On Succinct Non-Interactive Arguments in Relativized Worlds

Megan Chen[1], Alessandro Chiesa[2], and Nicholas Spooner[3]

[1] Boston University, Boston MA, USA. megchen@bu.edu
[2] EPFL, Lausanne, Switzerland. alessandro.chiesa@epfl.ch
[3] University of Warwick, Coventry, United Kingdom. Nicholas.Spooner@warwick.ac.uk

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.

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. 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

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. This property leads to exciting applications, and one that has received much attention recently, and motivates this paper, is *recursive proof composition*.

Recursive proof composition is a general technique for "bootstrapping" a SNARG of knowledge (SNARK) into an *incremental* proof system for ongoing computations [BCCT13]. This technique can be used to build incrementally-verifiable computation (IVC) [Val08] and proof-carrying data (PCD) [CT10]. 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; BCI+13; Gro16; GKM+18]) 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 [BCG+15; BGG17; BGM17; ABL+19]. 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; BCL+21; BDFG20; 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 giving a SNARK proof about a verifier that queries the random oracle. It is not known whether there exist SNARKs for such computations in the random oracle model. To avoid this issue, [COS20] performs a *heuristic step*: they assume that there exists a concrete hash function which yields a secure SNARK when used in place of the random oracle. Under this assumption they obtain a SNARK in the standard model, which can be (provably) recursively composed. 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 which can prove statements *about its own oracle*. Such an oracle model admits a proof of security for recursive composition "within the model", without any heuristic step.

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 [AD18; ACG+19; GKR+19; AABS+19; AGP+19]. 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.* What should be the standard-model analogue of a security property in the ROM? Choosing the correct heuristic assumption is a balancing act between the requirements of the standard model proof and what can be justified by the idealized proof. There are typically *many* details here, 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! In contrast, security proofs in idealized models provide clear "end-to-end" guarantees. Finally, heuristic assumptions make it difficult to assess the concrete security of a scheme, while security proofs in idealized models often lead to concrete security expressions (the random oracle model is an excellent example of this).

1.1 Our results

Our main result is to 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 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 $\mathsf{NP}^{\mathcal{O}}$.

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 makes use of 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) Transparent SNARKs in the low-degree random oracle model. We show that, if collision-resistant hash functions exist, there exist SNARKs in the low-degree random oracle model (LDROM) that can prove NP computations relative to the same oracle.

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

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. For the former component we use Micali's construction of a SNARK in the random oracle model [Mic00], whose security proof we adapt to the LDROM. The latter component is a new scheme and a key contribution of this work; we discuss this next.

(3) Accumulation scheme for low-degree random oracle queries. An accumulation scheme for an oracle θ 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, we require that the verifier cannot query the oracle, and that the size of the accumulator and the query complexity of the decider be independent of the number of accumulated queries. This is a variation on the notion of accumulation schemes for *predicates* as introduced by [BCMS20].

In this work, we build an accumulation scheme for the low-degree random oracle. To do so, we build on a prior interactive query reduction protocol [KR08; CFGS18], 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. Proving security of the accumulation scheme is the most technically involved part of this paper, and requires developing new tools that we deem of independent interest; see Section 2.2 for more details.

1.2 Related work

Below we summarize prior works related to idealized oracle models and accumulation schemes.

Generic group model. Shoup [Sho97] introduced the generic group model (GGM), an model 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 $L(x+y)$. The generic bilinear group model (GBM), introduced by Boneh and Boyen [BB04], augments this with an oracle implementing a bilinear map. [ZZ21] shows 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 $L(x)$.[4] A natural question, that we leave open for future

[4] This result is sensitive to the way that the generic group is modelled; in particular, it is not known to hold for the [Mau05] formalization of the GGM.

work, is whether there exist SNARKs in the GGM/GBM that can prove GGM/GBM computations.

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. The notion low-degree random oracles is 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].

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. Finally, in Section 2.5 we describe our SNARK construction that relativizes in the 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 $\mathrm{im}(f)$). 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 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.[5]

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 $\mathcal{C} \subseteq \{0,1\}^m \cup \{\Sigma\} \to \mathbb{F}$ (for a special symbol Σ), consisting of all functions c such that $c(\Sigma) = \sum_{a \in \{0,1\}^m} c(a)$. 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.

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 1. *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

[5] 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.

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) [BCF+17]. 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

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

for $x_1, \dots, x_m \in \mathbb{F}_p$. Note that if F is a random function then P is uniformly random in $\mathbb{F}_p^{\leq d}[X_1, \dots, X_m]$.[6] 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 on the query-reduction technique for IPCPs in [KR08; CFGS18], 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.

[6] 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).

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

- **Completeness:** *For all accumulators* acc *and query-answer pairs* (x, α), *if* $\mathrm{D}^{\theta}(\mathsf{acc}) = 1$ *and* $\theta(x) = \alpha$, *then for* $(\mathsf{acc}', \pi_{\mathrm{V}}) \leftarrow \mathrm{P}^{\theta}(x, \alpha, \mathsf{acc})$ *it holds that* $\mathrm{V}(x, \alpha, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}}) = 1$ *and* $\mathrm{D}^{\theta}(\mathsf{acc}') = 1$.
- **Soundness:** *For efficiently generated accumulators* $\mathsf{acc}, \mathsf{acc}'$, *query-answer pairs* (x, α) *and accumulation proofs* π_{V}, *if* $\mathrm{D}^{\theta}(\mathsf{acc}') = 1$ *and* $\mathrm{V}(x, \alpha, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}}) = 1$ *then* $\theta(x) = \alpha$ *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 $[(x_i, \alpha_i)]_{i=1}^{n}$ and m old accumulators $[\mathsf{acc}_j]_{j=1}^{\ell}$, as in the formal in Section 3.3. For simplicity, below we present our accumulation scheme below for the case of general n and $m = 1$. 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, [CFGS18] 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 $[\mathsf{q}_i]_{i=1}^{n} = \{(x_1, \alpha_1), \ldots, (x_n, \alpha_n)\}$ be a list of (alleged) query-answer pairs. The prover $\mathcal{P}$ wishes to convince the verifier $\mathcal{V}$ that $\hat{\rho}(x_i) = \alpha_i$ 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 $f(b_i) = \alpha_i$ 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 $\hat{\rho}(x_i) = \hat{\rho}(g(b_i)) = f(b_i) = \alpha_i$ 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}, α_{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 $\beta := \hat{\rho}(x_1, \dots, x_{n+1}, f)$ (the embedding from binary strings into the domain of $\hat{\rho}$ is implicit). Note that we include $x_1, \dots, x_{n+1}$ 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 $((x_1, \dots, x_{n+1}, f), \beta)$ for correctness. The natural approach is to store the pair $((x_1, \dots, x_{n+1}, f), \beta)$ in the accumulator, so that an accumulator contains an additional query-answer pair (x_{n+2}, α_{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 $\beta := \hat{\rho}(C)$ for $C := \mathsf{Commit}(x_1, \dots, x_{n+2}, f)$ 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 $[(x_i, \alpha_i)]_{i=1}^{n}$, 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}, α_{n+1}) and (x_{n+2}, α_{n+2}). Set the query-answer list $Q := [(x_i, \alpha_i)]_{i=1}^{n+2}$.
  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 $C := \mathsf{Commit}(x_1, \dots, x_{n+2}, f)$.
  5. *Fiat–Shamir challenge.* Compute the challenge $\beta := \hat{\rho}(C) \in \mathbb{F}$.
  6. *Output.* Output the new accumulator $\mathsf{acc}' := \{(g(\beta), f(\beta)), (C, \beta)\}$ and accumulation proof $\pi_{\mathrm{V}} := f$.

