilistic polynomial-time stateful simulator $\mathbb{S}$ such that for every polynomial-size honest (stateful) adversary $\mathcal{A}$, the following distributions are $\operatorname{negl}(\lambda)$-close in statistical distance:

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

An adversary $\mathcal{A}$ is *honest* if its output satisfies the implicant of the completeness condition with probability 1 (i.e., $\Phi \in \mathsf{F}$, $\Phi^{\theta}(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and either for all i, $z_i = \perp$, or for all i, $\mathbb{V}^{\theta}(\mathbb{ivk}, z_i, \pi_i) = 1$). Above, the notation $\mathcal{A}^{\mathbb{S}}$ indicates that the simulator $\mathbb{S}$ answers oracle queries of $\mathcal{A}$.

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

3.4 Accumulation schemes

We recall the definition of an accumulation scheme from [BCMS20], extended to any oracle distribution; then in Definition 3.3 below we describe how to specialize that notion to the case of accumulating oracle queries.

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

An **accumulation scheme for** $(\Phi, \mathcal{H})$ is a tuple of algorithms $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ that have access to the same oracle θ (except for G). These algorithms satisfy *completeness* and *soundness*, and optionally also *zero knowledge*, as specified below.

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

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

Note that for $\ell = n = 0$, 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 $\mathcal{A}$,

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

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

$$\left\{\begin{array}{c|r} & \theta \leftarrow \mathcal{O}(\lambda) \\ & \mathsf{pp} \leftarrow \mathrm{G}(1^\lambda) \\ & \mathsf{pp}_\Phi \leftarrow \mathcal{H}^{\theta}(1^\lambda) \\ \mathcal{A}^{\theta}(\mathsf{acc}) & (\mathsf{i}_\Phi, [\mathsf{q}_i]_{i=1}^k, [\mathsf{acc}_j]_{j=1}^{\ell}) \leftarrow \mathcal{A}^{\theta}(\mathsf{pp}, \mathsf{pp}_\Phi) \\ & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\theta}(\mathsf{pp}, \mathsf{pp}_\Phi, \mathsf{i}_\Phi) \\ & (\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\theta}(\mathsf{apk}, [\mathsf{q}_i]_{i=1}^k, [\mathsf{acc}_j]_{j=1}^{\ell}) \end{array}\right\}$$

and

$$\left\{ \mathcal{A}^{\theta}(\mathsf{acc}) \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda) \\ \mathsf{pp} \leftarrow \mathrm{S}(1^{\lambda}) \\ \mathsf{pp}_{\Phi} \leftarrow \mathcal{H}^{\theta}(1^{\lambda}) \\ (\mathsf{i}_{\Phi}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}; \mathsf{tr}) \leftarrow \mathcal{A}^{\theta}(\mathsf{pp}, \mathsf{pp}_{\Phi}) \\ \mathsf{acc} \leftarrow \mathrm{S}^{\theta}(\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}; \mathsf{tr}) \end{array} \right\} .$$

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

Accumulation scheme for oracle queries. We explain how specialize the general notion of an accumulation scheme above to the special case of accumulating queries to an oracle.

Definition 3.3. *Let $\mathcal{O}$ be an oracle distribution. An* **accumulation scheme for $\mathcal{O}$-queries** *is an accumulation scheme where: (i) the accumulation verifier* V *does not access the oracle; (ii) $\mathcal{H} = \bot$ (and so $\mathsf{pp}_{\Phi} = \bot$); (iii) predicate inputs* q *are of the form (x, y); (iv) the predicate Φ is defined such that $\Phi^{\theta}(\mathsf{pp}_{\Phi}, \mathsf{i}_{\Phi}, x, y) = 1$ if and only if $\theta(x) = y$ (in particular, pp_{Φ} and i_{Φ} are ignored).*

3.5 Commitment schemes

A *commitment scheme* is a tuple $\mathsf{CM} = (\mathsf{CM.Setup}, \mathsf{CM.Commit})$ with the following syntax.

- $\mathsf{CM.Setup}$, on input a security parameter 1^{λ}, outputs a commitment key ck.
- $\mathsf{CM.Commit}$, on input a commitment key ck, a message $m \in \{0,1\}^*$, and randomness ω, outputs a commitment cm.

The tuple CM satisfies a binding property and, optionally, a hiding property.

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

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \\ \wedge \\ \mathsf{CM.Commit}(\mathsf{ck}, m_0; \omega_0) = \mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega_1) \end{array} \;\middle|\; \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^{\lambda}) \\ ((m_0, \omega_0), (m_1, \omega_1)) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array} \right] \leq \mathrm{negl}(\lambda).$$

- **Hiding.** For every efficient stateful adversary $\mathcal{A}$ that outputs two messages of the same length, the following distributions are (statistically or computationally) indistinguishable:

$$\mathcal{D}_0(\lambda) := \left\{ (\mathsf{pp}, \mathsf{cm}, \mathsf{aux}) \;\middle|\; \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^{\lambda}) \\ (m_0, m_1, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \omega \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ \mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, m_0; \omega) \end{array} \right\}$$

$$\text{and } \mathcal{D}_1(\lambda) := \left\{ (\mathsf{pp}, \mathsf{cm}, \mathsf{aux}) \;\middle|\; \begin{array}{r} \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^{\lambda}) \\ (m_0, m_1, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \omega \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ \mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega) \end{array} \right\} .$$

Moreover, we say that CM is s-**succinct** if for every commitment key $\mathsf{ck} \in \mathsf{CM.Setup}(1^{\lambda})$, message $m \in \{0,1\}^*$, and randomness ω, it holds that $\mathsf{CM.Commit}(\mathsf{ck}, m; \omega) \in \{0,1\}^{s(\lambda)}$.

We conclude with a simple claim about any binding and hiding commitment scheme.

Claim 3.4. *Let* CM *be a binding and hiding commitment scheme. Then for every message* m*,*

$$\Pr_{\omega,\omega'}\left[\mathsf{CM.Commit}(\mathsf{ck},m,\omega)=\mathsf{CM.Commit}(\mathsf{ck},m,\omega')\right]=\operatorname{negl}(\lambda)\ .$$

Proof. Denote by ε the probability on the left above. Consider the following adversary $\mathcal{C}$ for the hiding game:

1. Output (m_0, m_1) for two arbitrary distinct messages m_0 and m_1.
2. Receive cm from the challenger.
3. Sample commitment randomness ω and compute $\mathsf{cm}' := \mathsf{CM.Commit}(\mathsf{ck}, m_0, \omega)$.
4. If $\mathsf{cm}' = \mathsf{cm}$, output 0; otherwise output 1.

If cm is a commitment to m_0, then $\mathcal{C}$ outputs 0 with probability ε. By the hiding guarantee, if cm is a commitment to m_1, then $\mathcal{C}$ outputs 0 with probability at least $\varepsilon - \operatorname{negl}(\lambda)$. In this case, $\mathcal{C}$ can break the binding property of CM (jointly with the challenger) by returning (m_0, ω). It follows that $\varepsilon = \operatorname{negl}(\lambda)$. □

3.6 Vector commitment schemes

Given a vector $\mathbf{m}$, we denote the i-th element of $\mathbf{m}$ as $\mathbf{m}[i]$. Further for a set of indices I, we write $\mathbf{m}[I]$ to mean the list of values $[\mathbf{m}_i]_{i\in I}$.

A *vector commitment scheme* over the alphabet Σ is a tuple of algorithms $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Commit}, \mathsf{VC.Open}, \mathsf{VC.Check})$ with the following syntax:

- $\mathsf{VC.Setup}(1^\lambda, 1^N) \to \mathsf{pp}_{\mathsf{VC}}$, on input a (unary) security parameter 1^λ and a (unary) input length 1^N, outputs public parameters $\mathsf{pp}_{\mathsf{VC}}$.
- $\mathsf{VC.Commit}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}; \omega) \to \mathsf{cm}$, on input public parameters $\mathsf{pp}_{\mathsf{VC}}$, a vector $\mathbf{m} \in \Sigma^N$ and randomness ω, outputs a commitment cm to $\mathbf{m}$.
- $\mathsf{VC.Open}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}, I; \omega) \to \pi_{\mathsf{VC}}$, on input public parameters $\mathsf{pp}_{\mathsf{VC}}$, a the input vector $\mathbf{m} \in \Sigma^N$, a set of indices $I \subseteq [N]$, and randomness ω, outputs an opening (a.k.a. proof) π_{VC} that $\mathbf{m}[i]$ is the i-th entry of $\mathbf{m}$ for all $i \in [I]$.
- $\mathsf{VC.Check}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}, I, \mathbf{a}, \pi_{\mathsf{VC}}) \to b$, on input public parameters $\mathsf{pp}_{\mathsf{VC}}$, a commitment cm, a query set I, a set of claimed answers corresponding to the query set $\mathbf{a} \in \Sigma^I$, and an opening π_{VC}, outputs a bit b indicating whether cm is a commitment consistent with a vector $\mathbf{m}$ for which the i-th entry is $\mathbf{a}[i]$ for all $i \in [I]$.

The tuple VC satisfies correctness and position binding properties:

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

$$\Pr\left[\mathsf{VC.Check}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}, \mathbf{m}[I], \pi_{\mathsf{VC}}) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N) \\ (\mathbf{m}, \omega, I) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{VC}}) \\ \mathsf{cm} \leftarrow \mathsf{VC.Commit}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}; \omega) \\ \pi_{\mathsf{VC}} \leftarrow \mathsf{VC.Open}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}, I; \omega) \end{array}\right] = 1\ .$$

- **Position binding.** For every efficient adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathsf{VC.Check}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}, I, \mathbf{a}, \pi_{\mathsf{VC}}) = 1 \\ \wedge\ \mathsf{VC.Check}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}, I', \mathbf{a}', \pi'_{\mathsf{VC}}) = 1 \\ \wedge\ \exists\, i \in I \cap I' \text{ s.t. } \mathbf{a}[i] \neq \mathbf{a}'[i] \end{array} \;\middle|\; \begin{array}{r} \mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N) \\ (\mathsf{cm}, (I, \mathbf{a}, \pi_{\mathsf{VC}}), (I', \mathbf{a}', \pi'_{\mathsf{VC}})) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{VC}}) \end{array}\right] \leq \operatorname{negl}(\lambda)\ .$$

A vector commitment scheme may also satisfy a hiding property, where openings do not reveal unopened values. The following definition formalizes this.

- **Hiding.** For every efficient stateful adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} & \mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^N) \\ & (\mathbf{m}_0, \mathbf{m}_1, I) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{VC}}) \\ \mathbf{m}_0|_I = \mathbf{m}_1|_I & b \leftarrow \{0,1\} \\ \wedge\ \mathcal{A}(\mathsf{cm}, \pi_{\mathsf{VC}}) = b & \omega \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ & \mathsf{cm} := \mathsf{VC.Commit}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}_b; \omega) \\ & \pi_{\mathsf{VC}} := \mathsf{VC.Open}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{m}_b, I; \omega) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda) \ .$$

VC in oracle models. In Section 8.2 we consider vector commitments in an oracle model. This means that given an oracle θ sampled from $\mathcal{O}(\lambda)$, all VC algorithms and the adversary $\mathcal{A}$ have access to θ.

4 Linear code random oracles

We define *linear code random oracles* and establish properties about them. We first recall the definition of a linear code and its dual.

Definition 4.1. *A* **linear code** *$\mathcal{C}$ with domain D and alphabet $\mathbb{F}$ is a subspace of $\mathbb{F}^D$. The* **dual code** *of $\mathcal{C}$ is the set $\mathcal{C}^\perp := \{z \in \mathbb{F}^D : \forall c \in \mathcal{C}, \sum_{x \in D} z(x)c(x) = 0\}$, which is also a subspace of $\mathbb{F}^D$.*

We focus on a particular class of linear codes which have a certain systematicity property.[11]

Definition 4.2. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code and $f \colon X \to D$ a function. We say that $\mathcal{C}$ is ψ-***systematic** *if $\mathcal{C}|_{\mathrm{im}(\psi)} = \mathbb{F}^{\mathrm{im}(\psi)} \simeq \mathbb{F}^X$ (in particular, f is an injection). Let $\mathscr{C} = \{\mathcal{C}_\lambda \subseteq (D_\lambda \to \mathbb{F}_\lambda)\}_{\lambda \in \mathbb{N}}$ be a family of linear codes and F an efficient algorithm. We say that $\mathscr{C}$ is* **Ψ-systematic with arity** *m if, for every $\lambda \in \mathbb{N}$, $\Psi(1^\lambda) \colon \{0,1\}^{m(\lambda)} \to D_\lambda$ is a circuit such that $\mathcal{C}_\lambda$ is $\Psi(1^\lambda)$-systematic.*

We are now ready to state an informal definition of a linear code random oracle:

> A **linear code random oracle model** is a model in which all honest and malicious parties have access to an oracle chosen uniformly at random from a member of a systematic family.

Formally, this means that in the definitions of oracle primitives in Section 3.2 and Section 3.4, for some systematic family $\mathscr{C} = \{\mathcal{C}_\lambda\}_\lambda$, the distribution $\mathcal{O}(\lambda)$ is the uniform distribution over $\mathcal{C}_\lambda$. To distinguish linear code random oracles from general oracles, we use the symbol $\hat{\rho}$ for the oracle (instead of θ).

We often omit Ψ and write $c(x)$ for $c(\Psi(1^\lambda)(x))$ when $c \in \mathcal{C}_\lambda$ and $q \in \{0,1\}^{m(\lambda)}$. The standard random oracle is the linear code random oracle $\{(\{0,1\}^{m(\lambda)} \to \mathbb{F}_\lambda)\}_\lambda$, which is I-systematic with $I(1^\lambda)(q) := q$ for $q \in \{0,1\}^{m(\lambda)}$.

An algorithm with oracle access to $c \in \mathcal{C}$, written $\mathcal{A}^c$, can query c at any point in D; we refer to such an algorithm as a $\mathcal{C}$-oracle algorithm. We say that an oracle algorithm is *systematic* if it only makes queries in $\{0,1\}^m$ for some $m \in \mathbb{N}$. Any systematic oracle algorithm $\mathcal{B}$ can be interpreted as a $\mathcal{C}_\lambda$-oracle algorithm via the efficient injection $\Psi(1^\lambda)$; we write $\mathcal{B}^{\hat{\rho}}$, omitting Ψ. The following statement follows immediately from the systematicity condition.

Claim 4.3. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$, $\psi \colon \{0,1\}^m \to D$ be such that $\mathcal{C}$ is ψ-systematic. For every systematic oracle algorithm $\mathcal{B}$,*

$$\Pr_{\rho \leftarrow (\{0,1\}^m \to \mathbb{F})}[\mathcal{B}^\rho \to 1] = \Pr_{\hat{\rho} \leftarrow \mathcal{C}}[\mathcal{B}^{\hat{\rho} \circ \psi} \to 1] \ .$$

4.1 Query transcripts and partial oracles

We define notions used in security proofs involving linear code random oracles. A $\mathcal{C}$-query transcript is a list of query-answer pairs consistent with the execution of a $\mathcal{C}$-oracle algorithm; equivalently, it is a partial function that is consistent with a codeword in $\mathcal{C}$.

Definition 4.4. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. A* **$\mathcal{C}$-transcript** *is a list $\mathsf{tr} = [(x_i, y_i)]_{i=1}^t \in (D \times \mathbb{F})^t$ for any $t \in \mathbb{N}$ such that there exists $c \in \mathcal{C}$ with $c(x_i) = y_i$ for every $i \in [t]$. A $\mathcal{C}$-transcript tr induces a partial function $D \rightharpoonup \mathbb{F}$ in the natural way, which we also denote by tr:*

$$\mathsf{tr}(x) = \begin{cases} y_i & \text{if } x = x_i \\ \perp & \text{if } x \notin \{x_1, \dots, x_t\} \end{cases} .$$

[11]The typical definition of a *systematic* linear code is defined with respect to an encoding function Enc: there exists a subset $S \subseteq D$ such that $\mathsf{Enc}(m)|_S = m$ for all messages m. Our property is related, in that if $\mathcal{C}$ is f-systematic then there exists a function Enc such that $\mathcal{C}$ is systematic with $S = \mathrm{im}(f)$.

A partial oracle extends a query transcript to include evaluations that are determined by the structure of the linear code.

Definition 4.5. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code and* tr *a $\mathcal{C}$-transcript. The* **partial oracle** $\overline{\mathsf{tr}}_{\mathcal{C}} \colon D \rightharpoonup \mathbb{F}$ *is defined as:*

$$\overline{\mathsf{tr}}_{\mathcal{C}}(x) := \begin{cases} \beta & \text{if for every } c \in \mathcal{C} \text{ it holds that } (\forall i \in [t],\ c(x_i) = y_i) \Rightarrow c(x) = \beta \\ \bot & \text{otherwise} \end{cases} .$$

When $\mathcal{C}$ is clear from context we omit it from the notation.

4.2 Constraints

We examine properties of a partial oracle arising from the linear structure of a linear code. We introduce the notion of a *constraint* for elements in the domain of a linear code, and then connect constraints and partial oracles. The results in this section hold for all linear codes, even those that are not systematic.

Definition 4.6. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. A subset $Q \subseteq D$ is* **constrained** *if there exists a nonzero $z \colon Q \to \mathbb{F}$ such that, for every $c \in \mathcal{C}$, $\sum_{x \in Q} z(x)c(x) = 0$ (equivalently, if there exists $z \neq 0 \in \mathcal{C}^{\perp}$ with $\mathrm{supp}(z) \subseteq Q$); we refer to z as a* **constraint** *on Q. We say that Q is* **unconstrained** *if it is not constrained. We say that $Q \subseteq D$* **determines** *$x \in D$ if $x \in Q$ or there exists a constraint z on $Q \cup \{x\}$ such that $z(x) \neq 0$.*

Claim 4.7. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code and* tr *be a $\mathcal{C}$-transcript. Then, for every $x \in D$, $\overline{\mathsf{tr}}_{\mathcal{C}}(x) \neq \bot$ if and only if $\mathrm{supp}(\mathsf{tr})$ determines x. In particular, $\overline{\mathsf{tr}}_{\mathcal{C}}(x) = y \in \mathbb{F}$ if and only if $\mathsf{tr}(x) = y$, or $\mathsf{tr}(x) = \bot$ and there exists $z \in \mathcal{C}^{\perp}$ such that $\mathrm{supp}(z) \subseteq \mathrm{supp}(\mathsf{tr}) \cup \{x\}$, $z(x) \neq 0$, and*

$$y = -z(x)^{-1} \sum_{x' \in \mathrm{supp}(\mathsf{tr})} z(x')\mathsf{tr}(x') .$$

We characterize the distribution of $c(x)$ conditioned on a prior query transcript tr in terms of the value of $\overline{\mathsf{tr}}_{\mathcal{C}}(x)$.

