Proof-Carrying Data From Arithmetized Random Oracles

Abstract. Proof-carrying data (PCD) is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. Known constructions of PCD are obtained by recursively-composing SNARKs or related primitives. SNARKs with desirable properties such as transparent setup are constructed in the random oracle model. However, using such SNARKs to construct PCD requires heuristically instantiating the oracle and using it in a non-black-box way. [CCS22] constructed SNARKs in the low-degree random oracle model, circumventing this issue, but instantiating their model in the real world appears difficult.

In this paper, we introduce a new model: the *arithmetized random oracle model* (AROM). We provide a plausible standard-model (software-only) instantiation of the AROM, and we construct PCD in the AROM, given only a standard-model collision-resistant hash function. Furthermore, our PCD construction is for arbitrary-depth compliance predicates. We obtain our PCD construction by showing how to construct SNARKs in the AROM for computations that query the oracle, given an accumulation scheme for oracle queries in the AROM. We then construct such an accumulation scheme for the AROM.

We give an efficient "lazy sampling" algorithm (an *emulator*) for the ARO up to some error. Our emulator enables us to prove the security of cryptographic constructs in the AROM and that zkSNARKs in the ROM also satisfy zero-knowledge in the AROM. The algorithm is non-trivial, and relies on results in algebraic query complexity and the combinatorial nullstellensatz.

Keywords: proof-carrying data; random oracle model; arithmetization

1 Introduction

Proof-carrying data (PCD) [CT10] is a powerful cryptographic primitive that allows mutually distrustful parties to perform distributed computation in an efficiently verifiable manner. The notion of PCD generalizes incrementally-verifiable computation (IVC) [Val08] and has recently found exciting applications in enforcing language semantics [CTV13], verifiable MapReduce computations [CTV15], image authentication [NT16], verifiable registries [TFZ+22], blockchains [Mina; KB20; BMRS20; CCDW20], and more.

All known PCD constructions (and practical IVC constructions) are obtained via *recursive proof composition*, a general framework for building PCD from simpler primitives such as SNARKs [BCCT13; BCTV14; COS20] or accumulation schemes [BGH19; BCMS20; BDFG21; BCL+21; KST22]. While the specific constructions differ, the high-level idea remains the same: to prove the correctness of t steps of computation given proof of correctness for $t-1$ steps, one proves that "the t-th step is correct *and* there exists a valid proof for the first $t-1$ steps".

The statement that "there exists a valid proof" refers to the *verifier* of the underlying SNARK or accumulation scheme. As such, the resulting PCD scheme makes non-black-box use of the verifier for the underlying scheme. This leads to a significant theoretical problem when trying to prove security for constructions based on recursive composition: almost all known constructions of SNARKs, and all known constructions of accumulation schemes, are proven secure in the random oracle model (ROM). The random oracle is an inherently black-box object; in particular, it is believed that there is no "nontrivial" proof system for statements about the random oracle.

Most prior work in the area [COS20; BCMS20; BCL+21] avoids this problem using a *heuristic step*: they assume that there exists some concrete hash function such that replacing the random oracle with the hash function yields a secure SNARK or accumulation scheme in the standard model (without oracles), and then apply recursive composition to this heuristic scheme.

Two prior works [CT10; CCS22] propose a different approach: endow the random oracle with some additional structure. The PCD construction in [CT10] is in a model where the random oracle additionally *signs* its responses using a standard-model signature scheme; the verifier can then check query-answer pairs by verifying the signature rather than querying the oracle. Trading cryptographic structure for algebraic structure, [CCS22] construct PCD in the *low-degree random oracle model* (LDROM), where parties have access to a random low-degree multivariate polynomial.

Both of these oracle models can be instantiated using hardware tokens. Unfortunately, we do not have any standard model (i.e., software-only) instantiation of these oracles, even heuristically. This is in contrast to the (usual) random oracle model, where empirical evidence suggests that "natural" schemes remain secure provided the oracle is replaced with a suitably "random-looking" hash function [BR93]. Our goal in this work is to design a new oracle model that simultaneously achieves both desiderata: (a) there exists a PCD scheme in this model under standard assumptions; and (b) the oracle can be heuristically instantiated.

1.1 Our results

In this work we introduce and study a new oracle model, the arithmetized random oracle model (AROM), which provides a random oracle and a corresponding "arithmetization" oracle. As in the standard ROM, the random oracle is an idealized model of some concrete hash function H. The arithmetization oracle is an idealized model of a certain *arithmetization* of H, which is a low-degree polynomial P_H that can be efficiently computed from the circuit of H. As such, the AROM has a plausible heuristic instantiation: replace the random oracle by H and the arithmetization oracle by P_H, for a suitable hash function H.

Our main result is a construction of PCD in the AROM, based on the [CCS22] construction of PCD in the LDROM. By instantiating the AROM with a suitable hash function, we obtain a candidate "real-world" construction of PCD. Formally, we prove the following theorem.

Theorem 1 (informal)**.** *There exists transparent*[1] *(zero-knowledge) PCD in the AROM (for computations in the AROM), assuming the existence of collision-resistant hash functions in the standard model.*

Our PCD construction is provably secure (in the AROM) for *all* efficient compliance predicates. This stands in contrast to all other constructions of PCD (with the exception of [CT10], but including [CCS22]), whose security proofs are limited to constant-depth recursion. This is because, like [CT10], our PCD construction preserves the straightline extraction property of the underlying SNARK.[2]

To prove our main theorem, we develop various tools for analyzing cryptographic constructions in the AROM. Our key result here is to show that the additional power provided by the AROM does not help the adversary win any game defined with respect to the random oracle alone.

Theorem 2 (informal)**.** *Any construction that is secure in the ROM is secure in the AROM.*

An immediate consequence of this theorem is that any construction that is secure in the standard model is secure in the AROM. In contrast, we do not know whether an analogous statement holds in the LDROM. We remark that this result is meaningful even outside the present context: it provides evidence that security in the ROM implies security against a specific type of non-black-box attack, namely, attacks that treat the *arithmetization* of the hash function as a black box.

Comparison to other oracle models. As discussed above, both the ROM and the LDROM fall short of our goal. While the ROM has a well-established heuristic instantiation, it is unlikely to support a PCD scheme. PCD exists in the LDROM, but we do not know how to instantiate the oracle. The AROM offers, in some sense, the "best of both worlds": a provable construction of PCD *and* a plausible heuristic instantiation. Moreover, the proposed instantiation of the AROM does not rely on any cryptography beyond "random-oracle-like" hash functions. As such, there are no barriers to implementing our scheme.

Post-quantum security. Our scheme does not rely on any pre-quantum assumption; it is plausibly post-quantum secure. Moreover, it is conceivable that the scheme is in fact *provably* post-quantum secure in the "quantum-accessible" AROM; we leave this intriguing question to future work.

1.2 Related work

PCD and IVC in the ROM. There is theoretical evidence that, unlike for SNARKs, there is no construction of PCD and IVC in the ROM (even allowing for additional "mild" cryptographic assumptions like standard-model CRHs). First, [CL20] shows that the PCP theorem does not hold for various cryptographically relevant oracle models,

[1] The only setup required is a uniform reference string.

[2] Some other prior PCD constructions are also based on SNARKs with straightline extraction (e.g., [Val08; COS20]). However, this property is lost after the heuristic step is applied.

such as the ROM and the LDROM. This suggests that succinct proofs for computations relative to these oracles may be out of reach. Nevertheless, [CCS22] shows that this is not the whole story by constructing SNARKs for LDROM computations, particularly PCD, from a cryptographic assumption. Second, [HN22] shows various impossibilities for IVC in the ROM. For example, if a particular type of commitment scheme exists, then zero-knowledge IVC (without a CRS) does not exist in the ROM. This result holds even if the IVC construction were to rely on "standard" cryptographic assumptions.[3]

Pseudorandom oracles. [JLLW22] introduce the *pseudorandom oracle model* (PROM) and apply it towards obfuscation. Similarly to the AROM, the PROM aims to capture cryptographic schemes that make a non-black-box use of the random oracle. We outline the PROM and explain how it differs from the AROM.

The PROM is specified relative to a (standard model) pseudorandom function family F_k, and has two interfaces. The first accepts a key k and outputs a random handle h (and stores (h, k)). The second accepts a handle h and an input x and outputs $F_k(x)$, where k is the key corresponding to h. By the security of the PRF, a party holding only h cannot distinguish the latter interface from a random oracle. On the other hand, a party holding the key k can use the circuit for F_k in a non-black-box way. [JLLW22] constructs ideal obfuscation from functional encryption in the PROM.

The key difference between the AROM and the PROM is that the PROM "separates" non-black-box and black-box access to the oracle. Specifically, non-black-box access to the PROM is available only to parties that know k, whereas random oracle security holds only against parties that do not know k. *In the AROM, there is no such asymmetry: all parties have the same access to the oracle.* This is important in the context of recursive composition (which we study) since completeness requires that both the prover and the verifier have non-black-box access to the oracle. Still, soundness relies on the security of the random oracle against the prover. It is an exciting open question to understand whether, despite this apparent barrier, recursive composition is possible in the PROM.