– *Accumulation verifier:* V receives as input $([\mathsf{q}_i]_{i=1}^{n}, \mathsf{acc}, \mathsf{acc}', \pi_{\mathrm{V}})$, where $\mathsf{acc}' = \{(x, \alpha), (C', \beta)\}$, 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 C

as in Step 4 of the prover. Then check that $C' = C$, $f(b_i) = \alpha_i$ for all $i \in [n+2]$, $x = g(\beta)$, and $\alpha = f(\beta)$.

– *Decider:* D checks that the input accumulator $\mathsf{acc} = \{(x, \alpha), (C, \beta)\}$ satisfies $\alpha = \hat{\rho}(x)$ and $\beta = \hat{\rho}(C)$.

Intuitively, the accumulation scheme is secure against efficient attackers because the binding property of the commitment scheme ensures that setting the challenge β to $\hat{\rho}(C)$ is as good as $\hat{\rho}(x_1, \ldots, x_{n+2}, f)$, and the latter 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 $\mathsf{acc}_i = \{(x, \alpha), (C, \beta)\}$, without access to inputs $[\mathsf{q}_i]_{i=1}^{n}$ 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 $[\mathsf{q}_i]_{i=1}^{n}$ 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 $\mathbb{F}^m$.[7] 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 C 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 $z := \hat{\rho}(\mathsf{Commit}(x_1, \ldots, x_{n+2}, f))$. In order 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

[7] Both [KR08] and [CFGS18] 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.

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.[8]

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 (q, o) such that $\mathsf{p}(\mathsf{q}, \mathsf{o}, \mathsf{tr}) = 1$ with probability δ, where $\mathsf{tr} = \{(\mathsf{q}_1, \alpha_1), \dots, (\mathsf{q}_t, \alpha_t)\}$ are the query-answer pairs of $\mathcal{A}$. We think of q as the adversary's "chosen" query, and o as some additional input to the predicate p.

Now suppose that we additionally run $\mathcal{A}$ in a *forked* execution. Let $i := \mathsf{FP}(\mathsf{q}, \mathsf{tr}) \in [t]$ be the location of the query q in its query-answer list tr (abort if q 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 $(\mathsf{q}', \mathsf{o}')$ 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) $\mathsf{p}(\mathsf{q}, \mathsf{o}, \mathsf{tr}) = 1$ ($\mathcal{A}$ wins the original game); (2) $\mathsf{p}(\mathsf{q}', \mathsf{o}', \mathsf{tr}') = 1$ ($\mathcal{A}$ wins the game in the forked execution); (3) $\mathsf{q} = \mathsf{q}'$ (the queries output by $\mathcal{A}$ in the two related executions are equal).[9]

Lemma 1. *Suppose that $\mathcal{A}$ is a t-query random oracle algorithm such that*

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

Then

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

Extending the forking lemma to any linear code random oracle. When extending Lemma 1 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 q 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 q.

Contrastingly, 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 $\hat{\rho}(\mathsf{q})$, 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 q_i at which $\mathcal{A}$ first learns the value of $\hat{\rho}$ at q (and before that $\mathcal{A}$ knows nothing about it); we define $\mathsf{FP}_{\mathcal{C}}(\mathsf{q}, \mathsf{tr}) := i$.

[8] 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.

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

Can $\mathsf{FP}_{\mathcal{C}}$ be efficiently computed? Note that $\mathcal{A}$ may try to obfuscate the fork point by trying to learn $\hat{\rho}(\mathsf{q})$ in some complicated way via the structure of the code. The problem of finding the fork point can be solved via *constraint detection* [BCF+17]: the fork point is the first query i at which there is a linear constraint over the set $\{\hat{\rho}(\mathsf{q}_1), \dots, \hat{\rho}(\mathsf{q}_i), \hat{\rho}(\mathsf{q})\}$. For low-degree random oracles, we can implement $\mathsf{FP}_{\mathcal{C}}$ efficiently using the efficient constraint detection algorithm for Reed–Muller codes of [BCF+17]. For many interesting linear codes, efficient constraint detection is an open problem.

To state the forking lemma requires one additional consideration. Strictly speaking, Lemma 1 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. *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*

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

Then

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

Note that the Fork algorithm here is *the same* as in Lemma 1, 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 C and a non-zero polynomial $f \in \mathbb{F}^{\leq d}[X_1, \dots, X_m]$ such that C is a commitment to f and $f(\rho(C)) = 0$. 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 $\rho(C)$, 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 1), as we now sketch. We define the forking lemma predicate $\mathsf{p}(\mathsf{q}, f, \mathsf{tr})$ to be satisfied if and only if: (i) $f \not\equiv 0$; (ii) $f(\mathsf{tr}(\mathsf{q})) \neq 0$; and (iii) q is a commitment to f. By increasing the query complexity of $\mathcal{A}$ to $t+1$ we can ensure that q 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 1, and so the forking experiment succeeds with probability at least $\delta^2/(t+1)$.

Suppose that the forking experiment succeeds: we obtain $(\mathsf{q}, f, f', \mathsf{tr}, \mathsf{tr}')$ such that f and f' are non-zero polynomials that are both valid openings of the same commitment q, and it holds that $f(\mathsf{tr}(\mathsf{q})) \neq 0$ and $f'(\mathsf{tr}'(\mathsf{q})) \neq 0$. In the forked execution, $\mathsf{tr}'(\mathsf{q})$ is sampled uniformly at random independently of f, and so $\Pr[f(\mathsf{tr}'(\mathsf{q})) = 0] \leq md/|\mathbb{F}|$. If $f = f'$ then the probability that $\mathsf{p}(\mathsf{q}, f', \mathsf{tr}')$ 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 C and two polynomials f, g such that: (1) C is a commitment to (f, g); (2) $f \not\equiv \hat{\rho} \circ g$; and (3) the oracle answer $z := \hat{\rho}(C)$ satisfies $f(z) = (\hat{\rho} \circ g)(z)$. For this we prove the following lemma.

Lemma 3 (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* $\mathcal{A}$*,*

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

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 $\hat{\rho}(C)$ determined by the oracle $\hat{\rho}$ (analogous to $\rho(C)$ 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 3 would be to adapt the proof of the zero-finding game in [BCMS20], using our forking lemma for linear code random oracles (Lemma 2) 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 $\mathsf{p}(C, (f, g, \omega), \mathsf{tr})$ that captures the three conditions on the left. Similarly to before, one condition is "$C = \mathsf{Commit}(f, g; \omega)$". 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 $f(\mathsf{tr}(C)) = 0$. This is justified in their setting because adversaries that do not query the oracle at C cannot win the game. This is *not* true for low-degree oracles! An adversary can, for example, learn $\hat{\rho}(C)$ by querying points on a curve that passes through C. To handle this issue, we use the structure of the linear code to define a partial function $\tilde{\rho}_{\mathsf{tr}}$ which captures all of the information that the adversary knows about $\hat{\rho}$ given the points it has queried. We show that if $\tilde{\rho}_{\mathsf{tr}}(x) = \bot$, 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 C, auxiliary input (f, g, ω), and query transcript tr, p accepts if the following three conditions hold:

(1) $C = \mathsf{Commit}(f, g; \omega)$,
(2) $f \not\equiv \tilde{\rho}_{\mathsf{tr}} \circ g$, and
(3) $f(\tilde{\rho}_{\mathsf{tr}}(C)) = \tilde{\rho}_{\mathsf{tr}}(g(\tilde{\rho}_{\mathsf{tr}}(C)))$.