Claim 4.8. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. For every $x \in D$, $Q \subseteq D$, $c' \in \mathcal{C}$, and $y \in \mathbb{F}$,*

$$\Pr_{c \leftarrow \mathcal{C}}[c(x) = y \mid c|_Q = c'|_Q] = \begin{cases} \frac{1}{|\mathbb{F}|} & \text{if } Q \text{ does not determine } x \\ 1 & \text{if } Q \text{ determines } x \text{ and } y = c'(x) \\ 0 & \text{otherwise} \end{cases} .$$

Equivalently, for every $x^* \in D$, $\mathcal{C}$-transcript $\mathsf{tr} = \{(x_i, y_i)\}_{i=1}^{t}$, and $y \in \mathbb{F}$,

$$\Pr_{c \leftarrow \mathcal{C}}[c(x^*) = y \mid \forall i \in [t],\ c(x_i) = y_i] = \begin{cases} \frac{1}{|\mathbb{F}|} & \text{if } \overline{\mathsf{tr}}_{\mathcal{C}}(x^*) = \bot \\ 1 & \text{if } \overline{\mathsf{tr}}_{\mathcal{C}}(x^*) = y \\ 0 & \text{otherwise} \end{cases} .$$

Proof. The equivalence follows from Claim 4.7 and the definition of a $\mathcal{C}$-transcript, so it suffices to prove the second statement. The cases when $\overline{\mathsf{tr}}(x) \neq \bot$ are clear, and so we consider the case when $\overline{\mathsf{tr}}(x) = \bot$. In this case there exist codewords $c, c' \in \mathcal{C}$ such that $c(x_i) = c'(x_i)$ for every i but $c(x) \neq c'(x)$. It follows by linearity of $\mathcal{C}$ that there exists $c^* = c - c' \in \mathcal{C}$ such that $c^*(x_i) = 0$ for every i and $c^*(x) \neq 0$. We can sample from the conditional distribution in the claim by choosing a random $c'' \in \mathcal{C}$ such that $c(x_i) = \alpha_i$ for every i and a random $\alpha \in \mathbb{F}$ and returning $c'' + \alpha c^*$. The claim follows since $c''(x) + \alpha c^*(x)$ is uniformly random in $\mathbb{F}$. $\square$

We recall the definition of a constraint detector [BCFGRS17], which is an algorithm that determines whether a set Q is constrained and, if so, outputs a constraint.

Definition 4.9. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. An algorithm CD is a* **constraint detector** *for $\mathcal{C}$ if, given as input a set $Q \subseteq D$, outputs: (i) a constraint z if Q is constrained; (ii) $\bot$ if Q is unconstrained. A code family $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ has* **efficient constraint detection** *if there exists a polynomial-time algorithm CD such that, for every $\lambda \in \mathbb{N}$, $\mathsf{CD}(1^\lambda, \cdot)$ is a constraint detector for $\mathcal{C}_\lambda$.*

A constraint detector directly yields an implementation of the partial oracle via Claim 4.7, which allows for efficient "lazy" simulation of query access to a random codeword via Claim 4.8. It also allows us to remove "redundant" queries from any $\mathcal{C}$-oracle algorithm.

Definition 4.10. *Let $\mathcal{A}$ be a $\mathcal{C}$-oracle algorithm. We say that a query made by $\mathcal{A}$ is* **redundant** *if it is determined by the prior queries made by $\mathcal{A}$. Also, $\mathcal{A}$ is* **non-redundant** *if it never makes a redundant query.*

Claim 4.11. *Let $\mathcal{A}$ be a t-query $\mathcal{C}$-oracle algorithm, and let CD be a constraint detector for $\mathcal{C}$. Then there exists a non-redundant t-query $\mathcal{C}$-oracle algorithm $\mathcal{A}'$ whose input-output behavior is identical to $\mathcal{A}$, and*

$$T_{\mathcal{A}'} \leq T_{\mathcal{A}} + t \cdot T_{\mathsf{CD}}(t) \ ,$$

where $T_{\mathsf{CD}}(t)$ bounds the running time of CD on a set Q of size at most t. In particular, if $\mathcal{A}$ and CD are efficient, so is $\mathcal{A}'$. Moreover, if $\mathcal{A}$ runs in expected polynomial time but makes a strict polynomial number of queries then $\mathcal{A}'$ runs in expected polynomial time.

4.3 Query complexity

We study query complexity in the linear code random oracle model. We first give an example showing an exponential gap between linear code random oracles and standard random oracles.

Claim 4.12. *For every $m \in \mathbb{N}$ the following holds. For every algorithm $\mathcal{A}$ making fewer than 2^m queries,*

$$\Pr_{\rho \leftarrow (\{0,1\}^m \to \mathbb{F})}\left[a = \textstyle\sum_{x\in\{0,1\}^m} \rho(x) \;\middle|\; a \leftarrow \mathcal{A}^\rho\right] = \frac{1}{|\mathbb{F}|} \ .$$

On the other hand, there exists $\psi \colon \{0,1\}^m \to \{0,1\}^m$, a ψ-systematic linear code $\mathcal{C}$ and a 1-query $\mathcal{C}$-oracle algorithm $\mathcal{B}$ such that

$$\Pr_{\hat{\rho} \leftarrow \mathcal{C}}\left[a = \textstyle\sum_{x\in\{0,1\}^m} \hat{\rho}(x) \;\middle|\; a \leftarrow \mathcal{B}^{\hat{\rho}}\right] = 1 \ .$$

Despite this gap, linear code random oracles are cryptographically useful. In particular, we show that they are collision-resistant. To prove this, we use a couple of linear-algebraic tools. First, we show a simple yet important claim about the existence of codewords with some entries fixed to zero.

Claim 4.13. *Let $\psi \colon \{0,1\}^m \to D$, and let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a ψ-systematic code. Then for every $x_1, \dots, x_t \in D$ there exists $c \in \mathcal{C}$ such that*
- *$c(x_1) = \cdots = c(x_t) = 0$, and*
- *there exists a set $S \subseteq \{0,1\}^m$, $|S| \geq 2^m - t$ such that for all $x \in S$, $c(\psi(x)) = 1$.*

Proof. For $i \in [2^m]$, let $e_i \in \mathbb{F}^{\mathrm{im}(\psi)}$ be the vector with zeroes everywhere except at $\psi(\bar{i})$, where $\bar{i}$ is the binary expansion of i. Since $\mathcal{C}$ is ψ-systematic, there is a basis for $\mathcal{C}$ where the first 2^m elements $c_1, \dots, c_{2^m}$ have $c_i|_{\mathrm{im}(\psi)} = e_i$ for every i. The claim follows by elementary linear algebra. $\square$

We use this to establish an upper bound on the number of points in $\{0,1\}^m$ determined by a set of size t.

Lemma 4.14. *Let $\psi\colon \{0,1\}^m \to D$, and let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a ψ-systematic code. Fix $Q \subseteq D$. Define $T := \{x \in \{0,1\}^m : Q$ determines $x\}$. Then $|T| \le |Q|$.*

Proof. By Claim 4.13, there exists $c \in \mathcal{C}$ such that for every $x \in Q$ it holds that $c(x) = 0$, and a set $S \subseteq \{0,1\}^m$ of size $2^m - |Q|$ such that for every $x \in S$ it holds that $c(x) = 1$. Suppose that $|T| > |Q|$. By the pigeonhole principle, there exists $y \in S \cap T$. By definition of T, there exists $z\colon D \to \mathbb{F}$ with $z(y) \neq 0$ such that $\sum_{x \in Q} z(x)c(x) + z(y)c(y) = 0$, which is a contradiction. ☐

Lemma 4.15. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a full-rank linear code of arity m. For every t-query $\mathcal{C}$-oracle adversary $\mathcal{A}$,*

$$\Pr_{\hat{\rho} \leftarrow \mathcal{C}}\left[\begin{array}{c} x, x' \in \{0,1\}^m \\ \wedge\, x \neq x' \\ \wedge\, \hat{\rho}(x) = \hat{\rho}(x') \end{array} \middle| \, (x, x') \leftarrow \mathcal{A}^{\hat{\rho}} \right] \le \frac{t^2}{|\mathbb{F}|} \ .$$

Proof. Consider running $\mathcal{A}$ and recording its oracle queries in tr. By Lemma 4.14, the set of points $T = \{x \in \{0,1\}^m : \overline{\mathsf{tr}}(x) \neq \perp\}$ is of size at most t. Moreover for every $x \in T$, $\hat{\rho}(x)$ is sampled uniformly at random. Hence the probability that there exist distinct $x, x' \in T$ such that $\hat{\rho}(x) = \hat{\rho}(x')$ is less than $t^2/|\mathbb{F}|$. By Claim 4.8, if x or x' are not in T, then $\Pr[\hat{\rho}(x) = \hat{\rho}(x') \mid \mathsf{tr}] = 1/|\mathbb{F}|$. The lemma follows by a union bound. ☐

4.4 Low-degree random oracles

We denote by $\mathbb{F}^{\le d}[X_1, \dots, X_m]$ the vector space of m-variate polynomials over $\mathbb{F}$ of individual degree at most d. The low-degree polynomial evaluation code is defined as follows:

$$\mathrm{LD}[\mathbb{F}, m, d] := \{c \in (\mathbb{F}^m \to \mathbb{F}) : \exists\, p \in \mathbb{F}^{\le d}[X_1, \dots, X_m] \text{ s.t. } \forall\, x \in \mathbb{F}^m, c(x) = p(x)\} \ .$$

For $d \ge 1$ this code is $\psi_{\mathbb{F},m}$-systematic for $\psi_{\mathbb{F},m}(b_1, \dots, b_m) := (i(b_1), \dots, i(b_{m(\lambda)}))$, where $i\colon \{0,1\} \to \mathbb{F}$ is the natural injection (that maps 0 to $0_{\mathbb{F}}$ and maps 1 to $1_{\mathbb{F}}$).

Definition 4.16. *Let $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of fields, $m\colon \mathbb{N} \to \mathbb{N}$ an arity function, and $d\colon \mathbb{N} \to \mathbb{N}$ a degree function. We define*

$$\mathrm{LD}[\mathscr{F}, m, d] := \{\mathrm{LD}[\mathbb{F}_\lambda, m(\lambda), d(\lambda)]\}_{\lambda \in \mathbb{N}} \ .$$

The above code family is Ψ-systematic for $\Psi(1^\lambda) := \psi_{\mathbb{F}_\lambda, m(\lambda)}$.

The following theorem is proved in [BCFGRS17]:

Theorem 4.17. *The code family* $\mathrm{LD}[\mathscr{F}, m, d]$ *has a constraint detector that runs in time* $\mathrm{poly}(m, d, \log|\mathbb{F}|)$. *In particular, it has efficient constraint detection.*

5 Forking lemmas for linear code random oracles

We state and prove our forking lemmas for linear code random oracles. Let $\psi\colon \{0,1\}^m \to D$ and let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a ψ-systematic linear code. The code $\mathcal{C}$ induces a corresponding fork-point function.

Definition 5.1. *Given a query/answer transcript* tr *of size t and a query x, the* **fork-point function** $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x)$ *outputs the smallest index $i \in [t]$ such that $\{x_1, \dots, x_i, x\}$ is constrained, or $\perp$ if there is no such i.*[12]

The fork-point function can be computed via constraint detection (see Definition 4.9): for $i = 1, \dots, t$ invoke the constraint detection algorithm on the set $\{x_1, \dots, x_i, x\}$ and output $i \in [t]$ if the algorithm outputs a constraint; if no constraint is found (in any iteration) then output $\perp$. In particular, if $\mathcal{C}$ has efficient constraint detection then the fork-point function for $\mathcal{C}$ can be computed efficiently.[13]

In Section 5.1 we introduce and analyze a basic forking algorithm that obtains two winning executions. In Section 5.2 we build on this to construct a forking algorithm that obtains any number of winning executions.

5.1 Basic forking algorithm

We describe the basic forking algorithm for linear code random oracles.

Definition 5.2. *Let $\mathcal{A}$ be a t-query adversary,* $\mathsf{tr} = ((x_1, y_1), \dots, (x_t, y_t))$ *a query-answer transcript, $i \in [t] \cup \{\perp\}$ an index (or abort symbol), and* ai *an auxiliary input. The* **forking algorithm** Fork*, given access to $\mathcal{A}$ and input* $(\mathsf{tr}, i, \mathsf{ai})$*, works as follows:*

(i) if $i = \perp$ then abort and output $\perp$;
(ii) run $\mathcal{A}(\mathsf{ai})$*, answering the first $i-1$ queries to the random oracle according to* tr*;*
(iii) answer each subsequent query with a random element in $\mathbb{F}$;
(iv) return $(\mathsf{o}', \mathsf{tr}')$ *where* (x', o') *is the output of $\mathcal{A}$ and* tr' *is the query transcript of this execution of $\mathcal{A}$.*

For a t-query $\mathcal{C}$-oracle algorithm $\mathcal{A}$, we denote by $(x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ai})$ the following procedure: run $\mathcal{A}(\mathsf{ai})$ with oracle access to $\hat{\rho}$ until it outputs a pair (x, o) (where $x \in D$ and o is arbitrary), and let $\mathsf{tr} := ((x_1, y_1), \dots, (x_t, y_t))$ be the query/answer transcript of this execution.

Below we state our forking lemma; note that it applies for non-redundant adversaries, which is without loss of generality if the linear code has efficient constraint detection (see Claim 4.11).

Lemma 5.3. *Let* $\mathsf{p}\colon \{0,1\}^* \to \{0,1\}$ *be a predicate, $\mathcal{Z}$ be an auxiliary-input distribution, and $\mathcal{A}$ be a t-query $\mathcal{C}$-oracle deterministic algorithm that is non-redundant. If we define*

$$\delta := \Pr\left[\begin{array}{l} i \neq \perp \\ \wedge\, x \in \{0,1\}^m \\ \wedge\, \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1 \end{array} \;\middle|\; \begin{array}{r} \hat{\rho} \leftarrow \mathcal{C} \\ \mathsf{ai} \leftarrow \mathcal{Z} \\ (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ai}) \\ i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) \end{array}\right] \tag{2}$$

[12]The fork-point function depends only on the queries; the answers are included in the transcript for notational simplicity. Moreover, since the zero word is a member of any linear code, any algorithm computing FP given query-answer pairs can be used to compute FP given queries alone by setting the answers to zero.

[13]Conversely, if the fork-point function for $\mathcal{C}$ can be computed efficiently then one can efficiently detect whether a constraint for $\mathcal{C}$ exists on a set of points $\{x_1, \dots, x_t, x\}$ (but not necessarily learn what the constraint is): set $\mathsf{tr} := \{(x_1, 0), \dots, (x_t, 0)\}$; and run $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x)$. $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) = \perp$ if and only if $\{x_1, \dots, x_t, x\}$ is unconstrained.

then it holds that

$$\Pr\left[\begin{array}{l|r} i \neq \bot & \hat{\rho} \leftarrow \mathcal{C} \\ \wedge\, x \in \{0,1\}^m & \mathsf{ai} \leftarrow \mathcal{Z} \\ \wedge\, \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1 & (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ai}) \\ \wedge\, \mathsf{p}(x, \mathsf{o}', \mathsf{tr}') = 1 & i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) \\ & (\mathsf{o}', \mathsf{tr}') \leftarrow \mathsf{Fork}^{\mathcal{A}}(\mathsf{tr}, i, \mathsf{ai}) \end{array}\right] \geq \delta^2/t \ . \tag{3}$$

Remark 5.4 (probabilistic adversaries)**.** The lemma can express probabilistic adversaries by considering auxiliary input distributions $\mathcal{Z}$ that output a random string (and possibly other relevant information). In fact, because the lemma does not require that $\mathcal{Z}$ has finite or even countable support, it can even capture infinite random tapes, which naturally arise when considering adversaries that run in expected polynomial time.

Proof. For $\vec{y} \in \mathbb{F}^t$, denote by $(x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\vec{y}}(\mathsf{ai})$ the same procedure as $\mathcal{A}^{\hat{\rho}}(\mathsf{ai})$, but where the i-th query $x_i \in \{0,1\}^m$ to the oracle is answered with the i-th entry of $\vec{y}$. For every $i \in [t]$ and $x \in \{0,1\}^m$, we define the (measurable) set

$$S_{i,x} := \{(\vec{y}, \mathsf{ai}) \in \mathbb{F}^t \times \mathrm{supp}(\mathcal{Z}) : (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\vec{y}}(\mathsf{ai}) \wedge \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) = i \wedge \mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1\} \ .$$

Next, for field elements $y_1, \ldots, y_{i-1} \in \mathbb{F}$ and $\mathsf{ai} \in \mathrm{supp}(\mathcal{Z})$, define the probability

$$\delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai}) := \Pr_{y'_i, \ldots, y'_t \in \mathbb{F}}\Big[\big((y_1, \ldots, y_{i-1}, y'_i, \ldots, y'_t), \mathsf{ai}\big) \in S_{i,x}\Big] \ . \tag{4}$$

We argue that $\delta = \mathbb{E}_{\vec{y},\mathsf{ai}}[\sum_{i \in [t], x \in \{0,1\}^m} \delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})]$, where $\vec{y} \leftarrow \mathbb{F}^t$ and $\mathsf{ai} \leftarrow \mathcal{Z}$. By linearity of expectation, it suffices to prove that $\delta = \sum_{i \in [t], x \in \{0,1\}^m} \mathbb{E}_{\vec{y},\mathsf{ai}}\Big[\delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})\Big]$. Let S be the (measurable) set of all pairs $(\vec{y}, \mathsf{ai})$ such that $(x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\vec{y}}(\mathsf{ai})$ and $\mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 1$. Because $\mathcal{A}$ is non-redundant, the answers to its queries to $\hat{\rho}$ are uniformly random field elements, and so $\delta = \Pr_{\vec{y},\mathsf{ai}}[(\vec{y}, \mathsf{ai}) \in S]$. It follows directly from the definition of $\delta_{i,x}$ that

$$\mathbb{E}_{\vec{y},\mathsf{ai}}[\delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})] = \mathbb{E}_{y_1,\ldots,y_{i-1},\mathsf{ai}}[\mathbb{E}_{y_i,\ldots,y_t}[\mathbb{1}_{S_{i,x}}(\vec{y}, \mathsf{ai})]] = \Pr_{\vec{y},\mathsf{ai}}[(\vec{y}, \mathsf{ai}) \in S_{i,x}] \ .$$

Since $\{S_{i,x}\}_{i \in [t], x \in \{0,1\}^m}$ is a partition of S,

$$\sum_{i,x} \mathbb{E}_{\vec{y},\mathsf{ai}}\big[\delta_{i,x}(\vec{y}; \mathsf{ai})\big] = \sum_{i,x} \Pr_{\vec{y},\mathsf{ai}}[(\vec{y}, \mathsf{ai}) \in S_{i,x}] = \Pr_{\vec{y},\mathsf{ai}}[(\vec{y}, \mathsf{ai}) \in S] = \delta \ .$$

Denote by E the event in Equation 3. First, we observe that by the definition of Fork (and since $\mathcal{A}$ is non-redundant) the following holds:

$$\Pr[E] = \Pr_{\vec{y},\vec{y}',\mathsf{ai}}[\exists i \in [t], x \in \{0,1\}^m,\ (\vec{y}, \mathsf{ai}) \in S_{i,x} \wedge \big((y_1, \ldots, y_{i-1}, y'_i, \ldots, y'_t), \mathsf{ai}\big) \in S_{i,x}]$$

where $\vec{y}, \vec{y}' \leftarrow \mathbb{F}^t$ and $\mathsf{ai} \leftarrow \mathcal{Z}$. Since the sets $\{S_{i,x}\}_{i \in [t], x \in \{0,1\}^m}$ are disjoint, we have that

$$\begin{aligned} \Pr[E] &= \sum_{i \in [t], x \in \{0,1\}^m} \Pr_{\vec{y},\vec{y}',\mathsf{ai}}[(\vec{y}, \mathsf{ai}) \in S_{i,x} \wedge \big((y_1, \ldots, y_{i-1}, y'_i, \ldots, y'_t), \mathsf{ai}\big) \in S_{i,x}] \\ &= \sum_{i,x} \mathbb{E}_{\vec{y},\vec{y}',\mathsf{ai}}\big[\mathbb{1}_{S_{i,x}}(\vec{y}, \mathsf{ai}) \cdot \mathbb{1}_{S_{i,x}}\big((y_1, \ldots, y_{i-1}, y'_i, \ldots, y'_t), \mathsf{ai}\big)\big] \end{aligned}$$

$$= \sum_{i,x} \mathbb{E}_{y_1,\ldots,y_{i-1},\mathsf{ai}}\Big[\mathbb{E}_{y_i,\ldots,y_t,y'_i,\ldots,y'_t}\big[\mathbb{1}_{S_{i,x}}(\vec{y},\mathsf{ai}) \cdot \mathbb{1}_{S_{i,x}}\big((y_1,\ldots,y_{i-1},y'_i,\ldots,y'_t),\mathsf{ai}\big)\big]\Big]$$
$$= \sum_{i,x} \mathbb{E}_{y_1,\ldots,y_{i-1},\mathsf{ai}}\big[\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai})^2\big]$$

where the last equality holds by independence, and by definition of $\delta_{i,x}$.

