Subquadratic SNARGs in the Random Oracle Model

Alessandro Chiesa[1] and Eylon Yogev[2]

[1] UC Berkeley
[2] BU and TAU

Abstract. In a seminal work, Micali (FOCS 1994) gave the first succinct non-interactive argument (SNARG) in the random oracle model (ROM). The construction combines a PCP and a cryptographic commitment, and has several attractive features: it is plausibly post-quantum; it can be heuristically instantiated via lightweight cryptography; and it has a transparent (public-coin) parameter setup. However, it also has a significant drawback: a large argument size.
In this work, we provide a new construction that achieves a smaller argument size. This is the first progress on the Micali construction since it was introduced over 25 years ago.
A SNARG in the ROM is (t, ϵ)-*secure* if every t-query malicious prover can convince the verifier of a false statement with probability at most ϵ. For (t, ϵ)-security, the argument size of all known SNARGs in the ROM (including Micali's) is $\tilde{O}((\log(t/\epsilon))^2)$ bits, *even* if one were to rely on conjectured probabilistic proofs well beyond current techniques. In practice, these costs lead to SNARGs that are much larger than constructions based on other (pre-quantum and costly) tools. This has led many to believe that SNARGs in the ROM are inherently quadratic.
We show that this is not the case. We present a SNARG in the ROM with a sub-quadratic argument size: $\tilde{O}(\log(t/\epsilon) \cdot \log t)$. Our construction relies on a strong soundness notion for PCPs and a weak binding notion for commitments. We hope that our work paves the way for understanding if a linear argument size, that is $O(\log(t/\epsilon))$, is achievable in the ROM.

Keywords: succinct arguments; random oracle; probabilistically checkable proofs

1 Introduction

A succinct non-interactive argument (SNARG) is a cryptographic proof system for non-deterministic languages whose communication complexity is "succinct" in the sense that it is sublinear in the witness size (or even in the size of the computation that checks the witness). In the last decade, SNARGs have drawn the attention of researchers from multiple communities, being a fundamental cryptographic primitive that has found applications in the real world.

A central goal in the study of SNARGs is improving their efficiency, which may include improving prover time, argument size, or verifier time. For example, achieving small argument size is crucial in real-world applications where SNARGs are broadcast in a peer-to-peer network and redundantly stored at every network node (as in privacy-preserving digital currencies [BCG+14; Zc14]).

SNARGs in the ROM. The goal of this paper is to improve the argument size of SNARGs in the random oracle model (ROM). A SNARG in the ROM is (t, ϵ)-*secure* if every malicious prover that makes at most t queries to the random oracle can convince the verifier of a false statement with probability at most ϵ (over the choice of random oracle). There are two known approaches to construct SNARGs in the ROM: the Micali transformation [Mic00] (building on [Kil92; FS86]), which uses probabilistically checkable proofs (PCPs); and the BCS transformation [BCS16], which uses public-coin interactive oracle proofs (IOPs). Both approaches adopt the same paradigm:

$$\begin{bmatrix} \text{information-theoretic} \\ \text{proof} \end{bmatrix} + \begin{bmatrix} \text{cryptographic commitment} \\ \text{with local opening} \end{bmatrix} \Longrightarrow \text{SNARG} \ .$$

Informally, they compile an information-theoretic proof system (PCP or IOP) into a SNARG by relying on a cryptographic commitment scheme that supports local openings. The commitment scheme is a Merkle tree, and each local opening is a path from the desired leaf to the root.