Note that since $\tilde{\rho}_{\mathsf{tr}}$ is a partial function, so is $\tilde{\rho}_{\mathsf{tr}} \circ g$, in general. If $\tilde{\rho}_{\mathsf{tr}} \circ g$ 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 3 game with probability δ. Invoking our forking lemma (Lemma 2), we obtain, with probability δ^2/t, $(C, (f, g, \omega), (f', g', \omega'), \mathsf{tr}, \mathsf{tr}')$,where p holds for $(C, (f, g, \omega), \mathsf{tr})$ and $(C, (f', g', \omega'), \mathsf{tr}')$. 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 $f - \tilde{\rho}_{\mathsf{tr}} \circ g$. 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 $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is the same function in both forks. Of course, it may be that $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is not total (even if $\tilde{\rho}_{\mathsf{tr}} \circ g$ is). We show that in this case the adversary is very unlikely to win: in particular, it can be shown that $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is not total only if $\tilde{\rho}_{\mathsf{tr}|_{i-1}}(g(\alpha)) = \bot$ for all but $m \cdot \deg(\hat{\rho}) \cdot \deg(g)$ choices of $\alpha \in \mathbb{F}$. But then since $\tilde{\rho}_{\mathsf{tr}'}(C)$ 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 $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is total, we can now apply Schwartz–Zippel to the polynomial $f - \tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$, 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 SNARK in the LDROM for computations in the LDROM, assuming the existence of collision-resistant hash functions in the standard model.

The proof is in two steps: first 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; then we explain how we instantiate the construction specifically for the low-degree random oracle model.

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 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 $\mathsf{AS} = (\mathrm{P}^{\theta}, \mathrm{V}, \mathrm{D}^{\theta})$ for queries to θ. We then combine these two components to obtain a SNARK in the oracle θ model for computations that query θ.

In more detail, our goal is to prove a relation $\mathcal{R}^{\theta} := \{(\mathbb{x}, \mathbb{w}) \colon M^{\theta}(\mathbb{x}, \mathbb{w}) = 1\} \in \mathsf{NP}^{\theta}$, where M is a polynomial-time oracle Turing machine. To build a SNARK for $\mathcal{R}^{\theta}$ using our high-level template, we define the following non-oracle relation:

$$\mathcal{R}' := \{((\mathbb{x}, \mathsf{acc}), (\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})) \colon M'(\mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1 \wedge \mathrm{V}(\mathsf{tr}, \perp, \mathsf{acc}, \pi_{\mathrm{V}}) = 1\} \in \mathsf{NP}\,,$$

where M' works exactly like M except that its oracle queries are answered using tr, V is the accumulation verifier, and $\perp$ denotes an empty accumulator. Intuitively, if $(\mathbb{x}, \mathsf{acc}) \in \mathcal{L}(\mathcal{R}')$ and acc is a valid accumulator, then for tr consistent with θ and some witness $\mathbb{w}$, $M'(\mathbb{x}, \mathbb{w}, \mathsf{tr}) = 1$. This implies that $M^{\theta}(\mathbb{x}, \mathbb{w}) = 1$, and so $\mathbb{x} \in \mathcal{L}(\mathcal{R})$.

Next, we describe our SNARK construction $\mathsf{ARG}_{\mathsf{out}} = (\mathcal{P}^{\theta}_{\mathsf{out}}, \mathcal{V}^{\theta}_{\mathsf{out}})$ in terms of $\mathcal{R}'$.

1. $\mathcal{P}^{\theta}_{\mathsf{out}}$ simulates M and records M's oracle transcript tr. Then, $\mathcal{P}^{\theta}_{\mathsf{out}}$ accumulates the queries in tr using the accumulation prover $\mathsf{acc} \leftarrow \mathrm{P}^{\theta}(\perp, \mathsf{tr})$. Next, $\mathcal{P}^{\theta}_{\mathsf{out}}$ runs the SNARK prover $\mathcal{P}_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), \mathsf{tr})$ to obtain a proof π_{in} that $(\mathbb{x}, \mathsf{acc}) \in \mathcal{R}'$. $\mathcal{P}^{\theta}_{\mathsf{out}}$ then outputs $\pi_{\mathsf{out}} = (\pi_{\mathsf{in}}, \mathsf{acc})$.
2. $\mathcal{V}^{\theta}_{\mathsf{out}}$ receives the input $\pi_{\mathsf{out}} = (\pi_{\mathsf{in}}, \mathsf{acc})$, then runs the SNARK verifier $\mathcal{V}_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}})$ and the accumulation decider $\mathrm{D}^{\theta}(\mathsf{acc})$. If both accept, $\mathcal{V}^{\theta}_{\mathsf{out}}$ accepts.

For (knowledge) soundness, consider a malicious prover $\tilde{\mathcal{P}}$ that outputs an instance $\mathbb{x}$ and a proof $(\pi_{\mathsf{in}}, \mathsf{acc})$ that $\mathcal{V}^{\theta}_{\mathsf{out}}$ accepts, i.e. $\mathcal{V}_{\mathsf{in}}((\mathbb{x}, \mathsf{acc}), \pi_{\mathsf{in}}) = 1$ and $\mathrm{D}^{\theta}(\mathsf{acc}) = 1$. By the knowledge guarantee of the SNARK, we can extract a witness $(\mathbb{w}, \mathsf{tr}, \pi_{\mathrm{V}})$. 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 $\mathcal{R}'$.

For clarity we have omitted parameter generation from the above description. The parameters of $\mathsf{ARG}_{\mathsf{out}}$ are simply the concatenation of the parameters for $\mathsf{ARG}_{\mathsf{in}}$ and AS. If, as in our construction below, both of these have transparent setup, then so does $\mathsf{ARG}_{\mathsf{out}}$.

Step 2: instantiating the building blocks. We explain how to instantiate the above generic construction in the case where θ is a low-degree random oracle. For the accumulation scheme, we use our construction for low-degree oracle queries from Section 2.2. Note that this construction requires a collision-resistant hash function, so our SNARK inherits its public parameters.

Next, we obtain a SNARK for non-oracle computations relative to the low-degree random oracle. It is well known that Micali's SNARG [Mic00], when instantiated with a PCP of knowledge, is a proof of knowledge in the random oracle model via a straightline extractor [Val08]. Naturally, we can carry this construction over to the low-degree random oracle model. However, it is unclear how to construct a straightline extractor. Instead, we make use of a folklore observation: there is an alternative extractor which uses a forking algorithm. This requires polynomially-many forked transcripts (rather than 2); we observe that our forking lemma in the LCROM (Lemma 2) can easily be extended to support this.

We observe that our technique for proving knowledge is not specific to the low-degree oracle. In fact, our forking lemma (Lemma 2) shows that efficient forking lemmas exist for any linear code random oracle with an efficient constraint detection algorithm. Similarly, all linear code random oracles are collision-resistant. Hence, 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.

3 Preliminaries

3.1 Notations

We use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. The notation $\mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ refers to the set of m-variate polynomials of individual degree at most d with coefficients in $\mathbb{F}$. In this paper, we write $\deg(\cdot)$ to denote individual degree. For a distribution D, we denote the support of D by $\mathrm{supp}(D)$. For a function f, we denote the image of f with $\mathrm{im}(f)$. We denote by $(X \to Y)$ the set of all functions $\{f \colon X \to Y\}$. We denote by $f \colon X \rightharpoonup Y$ a partial function from X to Y.

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

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

Distributions. For finite set X, we write $x \leftarrow X$ to denote that x is drawn uniformly at random from X.