Next, we bound the number of nonzero terms in the above sum. Let $Q_{i,\vec{y},\mathsf{ai}} := \{x \in \{0,1\}^m : \delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai}) > 0\}$.

Claim 5.5. *For every* $\vec{y} \in \mathbb{F}^t$ *and* $\mathsf{ai} \in \{0,1\}^*$, $\sum_{i\in[t]} |Q_{i,\vec{y},\mathsf{ai}}| \le t$.

Proof. Fix any t-query transcript $\mathsf{tr} = ((x_1,y_1),\ldots,(x_t,y_t))$ where the query answers are from $\vec{y}$, and define $Q_{i,\mathsf{tr}} := \{x \in \{0,1\}^m : \mathsf{FP}_{\mathcal{C}}(\mathsf{tr},x) = i\}$. Note that $Q_{i,\mathsf{tr}}$ is a function of $(x_1,\ldots,x_i)$ only, since $Q_{i,\mathsf{tr}}$ contains all queries x such that $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr},x) = i$ (as i is the earliest index where x is determined). Since $x \in \{0,1\}^m$, Lemma 4.14 implies that $\sum_{i\in[t]} |Q_{i,\mathsf{tr}}| \le t$.

We show that $Q_{i,\vec{y},\mathsf{ai}} \subseteq Q_{i,\mathsf{tr}}$ for every $i \in [t]$. Fix any $x \in Q_{i,\vec{y},\mathsf{ai}}$, that is, any $x \in \{0,1\}^m$ such that $\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai}) > 0$. Then there exists $y'_i,\ldots,y'_t$ such that $((y_1,\ldots,y_{i-1},y'_i,\ldots,y'_t),\mathsf{ai}) \in S_{i,x}$. This implies that

$$(x',\mathsf{o}';\mathsf{tr}') := \mathcal{A}^{(y_1,\ldots,y_{i-1},y'_i,\ldots,y'_t)}(\mathsf{ai}) \quad \text{and} \quad \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}',x') = i \quad \text{and} \quad x' = x$$

where $\mathsf{tr}' = ((x_1,y_1),\ldots,(x_{i-1},y_{i-1}),(x_i,y'_i),\ldots,(x_t,y'_t))$. For every $i \in [t]$, $(x_1,\ldots,x_i)$ is a function of $y_1,\ldots,y_{i-1},\mathsf{ai}$ only; when running $\mathcal{A}$, the i-th query depends only on prior queries and ai. So $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr}',x) = \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}',x') = i$ is a function of $y_1,\ldots,y_{i-1},\mathsf{ai}$ only. Since $Q_{i,\mathsf{tr}}$ is a function of $(x_1,\ldots,x_i)$ only, $Q_{i,\mathsf{tr}}$ is also a function of $y_1,\ldots,y_{i-1},\mathsf{ai}$ only. This implies that $\mathsf{FP}_{\mathcal{C}}(\mathsf{tr},x) = i$, so $x \in Q_{i,\mathsf{tr}}$. □

We conclude that

$$\begin{aligned}
\Pr[E] &= \mathbb{E}_{\vec{y},\mathsf{ai}}\Big[\textstyle\sum_{i\in[t]}\sum_{x\in Q_{i,\vec{y},\mathsf{ai}}}\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai})^2\Big] \\
&\ge \frac{1}{t}\cdot \mathbb{E}_{\vec{y},\mathsf{ai}}\Big[\Big(\textstyle\sum_{i\in[t]}\sum_{x\in\{0,1\}^m}\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai})\Big)^2\Big] \\
&\ge \frac{1}{t}\cdot \Big(\mathbb{E}_{\vec{y},\mathsf{ai}}\Big[\textstyle\sum_{i\in[t]}\sum_{x\in\{0,1\}^m}\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai})\Big]\Big)^2 \\
&= \frac{1}{t}\cdot\delta^2 \quad ,
\end{aligned}$$

where the first inequality holds because the number of terms in the sum is at most t (by Claim 5.5), and the second holds by the inequality $\mathbb{E}[X^2] \ge \mathbb{E}[X]^2$ (for general random variables X). □

5.2 Generic forking algorithm

We define the algorithm MFork, which uses the basic forking algorithm Fork (Definition 5.2) as a subroutine to produce N winning executions.

Definition 5.6. *Let $\mathcal{A}$ be a t-query adversary, $N \in \mathbb{Z}^+$ the target number of winning executions,* $\mathsf{tr} = ((x_1,y_1),\ldots,(x_t,y_t))$ *a query-answer transcript, $i \in [t] \cup \{\bot\}$ an index (or abort symbol), and* ai *an auxiliary input. The* **multi-fork algorithm** MFork*, given black-box access to $\mathcal{A}$ and predicate* p*, and input* $(N,\mathsf{tr},i,\mathsf{ai})$*, works as follows:*

(i) *if* $i = \bot$ *then abort; else run* $(x, \mathsf{o}) \leftarrow \mathcal{A}^{\mathsf{tr}}(\mathsf{ai})$*; if* $\mathsf{p}(x, \mathsf{o}, \mathsf{tr}) = 0$ *or* $x \notin \{0,1\}^m$ *then abort;*
(ii) *repeatedly run* $\mathsf{Fork}^{\mathcal{A}}(\mathsf{tr}, i, \mathsf{ai})$ *until we collect a set of* N *valid executions* $\mathsf{S} = \{(\mathsf{o}_i, \mathsf{tr}_i)\}_{i \in [N]}$ *(i.e., for every* $i \in [N]$*,* $\mathsf{p}(x, \mathsf{o}_i, \mathsf{tr}_i) = 1$*);*
(iii) *return* S*.*

We analyze the running time of MFork as a random variable. The *negative binomial* distribution naturally captures the number of repetitions of Fork required to obtain a given number of transcripts.

Definition 5.7. *For* $N \in \mathbb{N}$ *and* $\delta \in [0,1]$*, the* **negative binomial distribution** *with parameters* N, δ*, denoted* $\mathsf{NB}(N, \delta)$*, is the distribution of the random variable* X *in the following experiment: sample from a Bernoulli distribution with parameter* δ *until* N *successes are observed; let* X *be the total number of trials.*

Lemma 5.8. *Let* $\mathsf{p}\colon \{0,1\}^* \to \{0,1\}$ *be a predicate,* $\mathcal{Z}$ *be an auxiliary-input distribution, and* $\mathcal{A}$ *be a* t*-query* $\mathcal{C}$*-oracle deterministic algorithm that is non-redundant. Denote by* $K := K(N, \mathsf{tr}, i, \mathsf{ai})$ *the number of repetitions of Step (ii) in* $\mathsf{MFork}^{\mathcal{A},\mathsf{p}}(N, \mathsf{tr}, i, \mathsf{ai})$*, and let* $\delta := \delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})$ *where* $\mathsf{tr} = ((x_j, y_j))_{j=1}^t$*. If* MFork *does not abort in Step (i) then we have* $K \sim \mathsf{NB}(N, \delta)$*. Moreover,*

$$\mathbb{E}\left[\operatorname{time}(\mathsf{MFork}^{\mathcal{A},\mathsf{p}}(N, \mathsf{tr}, i, \mathsf{ai})) \;\middle|\; \begin{array}{r} \hat{\rho} \leftarrow \mathcal{C} \\ \mathsf{ai} \leftarrow \mathcal{Z} \\ (x, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ai}) \\ i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, x) \end{array}\right] \leq (tN + O(1)) \cdot \mathbb{E}[\operatorname{time}(\mathcal{A})] \ .$$

Proof. Fix a choice of $N, \mathsf{tr}, i, \mathsf{ai}$. Recall from Eq. 4 and the definition of Fork that

$$\Pr\left[\ \mathsf{p}(x, \mathsf{o}', \mathsf{tr}') = 1 \ \middle|\ (\mathsf{o}', \mathsf{tr}') \leftarrow \mathsf{Fork}^{\mathcal{A}}(\mathsf{tr}, i, \mathsf{ai})\right] = \delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai}) \ .$$

The distribution of K follows immediately from the description of MFork. Next, let $T := T(N, \vec{y}, \mathsf{ai})$ denote the running time of Step (ii) (as a random variable). Let $\tau(y_1, \ldots, y_{i-1}; \mathsf{ai})$ be the random variable denoting the running time of $\mathcal{A}$ inside $\mathsf{Fork}^{\mathcal{A}}(\mathsf{tr}, i, \mathsf{ai})$. Let E_{valid} be the event that Fork outputs a valid execution. Finally, define

$$\begin{aligned} \mathsf{t} := \mathsf{t}(y_1, \ldots, y_{i-1}; \mathsf{ai}) &:= \mathbb{E}[\tau(y_1, \ldots, y_{i-1}; \mathsf{ai})] \\ \mathsf{t}_{\mathsf{valid}} := \mathsf{t}_{\mathsf{valid}}(y_1, \ldots, y_{i-1}; \mathsf{ai}) &:= \mathbb{E}[\tau(y_1, \ldots, y_{i-1}; \mathsf{ai}) \mid E_{\mathsf{valid}}] \\ \mathsf{t}_{\mathsf{invalid}} := \mathsf{t}_{\mathsf{invalid}}(y_1, \ldots, y_{i-1}; \mathsf{ai}) &:= \mathbb{E}[\tau(y_1, \ldots, y_{i-1}; \mathsf{ai}) \mid \overline{E_{\mathsf{valid}}}] \end{aligned}$$

and observe that $\mathsf{t} = \delta \cdot \mathsf{t}_{\mathsf{valid}} + (1 - \delta) \cdot \mathsf{t}_{\mathsf{invalid}}$ by total expectation. Now we can compute

$$\mathbb{E}[T] = \sum_{k=N}^{\infty} \mathbb{E}[T \mid K = k] \Pr[K = k] = N \cdot \mathsf{t}_{\mathsf{valid}} + \sum_{k=N}^{\infty} \Pr[K = k](k - N)\mathsf{t}_{\mathsf{valid}} = N\mathsf{t}_{\mathsf{valid}} + (\mathbb{E}[K] - N)\mathsf{t}_{\mathsf{invalid}} \ .$$

Since $\mathbb{E}[K] = N/\delta$, $\mathbb{E}[T] = N(\mathsf{t}_{\mathsf{valid}} + (1/\delta - 1)\mathsf{t}_{\mathsf{invalid}}) = N\mathsf{t}/\delta$. Then, averaging over all $\vec{y}, \mathsf{ai}$ (and the randomness of MFork),

$$\begin{aligned} \mathbb{E}_{\vec{y},\mathsf{ai}}[T] &= \sum_{i=1}^{t} \mathbb{E}_{\vec{y},\mathsf{ai}}\Big[\sum_{x \in \{0,1\}^m} \mathbb{1}_{S_{i,x}}(\vec{y}, \mathsf{ai}) \cdot T(N, \vec{y}, \mathsf{ai})\Big] \\ &= \sum_{i=1}^{t} \mathbb{E}_{y_1,\ldots,y_{i-1},\mathsf{ai}}\Big[\sum_{x \in \{0,1\}^m} \mathbb{E}_{y_i,\ldots,y_t}[\mathbb{1}_{S_{i,x}}(\vec{y}, \mathsf{ai})] \cdot T(N, \vec{y}, \mathsf{ai})\Big] \end{aligned}$$

$$= \sum_{i=1}^{t} \mathbb{E}_{y_1,\ldots,y_{i-1},\mathsf{ai}} \left[\sum_{x \in Q_{i,\vec{y},\mathsf{ai}}} \delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai}) \cdot N \cdot \frac{\mathsf{t}(y_1,\ldots,y_{i-1};\mathsf{ai})}{\delta_{i,x}(y_1,\ldots,y_{i-1};\mathsf{ai})} \right]$$

$$= N \cdot \mathbb{E}_{\vec{y},\mathsf{ai}} \Big[\sum_{i=1}^{t} |Q_{i,\vec{y},\mathsf{ai}}| \cdot \mathsf{t}(y_1,\ldots,y_{i-1};\mathsf{ai}) \Big] \leq tN \cdot \mathbb{E}[\mathrm{time}(\mathcal{A})] \ .$$

The lemma follows by noting that the expected running time of Step (i) is $O(\mathbb{E}[\mathrm{time}(\mathcal{A})])$. □

6 Oracle zero-finding games

We state and prove our lemma for zero-finding games in the low-degree random oracle model.

Lemma 6.1 (formal restatement of Lemma 2.5)**.** *Let $\mathcal{C} = \mathrm{LD}[\mathbb{F}, m, d]$ be the code of m-variate polynomials of individual degree at most d. Let $\ell \in \mathbb{N}$ be a degree bound, and CM a commitment scheme (that is binding though not necessarily hiding). For every efficient probabilistic t-query oracle algorithm $\mathcal{A}$ that outputs tuples of the form $(f \in \mathbb{F}^{\leq dm\ell}[X], g \in (\mathbb{F}^{\leq \ell}[X])^m, \omega)$, the following holds:*

$$\Pr\left[\begin{array}{c|r} f(X) \not\equiv \hat{\rho}(g(X)) & \hat{\rho} \leftarrow \mathcal{C} \\ \wedge\, f(z) = \hat{\rho}(g(z)) & \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda) \\ & (f, g, \omega) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ck}) \\ & \mathsf{cm} \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, (f, g); \omega) \\ & z \in \mathbb{F} \leftarrow \hat{\rho}(\mathsf{cm}) \end{array}\right] \leq O\left(\sqrt{\frac{tdm\ell}{|\mathbb{F}|}}\right) + \mathrm{negl}(\lambda) \enspace .$$

Remark 6.2. The definition of a commitment scheme in Section 3.5 for simplicity allows the message to have any size. If instead the commitment scheme were to require specifying a bound on the message size, then this bound should be large enough to allow a message that describes the two polynomials f and g. In particular, the bound can be $O(dm\ell)$ field elements, because f can be described via $dm\ell + 1$ evaluations (it is a univariate polynomial of degree at most $dm\ell$) and g can be described via $m \cdot (\ell + 1)$ evaluations (it consists of m univariate polynomials of degree at most ℓ).

Proof. Let $\mathcal{A}$ be an adversary that wins the above game with probability δ. Theorem 4.17 tells us that $\mathcal{C}$ has efficient constraint detection and, therefore, by Claim 4.11 we may assume without loss of generality that $\mathcal{A}$ is non-redundant. We can therefore invoke Lemma 5.3, which applies only to non-redundant algorithms. To do so we define the following objects.

- An auxiliary-input distribution $\mathcal{Z}$ that is the joint distribution of commitment keys $\mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda)$ and the random tape σ for $\mathcal{A}$.
- A deterministic (forking lemma) adversary $\mathcal{B}$ that receives auxiliary input $(\mathsf{ck}, \sigma) \in \mathcal{Z}$, accesses an oracle $\hat{\rho}$, and works as follows: (i) run $\mathcal{A}^{\hat{\rho}}(\mathsf{ck})$ on random tape σ to obtain its output (f, g, ω); (ii) compute the commitment $\mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, (f, g); \omega)$; (iii) output $(x, \mathsf{o}, \mathsf{tr}) := (\mathsf{cm}, (f, g, \omega, \mathsf{ck}), \mathsf{tr})$.
- A forking predicate p that, on input $(x, \mathsf{o}, \mathsf{tr}) = (\mathsf{cm}, (f, g, \omega, \mathsf{ck}), \mathsf{tr})$, checks the following conditions:

  Condition 1. $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{ck}, (f, g); \omega)$;
  Condition 2. either $\overline{\mathsf{tr}} \circ g \colon \mathbb{F} \to \mathbb{F}$ is not total or $f(X) \not\equiv \overline{\mathsf{tr}}(g(X))$; and
  Condition 3. $f(z) = \overline{\mathsf{tr}}(g(z))$, where $z := \overline{\mathsf{tr}}(\mathsf{cm})$.

  Here $\overline{\mathsf{tr}}$ is as defined in Definition 4.5.

Recall from the assumption that $\mathcal{A}$ wins the zero-finding game with probability δ. We show that $\mathcal{B}$ satisfies p with probability at least $\delta - \frac{1}{|\mathbb{F}|}$.

First, any $\mathcal{A}$ winning the zero-finding game satisfies Condition 1 directly; hence so does $\mathcal{B}$.

Second, we show that whenever $\mathcal{A}$ wins the zero-finding game, then Condition 2 is satisfied. (Since $\mathcal{B}$'s queries are the same as $\mathcal{A}$'s, the same is then true for $\mathcal{B}$.) We argue this via the contrapositive: if $f(X) \equiv \overline{\mathsf{tr}}(g(X))$, then $\mathcal{A}$ does not win the zero-finding game. For any oracle input x, if $\overline{\mathsf{tr}}(x)$ is defined, then it is equal to $\hat{\rho}(x)$. Thus if $f(X) \equiv \overline{\mathsf{tr}}(g(X))$, $\overline{\mathsf{tr}}$ must be defined over the image of g, and so $f(X) \equiv \hat{\rho}(g(X))$; hence $\mathcal{A}$ cannot win the zero-finding game by definition.