Quadratic argument size. In both approaches, the argument size is *quadratic* in the desired security. If the random oracle has output length λ, then compiling a PCP/IOP with proof length l over alphabet Σ and query complexity q leads to an argument of size that is (up to constants):

$$\underbrace{\mathsf{q} \cdot \log |\Sigma|}_{\text{information-theoretic proof}} \quad + \quad \underbrace{\mathsf{q} \cdot \lambda \cdot \log \mathsf{l}}_{\text{cryptographic commitment}} \quad .$$

The term $\mathsf{q} \cdot \log |\Sigma|$ is the cost of the information-theoretic proof, and the term $\mathsf{q} \cdot \lambda \cdot \log \mathsf{l}$ is the cost of the cryptographic commitment (q authentication paths each consisting of $\log \mathsf{l}$ digests of size λ). To achieve (t, ϵ) security, the oracle output size is set to $\lambda = O(\log(t/\epsilon))$, and the soundness error of the PCP/IOP must be $O(\epsilon/t)$. For example, a PCP with this soundness error can be obtained by repeating $O(\log(t/\epsilon))$ times the verifier of any constant-query constant-soundness base PCP. This leads to a PCP with query complexity $\mathsf{q} = O(\log(t/\epsilon))$, and in turn to a quadratic argument size: $\tilde{O}((\log(t/\epsilon))^2)$. The quadratic complexity is due to the cost of the cryptographic commitment (while, for a small enough alphabet, the cost of the information-theoretic proof is linear).

One might hope to reduce the number of queries of the PCP/IOP to overcome this “quadratic barrier”, at the expense of a larger alphabet (even an alphabet of size $2^{O(\lambda)}$ would have a negligible effect on the argument size). However, using state-of-the-art PCPs (e.g., [DHK15]) or even conjectured PCPs (e.g., fulfilling the sliding scale conjecture [BGLR93]) would only shave off a $\log \mathsf{l}$ factor in the number of queries. Any PCP/IOP that has fewer queries would violate standard complexity-theoretic assumptions, e.g., the exponential-time hypothesis [GH98; CY20].

If one relies on cryptography with "more structure" then better argument sizes are possible. Known SNARGs based on bilinear groups (e.g., [Gro10; GGPR13; BCI+13]) have optimal size: $O(\log(t/\epsilon))$, which translates to a few hundred bytes in practice. Similarly, known SNARGs based on cyclic groups or unknown-order groups (e.g., [BCC+16; BBB+18; BFS20]) are almost as short: $O(\log(t/\epsilon) \cdot \log n)$ (here n is the size of the computation being proved), which translates to just a few kilobytes in practice. This has led to the belief that SNARGs that solely rely on a random function (the random oracle) are fundamentally long.

While the inferior asymptotics of argument sizes of SNARGs in the ROM have not prevented useful applications,[3] they do lead to relatively large concrete sizes (tens to hundreds of kilobytes in practice), which makes them undesirable for many other applications.

This state of affairs is unfortunate because SNARGs in the ROM have several attractive features. First, SNARGs in the ROM are to date the most efficient approach for post-quantum security, and so achieving post-quantum SNARGs with (public verification and) optimal argument size remains an open problem.[4] Moreover, by heuristically instantiating the random oracle with a suitable cryptographic hash function, one obtains SNARGs that are lightweight (no public-key cryptography is used) and easy to deploy (users only need to agree on which hash function to use without having to rely on a trusted party to sample a structured reference string).

1.1 Breaking the quadratic barrier

Since it seems implausible to improve the query complexity of the PCP/IOP, how could we reduce the argument size? One way would be to reduce the overhead of the commitment scheme. This would be an amazing achievement on its own but currently seems out of reach (and, in fact, many believe that improving the commitment scheme is impossible). Another way would be to completely deviate from the paradigm of constructing SNARGs from PCPs/IOPs. However, [CY20] tells us that any SNARG in the ROM inherently contains an IOP with closely related parameters. In light of this, the motivating question of our work is: is the quadratic barrier of SNARGs in the ROM inherent or, instead, one can do better by achieving subquadratic, or even linear, argument size?

A new paradigm. In this work, we show how to go beyond the quadratic barrier by changing the interplay between the information-theoretic proof and the cryptographic compiler. Instead of asking for better soundness with fewer queries, we rely on a *stronger soundness notion* of the PCP with the same number of queries: we compile these strong PCPs into SNARGs *without using a commitment.* The commitment is relaxed with a weak notion of binding that is coupled with

[3] E.g., to increase throughput in peer-to-peer systems such as Ethereum via "roll-up" architectures [zkr].