Oracle algorithms. For a function $\theta \colon X \to Y$, 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. We say that an oracle algorithm is *systematic* if it only makes queries in $\{0,1\}^m$ for some $m \in \mathbb{N}$.

Random oracles. Typically, a *random oracle* is a function ρ sampled uniformly at random from $(\{0,1\}^m \to \{0,1\}^n)$ for some $m, n \in \mathbb{N}$. It will be convenient for us to consider a slightly broader definition, where ρ is sampled uniformly at random from $(\{0,1\}^m \to \mathbb{F})$ for some finite field $\mathbb{F}$.

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 all $\theta \in \mathrm{supp}(\mathcal{O})$, $\mathcal{R}^\theta = \{(\mathbb{i}, \mathbb{x}, \mathbb{w}) : M^\theta(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 1\}$.

3.2 Non-interactive arguments in oracle models

We follow [CHM+20] and define the tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ to be a (preprocessing) *non-interactive argument* relative to an oracle distribution $\mathcal{O}$ for an indexed oracle relation $\mathcal{R}^\mathcal{O}$ if the algorithms satisfy the following syntax and properties:

1. $\mathcal{G}^\theta(1^\lambda) \to \mathsf{pp}$. On input a security parameter λ (in unary), $\mathcal{G}$ samples public parameters pp for the argument system.
2. $\mathcal{I}^\theta(\mathsf{pp}, \mathbb{i}) \to (\mathrm{ipk}, \mathrm{ivk})$. On input public parameters pp and an index $\mathbb{i}$ for the relation $\mathcal{R}$, $\mathcal{I}$ deterministically specializes pp to index-specific proving and verification keys $(\mathrm{ipk}, \mathrm{ivk})$.
3. $\mathcal{P}^\theta(\mathrm{ipk}, \mathbb{x}, \mathbb{w}) \to \pi$. On input an index-specific proving key ipk, an instance $\mathbb{x}$, and a corresponding witness $\mathbb{w}$, $\mathcal{P}$ computes a proof π that attests to the claim that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$.
4. $\mathcal{V}^\theta(\mathrm{ivk}, \mathbb{x}, \pi) \to b \in \{0,1\}$. On input an index-specific proving key ivk, an instance $\mathbb{x}$, and a corresponding proof π, $\mathcal{V}$ checks that π is a valid proof.

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

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

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

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

The above formulation of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on the 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.

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

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

3.3 Accumulation schemes

We recall the definition of an accumulation scheme from [BCMS20]. 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 $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ all of which (except G) have access to the same oracle θ. These algorithms must satisfy two properties, *completeness* and *soundness*, defined below.

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

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

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 polynomial-size adversary $\mathcal{A}$,

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

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

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

and

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

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 all $i \in [n]$ and $j \in [\ell]$. Note that the simulator S is *not* required to simulate the accumulation verifier proof π_{V}.

3.4 Commitment schemes

Definition 2. *A* **commitment scheme** *for a family of message universes* $\mathcal{M}_{\mathsf{ck}} = \{0,1\}^L$*, commitment universes* $\mathcal{C}_{\mathsf{ck}} = \{0,1\}^{\mathrm{poly}(\lambda, L)}$*, and randomness domains* $\mathcal{R}_{\mathsf{ck}} = \{0,1\}^{\mathrm{poly}(\lambda)}$ *a 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 *and a message format* L*, outputs a commitment key* ck.
- $\mathsf{CM.Commit}$, *on input a commitment key* ck*, a message* $m \in \mathcal{M}_{\mathsf{ck}}$ *and randomness* $\omega \in \mathcal{R}_{\mathsf{ck}}$*, outputs a commitment* $C \in \mathcal{C}_{\mathsf{ck}}$.

The commitment scheme CM *is* **hiding** *if* $\mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda, L)$*, then for every efficient adversary* $\mathcal{A}$ *that chooses* $m_0 \neq m_1 \in \mathcal{M}_{\mathsf{ck}}$*, we have that*

$$\{\mathsf{CM.Commit}(\mathsf{ck}, m_0; \omega_0) \mid \omega_0 \leftarrow \mathcal{R}_{\mathsf{ck}}\} \approx \{\mathsf{CM.Commit}(\mathsf{ck}, m_1; \omega_1) \mid \omega_1 \leftarrow \mathcal{R}_{\mathsf{ck}}\} .$$

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

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

Finally, CM *is* s-**compressing** *if for all* $\mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda, L)$*,* $\mathcal{C}_{\mathsf{ck}} = \{0,1\}^{s(\lambda)}$.

4 Linear code random oracles

In this section we define our main object of study, *linear code random oracles*. We first recall the definition of a linear code and its dual. For a set D and field $\mathbb{F}$, we write $(D \to \mathbb{F})$ or $\mathbb{F}^D$ for the vector space of functions from D to $\mathbb{F}$.

Definition 3. *Let* D *be a set, and* $\mathbb{F}$ *a field. A* **linear code** $\mathcal{C}$ *is a subspace of* $\mathbb{F}^D$*. The* **dual code** *of* $\mathcal{C}$ *is the set* $\mathcal{C}^\perp := \{z \in \mathbb{F}^D : \forall c \in \mathcal{C}, \sum_{x \in D} z(x)c(x) = 0\}$*; observe that* $\mathcal{C}^\perp$ *is also a subspace of* $\mathbb{F}^D$.

We are now ready to define a linear code random oracle.

Definition 4 (Linear code random oracle). *Let* $\mathcal{C} \subseteq (D \to \mathbb{F})$ *be a linear code, and* $f \colon \{0,1\}^k \to D$*. We say that* $(\mathcal{C}, f)$ *is* **full-rank** *if* $\mathcal{C}|_{\mathrm{im}(f)} = \mathbb{F}^{\mathrm{im}(f)} \simeq \mathbb{F}^{2^k}$ *(in particular,* f *is an injection).*

Let $\mathscr{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *be a family of linear codes and* F *an efficient algorithm. We say that* $(\mathscr{C}, F)$ *is a* **linear code random oracle** *of arity* m *if for each* $\lambda \in \mathbb{N}$ *if* $F(1^\lambda) \colon \{0,1\}^{m(\lambda)} \to D_\lambda$ *is a circuit such that* $(\mathcal{C}_\lambda, F(1^\lambda))$ *is full-rank for all* $\lambda \in \mathbb{N}$.

For the remainder of this work, when it is unambiguous we will typically omit F and write $c(q)$ for $c(F(1^\lambda)(q))$ when $c \in \mathcal{C}_\lambda$ and $q \in \{0,1\}^{m(\lambda)}$. The standard random oracle is the linear code random oracle $\big(\{(\{0,1\}^{m(\lambda)} \to \mathbb{F}_\lambda)\}_\lambda, I\big)$ where $I(1^\lambda)(q) = q$ for $q \in \{0,1\}^{m(\lambda)}$. We will often refer to a function sampled uniformly from a linear code random oracle simply as a "linear code random oracle", and we use the symbol $\hat{\rho}$.

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. Any systematic oracle algorithm $\mathcal{B}$ can be interpreted as a $\mathcal{C}_\lambda$-oracle algorithm via the efficient injection $F(1^\lambda, \cdot)$; we write $\mathcal{B}^{\hat{\rho}}$, omitting F. The following statement follows immediately from the full-rank condition.

Claim. Let $(\mathscr{C} = \{\mathcal{C}_\lambda \subseteq (D_\lambda \to \mathbb{F}_\lambda)\}_\lambda, F)$ be a linear code random oracle of arity m. For any algorithm $\mathcal{B}$ with access to an oracle $\{0,1\}^{m(\lambda)} \to \mathbb{F}_\lambda$,

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

4.1 Query transcripts and partial oracles

We define some 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. A partial oracle extends a query transcript to include evaluations that are fixed by the structure of the code.