Third, we argue that the probability that $\mathcal{A}$ wins the zero-finding game but $\mathcal{B}$ fails to satisfy Condition 3 is at most $\frac{1}{|\mathbb{F}|}$. There are two cases: either $\overline{\mathsf{tr}}(g(z))$ is defined, or not. If $\overline{\mathsf{tr}}(g(z))$ is defined then $\overline{\mathsf{tr}}(g(z)) = \hat{\rho}(g(z))$, and if $\mathcal{A}$ wins the zero-finding game then $\hat{\rho}(g(z)) = f(z)$; hence Condition 3 is satisfied in this case. If $\overline{\mathsf{tr}}(g(z)) = \bot$, then the probability that $\mathcal{A}$ wins the zero-finding game is $\frac{1}{|\mathbb{F}|}$, since it must guess the value of $\hat{\rho}(g(z))$.

Thus overall $\mathcal{B}$ satisfies p with probability at least $\delta - \frac{1}{|\mathbb{F}|}$. We deduce, via Lemma 5.3, that

$$\mu := \Pr\left[\begin{array}{l|r} \mathsf{cm} \neq \bot & \hat{\rho} \leftarrow \mathcal{C} \\ \wedge\, \mathsf{cm} \in \{0,1\}^m & (\mathsf{ck}, \sigma) \leftarrow \mathcal{Z} \\ \wedge\, \mathsf{p}(\mathsf{cm}, \mathsf{o}, \mathsf{tr}) = 1 & (\mathsf{cm}, \mathsf{o}; \mathsf{tr}) \leftarrow \mathcal{B}^{\hat{\rho}}(\mathsf{ck}, \sigma) \\ \wedge\, \mathsf{p}(\mathsf{cm}, \mathsf{o}', \mathsf{tr}') = 1 & i \leftarrow \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, \mathsf{cm}) \\ & (\mathsf{o}', \mathsf{tr}') \leftarrow \mathsf{Fork}^{\mathcal{B}}(\mathsf{tr}, i, (\mathsf{ck}, \sigma)) \end{array}\right] \geq \left(\delta - \frac{1}{|\mathbb{F}|}\right)^2 / t \ .$$

To conclude the proof, we upper bound μ. Denote by E_1 the event on the left of the above expression.

The probability that E_1 occurs and $\mathsf{o} \neq \mathsf{o}'$ is at most $\mathrm{negl}(\lambda)$. Indeed, by definition of p, if E_1 holds then $\mathsf{CM.Commit}(\mathsf{ck}, \mathsf{o}) = \mathsf{cm} = \mathsf{CM.Commit}(\mathsf{ck}, \mathsf{o}')$. However, by the binding property of the commitment scheme CM, the probability that this happens and $\mathsf{o} \neq \mathsf{o}'$ occurs is at most $\mathrm{negl}(\lambda)$.

Let $E_2 := E_1 \wedge (\mathsf{o} = \mathsf{o}')$. From the above it holds that $\Pr[E_2] \geq \mu - \mathrm{negl}(\lambda)$.

Let $i := \mathsf{FP}_{\mathcal{C}}(\mathsf{tr}, \mathsf{cm})$, and let $\mathsf{tr}|_{i-1}$ denote the truncation of tr to the first $i-1$ queries. Define the point $z' := \overline{\mathsf{tr}'}(\mathsf{cm}')$. We show that if E_2 occurs then, with high probability, $\mathsf{tr}|_{i-1} \circ g$ is total.

Claim 6.3. *The probability that E_2 occurs and $\overline{\mathsf{tr}|_{i-1}} \circ g$ is not total is at most $(dm\ell + 1)/|\mathbb{F}|$.*

Proof. Note that, for every $c \in \mathcal{C}$, $\deg(c \circ g) \leq d \cdot \ell$. Hence, if $\overline{\mathsf{tr}|_{i-1}} \circ g$ is not total, then there are at most $d \cdot \ell$ points $x \in \mathbb{F}$ such that $\overline{\mathsf{tr}|_{i-1}}(g(x)) \neq \bot$.

By the choice of fork point i, since $\mathsf{tr}|_{i-1}$ contains only queries before the i-th query, $\overline{\mathsf{tr}|_{i-1}}(\mathsf{cm}) = \bot$. Hence z' is uniformly random conditioned on $\mathsf{tr}|_{i-1}$. Thus $\Pr[\overline{\mathsf{tr}|_{i-1}}(g(z')) \neq \bot] \leq \frac{dm\ell}{|\mathbb{F}|}$. Finally, if $\overline{\mathsf{tr}|_{i-1}}(g(z')) = \bot$ then $\Pr[\overline{\mathsf{tr}'}(g(z')) = f(z')] \leq \frac{1}{|\mathbb{F}|}$, as f and tr' are independent conditioned on $\mathsf{tr}|_{i-1}$. ☐

By the claim, the probability that E_2 occurs and $\overline{\mathsf{tr}|_{i-1}} \circ g$ is total is at least $\mu - (dm\ell + 1)/|\mathbb{F}| - \mathrm{negl}(\lambda)$. In this case it holds that $\overline{\mathsf{tr}|_{i-1}} \circ g \not\equiv f$, but $\overline{\mathsf{tr}|_{i-1}}(g(z')) = f(z')$. Since z' is drawn independently of $\mathsf{tr}|_{i-1}, g, f$, this equality holds with probability at most $dm\ell/|\mathbb{F}|$.

Rearranging, we conclude that

$$\mu \leq \frac{2dm\ell + 1}{|\mathbb{F}|} + \mathrm{negl}(\lambda) \ .$$

Since $\mu \geq \left(\delta - \frac{1}{|\mathbb{F}|}\right)^2 / t$, we deduce that

$$\delta \leq \sqrt{t \cdot \left(\frac{2dm\ell + 1}{|\mathbb{F}|}\right)} + \frac{1}{|\mathbb{F}|} + \mathrm{negl}(\lambda) \ ,$$

which implies the statement. ☐

7 Accumulation scheme for low-degree random oracles

We construct an accumulation scheme to accumulate queries to any low-degree random oracle (over a sufficiently large field). See Definition 3.3 for the definition of an accumulation scheme for oracle queries.

Theorem 7.1. *Let* $\mathrm{LD}[\mathscr{F}, m, d] = \{\mathcal{C}_\lambda = \mathrm{LD}[\mathbb{F}_\lambda, m(\lambda), d(\lambda)]\}_{\lambda \in \mathbb{N}}$ *be a family of low-degree polynomial evaluation codes, where* $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a family of fields,* $m \colon \mathbb{N} \to \mathbb{N}$ *an arity function, and* $d \colon \mathbb{N} \to \mathbb{N}$ *a degree function (see Definition 4.16). Let* $\mathsf{CM} = (\mathsf{CM.Setup}, \mathsf{CM.Commit})$ *be a standard-model commitment scheme that is hiding, binding, and* m*-succinct (see Section 3.5). Then,* AS *from Construction 1 below is a zero-knowledge accumulation scheme for queries to* $\mathrm{LD}[\mathscr{F}, m, d]$ *with the following properties.*

- Soundness error. *When accumulating* n *oracle queries and* ℓ *old accumulators, the soundness error is*

$$O\left(\sqrt{\frac{t \cdot d(\lambda) \cdot m(\lambda) \cdot (n + \ell)}{|\mathbb{F}_\lambda|}}\right) + \mathrm{negl}(\lambda)$$

  against any t*-query efficient adversary. In particular, if* $t, m, d, n, \ell = \lambda^{O(1)}$ *and* $|\mathbb{F}_\lambda| = \lambda^{\omega(1)}$*, then the soundness error is negligible in* λ*.*
- Accumulator size. *The accumulator contains 2 oracle query-answer pairs (i.e.,* $2(m(\lambda)+1)$ *field elements).*
- Decider efficiency. *The decider consists of checking 2 oracle query-answer pairs.*

Remark 7.2. If CM is statistically hiding, then AS achieves statistical zero knowledge. Further, if AS is not required to be zero knowledge, then CM need not be hiding and so can be replaced by a collision-resistant hash function family.

We give the construction below. For notational simplicity, we omit the dependence on the security parameter λ on the low-degree random oracle's parameters and thus write: $\mathcal{C} = \mathcal{C}_\lambda$, $\mathbb{F} = \mathbb{F}_\lambda$, $d = d(\lambda)$, $m = m(\lambda)$. Moreover, we assume a global ordering of the field $\mathbb{F}$, so that $\mathbb{F} = \{b_1, \ldots, b_{|\mathbb{F}|}\}$.

Construction 1. The accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ is specified below. An accumulator acc is a tuple $((x_1, y_1), (x_2, y_2))$ where each (x_i, y_i) is a query-answer pair in $\mathbb{F}^m \times \mathbb{F}$. Note that a predicate input q is a query-answer pair $(x, y) \in \mathbb{F}^m \times \mathbb{F}$.

- $\mathrm{G}(1^\lambda)$: Sample a commitment key $\mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda)$, and output the public parameters $\mathsf{pp} := \mathsf{ck}$.
- $\mathrm{I}^{\hat{\rho}}(\mathsf{pp} = \mathsf{ck})$: Output $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := (\mathsf{ck}, \mathsf{ck}, 1^\lambda)$.
- $\mathrm{P}^{\hat{\rho}}(\mathsf{apk} = \mathsf{ck}, [\mathsf{q}_i]_{i=1}^k, [\mathsf{acc}_j]_{j=1}^\ell)$:
  1. Sample a random point $x_{n+2\ell+1} \in \mathbb{F}^m$, and set $y_{n+2\ell+1} := \hat{\rho}(x_{n+2\ell+1})$.
  2. Let $Q = [(x_k, y_k)]_{k=1}^{n+2\ell+1}$ be the concatenation of query-answer pairs in the predicate inputs $[\mathsf{q}_i]_{i=1}^k$, old accumulators $[\mathsf{acc}_j]_{j=1}^\ell$, and the above pair $(x_{n+2\ell+1}, y_{n+2\ell+1})$.
  3. Compute the polynomial $g \in (\mathbb{F}[X])^m$ of degree less than $n + 2\ell + 1$ that interpolates the query points in Q over the fixed domain $\{b_k\}_{k=1}^{n+2\ell+1}$ (i.e., such that $g(b_k) = x_k$ for every $k \in [n + 2\ell + 1]$).
  4. Compute the polynomial $f \in \mathbb{F}[X]$ such that $f(X) \equiv \hat{\rho}(g(X))$. Note that the degree of f is less than $m \cdot d \cdot (n + 2\ell + 1)$, and so f can also be computed by interpolation by evaluating the expression $\hat{\rho}(g(X))$ at $m \cdot d \cdot (n + 2\ell + 1)$ points (which involves a corresponding number of queries to $\hat{\rho}$).
  5. Sample commitment randomness ω and then compute the commitment

$$\mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1}), f); \omega) \in \{0,1\}^m \ .$$

6. Compute the challenge $\beta := \hat{\rho}(\mathsf{cm}) \in \mathbb{F}$. (Since $\mathcal{C}$ is Ψ-systematic, any query in $\{0,1\}^m$ to $\hat{\rho} \in \mathcal{C}$ is efficiently embedded in $\mathcal{C}$'s domain using the injection Ψ.) If $\hat{\rho}(\mathsf{cm}) \in \{b_1, \ldots, b_{n+2\ell+1}\}$, go to step 5; except if this has happened λ times, in which case we proceed.
7. Output the new accumulator acc and accumulation proof π_{V} defined as follows:

$$\mathsf{acc} := \big((\mathsf{cm}, \beta), (g(\beta), f(\beta))\big) \ ,$$
$$\pi_{\mathrm{V}} := \big((x_{n+2\ell+1}, y_{n+2\ell+1}), f, \omega\big) \ .$$

- $\mathrm{V}\Big(\mathsf{avk} = \mathsf{ck}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc} = \big((\mathsf{cm}, \beta), (x, y)\big), \pi_{\mathrm{V}} = ((x_{n+2\ell+1}, y_{n+2\ell+1}), f, \omega))\Big)$:

1. Compute the list $Q = [(x_k, y_k)]_{k=1}^{n+2\ell+1}$ and the polynomial g from $[\mathsf{q}_i]_{i=1}^{k}$, $[\mathsf{acc}_j]_{j=1}^{\ell}$, and $(x_{n+2\ell+1}, y_{n+2\ell+1})$ as P does. However, rather than sampling the $(n+2\ell+1)$-th pair, use the pair received in π_{V}.
2. Check that $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1}), f); \omega)$, $x = g(\beta)$, and $y = f(\beta)$.
3. For every $k \in [n+2\ell+1]$, check that $f(b_k) = y_k$.

- $\mathrm{D}^{\hat{\rho}}\Big(\mathsf{dk} = 1^{\lambda}, \mathsf{acc} = \big((x_1, y_1), (x_2, y_2)\big)\Big)$: Check that $\hat{\rho}(x_1) = y_1$ and $\hat{\rho}(x_2) = y_2$.

Proof of Theorem 7.1. We discuss completeness, soundness, zero knowledge, and efficiency.

Completeness. The accumulation prover P receives as input a list of predicate inputs $[\mathsf{q}_i]_{i=1}^{k}$ and a list of old accumulators $[\mathsf{acc}_j]_{j=1}^{\ell}$. Suppose that $\Phi^{\hat{\rho}}(\mathsf{q}_i) = 1$ for every $i \in [n]$ and $\mathrm{D}^{\hat{\rho}}(\mathsf{acc}_j) = 1$ for every $j \in [\ell]$. Completeness requires showing that $\mathrm{P}^{\hat{\rho}}$ outputs a new accumulator acc and accumulation proof π_{V} that satisfy two conditions.

- Condition 1: $\mathrm{V}(\mathsf{avk}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc}, [\mathsf{q}_i]_{i=1}^{k}, \pi_{\mathrm{V}}) = 1$.

  By construction of V, this holds if $\mathsf{acc} = \big((\mathsf{cm}, \beta), (x, y)\big)$ and $\pi_{\mathrm{V}} = ((x_{n+2\ell+1}, y_{n+2\ell+1}), f, \omega)$ are such that (i) $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1}), f); \omega)$; (ii) $x = g(\beta)$ (where V computes g exactly as P does); (iii) $y = f(\beta)$; and (iv) $f(b_k) = y_k$ for every $k \in [n+2\ell+1]$. The accumulation prover $\mathrm{P}^{\hat{\rho}}$ outputs $\mathsf{acc} := \big((\mathsf{cm}, \beta), (g(\beta), f(\beta))\big)$. Thus, Conditions ii and iii are satisfied directly. Condition i is satisfied because V computes cm using the same inputs and commmitment key (since $\mathsf{apk} = \mathsf{avk} = \mathsf{ck}$) as $\mathrm{P}^{\hat{\rho}}$. For Condition iv: $g(b_k) = x_k$ for every $k \in [n+2\ell+1]$ by definition of g; thus, $\hat{\rho}(g(b_k)) = \hat{\rho}(x_k) = y_k$ for every $k \in [n+2\ell+1]$; we defined $f := \hat{\rho} \circ g$, so $f(b_k) = y_k$ for every $k \in [n+2\ell+1]$.

- Condition 2: $\mathrm{D}^{\hat{\rho}}(\mathsf{acc}) = 1$.

  This occurs when both points in acc are valid query-answer pairs. In the first pair, the honest $\mathrm{P}^{\hat{\rho}}$ sets β to $\hat{\rho}(\mathsf{cm})$. For the second pair, $(g(\beta), f(\beta))$ satisfies $\hat{\rho}(g(\beta)) = f(\beta)$ because $f \equiv \hat{\rho} \circ g$.

Soundness. Since AS is an accumulation scheme for oracle queries, we show that the probability below is bounded from above by the expression in the theorem statement:

$$\Pr\left[\begin{array}{c|r} \mathrm{V}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc}, \pi_{\mathrm{V}}) = 1 & \hat{\rho} \leftarrow \mathcal{C} \\ \mathrm{D}^{\hat{\rho}}(\mathsf{dk}, \mathsf{acc}) = 1 & \mathsf{pp} \leftarrow \mathrm{G}(1^{\lambda}) \\ \wedge & \begin{pmatrix} [\mathsf{q}_i]_{i=1}^{k} & [\mathsf{acc}_j]_{j=1}^{\ell} \\ \mathsf{acc} & \pi_{\mathrm{V}} \end{pmatrix} \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{pp}) \\ \exists\, j \in [\ell],\ \mathrm{D}^{\hat{\rho}}(\mathsf{dk}, \mathsf{acc}_j) = 0 \vee & \\ \exists\, i \in [n],\ \Phi^{\hat{\rho}}(\mathsf{q}_i) = 0 & (\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) \leftarrow \mathrm{I}^{\hat{\rho}}(\mathsf{pp}) \end{array}\right] . \tag{5}$$

Since $\mathsf{i}_{\Phi} = \perp$ and $\mathsf{pp}_{\Phi} = \perp$ in oracle query accumulation schemes, we omit them from the above equation.

In the above equation, let $\mathsf{acc} = \big((\mathsf{cm}, \beta), (x, y)\big)$ and $\pi_{\mathrm{V}} = ((x_{n+2\ell+1}, y_{n+2\ell+1}), f, \omega)$.

We argue that the event in the left side of Equation 5, which we denote by E, is equivalent to the condition "$f(X) \not\equiv \hat{\rho}(g(X))$ and $f(\beta) = \hat{\rho}(g(\beta))$".

- First, we show that $f(\beta) = \hat{\rho}(g(\beta))$. Note that
  - $\mathrm{D}^{\hat{\rho}}(\mathsf{dk}, \mathsf{acc}) = 1$ implies that $\hat{\rho}(x) = y$, and
  - $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$ implies that $x = g(\beta)$ and $y = f(\beta)$.

  Then substitution gives $\hat{\rho}(g(\beta)) = f(\beta)$.

- Second, we show that $f(X) \not\equiv \hat{\rho}(g(X))$. Consider the expression

$$\exists\, j \in [\ell],\ \mathrm{D}^{\hat{\rho}}(\mathsf{dk}, \mathsf{acc}_j) = 0\ \vee \exists\, i \in [n],\ \Phi^{\hat{\rho}}(\mathsf{pp}_\Phi, \mathsf{i}_\Phi, \mathsf{q}_i) = 0\ \ .$$

  This means there is a query $\mathsf{q}_{k^*} = (x_{k^*}, y_{k^*}) \in [\mathsf{q}_i]_{i=1}^{k} \cup [\mathsf{acc}_j]_{j=1}^{\ell} \subseteq Q$ such that $\hat{\rho}(x_{k^*}) \neq y_{k^*}$, for some $k^* \in [n + 2\ell + 1]$. By g's definition, we have $x_k = g(b_k)$ for every $k \in [n + 2\ell + 1]$, so $\hat{\rho}(g(b_{k^*})) \neq y_{k^*}$. Also since $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$, we have $f(b_k) = y_k$ for every $k \in [n+2\ell+1]$. Hence, we get $f(b_{k^*}) \neq \hat{\rho}(g(b_{k^*}))$, and thus conclude that $f(X) \not\equiv \hat{\rho}(g(X))$.

Since the low-degree random oracle has efficient constraint detection (Theorem 4.17), any adversary in the LDROM that breaks CM can also break CM in the standard model (by simulating the oracle). Hence CM is a secure commitment scheme in the LDROM.