Augmented random oracles. [Zha22] defines the *augmented* random oracle model to analyze the resilience of cryptographic transformations in the ROM against uninstantiability results. While ideas about modeling non-black-box access to the random oracle (and the abbreviation "AROM") are common to both the augmented ROM and the arithmetized ROM, the models are very different both technically and in their applications. We briefly summarize [Zha22] and then explain how our model differs.

Let ro denote the random oracle, and Π denote some protocol. A cryptographic transformation T usually comes with a guarantee like "if Π is a secure X, then $T^{\mathsf{ro}}(\Pi)$ is a secure Y". An uninstantiability result for T typically shows that there exists some Π such that $T^H(\Pi)$ is insecure for every polynomial-size circuit H. Known uninstantiability results use some non-black-box technique to provide a "trapdoor" that can be used with respect to any H but is useless for ro. The augmented ROM captures this paradigm by requiring $T^{\mathsf{ro}}(\Pi)$ to be secure even if Π has access to an oracle M that provides some functionality permitted by non-black-box access to H, but with respect to

[3] The paper claims that this result holds for constructions that use *falsifiable* assumptions but does not show this explicitly. Nonetheless, one can check that the proof does work for "benign" cryptographic assumptions.

ro. [Zha22] shows that key uninstantiability results for transformations (e.g., Fiat–Shamir for arguments [GK03]) lead to insecure protocols in the augmented ROM.

The augmented ROM is a tool for proving a stronger form of security for random oracle transformations. In particular, no "honest" scheme ever accesses the oracle M; indeed, the oracle M is chosen adversarially (and may be trivial). On the other hand, in the arithmetized ROM, honest parties use the non-black-box access provided by the arithmetization oracle, whose functionality is (mostly) fixed by the model itself.

2 Techniques

Recall that our goal in this work is to construct proof-carrying data (PCD). Our approach follows the widely-used template of *recursive proof composition*. However, our setting imposes several technical and conceptual challenges. We begin by outlining a vital issue in proving security for this type of construction, which our work seeks to address.

Recursive proof composition refers to a set of techniques that enable the construction of PCD (and IVC) from SNARKs or accumulation schemes. With few notable exceptions (e.g., [Gro16]), all constructions of SNARKs and accumulation schemes rely on the Fiat–Shamir heuristic, which converts an interactive public-coin argument system into a non-interactive argument via a cryptographic hash function H. For all of these SNARK constructions, it is unknown whether this heuristic can be realized from any concrete (i.e., falsifiable) cryptographic assumption; indeed, there is evidence that this may not be possible [GW11]. However, we can prove these schemes secure in the ROM, treating the hash function H as a truly random function ro to which the adversary has black-box access.

This leads to a fundamental tension in proving security for the recursive composition of these protocols. On the one hand, to prove security for the protocol itself, we assume that the adversary treats the hash function H as a black box. On the other hand, when recursively composing, the *honest* protocol treats H in a non-black-box way: specifically, as a concrete polynomial-size circuit. The prior work [CL20; HN22] discussed in Section 1.2 suggests that non-black-box use of H may be necessary to achieve PCD (and IVC).

2.1 Starting point: the low-degree random oracle model

The work of [CCS22] addresses the aforementioned tension by introducing a new oracle model called the *low-degree random oracle model* (LDROM). They then show how to construct PCD via recursive composition in the LDROM (i.e., using the oracle as a black box).

In the LDROM, all parties have oracle access to a uniformly random low-degree multivariate polynomial $\hat{\rho}\colon \mathbb{F}^m \to \mathbb{F}$. Restricting $\hat{\rho}$ to $\{0,1\}^m \subseteq \mathbb{F}^m$ recovers the usual random oracle, and [CCS22] show that relevant security properties of the random oracle continue to hold in the LDROM; in particular, Micali's SNARK [Mic00] is secure in the LDROM. Unlike the random oracle, the LDROM admits a *query accumulation scheme*: a verifier, with the help of an untrusted accumulation proof, can check the correctness of

n queries to $\hat{\rho}$ using only $O(1)$ queries to $\hat{\rho}$. [CCS22] construct such an accumulation scheme and use it to build PCD.

Instantiating the LDROM. [CCS22] observe that the LDROM can be instantiated using a hardware token that implements the structured PRF of [BGV11]. Of course, schemes involving hardware tokens have significant drawbacks; finding a plausible "software-only" instantiation would be much preferable. [CCS22] suggest a natural strategy: given a "random-oracle-like" hash function H, convert it into an arithmetic circuit gate-by-gate. Such a circuit does define a polynomial with which we could instantiate the LDROM. Unfortunately, as noted in [CCS22], for widely-used hash functions, the degree of this polynomial will be large (at least 2^{25}). Since the complexity of the verifier in the query accumulation scheme is linear in the degree of the oracle, the resulting PCD scheme would be prohibitively expensive.

2.2 The arithmetized random oracle model

Given the above difficulty, a natural next step is to consider techniques for *reducing the degree* of the resulting arithmetic circuit. Since the degree of an arithmetic circuit grows exponentially in its depth, a natural approach is to try to reduce the depth of the circuit for H. This can be achieved via the well-known NP reduction from circuit satisfiability to 3-SAT (a depth-two formula). The output of the reduction is a boolean formula Φ_H with the following property: there is an efficiently computable *witness function* W_H such that

$$\Phi_H(x, y, z) = \begin{cases} 1 & \text{if } H(x) = y \text{ and } W_H(x) = z \\ 0 & \text{otherwise} \end{cases}.$$

Converting Φ_H into an arithmetic formula (gate-by-gate) yields a polynomial P_H of total degree $O(|H|)$ that agrees with Φ_H on boolean inputs.

P_H is *not* a low-degree extension of H (rather of Φ_H) and so this is not a candidate instantiation of the LDROM. As we note later, however, the low-degree structure of P_H will nonetheless allow us to build a query accumulation scheme, inspired by that of [CCS22]. Moreover, the statement "$H(x) = y$" can be verified by querying P_H only, given z as a witness. It is therefore plausible that, following the template developed in the prior work, we can obtain a secure construction of PCD that makes only black-box use of H and P_H.

Of course, given the current state of knowledge, we can only hope to prove that this PCD scheme is secure in some idealized model. In particular, we would like to model H as a random oracle. It is then necessary to answer the question: *if H is a random oracle, what should P_H look like?* A central modeling contribution of our work is to propose an answer to this question.

A new oracle model: the AROM. We refer to our proposed oracle model as the *arithmetized random oracle model* (AROM). Before presenting the model, we discuss two key modeling challenges that arise. Both relate to the fact that the black-box behavior of P_H depends in a non-black-box way on H.

– **Challenge #1: W_H is circuit-dependent.** For a concrete circuit H and input x, $W_H(x)$ is a vector representing the assignment to the internal wires of H on input x.

This of course depends on the size and structure of the circuit for H, which is no longer meaningful when H is replaced by a random oracle. We handle this conservatively, by allowing W_H to be adversarial. That is, we require that completeness, soundness, and zero-knowledge hold *regardless* of the choice of W_H, which we allow to depend on x and the random oracle, and may even itself be randomized.
There is, however, an important caveat. While we allow our W_H to depend on the random oracle, we must restrict this dependency; otherwise, the adversary could use W_H to learn information that it cannot otherwise obtain (e.g., W_H could encode a collision in H). Similarly, if W_H is computationally unbounded, the adversary could use it to break standard-model cryptography. As such, we restrict W_H to have an efficient implementation (in particular, it can only make polynomially-many queries to H).

– **Challenge #2: P_H is not the unique extension.** Even after we have fixed W_H (and hence Φ_H), P_H has a huge number of remaining degrees of freedom. This is because it is of individual degree larger than 1, but its behavior is specified only on boolean inputs. This is a more challenging issue to resolve: letting P_H be chosen adversarially from the set of extensions of Φ_H would make the adversary unrealistically powerful (see Remark 1). Instead, we model P_H as a uniformly random polynomial of the appropriate degree whose restriction to the hypercube is Φ_H. We propose that this captures the inability of the adversary to leverage the structure of H (and hence P_H) in breaking security. We leave to future work the question of whether this modeling choice can be weakened (again see Remark 1).

We now give an informal definition of the AROM; for details see Section 4. In the AROM, all parties (honest and malicious) have access to three oracles $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$:

– a *random oracle* $\mathsf{ro}\colon \{0,1\}^m \to \{0,1\}^\lambda$ drawn uniformly at random;
– a *witness oracle* $\mathsf{wo}\colon \{0,1\}^m \to \{0,1\}^w$ that is an arbitrary PPT-computable function (see below);
– an *extended verification oracle* (arithmetization oracle) $\widehat{\mathsf{vo}}\colon \mathbb{F}^{m+\lambda+w} \to \mathbb{F}$ that is a random extension of individual degree $d \geq 2$ of the verification oracle $\mathsf{vo}\colon \{0,1\}^{m+\lambda+w} \to \{0,1\}$ defined as follows:

$$\mathsf{vo}(x,y,z) := \begin{cases} 1 & \text{if } \mathsf{ro}(x) = y \text{ and } \mathsf{wo}(x) = z \\ 0 & \text{otherwise} \end{cases}.$$

We discuss each oracle in turn.