Definition 5. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. A $\mathcal{C}$-**query transcript** is a list $\mathsf{tr} = [(\mathsf{q}_i, \alpha_i)]_{i=1}^t \in (D \times \mathbb{F})^t$ for any $t \in \mathbb{N}$ such that there exists $c \in \mathcal{C}$ with $c(\mathsf{q}_i) = \alpha_i$ for all $i \in [t]$. A query transcript tr induces a partial function $D \rightharpoonup \mathbb{F}$ in the natural way, which we also denote by tr:*

$$\mathsf{tr}(\mathsf{q}) = \begin{cases} \alpha_i & \textit{if } \mathsf{q} = \mathsf{q}_i \\ \bot & \textit{if } \mathsf{q} \notin \{\mathsf{q}_1, \ldots, \mathsf{q}_t\}. \end{cases}$$

*We then define the **partial oracle** $\tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}} \colon D \rightharpoonup \mathbb{F}$ as follows:*

$$\tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(\mathsf{q}) = \begin{cases} \beta & \textit{if } \forall c \in \mathcal{C}, ((\forall i \in [t],\ c(\mathsf{q}_i) = \alpha_i) \Rightarrow c(\mathsf{q}) = \beta) \\ \bot & \textit{o.w.} \end{cases} .$$

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

4.2 Constraints

In this section we examine properties of the partial oracle arising from the linear structure of $\mathcal{C}$. We first introduce the notion of a *constraint* for elements in the domain of a linear code. The results in this section hold for all linear codes, even those that are not full-rank.

Definition 6. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. We say that a subset of the domain $Q \subseteq D$ is **constrained** if there exists a nonzero mapping $z \colon Q \to \mathbb{F}$ such that for all $c \in \mathcal{C}$, $\sum_{x \in Q} z(x)c(x) = 0$. Equivalently, Q is constrained if there exists $z \neq 0 \in \mathcal{C}^\perp$ with $\operatorname{supp}(z) \subseteq Q$. 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 either $x \in Q$ or there exists a constraint z on $Q \cup \{x\}$ such that $z(x) \neq 0$.*

The following claim connects constraints and partial oracles.

Claim. Let $\mathcal{C}$ be a linear code, and tr be a $\mathcal{C}$-query transcript. Then for all $x \in D$, $\tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(x) \neq \bot$ if and only if $\operatorname{supp}(\mathsf{tr})$ determines x.

In particular, $\tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(x) = \beta \in \mathbb{F}$ if and only if $\mathsf{tr}(x) = \beta$, or $\mathsf{tr}(x) = \bot$ and there exists $z \in \mathcal{C}^\perp$ such that $\operatorname{supp}(z) \subseteq \operatorname{supp}(\mathsf{tr}) \cup \{x\}$ and $z(x) \neq 0$, and

$$\beta = -z(x)^{-1}\Big(\sum_{y \in \operatorname{supp}(\mathsf{tr})} z(y)\mathsf{tr}(y)\Big) .$$

Next we characterize the distribution of $c(\mathsf{q})$ conditioned on a prior query transcript tr in terms of the value of $\tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(\mathsf{q})$.

Claim. Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a linear code. For all $\mathsf{q} \in D$, $Q \subseteq D$, $c' \in \mathcal{C}$ and $\beta \in \mathbb{F}$,

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

Equivalently, for all $\mathsf{q} \in D$ and all $\mathcal{C}$-query transcripts $\mathsf{tr} = \{(\mathsf{q}_i, \alpha_i)\}_{i=1}^{t}$ and $\beta \in \mathbb{F}$,

$$\Pr_{c \leftarrow \mathcal{C}}[c(x) = \beta \mid \forall i \in [t],\, c(\mathsf{q}_i) = \alpha_i] = \begin{cases} \frac{1}{|\mathbb{F}|} & \text{if } \tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(\mathsf{q}) = \bot, \\ 1 & \text{if } \tilde{\rho}^{\mathcal{C}}_{\mathsf{tr}}(\mathsf{q}) = \beta, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. The equivalence follows from Section 4.2 and the definition of a $\mathcal{C}$-query transcript, so it suffices to prove the second statement. The cases when $\tilde{\rho}_{\mathsf{tr}}(\mathsf{q}) \neq \bot$ are clear, and so we consider the case when $\tilde{\rho}_{\mathsf{tr}}(\mathsf{q}) = \bot$. In this case there exist codewords $c, c' \in \mathcal{C}$ such that $c(\mathsf{q}_i) = c'(\mathsf{q}_i)$ for all i but $c(\mathsf{q}) \neq c'(\mathsf{q})$. It follows by linearity of $\mathcal{C}$ that there exists $c^* = c - c' \in \mathcal{C}$ such that $c^*(\mathsf{q}_i) = 0$ for all i and $c^*(\mathsf{q}) \neq 0$. We can sample from the conditional distribution in the claim by choosing a random $c'' \in \mathcal{C}$ such that $c(\mathsf{q}_i) = \alpha_i$ for all i and a random $\alpha \in \mathbb{F}$ and returning $c'' + \alpha c^*$. The claim follows since $c''(\mathsf{q}) + \alpha c^*(\mathsf{q})$ is uniformly random in $\mathbb{F}$. □

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

Definition 7 (Constraint detector). *Let $\mathcal{C}$ be a linear code. An algorithm* CD *is a constraint detector for $\mathcal{C}$ if, given as input a set $Q \subseteq D$,*
– *if Q is constrained,* CD *outputs a constraint z;*
– *otherwise,* CD *outputs $\bot$.*
A code family $\{\mathcal{C}_\lambda\}_\lambda$ has efficient constraint detection if there is a polynomial-time algorithm CD *such that* $\mathsf{CD}(1^\lambda, \cdot)$ *is a constraint detector for $\mathcal{C}_\lambda$.*

A constraint detector directly yields an implementation of the partial oracle. It also allows us to remove "redundant" queries from any $\mathcal{C}$-oracle algorithm.

Definition 8. *We say that a $\mathcal{C}$-oracle algorithm $\mathcal{A}$ is* **non-redundant** *if it never makes any query that is determined by its previous queries.*

The following claim, which is straightforward to prove, shows that in many settings we may restrict our attention to non-redundant algorithms without loss of generality.

Claim. Let $\mathcal{A}$ be a t-query $\mathcal{C}$-oracle algorithm. Then there is a non-redundant t-query $\mathcal{C}$-oracle algorithm $\mathcal{A}'$ whose input-output behaviour is identical to $\mathcal{A}$. Moreover, if $\mathcal{C}$ has efficient constraint detection, then if $\mathcal{A}$ is efficient, $\mathcal{A}'$ is also.

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. For every algorithm $\mathcal{A}$ making fewer than 2^m queries,

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

On the other hand, there exists a linear code random oracle $(\mathscr{C}, F)$ and a 1-query algorithm $\mathcal{B}$ such that for all $\lambda \in \mathbb{N}$,

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

We now show that linear code random oracles are collision-resistant. To prove this, we will make use of 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. Suppose that $(\mathcal{C} \subseteq (D \to \mathbb{F}), f \colon \{0,1\}^m \to D)$ is full-rank. Then for all $q_1, \dots, q_t \in D$ there exists $c \in \mathcal{C}$ such that
- $c(q_1) = \cdots = c(q_t) = 0$, and
- there exists a set $S \subseteq \{0,1\}^m$, $|S| \geq 2^m - t$ such that for all $x \in S$, $c(f(x)) = 1$.