Next, we wish to invoke our oracle zero-finding game lemma (Lemma 6.1). We apply Lemma 6.1 with respect to the commitment scheme CM' obtained by modifying CM as follows: $\mathsf{CM}'.\mathsf{Commit}(\mathsf{ck}, (g, f); \omega) = \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, ((g(b_1), \ldots, g(b_{n+2\ell+1})), f); \omega)$. Note that this is binding to g since g is of degree less than $n + 2\ell$. The winning event in the zero-finding game is then that

$$\mathsf{cm} = \mathsf{CM}'.\mathsf{Commit}(\mathsf{ck}, (g, f); \omega) = \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1})\ , f); \omega),$$

$f(X) \not\equiv \hat{\rho}(g(X))$, and $f(\beta) = \hat{\rho}(g(\beta))$ for $\beta := \hat{\rho}(\mathsf{cm})$. Note that $\mathrm{V}(\mathsf{avk}, [\mathsf{q}_i]_{i=1}^{k}, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$ implies that $\mathsf{cm} = \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1}), f); \omega)$. Note also that $\deg(g) \leq n + 2\ell$. Thus, by our oracle zero-finding game lemma, the probability that $f(X) \not\equiv \hat{\rho}(g(X))$ and $f(\beta) = \hat{\rho}(g(\beta))$ is at most

$$\sqrt{t \cdot \left(\frac{2dm(n + 2\ell) + 1}{|\mathbb{F}|}\right)} + \frac{1}{|\mathbb{F}|} + \mathrm{negl}(\lambda)\ \ .$$

Since the event E is equivalent to $f(X) \not\equiv \hat{\rho}(g(X))$ and $f(\beta) = \hat{\rho}(g(\beta))$, the above probability is an upper bound on the probability in Equation 5, as desired.

Zero knowledge. We describe a zero-knowledge simulator S that has access to $\hat{\rho}$ and simulates: (i) the scheme's public parameters pp; and (ii) the distribution of P's accumulator acc without access to P's inputs (the predicate inputs $[\mathsf{q}_i]_{i=1}^{k}$ and old accumulators $[\mathsf{acc}_j]_{j=1}^{\ell}$). Then we show that if the commitment CM is statistically (resp. computationally) hiding, then the simulated joint distribution $(\hat{\rho}, \mathsf{pp}, \mathsf{acc})$ is statistically (resp. computationally) indistinguishable from $(\hat{\rho}, \mathsf{pp}, \mathsf{acc})$ produced in the real protocol.

- *Parameter generation:* $\mathrm{S}(1^\lambda) \to \mathsf{pp}$.
  1. Sample $\mathsf{ck} \leftarrow \mathsf{CM}.\mathsf{Setup}(1^\lambda)$.
  2. Output $\mathsf{pp} := \mathsf{ck}$ (and store ck in the internal state).

- *Proving:* $\mathrm{S}^{\hat{\rho}}(\mathsf{pp}_\Phi = \bot, \mathsf{i}_\Phi = \bot) \to \mathsf{acc}$.
  1. Sample commitment randomness ω and compute a commitment $\mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, ((0, \dots, 0), f'); \omega)$ where $f' \in \mathbb{F}[X]$ is the zero polynomial (appropriately padded).
  2. Query the oracle to compute $\beta := \hat{\rho}(\mathsf{cm})$. If $\beta \in \{b_1, \dots, b_{n+2\ell}\}$, resample ω and recompute cm until this is not the case. Abort if we resample more than $\kappa := \lambda/(\log|\mathbb{F}| - \log(n+2\ell))$ times.[14]
  3. Sample a random point $x \in \mathbb{F}^m$.
  4. Query the oracle to compute $y := \hat{\rho}(x)$.
  5. Output the accumulator $\mathsf{acc} := \big((\mathsf{cm}, \beta), (x, y)\big)$.

Observe that the probability that the simulator aborts in Step 2 is at most $\mathrm{negl}(\lambda)$. By Claim 3.4, with all but negligible probability, all of the sampled cm are distinct; hence the probability that $\hat{\rho}(\mathsf{cm}) \in \{b_1, \dots, b_{n+2\ell}\}$ for all sampled cm is at most $\big(\frac{n+2\ell}{|\mathbb{F}|}\big)^\kappa + \mathrm{negl}(\lambda) = \mathrm{negl}(\lambda)$.

We argue that $(\hat{\rho}, \mathsf{pp}, \mathsf{acc})$ above is indistinguishable from that produced by the accumulation prover. Since S does not program any oracle points, we fix an oracle $\hat{\rho} \leftarrow \mathcal{C}$, then argue that pp and acc output by S are distributed correctly, given $\hat{\rho}$.

First, the real and simulated pp have the same distribution: they are a commitment key $\mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda)$.

Next, we show that the real and simulated $\mathsf{acc} = \big((\mathsf{cm}, \beta), (x, y)\big)$ are indistinguishable. It suffices to argue the indistinguishability of cm and x (between the real protocol and the simulation), because $\beta = \hat{\rho}(\mathsf{cm})$ and $y = \hat{\rho}(x)$ in both the real protocol and the simulation.

- cm: Since CM is (statistically/computationally) hiding, cm is (statistically/computationally) indistinguishable from a commitment to any other message of the same length. This is true even if the adversary is given many independent commitments to the same message. Note that apart from the choice of message, the prover and simulator sample cm in the same way.
- x: Since $\beta \notin \{b_1, \dots, b_{n+2\ell}\}$ (else the simulator aborts), in the real protocol, as in the simulation, $g(\beta)$ is uniformly random in $\mathbb{F}^m$. This follows from a standard algebraic fact, stated below.

Fact 7.3. Let $b_1, \dots, b_{N+1} \in \mathbb{F}$ be distinct field elements, and let $x_1, \dots, x_N \in \mathbb{F}^m$. Sample $x_{N+1} \leftarrow \mathbb{F}^m$ uniformly and construct g to be the (unique) interpolation of the points $(b_1, x_1), \dots, (b_{N+1}, x_{N+1}) \in \mathbb{F} \times \mathbb{F}^m$ of minimal degree. Then for any point $\beta \leftarrow \mathbb{F} \setminus \{b_1, \dots, b_N\}$, $g(\beta)$ is uniformly random in $\mathbb{F}^m$.

Efficiency. We discuss the efficiency of Construction 1.

- *Generator.* Efficiency follows from the efficiency of $\mathsf{CM.Setup}$, which takes $\mathrm{poly}(\lambda)$ time.

- *Indexer.* This takes $\mathrm{poly}(\lambda)$ time.

- *Accumulation prover.* Running $\mathrm{P}^{\hat{\rho}}$ involves computing the polynomial g via polynomial interpolation, committing to (f, g), evaluating g at $n + 2\ell + 1$ points, computing the polynomial f via making $n + 2\ell + 1$ queries to $\hat{\rho}$ then interpolating. These are all operations that can be accomplished in time polynomial in λ.

- *Accumulator size.* Consider the first point $(\mathsf{cm}, \hat{\rho}(\mathsf{cm}))$. Since the commitment scheme CM has commitment size m, we know that $\mathsf{cm} \in \{0,1\}^m \subseteq \mathbb{F}^m$. Then, the point $(\mathsf{cm}, \hat{\rho}(\mathsf{cm}))$ is in $\mathbb{F}^m \times \mathbb{F}$. The second point of acc is $(g(\beta), f(\beta))$, which is in $\mathbb{F}^m \times \mathbb{F}$ due to the definitions of f, g.

- *Accumulation proof size.* The accumulation proof π_{V} includes of a single-variate polynomial f of degree at most $m \cdot d \cdot (n + 2\ell + 1)$, which can be represented with $d \cdot (n + 2\ell + 1) + 1$ elements of $\mathbb{F}$. It also includes commitment randomness, which is a bitstring of $\mathrm{poly}(\lambda)$ length.

[14]If the field $\mathbb{F}$ is of superpolynomial size (and n, ℓ are polynomially bounded) then resampling is not necessary.

- *Accumulation verifier.* The accumulation verifier V computes the polynomial g via interpolation, a single evaluation of g, $n+2\ell+1$ evaluations of f, and the commitment $\mathsf{CM.Commit}(\mathsf{ck}, ((x_1, \ldots, x_{n+2\ell+1}), f); \omega)$. Note that V makes no oracle queries.
- *Decider.* $\mathrm{D}^{\hat{\rho}}$ makes 2 queries to $\hat{\rho}$. $\square$

8 SNARKs for low-degree oracle computations

We prove that there exist SNARKs in the low-degree random oracle model for low-degree oracle computations.

Theorem 8.1 (restatement of Theorem 1)**.** *Let $\mathscr{C} = \mathrm{LD}[\mathscr{F}, m, d]$ be a low-degree random oracle, where $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of fields, $m \colon \mathbb{N} \to \mathbb{N}$ an arity function, and $d \colon \mathbb{N} \to \mathbb{N}$ a degree function (see Definition 4.16) such that $m(\lambda) \geq 2 \log |\mathbb{F}_\lambda|$ and $|\mathbb{F}_\lambda| = \lambda^{\omega(1)}$. If (standard model) collision-resistant functions exist, then there exists a transparent (zero-knowledge) SNARK relative to $\mathscr{C}$ for $\mathsf{NP}^{\mathscr{C}}$.*

The theorem follows immediately from the following three steps. First, in Section 8.1 we prove that, for any given oracle distribution $\mathcal{O}$, we can construct a SNARK in the $\mathcal{O}$-model for $\mathcal{O}$-oracle computations given (i) a SNARK in the $\mathcal{O}$-model for non-oracle computations, and (ii) an accumulation scheme for $\mathcal{O}$-queries. Second, in Section 8.2 we prove that there exist SNARKs for non-oracle computations in any linear code random oracle model with efficient constraint detection. Third, we instantiate the accumulation scheme from the first step with the one for low-degree random oracles in Section 7; this allows us to set $\mathcal{O} := \mathrm{LD}[\mathscr{F}, m, d]$.

8.1 SNARKs for oracle computations via query accumulation

We describe a generic SNARK construction in any oracle model from a SNARK for non-oracle computations and an accumulation scheme for queries to the oracle in that model.

Lemma 8.2. *Let $\mathcal{O}$ be an oracle distribution. Suppose that:*
- $\mathsf{ARG_{in}} = (\mathcal{G}_{\mathsf{in}}, \mathcal{I}_{\mathsf{in}}, \mathcal{P}_{\mathsf{in}}, \mathcal{V}_{\mathsf{in}})$ *is a SNARK in the $\mathcal{O}$-oracle model for* NP *computations (with no oracles), and*
- $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ *is an accumulation scheme for $\mathcal{O}$-queries (in particular,* V *makes no oracle query).*

Then $\mathsf{ARG_{out}} = (\mathcal{G}_{\mathsf{out}}, \mathcal{I}_{\mathsf{out}}, \mathcal{P}_{\mathsf{out}}, \mathcal{V}_{\mathsf{out}})$ *in Construction 2 is a SNARK relative to $\mathcal{O}$ for* $\mathsf{NP}^{\mathcal{O}}$*. Moreover, if* $\mathsf{ARG_{in}}$ *is zero-knowledge and* AS *is zero-knowledge, then* $\mathsf{ARG_{out}}$ *is zero-knowledge.*

Construction 2. Fix a security parameter $\lambda \in \mathbb{N}$ and an oracle θ in the support of the oracle distribution $\mathcal{O}(\lambda)$. Let $\mathcal{R}^\theta = \{(\mathbb{i}, \mathbb{x}, \mathbb{w}) : M^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 1\} \in \mathsf{NP}^\theta$ be an oracle (indexed) relation. We define a non-oracle (indexed) relation $\mathcal{R}_{\mathsf{in}}$ capturing the same computation:

$$\mathcal{R}_{\mathsf{in}} := \left\{ \big((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})\big) : \begin{array}{l} M_\circ(\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1 \\ \wedge\ \mathrm{V}(\mathsf{avk}, \mathsf{tr}, \bot, \mathsf{acc}, \pi_{\mathrm{V}}) = 1 \end{array} \right\} \in \mathsf{NP} \enspace ,$$

where $M_\circ$ works like M except that oracle queries are answered using the query-answer transcript tr, V is the verifier of the accumulation scheme AS, and $\bot$ denotes an empty (old) accumulator. Note that $M^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w}) = M_\circ(\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{tr})$ if tr is the correct query-answer transcript for the oracle θ.

We construct a SNARK $\mathsf{ARG_{out}}$ for $\mathcal{R}^\theta$ from a SNARK $\mathsf{ARG_{in}}$ for $\mathcal{R}_{\mathsf{in}}$ and AS, as follows.

- $\mathcal{G}_{\mathsf{out}}(1^\lambda)$:
  1. Sample public parameters for the accumulation scheme: $\mathsf{pp_{AS}} \leftarrow \mathrm{G}(1^\lambda)$.
  2. Sample public parameters for the SNARK: $\mathsf{pp_{in}} \leftarrow \mathcal{G}_{\mathsf{in}}(1^\lambda)$.
  3. Output the public parameters $\mathsf{pp_{out}} := (\mathsf{pp_{AS}}, \mathsf{pp_{in}})$.
- $\mathcal{I}^\theta_{\mathsf{out}}(\mathsf{pp_{out}} = (\mathsf{pp_{AS}}, \mathsf{pp_{in}}), \mathbb{i})$:
  1. Compute the accumulation keys: $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}^\theta(\mathsf{pp_{AS}})$.
  2. Compute the SNARK keys: $(\mathsf{ipk_{in}}, \mathsf{ivk_{in}}) := \mathcal{I}^\theta_{\mathsf{in}}(\mathsf{pp_{in}}, (\mathbb{i}, \mathsf{avk}))$.
  3. Set the proving key $\mathsf{ipk_{out}} := (\mathsf{apk}, \mathsf{ipk_{in}})$.

4. Set the verification key $\mathsf{ivk}_{\mathsf{out}} := (\mathsf{dk}, \mathsf{ivk}_{\mathsf{in}})$.
5. Output the key pair $(\mathsf{ipk}_{\mathsf{out}}, \mathsf{ivk}_{\mathsf{out}})$.

- $\mathcal{P}^{\theta}_{\mathsf{out}}(\mathsf{ipk}_{\mathsf{out}} = (\mathsf{apk}, \mathsf{ipk}_{\mathsf{in}}), \mathbb{x}, \mathbb{w})$:
  1. Run $M^{\theta}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and record the transcript tr of queries to θ and corresponding answers.
  2. Compute an accumulator for these queries: $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\theta}(\mathsf{apk}, \mathsf{tr}, \perp)$.
  3. Compute a SNARK: $\pi_{\mathsf{in}} \leftarrow \mathcal{P}^{\theta}_{\mathsf{in}}(\mathsf{ipk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}}))$.
  4. Output $\pi_{\mathsf{out}} := (\mathsf{acc}, \pi_{\mathsf{in}})$.

- $\mathcal{V}^{\theta}_{\mathsf{out}}\big(\mathsf{ivk}_{\mathsf{out}} = (\mathsf{dk}, \mathsf{ivk}_{\mathsf{in}}), \mathbb{x}, \pi_{\mathsf{out}} = (\mathsf{acc}, \pi_{\mathsf{in}})\big)$:
  1. Check that $\mathrm{D}^{\theta}(\mathsf{dk}, \mathsf{acc}) = 1$.
  2. Check that $\mathcal{V}^{\theta}_{\mathsf{in}}(\mathsf{ivk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$.

Proof of Lemma 8.2. We discuss completeness, knowledge soundness, and zero-knowledge.

Completeness. Fix an oracle $\theta \in \mathcal{O}(\lambda)$, $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\theta}$, public parameters $\mathsf{pp}_{\mathsf{out}} \in \mathcal{G}_{\mathsf{out}}(1^{\lambda})$, and SNARK keys $(\mathsf{ipk}_{\mathsf{out}}, \mathsf{ivk}_{\mathsf{out}}) \in \mathcal{I}^{\theta}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}}, \mathbb{i})$. We argue that for $\pi_{\mathsf{out}} \leftarrow \mathcal{P}^{\theta}_{\mathsf{out}}(\mathsf{ipk}_{\mathsf{out}}, \mathbb{x}, \mathbb{w})$ it holds that $\mathcal{V}^{\theta}_{\mathsf{out}}(\mathsf{ivk}_{\mathsf{out}}, \mathbb{x}, \pi_{\mathsf{out}}) = 1$ with probability 1, as follows.

- $\mathrm{D}^{\theta}(\mathsf{dk}, \mathsf{acc}) = 1$:

  The prover $\mathcal{P}^{\theta}_{\mathsf{out}}$ runs the accumulation prover $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\theta}(\mathsf{apk}, \mathsf{tr}, \perp)$ where tr is the (correct) query-answer transcript of the computation $M^{\theta}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ (and the list of old accumulators is set to $\perp$). By the completeness of the accumulation scheme AS (which is for accumulating queries to θ), we know that $\mathrm{D}^{\theta}(\mathsf{dk}, \mathsf{acc}) = 1$ and $\mathrm{V}(\mathsf{avk}, \mathsf{tr}, \perp, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$ (we use this latter fact below).

- $\mathcal{V}^{\theta}_{\mathsf{in}}(\mathsf{ivk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$:

  The prover $\mathcal{P}^{\theta}_{\mathsf{out}}$ runs the SNARK prover $\pi_{\mathsf{in}} \leftarrow \mathcal{P}^{\theta}_{\mathsf{in}}(\mathsf{ipk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}}))$, where $(\mathsf{ipk}_{\mathsf{in}}, \mathsf{ivk}_{\mathsf{in}})$ are a SNARK key pair computed by $\mathcal{I}^{\theta}_{\mathsf{in}}$ on input $(\mathsf{pp}_{\mathsf{in}}, (\mathbb{i}, \mathsf{avk}))$. Note that $((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})) \in \mathcal{R}_{\mathsf{in}}$ because: (i) $M_{\circ}(\mathbb{i}, \mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1$ since $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\theta}$ and tr contains M^{θ}'s correct query-answer transcript; and (ii) $\mathrm{V}(\mathsf{avk}, \mathsf{tr}, \perp, \mathsf{acc}, \pi_{\mathrm{V}}) = 1$ (by the previous bullet point). By the completeness of the SNARK $\mathsf{ARG}_{\mathsf{in}}$, we know that $\mathcal{V}^{\theta}_{\mathsf{in}}(\mathsf{ivk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$.

Knowledge soundness. Suppose that $\mathsf{ARG}_{\mathsf{in}}$ has knowledge soundness error $\varepsilon_{\mathsf{in}}(\lambda)$ and AS has soundness error $\varepsilon_{\mathsf{AS}}(\lambda)$. We argue that $\mathsf{ARG}_{\mathsf{out}}$ has knowledge soundness error $\varepsilon_{\mathsf{in}}(\lambda) + \varepsilon_{\mathsf{AS}}(\lambda)$.

We explain how to transform an adversary $\tilde{\mathcal{P}}_{\mathsf{out}}$ for $\mathsf{ARG}_{\mathsf{out}}$ for auxiliary-input distribution $\mathcal{D}_{\mathsf{out}}$ into an adversary $\tilde{\mathcal{P}}_{\mathsf{in}}$ for $\mathsf{ARG}_{\mathsf{in}}$ for a related auxiliary-input distribution $\mathcal{D}_{\mathsf{in}}$.