– The random oracle ro models the hash function H, as in the standard ROM.
– The witness oracle wo models the witness function W_H. It is defined via a polynomial-size oracle circuit B chosen arbitrarily before the oracle is sampled. On a query x, wo outputs $B^{\mathsf{ro}}(x, \mu_x)$ where μ_x is sampled uniformly at random (and is not resampled if x is queried again). The inclusion of μ_x allows our definition to subsume, e.g., modeling W_H as a random oracle. The efficiency requirement is necessary to allow for efficient simulation of wo (it prevents wo from being used to break standard-model cryptography).

– The verification function vo models the boolean formula Φ_H. Indeed, the definition of vo is directly obtained from the definition of Φ_H by replacing H with ro and W_H with wo.
– The extended verification oracle $\widehat{\mathsf{vo}}$ models the polynomial P_H. The requirement that $d \geq 2$ arises from a technical concern: as noted in [JKRS09], access to the unique multilinear ($d = 1$) extension of a function can be surprisingly powerful. (E.g., an adversary with access to the multilinear extension of $\widehat{\mathsf{vo}}$ can efficiently invert ro, see Remark 1.) Requiring $d \geq 2$ avoids this issue and is sufficient for our security proofs. In any case, we want to match the degree of $\widehat{\mathsf{vo}}$ to that of P_H for some concrete hash function H, and the degree of P_H will be at least 2 in each variable.[4]

A construction that makes black-box use of H, W_H, Φ_H can be analyzed in the AROM as suggested by the above discussion: replace H with ro, W_H with wo, and P_H with $\widehat{\mathsf{vo}}$ (with matching degree bound d).

In Section 2.3 we describe our construction of PCD in the AROM. This construction relies on a "lazy sampling" procedure for the AROM, a key technical contribution that we describe in Section 2.4.

AROM vs. LDROM. Superficially the AROM and LDROM seem quite similar; indeed, they both aim to capture some arithmetization of the random oracle. However, there are notable differences between the two models, even putting aside the differing instantiability considerations. We highlight a few such differences.
– The LDRO is a low-degree extension over a field $\mathbb{F}$ of a random function $\{0,1\}^m \to \mathbb{F}$. Hence the security of the LDRO as a random oracle depends on $|\mathbb{F}|$. The ARO decouples the choice of $\mathbb{F}$ from the random oracle: one may choose the codomain $\{0,1\}^\lambda$ of ro independently from the field $\mathbb{F}$ over which $\widehat{\mathsf{vo}}$ is defined. The security of ro (even in the presence of $\widehat{\mathsf{vo}}$) depends only on λ. That said, in *both* the LDROM and the AROM, the security of their respective query accumulation schemes depends on $|\mathbb{F}|$.
– The LDRO is a linear code random oracle; i.e., it is sampled at random from a linear space over $\mathbb{F}$. The ARO is also sampled uniformly from some set, but this set does not form a linear space. This means that tools developed in [CCS22] for analyzing linear code random oracles do not directly apply. That said, the ARO does have *some* linear structure: the oracle $\widehat{\mathsf{vo}}$ is sampled uniformly from the (affine) space of low-degree extensions of vo. This fact will be useful for emulating the AROM.
– The LDRO has security properties (e.g. collision resistance, unconditional SNARKs) even when $d = 1$ (i.e., it is a random *multilinear* polynomial). The ARO is not even one-way when $d = 1$.

Remark 1 (choice of extension). We set $\widehat{\mathsf{vo}}$ to be a random extension of vo of individual degree $d \geq 2$. We explain why setting $\widehat{\mathsf{vo}}$ to be an arbitrary extension of vo would grant the adversary too much power.

First consider the case when $\widehat{\mathsf{vo}}$ is the unique multilinear extension of vo ($d = 1$). Given oracle access to a multilinear polynomial P over a field $\mathbb{F}$ of characteristic different from 2, a single query to P suffices to efficiently evaluate the sum $\sum_{x \in \{0,1\}^n} P(x)$

[4] The degree of a variable in P_H is equal to the number of clauses in Φ_H in which it appears. Every wire appears in at least two clauses in Φ_H: once as an output and once as an input.

[JKRS09]. We can use this capability and the structure of vo to invert ro: given a target image $y \in \{0,1\}^\lambda$, perform a binary search for a preimage of y by evaluating the sum $\sum_{x_1,z} \widehat{\mathsf{vo}}((x_0,x_1),y,z)$ for different prefixes x_0.

Next consider the higher-degree case: $\widehat{\mathsf{vo}}$ is an *adversarially-chosen* extension of vo of degree $d \geq 2$. Given oracle access to a polynomial P of individual degree d, a single query to P suffices to efficiently evaluate the sum $\sum_{x \in H^n} P(x)$ where H is a multiplicative subgroup of $\mathbb{F}$ with $|H| > d$ [CFS17, Lemma A.4]. Assume that $\mathbb{F}$ has such a subgroup H of size $d+1$. We can embed $\{0,1\}$ into H via an affine shift, and so we may abuse notation to assume $\{0,1\} \subseteq H$. Choose $\widehat{\mathsf{vo}}$ to be the following extension of $\mathsf{vo}\colon \{0,1\}^n \to \mathbb{F}$: $\widehat{\mathsf{vo}}(x,y,z) = \mathsf{vo}(x,y,z)$ for $(x,y,z) \in \{0,1\}^n$, and $\widehat{\mathsf{vo}}(w) = 0$ for $w \in H^n \setminus \{0,1\}^n$. Note that $\widehat{\mathsf{vo}}$ has individual degree $|H| - 1 = d$. Given this extension we can then use binary search as in the multilinear case to invert ro.

The above gives some justification for modelling $\widehat{\mathsf{vo}}$ as a *random* low-degree extension of vo. Of course, there are many choices that lie in between adversarial and random. For example, one could set $\widehat{\mathsf{vo}}$ to be drawn from an adversarially-chosen distribution with "enough" entropy. It is not clear, however, whether such a choice would be substantially closer to "reality" than our choice.

2.3 Building PCD secure in the AROM

Prior work [CCS22] shows that to obtain PCD in an oracle model $\mathcal{O}$, it suffices to construct: (i) a SNARK for NP relative to $\mathcal{O}$; and (ii) an accumulation scheme for $\mathcal{O}$-queries relative to $\mathcal{O}$. Further, the resulting PCD scheme is zero-knowledge if the SNARK and accumulation scheme also satisfy zero-knowledge. The PCD construction in the LDROM in [CCS22] follows by establishing these results for the LDROM. Similarly, our construction of PCD will follow by establishing these results for the AROM.

(i) SNARKs in the AROM. [CCS22] prove that Micali's SNARK remains (information-theoretically) secure in the LDROM, via a rewinding argument. In the AROM, we show a much more general theorem.

Theorem 3 (informal)**.** *Let* p *be a predicate that queries* ro*, and let* $\mathcal{A}$ *be an algorithm querying* $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$ *that outputs* x *satisfying* p^{ro} *with probability* ε*. Then there is an algorithm* $\mathcal{B}$*, of similar efficiency to* $\mathcal{A}$*, that queries* ro *only and outputs* x *satisfying* p^{ro} *with probability* $\varepsilon - \mathrm{negl}(\lambda)$.

Theorem 3 follows directly from our emulator for $\widehat{\mathsf{vo}}$, which we discuss further in Section 2.4. It is *not* known whether a similar result holds for the LDROM.

As an illustrative example, we can use Theorem 3 to prove that the ARO is collision-resistant. By applying Theorem 3 to the predicate p^{ro} that, given $(x,x') \in \{0,1\}^m \times \{0,1\}^m$, checks that $x \neq x'$ and $\mathsf{ro}(x) = \mathsf{ro}(x')$, we deduce that the ARO is collision-resistant from the fact that the RO is collision-resistant.

We use Theorem 3 to prove knowledge soundness and zero knowledge of Micali's SNARK in the AROM.

– **Knowledge soundness.** We use Theorem 3 to prove that Micali's SNARK is secure in the AROM, via a *straightline* extractor. Informally, since we can cast knowledge soundness of Micali's SNARK as an oracle predicate p, any adversary $\mathcal{A}$ that breaks

that security property in the AROM can be transformed via Theorem 3 into an adversary $\mathcal{B}$ that breaks it in the ROM. We can then apply the straightline extractor for Micali's SNARK to $\mathcal{B}$. Since $\mathcal{B}$ invokes $\mathcal{A}$ in a straightline manner, the resulting AROM extractor is also straightline.

– **Zero-knowledge.** We prove that Micali's SNARK is zero knowledge in the AROM. Our zero knowledge simulator that *programs* the oracle; this is a commonality with the zero knowledge simulators for Micali's SNARK in both the ROM and in the LDROM (see [CCS22]). To program the oracle, the simulator relies on a slightly stronger version of our emulator, which emulates oracle queries conditioned on an input list of (real) oracle query-answer pairs. Our hybrid argument invokes Theorem 3 and the Micali SNARK's zero knowledge property in the ROM. Informally, we move between hybrids in the ROM vs AROM using Theorem 3, setting the predicate p to be any distinguisher between hybrids.