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

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 2. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a full-rank linear code of arity m, $Q \subseteq D$. Define $T := \{x \in \{0,1\}^m : Q \text{ determines } x\}$. Then $|T| \leq |Q|$.*

Proof. By Section 4.3, there exists $c \in \mathcal{C}$ such that for all $x \in Q$, $c(x) = 0$, and a set $S \subseteq \{0,1\}^m$ of size $2^m - |Q|$ such that for all $x \in S$, $c(x) = 1$. 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 3. *Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a full-rank linear code of arity m. For all t-query $\mathcal{C}$-oracle adversaries $\mathcal{A}$,*

$$\Pr_{\hat{\rho} \leftarrow \mathcal{C}}[x, y \in \{0,1\}^m \wedge \hat{\rho}(x) = \hat{\rho}(y) \mid (x,y) \leftarrow \mathcal{A}^{\hat{\rho}}] \leq t^2/|\mathbb{F}| \,.$$

Proof. Consider running $\mathcal{A}$ and recording its oracle queries in tr. By Lemma 2, the set of points $T = \{x \in \{0,1\}^m : \tilde{\rho}_{\mathsf{tr}}(x) \neq \bot\}$ 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 $x, y \in T$ such that $\hat{\rho}(x) = \hat{\rho}(y)$ is less than $t^2/|\mathbb{F}|$. By Section 4.2, if x or y are not in T, then $\Pr[\hat{\rho}(x) = \hat{\rho}(y) \mid \mathsf{tr}] = 1/|\mathbb{F}|$. The lemma follows by a union bound. □

4.4 Low-degree random oracles

We denote by $\mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ the vector space of m-variate polynomials over $\mathbb{F}$ of individual degree at most d.

Definition 9. *Let $\mathscr{F} = \{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of fields, $m, d \colon \mathbb{N} \to \mathbb{N}$. The $(\mathscr{F}, m, d)$-low-degree random oracle is the linear code random oracle $(\{\mathbb{F}_\lambda^{\leq d(\lambda)}[X_1, \ldots, X_{m(\lambda)}]\}_\lambda, F)$, where*

$$F(1^\lambda)(b_1, \ldots, b_{m(\lambda)}) := (i_\lambda(b_1), \ldots, i_\lambda(b_{m(\lambda)}))$$

for the natural injection $i_\lambda \colon \{0,1\} \to \mathbb{F}_\lambda$ mapping 0 to $0_{\mathbb{F}_\lambda}$ and 1 to $1_{\mathbb{F}_\lambda}$.

5 A forking lemma for linear code random oracles

Let $\mathcal{C} \subseteq (D \to \mathbb{F})$ be a full-rank linear code, and let A be a t-query $\mathcal{C}$-oracle algorithm.

For $x \in \{0,1\}^n$, $\vec{\alpha} \in \mathbb{F}^t$, $\sigma \in \{0,1\}^*$, denote by $(q, o; \mathsf{tr}) \leftarrow A^{\vec{\alpha}}(x; \sigma)$ the following procedure: Run A on input x and random tape σ. For every $i \in [t]$, answer A's i-th query $q_i \in \{0,1\}^n$ to the oracle with $\alpha_i \in \mathbb{F}$. Parse A's output as (q, o) for $q \in D$. Let $\mathsf{tr} = ((q_1, \alpha_1), \ldots, (q_t, \alpha_t))$ be the transcript of A's queries to the oracle. Denote by $(q, o; \mathsf{tr}, \sigma) \leftarrow A^{\hat{\rho}}(x)$ the same procedure, but where each α_i is adaptively set to $\hat{\rho}(q_i)$, and where σ is the random tape used by A (sampled uniformly).

For a query transcript tr, we define $\mathsf{FP}(\mathsf{tr}, q)$ to be the smallest $i \in [t]$ such that $\{q_1, \ldots, q_i, q\}$ is constrained, or $\perp$ if there is no such i. Note that given an efficient constraint detection algorithm CD, $\mathsf{FP}(\mathsf{tr}, q)$ can be computed in polynomial time. We now describe a general forking algorithm Fork.

```
$\mathsf{Fork}^A(q, o, \mathsf{tr}, \sigma)$:
1. Let $((q_1, \alpha_1), \ldots, (q_t, \alpha_t)) := \mathsf{tr}$.
2. Set $i := \mathsf{FP}(\mathsf{tr}, q)$. If $i = \perp$, abort and output $\perp$.
3. Otherwise, sample $\alpha'_i, \ldots, \alpha'_t \leftarrow \mathbb{F}$, and run $(q', o'; \mathsf{tr}') \leftarrow A^{\mathsf{tr}_{i-1}; \alpha'_i, \ldots, \alpha'_t}(\sigma)$.
4. Output $(q', o', \mathsf{tr}', i)$.
```

Lemma 4 (Forking Lemma). *For every predicate p and t-query non-redundant $\mathcal{C}$-oracle algorithm A, setting*

$$\delta := \Pr\left[\begin{array}{c} \mathsf{FP}(\mathsf{tr}, q) \neq \perp \wedge q \in \{0,1\}^m \\ \wedge\ \mathsf{p}(q, o, \mathsf{tr}) = 1 \end{array} \middle| \begin{array}{r} \hat{\rho} \leftarrow \mathcal{C}_\lambda \\ (q, o; \mathsf{tr}, \sigma) \leftarrow A^{\hat{\rho}} \end{array}\right]$$

we have that

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

Proof. For all $i \in [t], q \in \{0,1\}^m$, we define the set

$$S_{i,q} := \{(\vec{\alpha}, \sigma) : (q, o; \mathsf{tr}) \leftarrow A^{\vec{\alpha}}(\mathsf{pp}; \sigma) \ \wedge\ \mathsf{FP}(\mathsf{tr}, q) = i \ \wedge\ \mathsf{p}(\mathsf{pp}, o, \mathsf{tr}) = 1\} \ .$$

Next, define

$$\delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma) := \Pr_{\alpha'_i, \ldots, \alpha'_t \in \mathbb{F}}[((\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i, \ldots, \alpha'_t), \sigma) \in S_{i,q}].$$

Denote by E the event in Eq. (1). Observe that because A is non-redundant, $\Pr[\mathsf{Fork} \mid (\vec{\alpha}, \sigma) \in S_{i,q}]$ is exactly $\delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma)$.

We now analyze $\Pr[E]$.

$$\begin{aligned}
\Pr[E] &= \sum_{i \in [t], q \in \{0,1\}^m} \Pr[E \mid (\vec{\alpha}, \sigma) \in S_{i,q}] \cdot \Pr[(\vec{\alpha}, \sigma) \in S_{i,q}] \\
&= \sum_{i,q} \mathbb{E}_{\vec{\alpha}, \sigma}[\mathbb{1}_{S_{i,q}}(\vec{\alpha}, \sigma) \cdot \delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma)] \\
&= \sum_{i,q} \mathbb{E}_{\alpha_1, \ldots, \alpha_{i-1}, \sigma}\left[\delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma) \mathbb{E}_{\alpha_i, \ldots, \alpha_t}[\mathbb{1}_{S_{i,q}}(\vec{\alpha}, \sigma)]\right] \\
&= \sum_{i,q} \mathbb{E}_{\alpha_1, \ldots, \alpha_{i-1}, \sigma}\left[\delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma)^2\right]
\end{aligned}$$

where the last equality holds by definition of $\delta_{i,q}$. Let $Q_{i,\vec{\alpha},\sigma} := \{q : \delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma) \neq 0\}$. We prove the following claim.