- $\mathcal{D}_{\mathsf{in}}$ equals $\mathcal{D}_{\mathsf{out}}$ augmented with public parameters for the accumulation scheme AS: $\mathcal{D}_{\mathsf{in}}(\mathsf{pp}_{\mathsf{in}})$ samples $\mathsf{pp}_{\mathsf{AS}} \in \mathrm{G}(1^{\lambda})$, sets $\mathsf{pp}_{\mathsf{out}} := (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{in}})$, samples $\mathsf{ai}_{\mathsf{out}} \leftarrow \mathcal{D}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}})$, and outputs $\mathsf{ai}_{\mathsf{in}} := (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}_{\mathsf{out}})$.

- The adversary $\tilde{\mathcal{P}}_{\mathsf{in}}$ receives query access to $\theta \in \mathcal{O}(\lambda)$ and input $(\mathsf{pp}_{\mathsf{in}}, \mathsf{ai}_{\mathsf{in}})$ where $\mathsf{pp}_{\mathsf{in}} \leftarrow \mathcal{G}_{\mathsf{in}}(1^{\lambda})$ and $\mathsf{ai}_{\mathsf{in}} = (\mathsf{pp}_{\mathsf{AS}}, \mathsf{ai}_{\mathsf{out}}) \in \mathcal{D}_{\mathsf{in}}(\mathsf{pp}_{\mathsf{in}})$ and works as follows.

```
$\tilde{\mathcal{P}}^{\theta}_{\mathsf{in}}(\mathsf{pp}_{\mathsf{in}}, \mathsf{ai}_{\mathsf{in}})$:
1. Run $(\mathbb{i}, \mathbb{x}, \pi_{\mathsf{out}}) \leftarrow \tilde{\mathcal{P}}^{\theta}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{ai}_{\mathsf{out}})$ for $\mathsf{pp}_{\mathsf{out}} := (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{in}})$.
2. Parse $\pi_{\mathsf{out}}$ as $(\mathsf{acc}, \pi_{\mathsf{in}})$.
3. Run the accumulation indexer $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}^{\theta}(\mathsf{pp}_{\mathsf{AS}})$.
4. Output $((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}})$.
```

Let $\mathcal{E}_{\mathsf{in}}$ be an extractor (for the auxiliary-input distribution $\mathcal{D}_{\mathsf{in}}$) for $\mathsf{ARG}_{\mathsf{in}}$. Below we construct an extractor $\mathcal{E}_{\mathsf{out}}$ (for the auxiliary-input distribution $\mathcal{D}_{\mathsf{out}}$) for $\mathsf{ARG}_{\mathsf{out}}$.

```
$\mathcal{E}_{\mathsf{out}}^{\theta,\tilde{\mathcal{P}}_{\mathsf{out}}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{ai}_{\mathsf{out}})$:
1. Construct the algorithm $\tilde{\mathcal{P}}_{\mathsf{in}}$ from the algorithm $\tilde{\mathcal{P}}_{\mathsf{out}}$ as above (black-box access suffices).
2. Parse $\mathsf{pp}_{\mathsf{out}}$ as $(\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{in}})$.
3. Run $(\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}}) \leftarrow \mathcal{E}_{\mathsf{in}}^{\theta,\tilde{\mathcal{P}}_{\mathsf{in}}}(\mathsf{pp}_{\mathsf{in}}, \mathsf{ai}_{\mathsf{in}})$.
4. Output $\mathbb{w}$.
```

We argue that the probability that $\mathcal{V}_{\mathsf{out}}(\mathsf{ivk}_{\mathsf{out}}, \mathbb{x}, \pi_{\mathsf{out}}) = 1$ and $\mathcal{E}_{\mathsf{out}}^{\theta,\tilde{\mathcal{P}}_{\mathsf{out}}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{ai}_{\mathsf{out}})$ fails to output a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\theta}$ is at most $\varepsilon_{\mathsf{in}}(\lambda) + \varepsilon_{\mathsf{AS}}(\lambda)$.We first define the following events.

- E_1: $\mathcal{V}_{\mathsf{out}}^{\theta}(\mathsf{ivk}_{\mathsf{out}}, \mathbb{x}, \pi_{\mathsf{out}}) = 1$.
- E_2: $((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})) \notin \mathcal{R}_{\mathsf{in}}$.
- E_3: tr is inconsistent with θ.

Note that if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}^{\theta}$ then E_2 or E_3 holds (because $M_{\circ}$ agrees with M when the oracle transcript tr is consistent with θ). Hence it suffices to upper bound the probability:

$$\Pr[E_1 \wedge (E_2 \vee E_3)] \leq \Pr[E_1 \wedge E_2] + \Pr[E_1 \wedge E_3] \ .$$

We bound each term separately.

- $\Pr[E_1 \wedge E_2]$: By the knowledge soundness of $\mathsf{ARG}_{\mathsf{in}}$, we have that $\mathcal{V}_{\mathsf{in}}^{\theta}(\mathsf{ivk}_{\mathsf{in}}, (\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$ (E_1 holds in particular) and $((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})) \notin \mathcal{R}_{\mathsf{in}}$ (E_2 holds) with probability at most $\varepsilon_{\mathsf{in}}(\lambda)$.

- $\Pr[E_1 \wedge E_3]$: By the soundness of AS, we have that $\mathrm{D}^{\theta}(\mathsf{dk}, \mathsf{acc}) = 1$ (E_1 holds in particular) and tr is inconsistent with θ (E_3 holds) with probability at most $\varepsilon_{\mathsf{AS}}(\lambda)$.

Zero knowledge. We argue that if $\mathsf{ARG}_{\mathsf{in}}$ is zero knowledge and AS is zero knowledge then so is $\mathsf{ARG}_{\mathsf{out}}$.

Consider the simulator $\mathcal{S}_{\mathsf{out}}$ for $\mathsf{ARG}_{\mathsf{out}}$ defined as follows.

- *Parameter generation:* $\mathcal{S}_{\mathsf{out}}(1^{\lambda}) \rightarrow \mathsf{pp}_{\mathsf{out}}$.
  1. Simulate public parameters for AS: $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{S}_{\mathsf{AS}}(1^{\lambda})$.
  2. Simulate public parameters for $\mathsf{ARG}_{\mathsf{in}}$: $\mathsf{pp}_{\mathsf{in}} \leftarrow \mathcal{S}_{\mathsf{in}}(1^{\lambda})$.
  3. Output $\mathsf{pp}_{\mathsf{out}} := (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{in}})$.

- *Proving:* $\mathcal{S}_{\mathsf{out}}^{\theta}(\mathbb{i}, \mathbb{x}) \rightarrow \pi_{\mathsf{out}}$.
  1. Run the AS indexer to compute accumulation keys: $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}^{\theta}(\mathsf{pp}_{\mathsf{AS}})$.
  2. Run the AS simulator to sample an accumulator: $\mathsf{acc} \leftarrow \mathrm{S}_{\mathsf{AS}}^{\theta}(\mathsf{pp}_{\Phi} = \bot, \mathsf{i}_{\Phi} = \bot)$.
  3. Run the $\mathsf{ARG}_{\mathsf{in}}$ simulator to sample a SNARK: $\pi_{\mathsf{in}} \leftarrow \mathcal{S}_{\mathsf{in}}^{\theta}((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}))$.
  4. Output $\pi_{\mathsf{out}} := (\mathsf{acc}, \pi_{\mathsf{in}})$.

- *Query responses:* $\mathcal{S}_{\mathsf{out}}^{\theta}(x) \rightarrow y$.
  1. Run $y \leftarrow \mathcal{S}_{\mathsf{in}}^{\theta}(x)$ and output y.

To prove zero knowledge we consider the following hybrid simulator.

- *Parameter generation:* $\mathcal{S}_{\mathsf{hyb}}(1^{\lambda}) \rightarrow \mathsf{pp}_{\mathsf{out}}$.

1. Run the AS generator to get the accumulation scheme's public parameters $\mathsf{pp}_{\mathsf{AS}} \leftarrow \mathrm{G}(1^\lambda)$.
2. Simulate public parameters for $\mathsf{ARG}_{\mathsf{in}}$: $\mathsf{pp}_{\mathsf{in}} \leftarrow \mathcal{S}_{\mathsf{in}}(1^\lambda)$.
3. Output $\mathsf{pp}_{\mathsf{out}} := (\mathsf{pp}_{\mathsf{AS}}, \mathsf{pp}_{\mathsf{in}})$.

- *Proving:* $\mathcal{S}^\theta_{\mathsf{hyb}}(\mathbb{i}, \mathbb{x}, \mathbb{w}) \to \pi_{\mathsf{out}}$.

1. Run $M^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and record the transcript tr of queries to θ and the corresponding answers.
2. Run the AS indexer to compute the accumulation keys: $(\mathsf{apk}, \mathsf{avk}, \mathsf{dk}) := \mathrm{I}^\theta(\mathsf{pp}_{\mathsf{AS}})$.
3. Compute an accumulator: $(\mathsf{acc}, \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^\theta(\mathsf{apk}, \mathsf{tr}, \bot)$.
4. Simulate a SNARK proof: $\pi_{\mathsf{in}} \leftarrow \mathcal{S}^\theta_{\mathsf{in}}((\mathbb{i}, \mathsf{avk}), (\mathbb{x}, \mathsf{acc}))$.
5. Output $\pi_{\mathsf{out}} := (\mathsf{acc}, \pi_{\mathsf{in}})$.

- *Query responses:* $\mathcal{S}^\theta_{\mathsf{hyb}}(x) \to y$.

1. Run $y \leftarrow \mathcal{S}^\theta_{\mathsf{in}}(x)$ and output y.

Let $\mathcal{A}$ be a polynomial-size honest stateful adversary (per the definition of zero knowledge in Section 3.2). We define three hybrids.

- $\mathbf{H}_0$ (real experiment): The distribution of $\mathcal{A}^\theta(\pi_{\mathsf{out}})$ when interacting with $\mathcal{P}_{\mathsf{out}}$ (the LHS of Eq. 1).
- $\mathbf{H}_1$ (hybrid experiment): The distribution of $\mathcal{A}^{\mathcal{S}_{\mathsf{hyb}}}(\pi_{\mathsf{out}})$ when interacting with $\mathcal{S}_{\mathsf{hyb}}$.
- $\mathbf{H}_2$ (simulated experiment): The distribution of $\mathcal{A}^{\mathcal{S}_{\mathsf{out}}}(\pi_{\mathsf{out}})$ when interacting with $\mathcal{S}_{\mathsf{out}}$ (the RHS of Eq. 1).

Since $\mathcal{A}$ is honest, by the completeness of AS, $\mathcal{A}$ induces an honest adversary for $\mathsf{ARG}_{\mathsf{in}}$. Hence by the zero-knowledge property of $\mathsf{ARG}_{\mathsf{in}}$, $\mathbf{H}_0$ and $\mathbf{H}_1$ are $\mathrm{negl}(\lambda)$-close in statistical distance.

Since tr is generated using θ, $\mathcal{S}_{\mathsf{hyb}}$ induces an honest adversary for AS. Hence $\mathbf{H}_1$ and $\mathbf{H}_2$ are $\mathrm{negl}(\lambda)$-close in statistical distance by the zero-knowledge property of AS.

We conclude that the statistical distance between $\mathbf{H}_0$ and $\mathbf{H}_2$ is negligible. $\square$

Remark 8.3. We do not know whether if we merely assume that $\mathsf{ARG}_{\mathsf{in}}$ is a SNARG then we can conclude that $\mathsf{ARG}_{\mathsf{out}}$ in Construction 2 is a SNARG. In the above proof, we establish soundness of $\mathsf{ARG}_{\mathsf{out}}$ by efficiently extracting the query-answer transcript tr, which we then use to construct an efficient adversary against the computational soundness property of AS. But if we merely assume that $\mathsf{ARG}_{\mathsf{in}}$ is (computationally) sound then it is not clear how to appeal to the computational soundness of AS. More generally, we leave it as an open question to construct a SNARG relative to $\mathcal{O}$ for $\mathsf{NP}^{\mathcal{O}}$ from a SNARG relative to $\mathcal{O}$ for NP (and a suitable accumulation scheme for $\mathcal{O}$).

8.2 SNARKs for non-oracle computations in any oracle model

We show that there exists a SNARK for *non-oracle* computations in any linear code random oracle model with efficient constraint detection.

Lemma 8.4. *Let $\mathscr{C} = \{\mathcal{C}_\lambda \subseteq (D_\lambda \to \mathbb{F}_\lambda)\}_{\lambda \in \mathbb{N}}$ be a family of linear codes with efficient constraint detection where $|\mathbb{F}_\lambda| = \lambda^{\omega(1)}$. Suppose moreover that $\mathscr{C}$ is systematic with arity m for $m(\lambda) \geq 2 \log |\mathbb{F}_\lambda|$. Then there exists a SNARK relative to $\mathscr{C}$ for any $\mathcal{R} \in \mathsf{NP}$ (unconditionally).*

We prove Lemma 8.4 by instantiating the construction below with: (i) a PCP of knowledge with polynomial proof length and knowledge soundness error $\mathrm{negl}(\lambda)$ (e.g. from [BFLS91]); and (ii) the unconditional vector commitment in the $\mathscr{C}$-oracle model guaranteed by Lemma 8.6.

Construction 3 (Micali SNARK [Mic00] in a linear code random oracle model)**.** Fix a security parameter $\lambda \in \mathbb{N}$ and an oracle $\hat{\rho}$ in the support of $\mathcal{C}_\lambda$. Let $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ be a (holographic) PCP with proof length ℓ. The (preprocessing) Micali SNARK $\mathsf{ARG}_{\mathsf{Mic}} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is constructed as follows.

For simplicity we assume that $\mathbb{F}$ is sufficiently large to encode the PCP randomness r; if not, we can obtain additional randomness from the oracle in some canonical way. Throughout the construction, we use $\hat{\rho}$ as a function $\hat{\rho}\colon \{0,1\}^m \to \mathbb{F}$ (i.e., there is no need to query $\hat{\rho}$ outside of the systematic part); all calls to $\hat{\rho}$ should therefore be interpreted as calls to $\hat{\rho} \circ \psi$ for the canonical injection $\psi\colon \{0,1\}^m \to D$.

- $\mathcal{G}(1^\lambda)$:
  1. Sample public parameters for the vector commitment scheme: $\mathsf{pp}_{\mathsf{VC}} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^\ell)$.
  2. Output the public parameters $\mathsf{pp} := \mathsf{pp}_{\mathsf{VC}}$.
- $\mathcal{I}^{\hat{\rho}}(\mathsf{pp}, \mathbb{i})$:
  1. Compute the encoded index $\mathbf{I}(\mathbb{i})$.
  2. Compute a (vector) commitment $\mathsf{cm}_0 := \mathsf{VC.Commit}^{\hat{\rho}}(\mathsf{pp}, \mathbf{I}(\mathbb{i}); \bot)$.
  3. Create the proving key $\mathsf{ipk} := (\mathsf{pp}, \mathbb{i}, \mathbf{I}(\mathbb{i}))$ and the verification key $\mathsf{ivk} := (\mathsf{pp}, \mathsf{cm}_0, \hat{\rho}(\mathbb{i}))$.
  4. Output the key pair $(\mathsf{ipk}, \mathsf{ivk})$.
- $\mathcal{P}^{\hat{\rho}}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$:
  1. Parse ipk as $(\mathsf{pp}_{\mathsf{VC}}, \mathbb{i}, \mathbf{I}(\mathbb{i}))$.
  2. Compute the PCP string: $\Pi \leftarrow \mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$.
  3. Sample commitment randomness ω_1, and compute the commitment $\mathsf{cm}_1 := \mathsf{VC.Commit}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \Pi; \omega_1)$.
  4. Choose a random salt $s \leftarrow \{0,1\}^\lambda$.
  5. Derive the PCP randomness $r := \hat{\rho}(s, \hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1)$.
  6. Simulate $\mathbf{V}^{\mathbf{I}(\mathbb{i}),\Pi}(\mathbb{x}; r)$. Denote $\mathbf{V}$'s queries to $\mathbf{I}(\mathbb{i})$ as $Q_0 \subseteq [|\mathbf{I}(\mathbb{i})|]$ and to Π as $Q_1 \subseteq [\ell]$.
  7. Open $\mathbf{I}(\mathbb{i})$ at Q_0: $\pi_{\mathsf{VC},0} \leftarrow \mathsf{VC.Open}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{I}(\mathbb{i}), Q_0; \bot)$.
  8. Open Π at Q_1: $\pi_{\mathsf{VC},1} \leftarrow \mathsf{VC.Open}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \Pi, Q_1; \omega_1)$.
  9. Output the proof $\pi := (s, \hat{\rho}(\mathbb{i}), \mathsf{cm}_1, (Q_0, \mathbf{I}(\mathbb{i})|_{Q_0}, \pi_{\mathsf{VC},0}), (Q_1, \Pi|_{Q_1}, \pi_{\mathsf{VC},1}))$.
- $\mathcal{V}^{\hat{\rho}}(\mathsf{ivk}, \mathbb{x}, \pi)$:
  1. Parse ivk as $(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_0, \hat{\rho}(\mathbb{i}))$.
  2. Parse π as $(\mathsf{cm}_1, (Q_0, \mathbf{a}_0, \pi_{\mathsf{VC},0}), (Q_1, \mathbf{a}_1, \pi_{\mathsf{VC},1}))$, where $\mathbf{a}_0\colon Q_0 \to \Sigma$ and $\mathbf{a}_1 : Q_1 \to \Sigma$.
  3. Derive the PCP randomness $r := \hat{\rho}(\hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1)$.
  4. Check that
     - $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_0, Q_0, \mathbf{a}_0, \pi_{\mathsf{VC},0}) = 1$,
     - $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_1, Q_1, \mathbf{a}_1, \pi_{\mathsf{VC},1}) = 1$, and
     - Simulate the PCP verifier $\mathbf{V}$ to see if it accepts, on input $\mathbb{x}$, randomness r, and with
       * queries to $\mathbf{I}(\mathbb{i})$ answered with the corresponding answer in $\mathbf{a}_0$, and
       * queries to Π answered with the corresponding answer in $\mathbf{a}_1$.

       If any query answer is undefined, abort and output 0.

Proof. We prove completeness, knowledge soundness, and zero knowledge.

Completeness. Fix an oracle $\hat{\rho} \in \mathrm{supp}(\mathcal{C}_\lambda)$, tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^\theta$, public parameters $\mathsf{pp} \in \mathcal{G}^{\hat{\rho}}(1^\lambda, 1^\ell)$, and SNARK keys $(\mathsf{ipk}, \mathsf{ivk}) \in \mathcal{I}^{\hat{\rho}}(\mathsf{pp}, \mathbb{i})$. We argue that for every $\pi \in \mathcal{P}^{\hat{\rho}}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ it holds that $\mathcal{V}^{\hat{\rho}}(\mathsf{ivk}, \mathbb{x}, \pi) = 1$, by establishing that the three conditions below hold.