(ii) An accumulation scheme for ARO queries. The accumulation scheme for LDRO queries in [CCS22] is obtained by applying the Fiat–Shamir transformation to the (interactive public-coin) query reduction protocol of [KR08]. We follow the same template in the case of the ARO. The first observation is that it suffices to accumulate queries to $\widehat{\mathsf{vo}}$ only, because a query to ro or wo can be verified via a query to $\widehat{\mathsf{vo}}$.[5]

The [KR08] query reduction protocol itself works for any low-degree polynomial: in particular, for $\widehat{\mathsf{vo}}$. As in [CCS22], the central challenge is showing soundness of the Fiat–Shamir transformation in this setting. Note that here we *cannot* appeal to our general theorem above because the verification predicate queries $\widehat{\mathsf{vo}}$.

The soundness of our accumulation scheme is captured by a *zero-finding game* (ZFG). First explicitly described by [BCMS20], the most basic form of a ZFG challenges the adversary to output a commitment cm (under a standard-model commitment scheme) to a low-degree polynomial $f \not\equiv 0$ such that $f(\mathsf{ro}(\mathsf{cm})) = 0$. Intuitively this is hard because f is fixed by cm before $\mathsf{ro}(\mathsf{cm})$ is known, and so the probability that $f(\mathsf{ro}(\mathsf{cm})) = 0$ cannot be much larger than the probability that $f(\alpha) = 0$ for a random $\alpha \in \mathbb{F}$, which is negligible for large fields. [CCS22] shows that a more general version of the ZFG holds in the LDROM, where the ZFG polynomial may depend in a restricted way on the LDRO itself. That is, they show that it is hard to find a commitment cm to polynomials f, g such that $f - \hat{\rho} \circ g \not\equiv 0$ but $(f - \hat{\rho} \circ g)(\hat{\rho}(\mathsf{cm})) = 0$.

The security of our construction depends on the hardness of a similar problem in the AROM, captured by the following lemma.

Lemma 1 (informal)**.** *It is hard for any polynomial-size adversary with access to the ARO* $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$ *to find a commitment* cm *to a pair of low-degree polynomials* f, g *such that* $f - \widehat{\mathsf{vo}} \circ g \not\equiv 0$ *but* $(f - \widehat{\mathsf{vo}} \circ g)(\mathsf{ro}(\mathsf{cm})) = 0$.

We prove Lemma 1 by adapting the proof of the ZFG in [CCS22]. The proof relies on a *forking lemma* in the LDROM, which in turn relies on the ability to efficiently simulate the oracle in order to sample a forking transcript. For the AROM, we will rely on the emulator described in Section 2.4. The proof proceeds as follows. Looking ahead, we note that the emulator maintains a polynomial P that it uses to answer queries to $\widehat{\mathsf{vo}}$.We

[5] Recall that $\mathsf{ro}(x) = y$ and $\mathsf{wo}(x) = z$ if and only if $\widehat{\mathsf{vo}}(x, y, z) = 1$.

show that the adversary cannot win the ZFG when $\widehat{\mathsf{vo}}$ is replaced by P. This argument uses the forking lemma with respect to the emulator, and follows [CCS22], with one difference: this approach doesn't require a bespoke forking lemma as in [CCS22], and can be carried out using a general forking lemma [BN06, Lemma 1]. This general forking lemma is designed for random oracle adversaries, however, as we have already replaced $\widehat{\mathsf{vo}}$ with the emulator we can "perfectly emulate" P using the emulator. Further, we can perfectly emulate wo using the witness circuit B. This allows us to reduce the ZFG adversary to a random oracle adversary and thus apply the general forking lemma. Then, since the emulator is statistically indistinguishable from $\widehat{\mathsf{vo}}$, the adversary cannot win the original ZFG.

Before we describe our emulator, we discuss an important feature of our PCD construction.

Extraction and PCD depth. Almost all constructions of PCD suffer from the "extractor blowup" problem. To obtain a PCD transcript of depth d, we apply the SNARK extractor to itself d times. If the extractor corresponding to a size-S adversary is of size S^c, then the final extractor size is s^{c^d}, where s is the size of the original PCD adversary. As a result, one obtains meaningful security guarantees when d is a constant.

There is a single construction that does not suffer from this issue: the construction of [CT10]. This is because their SNARK (in their signed random oracle model) is straightline (or "list") extractable. Micali's SNARK is also straightline extractable in the ROM [Val08]. Of course, after heuristically instantiating the oracle there is no longer any notion of "straightline". On the other hand, we can easily show that Micali's SNARK is straightline extractable in the AROM. (We do not know how to show this in the LDROM; [CCS22] instead gives a rewinding extractor for Micali's SNARK.)

As a result, our PCD construction is *secure for arbitrary recursion depth*.

2.4 Emulation of the ARO

As discussed in Section 2.3, we aim to construct PCD in the AROM by proving that cryptographic properties in the ROM, specifically knowledge soundness and zero-knowledge of the Micali SNARK, also hold in the AROM. To this end, we design an efficient algorithm $\mathcal{M}$ that answers queries in a way that is statistically indistinguishable from answers of the ARO. We refer to such an algorithm as an *emulator* $\mathcal{M}$ for the AROM.[6]

Recall that the ARO consists of a tuple of oracles $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$. Our emulator $\mathcal{M}$ achieves a special (stronger) type of emulation: given oracle access to some ro and wo, $\mathcal{M}$ can efficiently emulate $\widehat{\mathsf{vo}}$ drawn from the ARO distribution *conditioned* on $(\mathsf{ro}, \mathsf{wo})$.We use this type of emulation to prove Theorem 3.

Lemma 2 (informal)**.** *There exists a probabilistic algorithm $\mathcal{M}$ such that for every security parameter $\lambda \in \mathbb{N}$, query bound $t \in \mathbb{N}$, and t-query adversary,*

$$\left| \Pr_{(\mathsf{ro},\mathsf{wo},\widehat{\mathsf{vo}}) \leftarrow \mathcal{O}(\lambda)} \left[\mathcal{A}^{(\mathsf{ro},\mathsf{wo},\widehat{\mathsf{vo}})} = 1 \right] - \Pr_{(\mathsf{ro},\mathsf{wo},\widehat{\mathsf{vo}}) \leftarrow \mathcal{O}(\lambda)} \left[\mathcal{A}^{\mathcal{M}^{(\mathsf{ro},\mathsf{wo})}} = 1 \right] \right| \leq \frac{t}{2^\lambda} \ . \quad (1)$$

[6] Emulators are sometimes known as "lazy samplers" or "simulators". In this paper we reserve the word *simulator* to refer to zero knowledge simulators.

Moreover, $\mathcal{M}$ is **degenerate** *with respect to $(\mathsf{ro}, \mathsf{wo})$: it answers queries to those oracles by forwarding them to the corresponding "real" oracle (and recording the answers).*

We refer to the absolute difference in Equation 1 as the *emulation error*. An emulator is *perfect* if it has zero emulation error.

Prior oracle emulators. Recall that a random oracle is a function ro chosen uniformly from $(\{0,1\}^m \to \{0,1\}^\lambda)$. It has a well-known perfect (stateful) emulator $\mathcal{M}_{\mathsf{ro}}$ that "lazily" samples answers: when $\mathcal{M}_{\mathsf{ro}}$ receives a new query $x \in \{0,1\}^m$, it uniformly samples and returns $y \in \{0,1\}^\lambda$, and then saves the query-answer pair (x, y) into its state; when $\mathcal{M}_{\mathsf{ro}}$ receives a repeat query, it returns the saved answer.

The low-degree random oracle [CCS22] also has a perfect emulator, based on *succinct constraint detection* for the Reed–Muller code [BCF+17].

Challenges for the ARO. The low-degree structure of $\widehat{\mathsf{vo}}$ may suggest that succinct constraint detection directly yields a construction of $\mathcal{M}^{(\mathsf{ro},\mathsf{wo})}$ with perfect emulation. However, the "sparsity" of vo implies that the set of all possible $\widehat{\mathsf{vo}}$ is *not* a linear space, as we now explain. Recall that for $x \in \{0,1\}^m, y \in \{0,1\}^\lambda, z \in \{0,1\}^w$, $\widehat{\mathsf{vo}}(x, y, z) = \mathsf{vo}(x, y, z) = 1$ if and only if $y = \mathsf{ro}(x)$ and $z = \mathsf{wo}(x)$, and 0 otherwise. Hence, if $\widehat{\mathsf{vo}}_1, \widehat{\mathsf{vo}}_2$ are extended verification oracles, $\widehat{\mathsf{vo}}' = \widehat{\mathsf{vo}}_1 + \widehat{\mathsf{vo}}_2$ many not be an extended verification oracle because there may exist x, y_1, y_2, z_1, z_2 such that $\widehat{\mathsf{vo}}'(x, y_1, z_1) = \widehat{\mathsf{vo}}'(x, y_2, z_2) = 1$ and $y_1 \neq y_2$. Hence, unlike for the LDRO, *we cannot directly construct $\mathcal{M}^{(\mathsf{ro},\mathsf{wo})}$ from succinct constraint detection.*

Our approach. We adopt a novel approach to simulation. First, we design a query-efficient but time-inefficient perfect emulator for a random low-degree extension $\hat{f}$ of a given arbitrary function f.[7] This *almost* suffices for our goal because $\widehat{\mathsf{vo}}$ is a random low-degree extension of the function vo defined by $(\mathsf{ro}, \mathsf{wo})$, which we can efficiently compute at any point by querying ro and wo. Second, we additionally achieve time-efficient emulation by leveraging the *sparsity* of vo, at the cost of a small statistical emulation error.