[4] Known approaches based on lattices achieve privately-verifiable SNARGs with optimal size [BISW17; BISW18] or publicly-verifiable SNARGs with square-root communication complexity [BBC+18].

our strong soundness notion for the PCP. In particular, the SNARG prover might not be committed to *any* location of a proof string. Informally, our approach can be summarized as follows:

$$\begin{bmatrix} \text{strong information} \\ \text{theoretic proof} \end{bmatrix} + \begin{bmatrix} \text{weak cryptographic} \\ \text{commitment} \end{bmatrix} \Longrightarrow \begin{matrix} \text{subquadratic} \\ \text{SNARG} \end{matrix} .$$

Results. We use this paradigm to construct the first SNARGs in the ROM of *subquadratic* size.

Theorem 1. *There exists a SNARG for* NP *in the random oracle model that achieves argument size $\tilde{O}(\log(t/\epsilon) \cdot \log t)$ with soundness error ϵ against t-query adversaries.*

Our construction is the first progress on the Micali construction since it was introduced over 25 years ago. Our construction relies solely on a random oracle and so is plausibly post-quantum secure. Previous SNARGs in the ROM have been proven to be secure in the *quantum* random oracle model [CMS19], and we leave it for future work to adapt these technique to our construction.

The argument size that we achieve, while better than quadratic, is still far from the lower bound of $\Omega(\log(t/\epsilon))$. In particular, our work leaves open the intriguing question:

Are there SNARGs in the ROM that have argument size $O(\log(t/\epsilon))$?

We hope that our work will lead to a better understanding of this fundamental question, where a positive answer is likely to have significant practical benefits.

1.2 Concrete efficiency

Our new construction achieves argument sizes that are not only asymptotically smaller but also concretely smaller: we obtain up to $2\times$ improvement in argument size over Micali's construction for an illustrative instantiation across different values of t and ϵ. Moreover, the running times of the verifier and the prover of our construction are essentially the same as in Micali's construction.

In more detail, Micali's construction is typically instantiated with a PCP whose verifier is repeated many times to reduce soundness error to $O(\epsilon/t)$. Looking ahead, our construction requires PCPs that satisfy a stronger notion of soundness that (as we will prove) is satisfied by repeated PCPs. Thus, conveniently, we can instantiate both the Micali construction and our construction via the same class (repeated PCPs), and in particular we can directly compare their argument sizes.

For example, in Table 1 we demonstrate the arguments sizes for various values of t and ϵ. We instantiate both constructions with the same repetition of a "base" PCP with soundness error $1/2$, query complexity 3, and proof length 2^{30} over a binary alphabet. By repeating the PCP verifier $\log \frac{1}{2} \frac{t}{\epsilon}$ times, the amplified PCP has soundness error $\varepsilon_{\text{PCP}} \leq \epsilon/t$ and $3 \log \frac{1}{2} \frac{t}{\epsilon}$ queries.

$\log t$ \ $-\log \epsilon$	64	96	128	160
64	$\frac{180\,\text{KB}}{131\,\text{KB}} \approx 1.37\times$	$\frac{257\,\text{KB}}{164\,\text{KB}} \approx 1.57\times$	$\frac{346\,\text{KB}}{188\,\text{KB}} \approx 1.84\times$	$\frac{448\,\text{KB}}{219\,\text{KB}} \approx 2.05\times$
96	$\frac{293\,\text{KB}}{237\,\text{KB}} \approx 1.24\times$	$\frac{389\,\text{KB}}{272\,\text{KB}} \approx 1.43\times$	$\frac{498\,\text{KB}}{317\,\text{KB}} \approx 1.57\times$	$\frac{618\,\text{KB}}{361\,\text{KB}} \approx 1.71\times$
128	$\frac{432\,\text{KB}}{357\,\text{KB}} \approx 1.21\times$	$\frac{547\,\text{KB}}{415\,\text{KB}} \approx 1.32\times$	$\frac{674\,\text{KB}}{473\,\text{KB}} \approx 1.42\times$	$\frac{814\,\text{KB}}{533\,\text{KB}} \approx 1.53\times$
160	$\frac{597\,\text{KB}}{513\,\text{KB}} \approx 1.16\times$	$\frac{730\,\text{KB}}{585\,\text{KB}} \approx 1.25\times$	$\frac{876\,\text{KB}}{659\,\text{KB}} \approx 1.33\times$	$\frac{1032\,\text{KB}}{730\,\text{KB}} \approx 1.41\times$

Table 1. Comparison of argument sizes between the Micali construction (in red) and our construction (in blue), for different settings of (t, ϵ). Both constructions are based on the same illustrative PCP.