- $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_0, Q_0, \mathbf{a}_0, \pi_{\mathsf{VC},0}) = 1$:

  The indexer $\mathcal{I}^{\hat{\rho}}(\mathsf{pp}, \mathbb{i})$ outputs a verification key ivk containing $\mathsf{cm}_0 := \mathsf{VC.Commit}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{I}(\mathbb{i}); \bot)$. The prover $\mathcal{P}^{\hat{\rho}}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ simulates a set of queries to the encoded index $\mathbf{I}(\mathbb{i})$, denoted Q_0, and records corresponding set of query answers $\mathbf{I}(\mathbb{i})|_{Q_0}$ from $\mathbf{I}(\mathbb{i})$. Next, $\mathcal{P}^{\hat{\rho}}$ computes the VC opening $\pi_{\mathsf{VC},0} = \mathsf{VC.Open}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathbf{I}(\mathbb{i}), Q_0; \bot)$. By the correctness of VC, $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_0, Q_0, \mathbf{I}(\mathbb{i})|_{Q_0}, \pi_{\mathsf{VC},0}) = 1$.

- $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_1, Q_1, \mathbf{a}_1, \pi_{\mathsf{VC},1}) = 1$:

  The prover $\mathcal{P}^{\hat{\rho}}(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$ runs the PCP prover $\Pi \leftarrow \mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$, samples commitment randomness ω_1, and computes the commitment $\mathsf{cm}_1 := \mathsf{VC.Commit}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \Pi; \omega_1)$. Next, $\mathcal{P}^{\hat{\rho}}$ simulates a set of queries to the PCP string Π, denoted Q_1, and records corresponding set of query answers $\Pi|_{Q_1}$ from Π. Finally, $\mathcal{P}^{\hat{\rho}}$ computes the VC opening $\pi_{\mathsf{VC},1} = \mathsf{VC.Open}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \Pi, Q_1; \omega_1)$. By the correctness of VC, $\mathsf{VC.Check}^{\hat{\rho}}(\mathsf{pp}_{\mathsf{VC}}, \mathsf{cm}_1, Q_1, \Pi|_{Q_1}, \pi_{\mathsf{VC},1}) = 1$.

- The simulated PCP verifier $\mathbf{V}$ accepts, on input $\mathbb{x}$, randomness r, and with
  - queries to $\mathbf{I}(\mathbb{i})$ answered with the corresponding answer in $\mathbf{a}_0$: $q_0 \to \Sigma$, and
  - queries to Π answered with the corresponding answer in $\mathbf{a}_1$: $q_1 \to \Sigma$.

  Both the prover $\mathcal{P}^{\hat{\rho}}$ and verifier $\mathcal{V}^{\hat{\rho}}$ compute the same PCP randomness r, which is used by the PCP verifier $\mathbf{V}$. Since $\mathcal{P}^{\hat{\rho}}$ simulates $\mathbf{V}^{\mathbf{I}(\mathbb{i}),\Pi}(\mathbb{x}; r)$ honestly, $\mathcal{V}^{\hat{\rho}}$'s execution of $\mathbf{V}$ will make the same queries (but answering queries with $\mathcal{P}^{\hat{\rho}}$'s recorded oracle answers). Since $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the completeness of the PCP implies that the simulated PCP verifier $\mathbf{V}$ accepts.

Knowledge soundness. We argue that $\mathsf{ARG}_{\mathsf{Mic}}$ has knowledge soundness error $\kappa = O(t \cdot \kappa_{\mathsf{PCP}}) + t^2/|\mathbb{F}_\lambda|$.[15] We define the extractor $\mathcal{E}$ as follows.

```
$\mathcal{E}^{\hat{\rho},\tilde{\mathcal{P}}}(\mathsf{pp}, \mathsf{ai})$:
 1. Run the malicious prover $(\mathbb{i}, \mathbb{x}, \pi; \mathsf{tr}) \leftarrow \tilde{\mathcal{P}}^{\hat{\rho}}(\mathsf{ai})$.
 2. Compute $(\mathsf{ipk}, \mathsf{ivk}) := \mathcal{I}(\mathbb{i})$. If $\mathcal{V}^{\hat{\rho}}(\mathsf{ivk}, \mathbb{x}, \pi) = 0$, stop.
 3. Let $y^* := \hat{\rho}(\mathbb{i})$. Compute the fork index $i := \mathsf{FP}_{\mathcal{C}_\lambda}(\mathsf{tr}, (y^*, \mathbb{x}, \mathsf{cm}_1))$.
 4. Compute the number of forks $N := 8 \cdot (\lambda + \ell \log |\Sigma|)$.
 5. Run the $N$-fork forking algorithm MFork: $((\mathbb{i}_j, \pi_j), \mathsf{tr}_j)_{j \in [N]} \leftarrow \mathsf{MFork}^{\tilde{\mathcal{P}},\mathsf{p}}(N, \mathsf{tr}, i, \mathsf{ai})$, where
    $\mathsf{p}((y^*, \mathbb{x}, \mathsf{cm}_1), \mathbb{i}', \pi', \mathsf{tr}') := \mathcal{V}^{\mathsf{tr}}(\mathsf{ivk}, \mathbb{x}, \pi')$ for $(\mathsf{ipk}, \mathsf{ivk}) := \mathcal{I}^{\mathsf{tr}}(\mathbb{i})$.
 6. If $\mathbb{i}_j \neq \mathbb{i}$ for any $j \in [N]$, output COL and abort.
 7. If there exist $q \in [\ell]$ and $j, j' \in [N]$ such that $\mathbf{a}_1^{(j)}[q] \neq \mathbf{a}_1^{(j')}[q]$, output COL and abort.
 8. Construct a PCP string $\Pi$: for each $q \in [\ell]$, if there exists $j \in [N]$ such that $q \in Q_1^{(j)}$, set
    $\Pi[q] := \mathbf{a}_1^{(j)}[q]$. If not, set $\Pi[q] := 0$.
 9. Run the PCP extractor $\mathbb{w} := \mathcal{E}_{\mathsf{PCP}}(\mathbb{i}, \mathbb{x}, \Pi)$.
10. Output $\mathbb{w}$.
```

We fix some $(\mathsf{tr}, i, \mathsf{ai})$. Let $((x_1, y_1), \ldots, (x_t, y_t)) := \mathsf{tr}$, and $(y^*, \mathbb{x}, \mathsf{cm}_1) := x_i$. We bound the probability, for any fixed choice of tr, that the extractor outputs a "bad" PCP. For a proof string Π, define $\mu(\mathbb{i}, \mathbb{x}, \Pi) := \Pr_r[\mathbf{V}^{\mathbf{I}(\mathbb{i}),\Pi}(\mathbb{x}; r) = 1]$.

Claim 8.5. $\Pr[\neg\mathtt{COL} \wedge \mu(\mathbb{i}, \mathbb{x}, \Pi) < \delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})/4] \leq 2^{-\lambda}$*, where* $\delta_{i,x}$ *is as defined in Eq. 4.*

[15] An identical soundness argument yields $\kappa = O(t \cdot \kappa_{\mathsf{PCP}}) + \mathrm{negl}(\lambda)$ when VC is instantiated with any (computationally) binding vector commitment.

Proof. Let $\delta := \delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai})$. We show that for any fixed string $\Pi^* \in \Sigma^\ell$ such that $\mu := \mu(\mathbb{i}, \mathbb{x}, \Pi^*) < \delta/4$,

$$\Pr_{\Pi \leftarrow \mathcal{E}^{\hat{\rho}, \tilde{\mathcal{P}}}}[\Pi = \Pi^*] \leq |\Sigma|^{-\ell} \cdot 2^{-\lambda} \ .$$

We consider an execution of the MFork algorithm as in $\mathcal{E}$, and define the following random variables over this joint probability space. For K is as in Lemma 5.8, define the random variable

$$X := \begin{cases} K & \text{if } \mathsf{MFork} \text{ outputs } \Pi^* \\ \infty & \text{otherwise} \end{cases} \ .$$

Let $r_i := \overline{\mathsf{tr}_i}(\mathsf{cm})$ be the PCP verifier randomness sampled in the i-th invocation of Fork. Let Y be the random variable whose value is the smallest integer ℓ such that $\sum_{i=1}^{\ell} \mathbf{V}^{\Pi^*}(\mathbb{i}, \mathbb{x}; r_i) = N$, or ∞ if there is no such ℓ. Observe that $Y \leq X$ with probability 1. Hence for all τ,

$$\Pr_{\Pi \leftarrow \mathcal{E}^{\hat{\rho}, \tilde{\mathcal{P}}}}[\Pi = \Pi^*] = \Pr[X < \infty] \leq \Pr[Y \leq \tau] + \Pr[\tau < X < \infty] \ .$$

Let $\tau := 2N/\delta$. Note that $Y \leq \tau$ if and only if $\sum_{i=1}^{K} \mathbf{V}^{\Pi^*}(\mathbb{i}, \mathbb{x}; r_i) \geq N$; the mean of this sum is $2N\mu/\delta < N/2$ by assumption on Π^*. Hence by a Chernoff bound, $\Pr[Y \leq \tau] \leq e^{-N/6}$. Next observe that

$$\Pr[\tau < X < \infty] \leq \Pr[K > \tau] \ .$$

By Lemma 5.8, $\mathbb{E}[K] = N/\delta$. Hence by a Chernoff bound, $\Pr[K > 2N/\delta] \leq e^{-N/8}$. Thus $\Pr_{\Pi \leftarrow \mathcal{E}^{\hat{\rho}, \tilde{\mathcal{P}}}}[\Pi = \Pi^*] \leq 2e^{-N/8} \leq |\Pi|^{-\ell} \cdot 2^{-\lambda}$. The claim follows by a union bound over all Π^* with $\mu(\mathbb{i}, \mathbb{x}, \Pi^*) < \delta/4$ (of which there are at most $|\Sigma|^\ell$). □

Let κ_{PCP} be the knowledge error of PCP. By the construction of $\mathcal{E}$, it holds that

$$\begin{aligned} \kappa &= \mathbb{E}_{\vec{y}, \mathsf{ai}}[\sum_{i,x} \mathbb{1}_{S_{i,x}}(\vec{y}, \mathsf{ai}) \cdot \mathbb{1}(\mu(\mathbb{i}, \mathbb{x}, \Pi) < \kappa_{\mathsf{PCP}})] \\ &= \sum_{i,x} \mathbb{E}_{y_1, \ldots, y_{i-1}, \mathsf{ai}}[\delta_{i,x}(y_1, \ldots, y_{i-1}; \mathsf{ai}) \cdot \Pr[\mu(\mathbb{i}, \mathbb{x}, \Pi) < \kappa_{\mathsf{PCP}}]] \\ &\leq t \cdot \max\{4\kappa_{\mathsf{PCP}}, 2^{-\lambda}\} + \Pr[\mathtt{COL}] \ . \end{aligned}$$

Zero knowledge. Let $\mathbf{S}_{\mathsf{PCP}}$ be the honest-verifier zero-knowledge simulator for PCP.

- *Parameter generation:* $\mathcal{S}(1^\lambda) \to \mathsf{pp}$.
  1. Run $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and output pp.
- *Proving:* $\mathcal{S}^\theta(\mathsf{pp}, \mathbb{i}, \mathbb{x}, \mathsf{tr}) \to \pi$.
  1. Sample PCP randomness $r \leftarrow \mathbb{F}$.
  2. Sample a PCP local view $(\mathbf{a}_1 \colon Q_1 \to \Sigma) \leftarrow \mathbf{S}_{\mathsf{PCP}}(\mathbb{i}, \mathbb{x}, r)$.
  3. For $x \in Q_1$, set $\Pi[x] := \mathbf{a}[x]$; for $x \notin Q_1$, set $\Pi[x] := 0$.
  4. Sample a salt $s \leftarrow \{0,1\}^\lambda$.
  5. Generate π as in Steps 6 to 9 of the honest prover $\mathcal{P}$.
  6. If $\overline{\mathsf{tr}}(s, \hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1) \neq \bot$, abort.
  7. If $\overline{\mathsf{tr}}(s, \hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1) = \bot$, add the mapping $(s, \hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1) \mapsto r$ to tr, and store tr as the state.

8. Output π.

- *Query responses:* $\mathcal{S}^{\theta}(x) \to y$.
  1. If $\overline{\mathsf{tr}}(x) \neq \perp$, return $\overline{\mathsf{tr}}(x)$. ($\mathsf{tr}$ is stored as the state of $\mathcal{S}$.)
  2. If $\overline{\mathsf{tr}}(x) = \perp$, sample a random $y \in \mathbb{F}$, add the mapping $x \mapsto y$ to tr, and return y.

Above, the simulator uses efficient constraint detection to evaluate $\overline{\mathsf{tr}}$ given tr. Provided $\overline{\mathsf{tr}}(s, \hat{\rho}(\mathbb{i}), \mathbb{x}, \mathsf{cm}_1) = \perp$, indistinguishability can be argued by the hiding property of VC and the honest-verifier zero-knowledge guarantee of PCP. By Lemma 4.14, the number of points $x \in \{0,1\}^m$ such that $\overline{\mathsf{tr}}(x) \neq \perp$ is at most t. Hence the probability (over the choice of s) that $\overline{\mathsf{tr}}(s, x) \neq \perp$ for *any* x is at most $t/2^{\lambda}$. □

The following lemma is a straightforward application of the collision-resistance of linear code random oracles (Lemma 4.15).

Lemma 8.6. *Merkle trees [Mer87] are a secure vector commitment in any systematic linear code random oracle model with arity $m(\lambda) \geq 2 \log |\mathbb{F}_{\lambda}|$ and $|\mathbb{F}_{\lambda}| = \lambda^{\omega(1)}$. Moreover, this construction can be made hiding following [BCS16].*

The arity requirement ensures that the oracle is sufficiently compressing to be used as the hash function in the Merkle tree construction (and could be relaxed somewhat).

8.3 Lower bound

We show that Theorem 8.1 is false, even for deterministic computations, if we replace $\mathscr{C}$ with a standard random oracle. It follows by Lemma 8.2 that there does not exist an accumulation scheme for oracle queries in the ROM (since there exist SNARKs for NP in the ROM).

Theorem 8.7. *Let $\mathcal{O}$ be the standard random oracle. For any polynomial T there is a language $\mathcal{L}^{\mathcal{O}} \in \mathsf{DTIME}(T)^{\mathcal{O}}$ such that if $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a (preprocessing) non-interactive argument system for $\mathcal{L}^{\mathcal{O}}$ relative to $\mathcal{O}$ then $\mathcal{V}$ must make $T(\lambda)$ queries.*

Proof. Consider the indexed language $\mathcal{L}^{\mathcal{O}}$ given by: for each $\lambda \in \mathbb{N}$, $B \in [2^{\lambda}]$ and $\theta \in \mathrm{supp}(\mathcal{O}(\lambda))$, $(\perp, B) \in \mathcal{L}^{\theta}$ if and only if $\bigoplus_{i=1}^{T(\lambda)} \theta(B+i)_1 = 1$, where $\theta(B+i)_1$ denotes the first bit of $\theta(B+i)$. Let $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a SNARK for $\mathcal{L}^{\mathcal{O}}$, and suppose that $\mathcal{V}$ makes fewer than T queries.

To attack the scheme we generate a proof π as follows. Choose $B \in [2^{\lambda}]$ uniformly at random, and output $(\perp, B)$ as the index-instance pair. We have that $(\perp, B) \in \mathcal{L}^{\theta}$ with probability $1/2$ over the choice of θ. Suppose now that $(\perp, B) \notin \mathcal{L}^{\theta}$. Choose $i \in [T]$ uniformly at random, and let θ_i be defined as

$$\theta_i(x) := \begin{cases} \overline{\theta(x)} & \text{if } x = B+i \text{, and} \\ \theta(x) & \text{otherwise.} \end{cases}$$

Compute $(\mathsf{ipk}', \mathsf{ivk}') := \mathcal{I}^{\theta_i}(\perp)$, and compute π as $\pi \leftarrow \mathcal{P}^{\theta_i}(\mathsf{ipk}', B)$. (Note that θ_i can be efficiently simulated via queries to θ.)

Observe that $(\perp, B) \in \mathcal{L}^{\theta_i}$. Hence by the completeness guarantee, $\mathcal{V}^{\theta_i}(\mathsf{ivk}', B, \pi) = 1$. If $\mathcal{V}^{\theta}(\mathsf{ivk}, B, \pi) = 0$ where $(\mathsf{ipk}, \mathsf{ivk}) := \mathcal{I}^{\theta}(\perp)$, then $\mathcal{I}$ or $\mathcal{V}$ must query θ at $B+i$. Since $\mathcal{I}$ does not know B, the probability that $\mathcal{I}$ queries θ at $B+i$ is negligible. Since $\mathcal{V}$ makes fewer than T queries, the probability that $\mathcal{V}$ queries θ at $B+i$ is at most $1 - \frac{1}{T}$. Thus $\mathcal{V}^{\theta}(\mathsf{pp}, 1^{\lambda}, \pi) = 1$ with probability at least $1/T - \mathrm{negl}(\lambda)$. □

We remark that the step at which the above proof fails in the LDROM is "either $\mathcal{I}$ or $\mathcal{V}$ must query θ at $B+i$." Since low-degree oracles have distance, changing the value of $\theta(B+i)$ has a global effect.

9 Proof-carrying data from relativized SNARKs

We give a generic construction, for any oracle $\mathcal{O}$, of a PCD scheme relative to $\mathcal{O}$ (for compliance predicates with access to $\mathcal{O}$) from a SNARK relative to $\mathcal{O}$ for $\mathsf{NP}^{\mathcal{O}}$. The construction and its proof are a direct adaptation to the relativized setting of the construction in [COS20]. The only changes are that we grant oracle access to the appropriate algorithms in the construction, and we adapt the proof of knowledge soundness to our definition of knowledge soundness for non-interactive arguments (see Section 3.2), which differs somewhat from that of [COS20]. We first recall the definition of *depth* for compliance predicates.

Definition 9.1. *The* depth *of a PCD transcript* T*, denoted* $\Delta(\mathsf{T})$*, is the depth of the underlying rooted DAG. For* $\lambda \in \mathbb{N}$*, the depth of a compliance predicate* Φ *with access to* $\mathcal{O}(\lambda)$*, denoted* $\Delta(\Phi, \lambda)$*, is defined to be maximum of* $\Delta(\mathsf{T})$ *over all* Φ^{θ}*-compliant transcripts* T *and* $\theta \in \operatorname{supp}(\mathcal{O}(\lambda))$*.*

Theorem 9.2. *There exists a polynomial-time transformation* $\mathbb{T}$ *such that if* $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a (preprocessing) SNARK relative to* $\mathcal{O}$ *for* $\mathsf{NP}^{\mathcal{O}}$*, then* $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathbb{T}(\mathsf{ARG})$ *is a (preprocessing) PCD scheme relative to* $\mathcal{O}$ *for constant-depth compliance predicates with access to* $\mathcal{O}$*. Moreover, if* ARG *is zero-knowledge, then* PCD *is also zero-knowledge.*

Remark 9.3. As in [COS20], the efficiency requirement for the SNARK verifier in Theorem 9.2 can be relaxed to *sublinear* in the index (circuit) size.