Claim. For all $\vec{\alpha}, \sigma$, $\sum_i |Q_{i,\vec{\alpha},\sigma}| \leq t$.

Proof. For any t-query transcript $\mathsf{tr} = ((q_1, \alpha_1), \ldots, (q_t, \alpha_t))$, let $Q_{i,\mathsf{tr}} := \{q : \mathsf{FP}(\mathsf{tr}, q) = i\}$. By Lemma 2, for all tr it holds that $\sum_i |Q_{i,\mathsf{tr}}| \leq t$. Note that $Q_{i,\mathsf{tr}}$ is a function of $(q_1, \ldots, q_i)$ only.

Now let $(q, o; \mathsf{tr}) := A^{\vec{\alpha}}(\mathsf{pp}; \sigma)$. For each i, $(q_1, \ldots, q_i)$ is a function of $\alpha_1, \ldots, \alpha_{i-1}, \sigma$ only; hence so is $Q_{i,\mathsf{tr}}$. It follows that $Q_{i,\vec{\alpha},\sigma} \subseteq Q_{i,\mathsf{tr}}$ for all i, which proves the claim. □

It follows that

$$\begin{aligned}
\Pr[E] &= \mathbb{E}_{\vec{\alpha}, \sigma}\left[\sum_i \sum_{q \in Q_{i,\vec{\alpha},\sigma}} \delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma)^2\right] \\
&\geq \frac{1}{t} \cdot \mathbb{E}_{\vec{\alpha}, \sigma}\left[\Big(\sum_i \sum_q \delta_{i,q}(\alpha_1, \ldots, \alpha_{i-1}; \sigma)\Big)^2\right] \geq \frac{\delta^2}{t} \ ,
\end{aligned}$$

where the first inequality holds because the number of terms in the sum is at most t, and the second holds by the inequality $\mathbb{E}[X^2] \geq \mathbb{E}[X]^2$. (The outer expectation is taken over $\vec{\alpha} \in \mathbb{F}^t$ so that all events are over the same probability space.) □

6 Oracle zero-finding games

Lemma 5 (Oracle zero-finding game). *Let $\{\mathbb{F}_\lambda\}_\lambda$ be a family of fields. Fix a number of variables $m \in \mathbb{N}$ and a maximum individual degree $d \in \mathbb{N}$. Further, let CM be a binding commitment scheme with message format L that is two polynomials $(f\colon \mathbb{F}_\lambda \to \mathbb{F}_\lambda,\ g\colon \mathbb{F}_\lambda \to (\mathbb{F}_\lambda)^m)$. Then for every efficient t-query oracle algorithm $\mathcal{A}$, the following holds.*

$$\Pr\left[\begin{array}{c|r} f(X) \not\equiv \hat{\rho}(g(X)) & \hat{\rho} \leftarrow \mathbb{F}_\lambda^{\leq d}[X_1,\ldots,X_m] \\ \wedge & \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda, L) \\ f(z) = \hat{\rho}(g(z)) & (f,g,\omega) \leftarrow \mathcal{A}^{\hat{\rho}}(\mathsf{ck}) \\ & C \leftarrow \mathsf{CM.Commit}(\mathsf{ck}, f, g, \omega) \\ & z \in \mathbb{F}_\lambda \leftarrow \hat{\rho}(C) \end{array}\right] \leq \sqrt{t \cdot \left[\frac{2md \cdot \deg(g) + 1}{|\mathbb{F}_\lambda|}\right]} + \mathrm{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that wins the above game with probability δ. By Section 4.2, we may assume without loss of generality that $\mathcal{A}$ is non-redundant.

We will apply Lemma 4. The forking predicate $\mathsf{p}(\mathsf{pp}, (\mathsf{q}, \alpha), o, \mathsf{tr})$ is the conjunction of the following conditions, where $(f, g) := o$ and $\tilde{\rho}_{\mathsf{tr}}$ is as defined in Section 4.1.
- $\mathsf{q} = \mathsf{CM.Commit}(f, g; \omega)$.
- Either $\tilde{\rho}_{\mathsf{tr}} \circ g\colon \mathbb{F} \to \mathbb{F}$ is not total, or $f(X) \not\equiv \tilde{\rho}_{\mathsf{tr}}(g(X))$.
- $f(\alpha) = \tilde{\rho}_{\mathsf{tr}}(g(\alpha))$, where $\alpha := \tilde{\rho}_{\mathsf{tr}}(\mathsf{q})$.

By assumption, from $\mathcal{A}$ we can obtain an adversary satisfying p with probability $> \delta$.

Lemma 4 guarantees that

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

To conclude the proof, we will bound p. Denote by E the event on the left of the above expression.

We first bound the probability that E occurs and $o \neq o'$. By definition of p, if $\mathcal{B}$ succeeds then $\mathsf{CM.Commit}(o) = \mathsf{q} = \mathsf{q}' = \mathsf{CM.Commit}(o')$. However, by the binding property of the commitment scheme CM, the probability $\mathcal{B}$ succeeds and $o \neq o'$ occurs with probability $\leq \mathrm{negl}(\lambda)$. Then the probability that E occurs and $o = o'$ is at least $p - \mathrm{negl}(\lambda)$; call this event E'.

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

Claim. The probability that E' occurs and $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is not total is at most $(md \cdot \deg(g) + 1)/|\mathbb{F}|$.

Proof. Since for all $c \in \mathcal{C}$, $\deg(c \circ g) \leq d \cdot \deg(g)$, if $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is not total then there are at most $d \cdot \deg(g)$ points $x \in \mathbb{F}$ such that $\tilde{\rho}_{\mathsf{tr}|_{i-1}}(g(x)) \neq \bot$. Then since α' is chosen independently of g and tr, $\Pr[\tilde{\rho}_{\mathsf{tr}|_{i-1}}(g(\alpha')) \neq \bot] \leq md \cdot \deg(g)/|\mathbb{F}|$. Finally, if $\tilde{\rho}_{\mathsf{tr}|_{i-1}}(g(\alpha')) = \bot$, $\Pr[\tilde{\rho}_{\mathsf{tr}'}(g(\alpha')) = f(\alpha')] \leq \frac{1}{|\mathbb{F}|}$, since f and tr' are independent conditioned on $\mathsf{tr}|_{i-1}$. □

It follows that the probability that E' occurs and $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g$ is total is at least $p - (md \cdot \deg(g) + 1)/|\mathbb{F}| - \operatorname{negl}(\lambda)$. In this case it holds that $\tilde{\rho}_{\mathsf{tr}|_{i-1}} \circ g \not\equiv f$, but $\tilde{\rho}_{\mathsf{tr}|_{i-1}}(g(\alpha')) = f(\alpha')$. Since α' is drawn independently of $\mathsf{tr}|_{i-1}$, g and f, this holds with probability at most $md \cdot \deg(g)/|\mathbb{F}|$.

Rearranging, it follows that

$$p \leq \frac{2md \cdot \deg(g) + 1}{|\mathbb{F}|} + \operatorname{negl}(\lambda) \ ;$$

the statement follows since $p \geq \delta^2/t$. □

7 Accumulation scheme for low-degree random oracles

We construct an accumulation scheme $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ for any low-degree random oracle over a sufficiently large field.

Note that AS is typically defined with respect to a predicate Φ and a randomized algorithm $\mathcal{H}$, which accesses the oracle and outputs predicate parameters pp_{Φ}. For this construction, $\mathcal{H} = \perp$.