(1) Time-inefficient emulation of a random low-degree extension. Let $f\colon \{0,1\}^n \to \mathbb{F}$ be a function and $d \in \mathbb{N}$ a degree bound. We seek an emulator $\mathcal{M}_{\mathsf{LD}}$ such that $\mathcal{M}_{\mathsf{LD}}^f$ answers queries in a way that is identically distributed to a random extension $\hat{f}$ of f with individual degree at most d.

We fix some notation. For $S \subseteq \mathbb{F}^n$ and $w \in \{0,1\}^n$, we say that w is S-good if there exists an n-variate polynomial $Q_{w,S}$ of individual degree at most d such that: (i) $Q_{w,S}(w) = 1$; (ii) $Q_{w,S}(x) = 0$ for every $x \in \{0,1\}^n \setminus \{w\}$; and (iii) $Q_{w,S}(z) = 0$ for every $z \in S$. We say that w is S-bad if it is not S-good. Intuitively, w is S-bad if $f(w)$ can be deduced from $\hat{f}|_S$ (given the structure of a low-degree extension $\hat{f}$). Note that S-badness is monotone with respect to S, and that if $w \in S$ then w is S-bad.

The query-efficient but time-inefficient emulator $\mathcal{M}_{\mathsf{LD}}$ works as follows.

$\mathcal{M}_{\mathsf{LD}}^f$:

[7] In contrast, emulating the low-degree random oracle (as in [CCS22]) corresponds to emulating $\hat{f}$ for a *random* function f that the emulator *samples itself*. This considerably simplifies the task, and in particular enables a time-efficient perfect emulation.

1. Initially sample a random low-degree extension P of the all-zero function on $\{0,1\}^n$.
2. For each new query x, answer it as follows.
   - Let $S \subseteq \mathbb{F}^m$ be the set of points queried prior to x.
   - Let W denote the set of $w \in \{0,1\}^n$ that are S-good and $(S \cup \{x\})$-bad.
   - Update $P := P + \sum_{w \in W} f(w) \cdot Q_{w,S}$.
   - Return $P(x)$ as the answer.

The emulator $\mathcal{M}_{\mathsf{LD}}$ maintains the invariant that P is a low-degree extension of the function $g\colon \{0,1\}^n \to \mathbb{F}$ given by $g(w) = 0$ for S-good w and $g(w) = f(w)$ for S-bad w. That is, g is consistent with f at every point the adversary "knows", and is zero elsewhere. It is also crucial that $\mathcal{M}_{\mathsf{LD}}$ does not change $P(x)$ for any $x \in S$, since such a change would be detectable by the adversary; this is achieved since $Q_{w,S}(x) = 0$ for all $x \in S$. Together these facts imply that $\mathcal{M}_{\mathsf{LD}}$ achieves perfect simulation.

The query complexity of $\mathcal{M}_{\mathsf{LD}}$ is equal to the size of the union of all sets W across all invocations of Step 2, which is equal to the number of S-bad points when S is the set of *all* queries made to $\hat{f}$. Aaronson and Wigderson [AW09, Lemma 4.3] proved that, provided $d \geq 2$, the number of S-bad points is at most $|S|$, which in the context of Lemma 2 is the query complexity of $\mathcal{A}$.

(2) Time-efficient emulation from sparsity. There are three main sources of time-inefficiency in the emulator $\mathcal{M}_{\mathsf{LD}}$: (i) sampling the initial polynomial P; (ii) computing the polynomials $Q_{w,S}$; and (iii) computing the set W. We consider each of these difficulties in turn. Throughout we will make use of the random multivariate polynomial emulation algorithm of [BCF+17], which achieves the following guarantee.

Lemma 3. *There is an efficient probabilistic algorithm* LDSample *such that for every degree bound $d \in \mathbb{N}$, set $S \subseteq \mathbb{F}^m$, map $h\colon S \to \mathbb{F}$, $q \in \mathbb{F}^m$, and $\alpha \in \mathbb{F}$,*

$$\Pr[\mathsf{LDSample}(1^d, S, h, q) = \alpha] = \Pr[P(q) = \alpha \mid P|_S = h] \ ,$$

where P is a uniformly random m-variate polynomial of individual degree at most d.[8]

We address the first two difficulties via suitable use of algebra.

(i) *Sampling P.* The polynomial P is initially a random low-degree extension of the all-zero function. We do not know how to use LDSample to sample P directly, since that would need $S = \{0,1\}^m$, which is an exponentially-sized set. Instead, we use a structural result about low-degree extensions of the zero function, the *combinatorial nullstellensatz* [Alo99].

Lemma 4 (informal)**.** *If a polynomial P is zero on $\{0,1\}^m$, then there exist polynomials $(R_i)_{i=1}^m$ such that*

$$P(\vec{X}) \equiv \sum_{i=1}^{m} X_i(X_i - 1)R_i(\vec{X}) \ . \tag{2}$$

[8] If the RHS is not well-defined, LDSample outputs $\perp$.

Combining Lemma 4 with a linear-algebraic argument, we show that sampling each R_i in Equation 2 uniformly at random yields a uniformly random low-degree extension of the all-zero function. We can then sample each R_i via $\mathsf{LDSample}$.

(ii) *Computing $Q_{w,S}$.* [AW09] sets the polynomial $Q_{w,S} := \delta_w \cdot p_{w,S}$ where:
- δ_w is the unique multilinear polynomial with $\delta_w(w) = 1$ and $\delta_w(x) = 0$ for all $x \in \{0,1\}^m \setminus \{w\}$;
- $p_{w,S}$ is a multilinear polynomial with $p_{w,S}(w) = 1$ and $p_{w,S}(z) = 0$ for all $z \in S$.

The polynomial δ_w is easy to compute because it has a succinct expression. In contrast, the polynomial $p_{w,S}$ may not have a succinct expression, but is specified via its evaluations on the polynomial-sized set $S \cup \{w\}$, so queries to $p_{w,S}$ can be answered via $\mathsf{LDSample}$.

We do not know of an algorithm that can efficiently compute, given a set $S \subseteq \mathbb{F}^n$, the set of all S-bad points. As a result, we do not know how to obtain an efficient emulator for a random low-degree extension of an arbitrary function f. Nevertheless we address the third difficulty by leveraging the structure of vo.

(iii) *The set W.* We observe that vo is sparse: it is nonzero only at points (x, y, z) for $\mathsf{ro}(x) = y, \mathsf{wo}(x) = z$. If the adversary has not queried ro at x, then intuitively (since $\mathsf{ro}(x)$ is random) it will not be able to find any y, z such that $\mathsf{vo}(x, y, z) = 1$, even given access to $\widehat{\mathsf{vo}}$. In particular, the probability that the set W contains any $(x, y, z) \in \{0,1\}^{m+\lambda+w}$ such that $\mathsf{vo}(x, y, z) = 1$, but $\mathsf{ro}(x)$ was not yet queried, should be negligible. Indeed, we show that this probability is at most $|S|/2^\lambda$.
Observe that Step 2 of the time-inefficient emulator does nothing if $f(w) = 0$ for all $w \in W$. It follows from the above that to achieve simulation accuracy $O(|S|/2^\lambda)$, it suffices to perform Step 2 only for points (x, y, z) for which the adversary has already queried ro at x and $\mathsf{ro}(x) = y, \mathsf{wo}(x) = z$. Since we observe the adversary's queries to ro, this set of points is easy to determine.
To show this formally we follow an "identical-until-bad-is-set" analysis [BR06].

3 Preliminaries

3.1 Notations

We define $[n] := \{1, \ldots, n\}$. We use $\mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ to denote the set of m-variate polynomials of individual degree at most d with coefficients in $\mathbb{F}$; we write $\deg(\cdot)$ to denote the individual degree. For $\vec{d} = (d_1, \ldots, d_m)$, we use $\mathbb{F}^{\leq \vec{d}}[X_1, \ldots, X_m]$ to denote the set of m-variate polynomials such that the variable X_i has individual degree at most d_i for each $i \in [m]$.