Construction 4. Fix a security parameter $\lambda \in \mathbb{N}$ and an oracle θ in the support of the oracle distribution $\mathcal{O}(\lambda)$. We denote by $\mathcal{V}^{(\lambda,N,k)}$ the circuit corresponding to the computation of the SNARK verifier $\mathcal{V}$, for the security parameter λ, checking indices of size at most N, and instances of size at most k. Let $\Phi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ be an oracle compliance predicate represented by a circuit with θ-gates. The circuit that realizes the recursion is as follows:

```
$[R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{pp}}]^{\theta}((\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, (z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})_{i\in[m]}))$:
1. Check that the compliance predicate $\Phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \dots, z^{(m)}_{\mathsf{in}})$ accepts.
2. If there exists $i$ such that $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}) \neq \bot$,
   check that the SNARK verifier $[\mathcal{V}^{(\lambda,N,k)}]^{\theta}(\mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}})$ accepts for every $i \in [m]$.
```

We describe the PCD scheme's generator, indexer, prover, and verifier.

- $\mathbb{G}(1^{\lambda})$:
  1. Sample public parameters for the SNARK: $\mathsf{pp} := \mathcal{G}(1^{\lambda})$.
  2. Output pp.
- $\mathbb{I}^{\theta}(\mathsf{pp}, \Phi)$:
  1. Compute $N := N(\lambda, |\Phi|, m, \ell)$, as given in [COS20, Lemma 11.8].[16]
  2. Construct the (oracle) circuit $R := R^{(\lambda,N,|\mathsf{ivk}(\lambda,N)+\ell|)}_{\mathcal{V},\Phi,\mathsf{pp}}$.
  3. Compute the index key pair $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\theta}(\mathsf{pp}, R)$.
- $\mathbb{P}^{\theta}(\mathsf{ipk}, z_{\mathsf{out}}, z_{\mathsf{loc}}, \vec{z_{\mathsf{in}}}, \vec{\pi_{\mathsf{in}}})$: output the proof $\pi_{\mathsf{out}} \leftarrow \mathcal{P}^{\theta}(\mathsf{ipk}, (\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, \vec{z_{\mathsf{in}}}, \vec{\pi_{\mathsf{in}}}))$.

[16]Since Fractal ([COS20, Construction 11.7]) and Construction 4 are the same, except that Construction 4's SNARK verifier also checks the oracle transcript, we can define $N(\lambda, |\Phi|, m, \ell)$ using [COS20, Lemma 11.8]. Note that the lemma's v describes the size of the SNARK verifier relative to $\mathcal{O}$ for $\mathsf{NP}^{\mathcal{O}}$ computations and includes the complexity of checking oracle queries.

- $\mathbb{V}^\theta(\mathsf{ivk}, z_{\mathsf{out}}, \pi_{\mathsf{out}})$: accept if $\mathcal{V}^\theta(\mathsf{ivk}, (\mathsf{ivk}, z_{\mathsf{out}}), \pi_{\mathsf{out}})$ accepts.

The correctness and efficiency of the PCD scheme is essentially the same as in [COS20]. Below we provide an adapted knowledge soundness analysis.

Proof of knowledge soundness. We present and then analyze the knowledge extractor.

Construction. The construction of the malicious SNARK prover $\tilde{\mathcal{P}}$ and the PCD extractor $\mathbb{E}^{\theta,\tilde{\mathbb{P}}}$ are the same as in [COS20], except for the following differences.

- All SNARK and PCD algorithms (except $\mathcal{G}$ and $\mathbb{G}$) have access to the oracle θ. In particular, this includes sequence of extractors $\mathbb{E}^\theta_1, \ldots, \mathbb{E}^\theta_\Delta$ that define $\mathbb{E}^{\theta,\tilde{\mathbb{P}}}$, where $\Delta := \Delta(\lambda) := \max_{\Phi \in \mathsf{F}} \Delta(\Phi, \lambda)$.
- The SNARK's malicious prover $\tilde{\mathcal{P}}$ and extractor $\mathcal{E}^{\theta,\tilde{\mathcal{P}}}$ run in (non-uniform) expected polynomial-time. Hence, we discuss the efficiency of $\mathbb{E}^{\theta,\tilde{\mathbb{P}}}$ via running time instead of circuit size.

Running time. This analysis is essentially identical to that of [COS20], except that we analyze expected running time because $\tilde{\mathcal{P}}$ and $\mathcal{E}^{\theta,\tilde{\mathcal{P}}}$ run in expected polynomial-time.

The j-th adversary $\tilde{\mathcal{P}}^\theta_j$ runs in expected time: $\mathbb{E}[\mathrm{time}(\mathbb{E}^\theta_{j-1})] + \mathrm{poly}(\lambda) + O(2^j)$. Thus, $\mathcal{E}^{\theta,\tilde{\mathcal{P}}^\theta_j}$ runs in (expected) time: $e(\mathbb{E}[\mathrm{time}(\mathbb{E}^\theta_{j-1})] + \mathrm{poly}(\lambda) + O(2^j))$, where e is a function mapping the expected running time of $\tilde{\mathcal{P}}^\theta_j$ to the expected running time of $\mathcal{E}^{\theta,\tilde{\mathcal{P}}^\theta_j}$. This means that $\mathbb{E}[\mathrm{time}(\mathbb{E}^\theta_j)] \le e(\mathbb{E}[\mathrm{time}(\mathbb{E}^\theta_{j-1})] + \mathrm{poly}(\lambda) + c \cdot 2^j)$ for some $c \in \mathbb{N}$.

A solution for this recurrence (for $e(n) \ge n$) is $\mathbb{E}[\mathrm{time}(\mathbb{E}^\theta_\Delta)] \le e^{(\Delta)}(\mathbb{E}[\mathrm{time}(\tilde{\mathbb{P}})] + \Delta \cdot \mathrm{poly}(\lambda) + 2c \cdot 2^\Delta)$ for some $c \in \mathbb{N}$, where $e^{(\Delta)}$ is the function e iterated Δ times. In particular, since e is some polynomial, if $\Delta(\Phi)$ is a constant, $\mathbb{E}^{\theta,\tilde{\mathbb{P}}}$ runs in expected polynomial time.

Correctness. Suppose that $\tilde{\mathbb{P}}$ causes $\mathbb{V}^\theta$ to accept with probability μ. We show by induction that for all $j \in \{0, \ldots, \Delta\}$, the transcript T_j output by $\mathbb{E}^\theta_j$ is Φ-compliant up to depth j, $\mathcal{V}^\theta(\mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)}) = 1$ for all $v \in \mathsf{T}$, $\mathsf{o}(\mathsf{T}_j) = \mathsf{o}$, and the predicate $\Phi \in \mathsf{F}$ with probability $\mu - \mathrm{negl}(\lambda)$.

For $j = 0$, this follows from $\mathbb{V}^\theta$ accepting.

Now suppose that T_{j-1} output by $\mathbb{E}^\theta_{j-1}$ is Φ-compliant up to depth $j-1$, $\mathcal{V}^\theta(\mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)}) = 1$ for all $v \in \mathsf{T}$, $\mathsf{o}(\mathsf{T}_{j-1}) = \mathsf{o}$, and the predicate $\Phi \in \mathsf{F}$ with probability $\mu - \mathrm{negl}(\lambda)$.

Let $(\vec{\mathbb{i}}, (\mathsf{ivk}, z^{(v)})_v, (\pi^{(v)})_v, (\Phi, \mathsf{T}'), \vec{\mathbb{w}}) \leftarrow \mathcal{E}^{\theta,\tilde{\mathcal{P}}}$. By the knowledge soundness of ARG, with probability $\mu - \mathrm{negl}(\lambda)$, for $v \in l_{\mathsf{T}'}(j)$, we have $([R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{pp}}]^\theta, (\mathsf{ivk}_v, z^{(v)}), \vec{\mathbb{w}}) \in \mathcal{R}^\theta$, where $\mathsf{ivk}_v = \mathsf{ivk}$ where $(\mathsf{ivk}, \mathsf{ipk}) \leftarrow \mathcal{I}^\theta(\mathsf{pp}, [R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{pp}}]^\theta)$.

Next, consider some $v \in l_{\mathsf{T}'}(j)$. Since $([R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{pp}}]^\theta, (\mathsf{ivk}_v, z^{(v)}), \vec{\mathbb{w}}) \in \mathcal{R}^\theta$, we obtain from $\vec{\mathbb{w}}$ either:

- local data $z^{(v)}_{\mathsf{loc}}$ and input messages $(z^{(i)}_{\mathsf{in}}, \pi^{(i)})_{i\in[m]}$ such that $\Phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \ldots, z^{(m)}_{\mathsf{in}}) = 1$ and for all $i \in [m]$ the SNARK verifier $[\mathcal{V}^{(\lambda,N,k)}]^\theta(\mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}}) = 1$; or
- local data $z^{(v)}_{\mathsf{loc}}$ such that $\Phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, \bot, \ldots, \bot) = 1$.

In both cases, $z^{(v)}_{\mathsf{loc}}$ is appended to the label of v. In the former case, we label v's children with $(z^{(i)}_{\mathsf{in}}, \pi^{(i)})_{i\in[m]}$ and so v is Φ-compliant, and for all of its descendants w we have that $[\mathcal{V}^{(\lambda,N,k)}]^\theta(\mathsf{ivk}, (\mathsf{ivk}, z^{(w)}_{\mathsf{in}}), \pi^{(w)}_{\mathsf{in}}) = 1$. In the latter case, v has no children; by the base case, v is Φ-compliant. Thus, $\mathsf{T}_j \leftarrow \mathbb{E}^\theta_j$ is Φ-compliant up to depth j.

Since $\Delta(\Phi, \lambda) \le \Delta$, it must be the case that all vertices v at depth Δ have no children, i.e. are in the base case. Hence by induction, $(\Phi, \mathsf{T}) \leftarrow \mathbb{E}^\theta_\Delta$ outputs a Φ-compliant T, $\mathsf{o}(\mathsf{T}) = \mathsf{o}$, and $\Phi \in \mathsf{F}$ with probability at least $\mu - \mathrm{negl}(\lambda)$. □

Remark 9.4. The generator $\mathcal{G}$ of the SNARK that we construct in Section 8 does not require access to the oracle, and neither does the generator $\mathbb{G}$ of the resulting PCD scheme via the transformation Theorem 9.2. In light of this, the definitions for SNARK and PCD in Sections 3.2 and 3.3 do not give generators access to the oracle. That said, the transformation in Theorem 9.2 also works with generators that require the oracle: if the SNARK generator $\mathcal{G}$ needs oracle access to θ, then the PCD generator $\mathbb{G}$ will also have oracle access to θ.

9.1 PCD in the LDROM

The following corollary is directly obtained by combining our SNARK in the low-degree random oracle model (Theorem 8.1) with Theorem 9.2.

Corollary 9.5. *Let* $\mathscr{C} = \mathrm{LD}[\mathscr{F}, m, d] = \{\mathcal{C}_\lambda = \mathrm{LD}[\mathbb{F}_\lambda, m(\lambda), d(\lambda)]\}_{\lambda\in\mathbb{N}}$ *be a family of low-degree polynomial evaluation codes, where* $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a family of fields,* $m\colon \mathbb{N} \to \mathbb{N}$ *an arity function, and* $d\colon \mathbb{N} \to \mathbb{N}$ *a degree function (see Definition 4.16) such that* $m(\lambda) \geq 2\log|\mathbb{F}_\lambda|$ *and* $|\mathbb{F}_\lambda| = \lambda^{\omega(1)}$*. There exists a PCD scheme relative to* $\mathscr{C}$ *for constant-depth compliance predicates with access to* $\mathscr{C}$*, assuming the existence of (standard-model) collision-resistant hash functions.*

Since the proof of Theorem 8.1 yields an explicit knowledge extractor, we can provide a tighter characterization of the running time of the PCD extractor than the general bound in the prior section. This enables us to give meaningful guarantees in the logarithmic depth regime, as we discuss below.

Running time of the extractor. Since $\mathscr{C}$ has efficient constraint detection (Theorem 4.17) and $\tilde{\mathbb{P}}$ makes a strict polynomial number of queries, we may assume without loss of generality that $\tilde{\mathbb{P}}$ is non-redundant (by Claim 4.11). Moreover, if $\tilde{\mathcal{P}}$ is non-redundant then so is $\mathcal{E}^{\theta,\tilde{\mathcal{P}}}$.

For non-redundant adversaries $\tilde{\mathcal{P}}$, by examining Construction 3 and by Lemma 5.8, it holds that

$$e(\tau) \leq 2tN \cdot \tau + \mathrm{poly}(\lambda)$$

where t is the number of queries made by $\tilde{\mathcal{P}}$ and $N = \mathrm{poly}(\lambda)$ is the number of forks generated by MFork.

Plugging e into the generic expression for the running time of the PCD extractor, we obtain

$$\mathrm{time}(\mathbb{E}^{\theta}_{\Delta}) \leq (2tN)^{\Delta} \cdot \Big(\mathrm{time}(\mathbb{P}^{\theta}) + 2c \cdot 2^{\Delta} + \mathrm{poly}(\lambda)\Big) \ .$$

Considering the regime $\Delta = O(\log\lambda)$, this is $\mathrm{poly}(\lambda)^{\Delta}$; in particular, we can obtain meaningful guarantees for logarithmic depth from superpolynomial assumptions. By comparison, the generic bound $e(\tau) \leq \tau^c$ for unknown constant c yields $\mathrm{time}(\mathbb{E}^{\theta}_{\Delta}) \leq \mathrm{poly}(\lambda)^{c^{\Delta}}$, which is exponential in λ in the same regime.

Acknowledgments

This research was supported in part by a donation from the Ethereum Foundation, and by DARPA under Agreement No. HR00112020023.

References

[AABDS20] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 1–45.

[ABLSZ19] B. Abdolmaleki, K. Baghery, H. Lipmaa, J. Siim, and M. Zajac. "UC-Secure CRS Generation for SNARKs". In: *Proceedings of the 11th International Conference on Cryptology in Africa*. AFRICACRYPT '19. 2019, pp. 99–117.

[Alb+19a] M. R. Albrecht et al. "Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC". In: *Proceedings of the 25th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '19. 2019, pp. 371–397.

[Alb+19b] M. R. Albrecht et al. "Feistel Structures for MPC, and More". In: *Proceedings of the 24th European Symposium on Research in Computer Security*. ESORICS '19. 2019, pp. 151–171.

[AW09] S. Aaronson and A. Wigderson. "Algebrization: A New Barrier in Complexity Theory". In: *ACM Transactions on Computation Theory* 1.1 (2009), 2:1–2:54.

[BB04] D. Boneh and X. Boyen. "Short Signatures Without Random Oracles". In: *Proceedings of the 23rd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '04. 2004, pp. 56–73.

[BCCT12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. 2012, pp. 326–349.

[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.

[BCFGRS17] E. Ben-Sasson, A. Chiesa, M. A. Forbes, A. Gabizon, M. Riabzev, and N. Spooner. "Zero Knowledge Protocols from Succinct Constraint Detection". In: *Proceedings of the 15th Theory of Cryptography Conference*. TCC '17. 2017, pp. 172–206.

[BCGTV15] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs". In: *Proceedings of the 36th IEEE Symposium on Security and Privacy*. S&P '15. 2015, pp. 287–304.

[BCIOP13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. "Succinct Non-Interactive Arguments via Linear Interactive Proofs". In: *Proceedings of the 10th Theory of Cryptography Conference*. TCC '13. 2013, pp. 315–333.

[BCLMS21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 681–710.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: *Proceedings of the 18th Theory of Cryptography Conference*. TCC '20. 2020.

[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.

[BDFG21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. “Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments”. In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO ’21. 2021, pp. 649–680.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. “Checking computations in polylogarithmic time”. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC ’91. 1991, pp. 21–32.

[BG08] B. Barak and O. Goldreich. “Universal Arguments and their Applications”. In: *SIAM Journal on Computing* 38.5 (2008). Preliminary version appeared in CCC ’02., pp. 1661–1694.

[BGG17] S. Bowe, A. Gabizon, and M. Green. *A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK*. Cryptology ePrint Archive, Report 2017/602. 2017.

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

[BGM17] S. Bowe, A. Gabizon, and I. Miers. *Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model*. Cryptology ePrint Archive, Report 2017/1050. 2017.

[BGV11] S. Benabbas, R. Gennaro, and Y. Vahlis. “Verifiable Delegation of Computation over Large Datasets”. In: *Proceedings of the 31st Annual International Cryptology Conference*. CRYPTO ’11. 2011, pp. 111–131.

[BN06] M. Bellare and G. Neven. “Multi-signatures in the plain public-Key model and a general forking lemma”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS ’06. 2006, pp. 390–399.

[CFGS22] A. Chiesa, M. A. Forbes, T. Gur, and N. Spooner. “Spatial Isolation Implies Zero Knowledge Even in a Quantum World”. In: *Journal of the ACM* 69.2 (2022), pp. 1–44.

[CFS17] A. Chiesa, M. A. Forbes, and N. Spooner. *A Zero Knowledge Sumcheck and its Applications*. Cryptology ePrint Archive, Report 2017/305. 2017.

[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.

[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, pp. 769–793.

[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.

[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.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs and Succinct NIZKs without PCPs”. In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT ’13. 2013, pp. 626–645.

[GKMMM18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. “Updatable and Universal Common Reference Strings with Applications to zk-SNARKs”. In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO ’18. 2018, pp. 698–728.

[GKRRS21] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems”. In: *Proceedings of the 30th USENIX Security Symposium*. USENIX Security ’21. 2021, pp. 519–535.

[Gro10] J. Groth. “Short Pairing-Based Non-interactive Zero-Knowledge Arguments”. In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’10. 2010, pp. 321–340.

[Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: *Proceedings of the 35th Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’16. 2016, pp. 305–326.

[Kil92] J. Kilian. “A note on efficient zero-knowledge proofs and arguments”. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC ’92. 1992, pp. 723–732.

[KR08] Y. Kalai and R. Raz. “Interactive PCP”. In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*. ICALP ’08. 2008, pp. 536–547.

[KST21] A. Kothapalli, S. Setty, and I. Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*. Cryptology ePrint Archive, Report 2021/370. 2021.

[Mau05] U. M. Maurer. “Abstract Models of Computation in Cryptography”. In: *Proceedings of the 10th IMA International Conference on Cryptography and Coding*. IMA ’05. 2005, pp. 1–12.

[Mer87] R. C. Merkle. “A Digital Signature Based on a Conventional Encryption Function”. In: *Proceedings of the 10th Annual International Cryptology Conference*. CRYPTO ’87. 1987, pp. 369–378.

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

[Sho97] V. Shoup. “Lower bounds for discrete logarithms and related problems”. In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptographic Techniques*. EUROCRYPT ’97. 1997, pp. 256–266.

[Sma] N. P. Smart. *‘Bristol Fashion’ MPC Circuits*. `https://homes.esat.kuleuven.be/~nsmart/MPC/`.

[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.

[ZZ21] M. Zhandry and C. Zhang. *The Relationship Between Idealized Models Under Computationally Bounded Adversaries*. Cryptology ePrint Archive, Report 2021/240. 2021.