2 Techniques

We summarize the main ideas behind our main result (Theorem 1).

2.1 The Micali construction is inherently quadratic

We review the SNARG construction of Micali [Mic00] and explain why its argument size is quadratic.

The Micali construction. Micali [Mic00] combined ideas from Fiat and Shamir [FS86] and Kilian [Kil92] in order to compile any probabilistically checkable proof (PCP) into a corresponding SNARG. Informally, the SNARG prover uses the random oracle to Merkle hash the PCP to a short root that acts as a short commitment to the PCP string; then, the SNARG prover uses the random oracle to derive randomness for the PCP verifier's queries; finally, the SNARG prover outputs an argument that includes the Merkle root, answers to the PCP verifier's queries, and authentication paths for each of those answers (which act as local openings to the commitment). The SNARG verifier re-derives the PCP verifier's queries from the Merkle root and then runs the PCP verifier with the provided answers, ensuring that those answers are indeed authenticated.

On the output length of the random oracle. As mentioned in the introduction, the argument size in the Micali construction is $\tilde{O}(\mathsf{q} \cdot \lambda)$, ignoring low-order terms; moreover, under standard complexity assumptions, the number of queries must be $\mathsf{q} = \Omega(\log(t/\epsilon))$ (up to low-order terms) even if using conjectured "best possible" PCPs. What about the oracle output size λ? If we were to set $\lambda = O(\log t)$ then we would obtain the argument size $\tilde{O}(\log(t/\epsilon) \cdot \log t)$ claimed in Theorem 1. However, the Micali construction is *not* secure in this regime, as we now explain.

Consider the following attack. A cheating prover selects an arbitrary Merkle root; uses the random oracle to derive PCP randomness from this Merkle root; finds a PCP string that satisfies the PCP verifier for this choice of PCP randomness; computes the Merkle tree on this PCP string; and hopes that the resulting Merkle root equals the Merkle root that was previously chosen. A success would constitute an "inversion". If this did not work, the cheating prover re-tries until he succeeds (or runs out of queries). If we want the construction to be (t, ϵ)-secure, then any t-query attack can succeed with probability at most ϵ. However, the described attack would succeed with probability roughly $t \cdot 2^{-\lambda}$ which gives us the lower bound $\lambda = \Omega(\log(t/\epsilon))$ (for the Micali scheme to be secure we actually need to set $\lambda = \Omega(\log(t^2/\epsilon))$).

Looking beyond the Micali construction. Our goal is to change the construction such that we can set the output length of the random oracle to $\lambda = \tilde{O}(\log t)$. This means that a cheating prover in this regime may find inversions or collisions in the random oracle. In particular, a Merkle tree with this choice of λ is *not* a commitment scheme (e.g., a collision will allow him to open in different ways). Therefore, we will need to find ways to handle this new class of attacks and, in particular, prevent the inversion attack described above.

2.2 Our construction

Our construction shares features with the Micali construction: the argument prover constructs a PCP string; commits to this PCP string using the random oracle; derives PCP randomness (and thus PCP queries) from the resulting commitment again using the random oracle; and outputs the commitment and certified answers to each PCP query.

At the same time, our construction differs from the Micali construction in several crucial ways. In the sequel, we describe the differences and provide intuition for why these differences are useful towards reducing argument size. In subsequent subsections, we will provide more information about how we establish the security of our construction.