Functions. We use $\mathrm{Dom}(f)$ to denote the domain and $\mathrm{Cod}(f)$ to denote the codomain of a function f. We use $(X \to Y)$ to denote the set of all functions $\{f \colon X \to Y\}$. For a linear map Φ, we use $\ker(\Phi)$ to denote the kernel of Φ and $\mathrm{im}(\Phi)$ to denote the image of Φ. We say that a function is total if it is defined for all elements of its domain, and say that it is not total otherwise.

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

Oracles. A ***random oracle*** is defined as a function ro sampled uniformly at random from $(\{0,1\}^m \to \{0,1\}^n)$ for some $m, n \in \mathbb{N}$.

Oracle algorithms. For a function $\theta\colon X \to Y$, we write A^{θ} for an algorithm with oracle access to θ. Further, for a tuple of functions $(\theta_1, \ldots, \theta_\nu)$, where $\nu \in \mathbb{N}$, with $\theta_i\colon X_i \to Y_i$, we write $A^{(\theta_1,\ldots,\theta_\nu)}$ for an algorithm with oracle access to each θ_i for $i \in [\nu]$, and A^{θ_S} for an algorithm with oracle access to a subset of functions $\{\theta_i |\ i \in S\}$ where $S \subseteq [\nu]$.

Oracle identifiers and oracle transcripts. Given an oracle distribution $\mathcal{O}$ such that $\mathrm{supp}(\mathcal{O})$ contains tuples of $\nu \in \mathbb{N}$ oracles, we assign to each oracle in the tuple a unique oracle identifier: $\mathsf{oid} \in [\nu]$. Then, an $\mathcal{O}$-query-answer transcript tr is a list consisting of query-answer pairs, along with the oracle identifier corresponding to the oracle to which each query was made. That is, $\mathsf{tr} := [(\mathsf{oid}_i, x_i, y_i)]_{i=1}^{t}$ such that there exists $(\theta_1, \theta_2, \ldots, \theta_\nu) \in \mathrm{supp}(\mathcal{O})$ satisfying $\theta_{\mathsf{oid}_i}(x_i) = y_i$ for all $i \in [t]$. (Note that $\mathsf{oid}_i \in [\nu]$ indicates that the i-th query was made to θ_{oid_i}.)

When considering oracle distributions $\mathcal{O}$ whose support contains tuples of oracles, it is often useful to view this tuple $(\theta_1, \ldots, \theta_\nu)$ as a bundle of oracles $\theta\colon [\nu] \times \bigcup_{(\theta_1,\ldots,\theta_\nu)\in\mathrm{supp}(\mathcal{O})} \bigcup_{i\in[\nu]} \mathrm{Dom}(\theta_i) \to \bigcup_{(\theta_1,\ldots,\theta_\nu)\in\mathrm{supp}(\mathcal{O})} \bigcup_{i\in[\nu]} \mathrm{Cod}(\theta_i)$ which takes an oracle identifier as input alongside the query point, i.e., $\theta \leftarrow \mathcal{O}$ and $\theta(\mathsf{oid}_i, x) = \theta_{\mathsf{oid}_i}(x)$.

Query complexity. An algorithm with access to an oracle $\mathcal{O}$ is t-query if its $\mathcal{O}$-query-answer transcript has length $\leq t$.

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

Oracle relations. For a set of oracle distributions $\mathcal{X}$, we write $\mathcal{R}^{\mathcal{X}}$ to denote the set of indexed relations $\{\mathcal{R}^{\theta} : \theta \in \bigcup_{\mathcal{O}\in\mathcal{X}} \mathrm{supp}(\mathcal{O})\}$. When considering sets of oracle distributions $\mathcal{X}$ for which each $\mathcal{O} \in \mathcal{X}$ is such that $\mathrm{supp}(\mathcal{O})$ contains tuples of oracles $(\theta_1, \ldots, \theta_\nu)$, for $\nu \in \mathbb{N}$, with oracle identifiers $(\mathsf{oid}_1, \ldots \mathsf{oid}_\nu)$, we write $\mathcal{R}^{(\mathcal{X},\mathsf{oid})}$ to denote the set of indexed relations $\{\mathcal{R}^{\theta_{\mathsf{oid}}} : (\theta_1, \ldots, \theta_\nu) \in \bigcup_{\mathcal{O}\in\mathcal{X}} \mathrm{supp}(\mathcal{O})\}$. We define $\mathcal{R}^{(\mathcal{X},\mathsf{oid})} \in \mathsf{NP}^{(\mathcal{X},\mathsf{oid})}$ if and only if there exists a polynomial-time oracle Turing machine M such that, for every $(\theta_1, \ldots, \theta_\nu) \in \bigcup_{\mathcal{O}\in\mathcal{X}} \mathrm{supp}(\mathcal{O})$, $\mathcal{R}^{\theta_{\mathsf{oid}}} = \{(\mathtt{i}, \mathtt{x}, \mathtt{w}) : M^{\theta_{\mathsf{oid}}}(\mathtt{i}, \mathtt{x}, \mathtt{w}) = 1\}$.

Security parameters. We assume for simplicity that all public parameters have a length of at least λ so that efficient algorithms that 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 $\mathcal{A}$.

For $\nu \in \mathbb{N}$ and a distribution $\mathcal{O}$, whose support contains tuples of oracles $(\theta_1, \ldots, \theta_\nu)$, we refer to an adversary with access to $(\theta_1, \ldots, \theta_\nu) \leftarrow \mathcal{O}$ as an $\mathcal{O}$-adversary.

Stateful algorithms. An algorithm A is *stateful* if it has the following syntax:

- $A.\mathsf{Initialize}(\mathsf{pp}) \to \mathsf{z}$, on input parameters pp, outputs an initial state z.
- $A.\mathsf{Evaluate}(\mathsf{pp}, \mathsf{z}_0, x) \to (\mathsf{z}_1, y)$, on input an old state z_0 and a query x, outputs a new state z_1 and an output y.

3.2 Non-interactive arguments in oracle models

Given a set of oracle distributions $\mathcal{X}$, a (preprocessing) *non-interactive argument* relative for an indexed oracle relation $\mathcal{R}^{\mathcal{X}}$ is a tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ that works as follows. Below we denote by θ an oracle (or tuple of oracles) in the set $\bigcup_{\mathcal{O} \in \mathcal{X}} \mathrm{supp}(\mathcal{O})$.

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

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

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

– *Soundness.* For every oracle distribution $\mathcal{O} \in \mathcal{X}$ and polynomial-size adversary $\tilde{\mathcal{P}}$,

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

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

We also consider straightline knowledge soundness properties and zero knowledge for ARG.

Straightline knowledge soundness. ARG has *straightline knowledge soundness* (with respect to auxiliary input distribution $\mathcal{D}$) if there exists a deterministic polynomial-time extractor $\mathcal{E}$ such that for every oracle distribution $\mathcal{O} \in \mathcal{X}$ and (non-uniform) polynomial-time adversary $\tilde{\mathcal{P}}$,

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

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

$$\left\{ \mathcal{A}^{\theta}(\pi) \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{G}(1^{\lambda}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\theta}(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\theta}(\mathsf{pp}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}^{\theta}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array} \right\} \quad \text{and} \quad \left\{ \mathcal{A}^{\mathcal{S}^{\theta}}(\pi) \;\middle|\; \begin{array}{r} \theta \leftarrow \mathcal{O}(\lambda) \\ \mathsf{pp} \leftarrow \mathcal{S}(1^{\lambda}) \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \xleftarrow{\mathsf{tr}} \mathcal{A}^{\theta}(\mathsf{pp}) \\ \pi \leftarrow \mathcal{S}^{\theta}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) \end{array} \right\} \ . \tag{3}$$

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 $\mathcal{R}^{\mathcal{X}}$ 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, 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 a set of oracle distributions $\mathcal{X}$ if the properties below hold.

Definition 1. *A* **transcript** T *is a directed acyclic graph where each vertex* $u \in V(\mathsf{T})$ *is labeled by local data* $z^{(u)}_{\mathsf{loc}}$ *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 2. *A vertex* $u \in V(\mathsf{T})$ *is* Φ**-compliant** *for* $\Phi \in \mathsf{F}$ *if for all outgoing edges* $e = (u, v) \in E(\mathsf{T})$ *and for all* $\theta \in \bigcup_{\mathcal{O} \in \mathcal{X}} \mathrm{supp}(\mathcal{O})$*:*

- *(base case) if* u *has no incoming edges,* $\Phi^\theta(z^{(e)}, z^{(u)}_{\mathsf{loc}}, \bot, \ldots, \bot) = 1$*;*
- *(recursive case) if* u *has incoming edges* $e_1, \ldots, e_m$*,* $\Phi^\theta(z^{(e)}, z^{(u)}_{\mathsf{loc}}, z^{(e_1)}, \ldots, z^{(e_m)}) = 1$.