Theorem 1. *Let $\mathcal{C}$ be a $(\mathbb{F}_\lambda, m, d)$-low-degree random oracle.. Let $\mathsf{CM} = (\mathsf{CM.Setup}, \mathsf{CM.Commit})$ be a commitment scheme that is hiding, binding, and has compression to size m. Then, the scheme AS from Construction 1 is an accumulation scheme for $(\Phi, \perp)$, where $\Phi([\mathsf{q}_i]_{i=1}^n) = 1$ if for all $i \in [n]$, the query $\mathsf{q}_i = (x_i, \alpha_i)$ satisfies $\hat{\rho}(x_i) = \alpha_i$.*

We give the construction below. We assume a global ordering of the field $\mathbb{F}_\lambda$, so that $\mathbb{F}_\lambda = \{b_1, \ldots, b_{|\mathbb{F}_\lambda|}\}$.

Construction 1. $\mathsf{AS} = (\mathrm{G}, \mathrm{I}, \mathrm{P}, \mathrm{V}, \mathrm{D})$ is defined as follows:

- Accumulator: The scheme's accumulators are of the form $\mathsf{acc} \in ((\mathbb{F}_\lambda)^n, \mathbb{F}_\lambda)^2$.
- $\mathrm{G}(1^\lambda)$: Output a commitment scheme's commitment key and public parameters $(\mathsf{ck}, \mathsf{pp}_{\mathsf{CM}}) \leftarrow \mathsf{CM.Setup}(L)$, such that the message format L is two polynomials $(f \colon \mathbb{F}_\lambda \to \mathbb{F}_\lambda,\ g \colon \mathbb{F}_\lambda \to (\mathbb{F}_\lambda)^m)$.
- $\mathrm{I}^{\hat{\rho}}(\mathsf{pp} = \mathsf{ck})$: Output $(\mathsf{apk} = \mathsf{ck}, \mathsf{avk} = \mathsf{ck}, \mathsf{dk} = 1^\lambda)$.
- $\mathrm{P}^{\hat{\rho}}(\mathsf{apk} = \mathsf{ck}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^{\ell})$:
  1. Let $Q = [(x_k, \alpha_k)]_{k=1}^{n+2\ell}$ be the concatentation of $[\mathsf{q}_i]_{i=1}^n$ and $[\mathsf{acc}_j]_{j=1}^{\ell}$.
  2. Sample a random point $x_{n+2\ell+1} \in (\mathbb{F}_\lambda)^\ell$, and set $\alpha_{n+2\ell+1} := \hat{\rho}(x_{n+2\ell+1})$.
  3. Compute the polynomial $g \colon \mathbb{F}_\lambda \to (\mathbb{F}_\lambda)^\ell$ of degree at most $n + 2\ell$ such that for each $k \in [n + 2\ell + 1]$, $g(b_k) = x_k$.
  4. Compute the polynomial $f \colon \mathbb{F}_\lambda \to \mathbb{F}_\lambda$ as $f(X) \equiv \hat{\rho}(g(X))$. Note that the degree of f is at most $m \cdot d \cdot (n + 2\ell + 1)$.
  5. Sample randomness ω for the commitment scheme, then compute $C := \mathsf{CM.Commit}(\mathsf{ck}, (f, g); \omega) \in \{0,1\}^m$ and $\beta := \hat{\rho}(C)$.
  6. Output the new accumulator $\mathsf{acc} = \{(g(\beta), f(\beta)), (C, \beta)\}$ and proof $\pi_{\mathrm{V}} = (f, \omega, (x_{n+2\ell+1}, \alpha_{n+2\ell+1}))$.
- $\mathrm{V}(\mathsf{avk} = \mathsf{ck}, [\mathsf{q}_i]_{i=1}^n, [\mathsf{acc}_j]_{j=1}^{\ell}, \mathsf{acc} = \{(x, \alpha), (C, \beta)\}, \pi_{\mathrm{V}} = (f, \omega, (x_{n+2\ell+1}, \alpha_{n+2\ell+1})))$:

1. Compute the list $Q = [(x_k, \alpha_k)]_{k=1}^{n+2\ell}$ and the polynomial g from $[\mathsf{q}_i]_{i=1}^{n}$ and $[\mathsf{acc}_j]_{j=1}^{\ell}$ as P does. However, rather than sampling $(x_{n+2\ell+1}, \alpha_{n+2\ell+1})$, use the value received in π_{V}.
2. Check that $x = g(\beta)$, $\alpha = f(\beta)$ and $C = \mathsf{CM}.\mathsf{Commit}(\mathsf{ck}, (f, g); \omega)$.
3. For each $k \in [n + 2\ell + 1]$, check that $f(b_k) = \alpha_k$.
4. Accept if and only if both checks pass.

– $\mathrm{D}^{\hat{\rho}}(\mathsf{dk} = 1^{\lambda}, \mathsf{acc} = \{(x_1, \alpha_1), (x_2, \alpha_2)\})$: Accept if and only if $\hat{\rho}(x_1) = \alpha_1$ and $\hat{\rho}(x_2) = \alpha_2$.

Remark 1. CM can be replaced by a hash function (sampled from an appropriate hash family) if AS isn't required to be zero knowledge.

Acknowledgments

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

References

[AABS+19] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". IACR Cryptology ePrint Archive, Report 2019/426.

[ABL+19] B. Abdolmaleki, K. Baghery, H. Lipmaa, J. Siim, and M. Zajac. "UC-Secure CRS Generation for SNARKs". In: AFRICACRYPT '19.

[ACG+19] M. R. Albrecht, C. Cid, L. Grassi, D. Khovratovich, R. Lüftenegger, C. Rechberger, and M. Schofnegger. "Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC". IACR Cryptology ePrint Archive, Report 2019/419.

[AD18] T. Ashur and S. Dhooghe. "MARVELlous: a STARK-Friendly Family of Cryptographic Primitives". IACR Cryptology ePrint Archive, Report 2018/1098.

[AGP+19] M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger. "Feistel Structures for MPC, and More". IACR Cryptology ePrint Archive, Report 2019/397.

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

[BB04] D. Boneh and X. Boyen. "Short Signatures Without Random Oracles". In: EUROCRYPT '04.

[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.

[BCG+15] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs". In: S&P '15.

[BCI+13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. "Succinct Non-Interactive Arguments via Linear Interactive Proofs". In: TCC '13.

[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 (2020).

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

[BDFG20] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. “Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme”. ePrint Report 2020/1536.

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

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

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

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

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

[CFGS18] A. Chiesa, M. A. Forbes, T. Gur, and N. Spooner. “Spatial Isolation Implies Zero Knowledge Even in a Quantum World”. In: FOCS ’18.

[CHM+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. “Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS”. In: EUROCRYPT ’20.

[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.

[FS86] A. Fiat and A. Shamir. “How to prove yourself: practical solutions to identification and signature problems”. In: CRYPTO ’86.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs and Succinct NIZKs without PCPs”. In: EUROCRYPT ’13.

[GKM+18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. “Updatable and Universal Common Reference Strings with Applications to zk-SNARKs”. In: CRYPTO ’18.

[GKR+19] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems”. IACR Cryptology ePrint Archive, Report 2019/458.

[Gro10] J. Groth. “Short Pairing-Based Non-interactive Zero-Knowledge Arguments”. In: ASIACRYPT ’10.

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

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

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

[Mau05] U. M. Maurer. “Abstract Models of Computation in Cryptography”. In: IMA ’05.

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

[Sho97] V. Shoup. “Lower bounds for discrete logarithms and related problems”. In: EUROCRYPT ’97.

[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: TCC '08.

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