We say that T *is* Φ**-compliant** *if* $E(\mathsf{T})$ *is non-empty and all vertices incident to an edge are* Φ*-compliant.*

Completeness. For every oracle distribution $\mathcal{O} \in \mathcal{X}$ and adversary $\mathcal{A}$,

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

Straightline knowledge soundness. $\mathsf{PCD} = (\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$ has straightline knowledge soundness (with respect to auxiliary input distribution $\mathcal{D}$) if there exists a deterministic polynomial-time extractor $\mathbb{E}$ such that for every oracle distribution $\mathcal{O} \in \mathcal{X}$ and (non-uniform) polynomial-time adversary $\tilde{\mathbb{P}}$,

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

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

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

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, $z_i = \bot$,

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 set of oracle distributions; then, in Definition 3 below, we describe how to specialize that notion to the case of accumulating oracle queries.

Let $\Phi\colon \bigcup_{\mathcal{O}\in\mathcal{X}} \mathrm{supp}(\mathcal{O}(*)) \times (\{0,1\}^*)^3 \to \{0,1\}$ 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 probabilistic 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 oracle distribution $\mathcal{O} \in \mathcal{X}$ and (unbounded) adversary $\mathcal{A}$,

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

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 oracle distribution $\mathcal{O} \in \mathcal{X}$ and polynomial-size adversary $\mathcal{A}$,

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

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

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

and

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

Here $\mathcal{A}$ is *honest* if it outputs, with probability 1, a tuple $(\mathsf{i}_{\Phi}, [\mathsf{q}_i]_{i=1}^{n}, [\mathsf{acc}_j]_{j=1}^{\ell})$ 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 to specialize the general notion of an accumulation scheme above to the particular case of accumulating queries to a tuple of oracles.

Definition 3. *Let $\mathcal{X}$ be a set of oracle distributions. An* **accumulation scheme for $\mathcal{X}$-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);*[9] *(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

Let $\nu \in \mathbb{N}$ and let $\mathcal{X}$ be a set of oracle distributions, such that each $\mathcal{O} \in \mathcal{X}$ is a distribution over tuples of oracles $(\theta_1, \ldots, \theta_\nu)$. A *commitment scheme in $\mathcal{X}$* 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 $\mathcal{O} \in \mathcal{X}$ and efficient adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \\ \wedge \\ \mathsf{CM.Commit}(\mathsf{ck}, m_0; \omega_0) = \mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega_1) \end{array} \;\middle|\; \begin{array}{r} (\theta_1, \ldots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda) \\ ((m_0, \omega_0), (m_1, \omega_1)) \leftarrow \mathcal{A}^{(\theta_1, \ldots, \theta_\nu)}(\mathsf{ck}) \end{array} \right] \leq \mathrm{negl}(\lambda) \ .$$

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

$$\mathcal{D}_0(\lambda) := \left\{ (\mathsf{pp}, \mathsf{cm}, \mathsf{aux}) \;\middle|\; \begin{array}{r} (\theta_1, \ldots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda) \\ (m_0, m_1, \mathsf{aux}) \leftarrow \mathcal{A}^{(\theta_1, \ldots, \theta_\nu)}(\mathsf{ck}) \\ \omega \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ \mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, m_0; \omega) \end{array} \right\}$$

[9] If $\mathcal{X}$ is a set of oracle distributions whose support contains tuples of oracles, then x is assumed to start with the oracle identifier corresponding to the oracle being queried.

$$\text{and } \mathcal{D}_1(\lambda) := \left\{ (\mathsf{pp}, \mathsf{cm}, \mathsf{aux}) \;\middle|\; \begin{array}{r} (\theta_1, \ldots, \theta_\nu) \leftarrow \mathcal{O}(\lambda) \\ \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda) \\ (m_0, m_1, \mathsf{aux}) \leftarrow \mathcal{A}^{(\theta_1,\ldots,\theta_\nu)}(\mathsf{ck}) \\ \omega \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ \mathsf{cm} := \mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega) \end{array} \right\} .$$

Note that in this definition CM does not have access to $(\theta_1, \ldots, \theta_\nu)$. The above generalizes the notion of a commitment scheme, which is recovered from the above by setting the oracles to be empty.

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 have the following simple claim about any binding and hiding commitment scheme.

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

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

A proof of the above claim appears in Claim 3.4 of [CCS22].

3.6 Constraint detection for low-degree polynomials

Definition 4. *Let $\vec{d} = (d_1, \ldots, d_m) \in \mathbb{N}^m$. The low-degree polynomial evaluation code is defined as follows:*

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

Further, 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 $\vec{d}\colon \mathbb{N} \to \mathbb{N}^m$ a degree function. We define

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

We recall the notion of constraints for linear codes.

Definition 5. *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 $z(x) \neq 0$.*

We recall the definition of a constraint detector [BCF+17], which is an algorithm that determines whether a set of queries Q is constrained and, if so, outputs a constraint.

Definition 6. *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 basis for the space of constraints $\{z\colon Q \to \mathbb{F} : \forall\ c \in \mathcal{C}, \sum_{x \in Q} z(x)c(x) = 0\}$ on Q if Q is constrained; (ii) $\perp$ 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 $\mathcal{C}_\lambda$.*

The following theorem is proved in [BCF+17]:

Theorem 1. *The code family $\mathrm{LD}[\mathscr{F}, m, \vec{d}]$ has a constraint detector $\mathsf{CD}(1^\lambda, \cdot)$ that runs in time $\mathrm{poly}(m(\lambda), d(\lambda), \log|\mathbb{F}_\lambda|)$, where $d(\lambda) := \max_{i \in [m]} d(\lambda)_i$. In particular, it has efficient constraint detection.*

3.7 Forking lemmas

We state a general forking lemma proved in [BN06].

Lemma 1. *Fix $t, \lambda \in \mathbb{N}$. Let $\mathcal{A}$ be a probabilistic algorithm that on input $x, y_1, \ldots, y_t$ returns a pair (I, σ), where $I \in [t]$ and σ is referred to as a side output. Let* IG *be a probabilistic algorithm that we call the input generator. The accepting probability of $\mathcal{A}$, denoted* acc*, is defined as follows:*

$$\mathrm{acc} := \Pr\left[I \geq 1 \;\middle|\; \begin{array}{r} x \leftarrow \mathsf{IG} \\ y_1, \ldots, y_t \leftarrow \mathcal{U}(\{0,1\}^\lambda) \\ (I, \sigma) \leftarrow \mathcal{A}(x, y_1, \ldots, y_t) \end{array} \right] .$$

The forking algorithm $\mathsf{Fork}_\mathcal{A}$ associated to $\mathcal{A}$ is the probabilistic algorithm that takes input x and proceeds as follows:

(i) Pick coins ρ for $\mathcal{A}$ at random.
(ii) Sample $y_1, \ldots, y_t \leftarrow \mathcal{U}(\{0,1\}^\lambda)$, and run $\mathcal{A}(x, y_1, \ldots, y_t; \rho)$ to obtain (I, σ).
(iii) If $I = 0$ then return $(0, \varepsilon, \varepsilon)$.
(iv) Otherwise, sample $y'_I, \ldots, y'_t \leftarrow \mathcal{U}(\{0,1\}^\lambda)$ and run $\mathcal{A}(x, y_1, \ldots, y_{I-1}, y'_I, \ldots, y'_t; \rho)$ to obtain (I', σ').
(v) If $(I = I'$ and $y_I \neq y'_I)$ then return $(1, \sigma, \sigma')$.
(vi) Otherwise return $(0, \varepsilon, \varepsilon)$.

Let

$$\mathrm{frk} := \Pr\left[b = 1 \;\middle|\; \begin{array}{r} x \leftarrow \mathsf{IG} \\ (b, \sigma, \sigma') \leftarrow \mathsf{Fork}_\mathcal{A}(x) \end{array} \right] .$$

Then

$$\mathrm{frk} \geq \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{t} - \frac{1}{2^\lambda} \right) ,$$

alternatively,

$$\mathrm{acc} \leq \frac{t}{2^\lambda} + \sqrt{t \cdot \mathrm{frk}} .$$

3.8 Identical-until-bad

We consider two programs, G and H, which are written in some pseudocode. We say that G and H are *identical-until-bad* if they are syntactically identical except for statements that follow the setting of a bad flag to true. Somewhat more formally, let G and H be programs written in some pseudocode and let bad be a flag that occurs in both of them. We say that G and H are *identical-until-bad* if their code is the same except possibly places where G has a statement "set the bad flag" followed by some statements S_G while H has a corresponding statement "set the bad flag" followed by some statements S_H, different from S_G.

We refer the reader to [BR06] for further details and a full formal treatment of the notion of identical-until-bad, which requires specification of the programming language in question to fully formalise. We stress that that identical-until-bad is a purely syntactic requirement.

We state the fundamental lemma of game-playing, which is proved in [BR06].

Lemma 2. *Let G and H be identical-until-bad programs and let $\mathcal{A}$ be an adversary. Then*

$$\left|\Pr[\mathcal{A}^G = 1] - \Pr[\mathcal{A}^H = 1]\right| \leq \Pr[\mathcal{A}^G \textit{ sets } \mathsf{bad}] \ .$$

4 Arithmetized random oracle model

We define the arithmetized random oracle model. As a first step, we define the arithmetized random oracle *distribution*, which is defined over tuples $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$, and explain how the oracles $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$ are sampled.

Definition 7. *Let $m \in \mathbb{N}$ be an arity parameter, $\lambda \in \mathbb{N}$ be a security parameter, $r \in \mathbb{N}$ be a randomness-size parameter, $w \in \mathbb{N}$ be a witness-size parameter, and $d \in \mathbb{N}$ be a degree parameter. For all oracle circuits $B\colon \{0,1\}^{m+r} \to \{0,1\}^w$, we define an* **arithmetized random oracle distribution** $\mathrm{ARO}[\mathbb{F}, m, \lambda, d, B]$,[10] *where $\mathbb{F}$ is a finite field and the support of $\mathrm{ARO}[\mathbb{F}, m, \lambda, d, B]$ contains triples $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$ that are sampled as follows:*

1. *Sample the* **random oracle** ro *uniformly at random from $(\{0,1\}^m \to \{0,1\}^\lambda)$.*
2. *For every $x \in \{0,1\}^m$, sample a random string $\mu_x \in \{0,1\}^r$. Then define the* **witness oracle** $\mathsf{wo}\colon \{0,1\}^m \to \{0,1\}^w$ *as $\mathsf{wo}(x) := B^{\mathsf{ro}}(x, \mu_x)$.*
3. *Define the* **verification function** $\mathsf{vo}\colon \{0,1\}^{m+\lambda+w} \to \{0,1\}$ *as*

$$\mathsf{vo}(x, y, z) := \begin{cases} 1 & \textit{if } \mathsf{ro}(x) = y \wedge \mathsf{wo}(x) = z \\ 0 & \textit{o.w.} \end{cases} \ .$$

4. *Sample the* **(extended) verification oracle** $\widehat{\mathsf{vo}}\colon \mathbb{F}^{m+\lambda+w} \to \mathbb{F}$ *uniformly at random from the set*

$$\left\{p \in \mathbb{F}^{\leq d}[X_1, \ldots, X_{m+\lambda+w}] : p \textit{ equals } \mathsf{vo} \textit{ on } \{0,1\}^{m+\lambda+w}\right\} \ .$$

[10] Given $m \in \mathbb{N}$ and the oracle circuit B, the randomness length r and witness size w parameters are determined. Thus r, w do not appear in the parameterization of ARO.

5. *Output* $(\mathsf{ro}, \mathsf{wo}, \widehat{\mathsf{vo}})$.

Next, we define a *family* of ARO distributions, which is parameterized by a family of finite fields $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda\in\mathbb{N}}$ and a family of oracle circuits $\mathcal{B} = \{B_\lambda^{(\cdot)} \colon \{0,1\}^{m(\lambda)} \to \{0,1\}^{w(\lambda)}\}_{\lambda\in\mathbb{N}}$. Here, $\mathcal{B}$ can be interpreted as the set of all possible adversarial strategies for learning information about the random oracle, and λ is the security parameter.

Definition 8. *Let $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of fields, $m\colon \mathbb{N}\to\mathbb{N}$ be an arity function, $w\colon \mathbb{N}\to\mathbb{N}$ be a witness-size function, $\mathcal{B} = \{B_\lambda^{(\cdot)} \colon \{0,1\}^{m(\lambda)} \to \{0,1\}^{w(\lambda)}\}_{\lambda\in\mathbb{N}}$ be a family of oracle circuits, and $d\colon \mathbb{N}\to\mathbb{N}$ be a degree function. We define the* **arithmetized random oracle family** *as*

$$\mathrm{ARO}[\mathscr{F}, m, d, \mathcal{B}] := \{\ \mathrm{ARO}[\mathbb{F}_\lambda, m(\lambda), \lambda, d(\lambda), B_\lambda^{(\cdot)}]\ \}_{\lambda\in\mathbb{N}}\ .$$

The "arithmetized random oracle" is the set of all ARO distributions for polynomial-sized circuit families $\mathcal{B}$.

Definition 9. *Let $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of fields, $m\colon \mathbb{N}\to\mathbb{N}$ be an arity function, $w\colon \mathbb{N}\to\mathbb{N}$ be a witness-size function, and $d\colon \mathbb{N}\to\mathbb{N}$ be a degree function. Then, we define a* **set of arithmetized random oracle families** *as*

$$\mathbf{ARO}[\mathscr{F}, m, d] := \{\mathrm{ARO}[\mathscr{F}, m, d, \mathcal{B}] \colon \mathcal{B} \textit{ is a family of } \mathrm{poly}(\lambda)\textit{-size oracle circuits}\}\ ,$$

where above $\mathcal{B} = \{B_\lambda^{(\cdot)} \colon \{0,1\}^{m(\lambda)} \to \{0,1\}^{w(\lambda)}\}_{\lambda\in\mathbb{N}}$.

Acknowledgments

We thank Giacomo Fenzi for pointing out some inaccuracies in an earlier draft of this paper.

Tom Gur is supported by the UKRI Future Leaders Fellowship MR/S031545/1 and EPRSC New Horizons Grant EP/X018180/1. Jack O'Connor is supported by the Engineering and Physical Sciences Research Council through the Mathematics of Systems Centre for Doctoral Training at the University of Warwick (reference EP/S022244/1). Megan Chen is supported by DARPA under Agreement No. HR00112020023.

References

[Alo99] N. Alon. "Combinatorial Nullstellensatz". In: *Combinatorics, Probability and Computing* (1999).

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

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: STOC '13.

[BCF+17] E. Ben-Sasson, A. Chiesa, M. A. Forbes, A. Gabizon, M. Riabzev, and N. Spooner. "Zero Knowledge Protocols from Succinct Constraint Detection". In: TCC '17.

[BCL+21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-Carrying Data Without Succinct Arguments". In: CRYPTO '21.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. “Proof-Carrying Data from Accumulation Schemes”. In: TCC ’20.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Scalable Zero Knowledge via Cycles of Elliptic Curves”. In: CRYPTO ’14.

[BDFG21] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. “Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments”. In: CRYPTO ’21.

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

[BGV11] S. Benabbas, R. Gennaro, and Y. Vahlis. “Verifiable Delegation of Computation over Large Datasets”. In: CRYPTO ’11.

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

[BN06] M. Bellare and G. Neven. “Multi-signatures in the plain public-Key model and a general forking lemma”. In: CCS ’06.

[BR06] M. Bellare and P. Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: EUROCRYPT ’06.

[BR93] M. Bellare and P. Rogaway. “Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols”. In: CCS ’93.

[CCDW20] W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. “Reducing Participation Costs via Incremental Verification for Ledger Systems”. Cryptology ePrint Archive, Report 2020/1522.

[CCS22] M. Chen, A. Chiesa, and N. Spooner. “On Succinct Non-interactive Arguments in Relativized Worlds”. In: EUROCRYPT ’22.

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

[CL20] A. Chiesa and S. Liu. “On the Impossibility of Probabilistic Proofs in Relativized Worlds”. In: ITCS ’20.

[COS20] A. Chiesa, D. Ojha, and N. Spooner. “Fractal: Post-Quantum and Transparent Recursive Proofs from Holography”. In: EUROCRYPT ’20.

[CT10] A. Chiesa and E. Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards”. In: ICS ’10.

[CTV13] S. Chong, E. Tromer, and J. A. Vaughan. “Enforcing Language Semantics Using Proof-Carrying Data”. ePrint Report 2013/513.

[CTV15] A. Chiesa, E. Tromer, and M. Virza. “Cluster Computing in Zero Knowledge”. In: EUROCRYPT ’15.

[GK03] S. Goldwasser and Y. T. Kalai. “On the (In)security of the Fiat-Shamir Paradigm”. In: FOCS ’03.

[Gro16] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: EUROCRYPT ’16.

[GW11] C. Gentry and D. Wichs. “Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions”. In: STOC ’11.

[HN22] M. Hall-Andersen and J. B. Nielsen. “On Valiant’s Conjecture: Impossibility of Incrementally Verifiable Computation from Random Oracles”. Cryptology ePrint Archive, Paper 2022/542.

[JKRS09] A. Juma, V. Kabanets, C. Rackoff, and A. Shpilka. “The Black-Box Query Complexity of Polynomial Summation”. In: *Computational Complexity* (2009).

[JLLW22] A. Jain, H. Lin, J. Luo, and D. Wichs. “The Pseudorandom Oracle Model and Ideal Obfuscation”. Cryptology ePrint Archive, Paper 2022/1204.

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

[KR08] Y. Kalai and R. Raz. “Interactive PCP”. In: ICALP ’08.

[KST22] A. Kothapalli, S. Setty, and I. Tzialla. “Nova: Recursive Zero-Knowledge Arguments from Folding Schemes”. In: CRYPTO ’22.

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

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

[NT16] A. Naveh and E. Tromer. “PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations”. In: S&P ’16.

[TFZ+22] N. Tyagi, B. Fisch, A. Zitek, J. Bonneau, and S. Tessaro. “VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries”. In: CCS ’22.

[Val08] P. Valiant. “Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency”. In: TCC ’08.

[Zha22] M. Zhandry. “Augmented Random Oracles”. In: CRYPTO ’22.