(1) Chopped tree. The argument prover commits to the PCP string via a *chopped* Merkle tree: the Merkle tree is computed layer by layer from the leaves but stops at a specific stop layer i^*. In the Micali construction, the stop layer is $i^* = 0$ (a single vertex called the Merkle root); in our construction, the stop layer is (roughly) $i^* = \log \mathsf{q}$ (where q is the query complexity of the PCP), which consists of 2^{i^*} vertices that we collectively call a Merkle *cap*.[5] The argument prover then uses the Merkle cap similarly to a Merkle root in the Micali construction: it derives PCP randomness from the Merkle cap by using the random oracle (in a single query); and subsequently authenticates answers to PCP queries via paths that are truncated at layer i^*.

As the stop layer i^* increases, argument size increases as well. In the extreme, if $i^* = \log \mathsf{l}$ (the stop layer is the leaf layer), then the argument contains the entire

[5] Equivalently, the Merkle cap is an ordered list of Merkle roots for smaller sub-trees.

PCP string. In our construction, we set (roughly) $i^* = \log \mathsf{q}$, in which case the argument size is (almost) the same as when $i^* = 0$ (for the same output size of the random oracle). Intuitively, if the argument prover supplies q authentications paths then, with high probability, most of the vertices in layer $\log \mathsf{q}$ would have been already included, so that truncating the paths to layer $\log \mathsf{q}$ and including in the argument all the digests in layer $\log \mathsf{q}$ does not affect argument size by much. (This is not just asymptotically: our experiments show that this has a negligible effect on the argument size in practice as well.)

Our main observation is that as the stop layer increases, *security increases as well.* In the Micali construction, a single inversion of the Merkle root breaks the scheme: the attacker selects an arbitrary Merkle root, derives corresponding PCP randomness (and thus PCP queries), finds a PCP string that makes the PCP verifier accept with that PCP randomness, computes a (full) Merkle tree on this PCP string, and hopes that the resulting root equals the previously selected root. If the root has output size λ, this takes about 2^λ attempts (which is roughly 2^λ queries). In contrast, in our construction, an inversion of a single vertex in the Merkle cap affects only a $1/\mathsf{q}$ fraction of the PCP string, which (in general) is not a winning strategy for a cheating prover. To emulate the prior strategy, the attacker would need to invert all q vertices in the Merkle cap, which is much harder.

(2) Domain separation. The Micali construction involves two random oracles: an oracle for computing the Merkle tree, and another oracle for deriving PCP randomness. (See [Mic00].) In our construction, we use domain separation to "split up" the oracle for the Merkle tree into a separate oracle for each vertex in the (in our case, chopped) Merkle tree. To compute the digest located in position j of layer i in the tree, the argument prover uses the prefix (i, j) in the query to the tree oracle. This does not increase argument size (the indices i and j are known so are not included in the argument) and has essentially no effect on the prover time and verifier time.

This domain separation is crucial for security because, without it, a cheating prover could recycle a single inversion or collision many times. For example, the cheating prover could find a collision in a leaf vertex between the values 0 and 1 and then re-use the same collision for *all* leaves to compute a Merkle tree for which any location can be opened to 0 or 1. This is insecure, e.g., for any PCP over the binary alphabet.

(3) Permuting the proof. In our construction, the argument prover randomly permutes the PCP string before applying the (chopped) Merkle tree. This requires a random permutation $\mathtt{Perm}\colon [\mathsf{l}] \to [\mathsf{l}]$ that is also known to the argument verifier, and can be derived via the Luby–Rackoff construction from the random oracle (see Section 4.1). Thus, if the PCP verifier wishes to read the i-th symbol of the PCP string, the $\mathtt{Perm}(i)$-th leaf should be accessed. This modification also does not increase argument size and has a negligible effect on the time complexity of the argument prover and argument verifier (each permutation call translates to a few calls to the random oracle).

Permuting the PCP string creates the effect of a PCP with *uniform random queries.* This property ensures that there is no "weak" block of symbols in the PCP. Indeed, recall that we chopped the Merkle tree in order to have a Merkle cap instead of a Merkle root, so that if a cheating prover makes a single inversion then this will affect only a small block of the PCP string. However, if all the PCP queries were to this block, then the cheating prover could still win with this single inversion. In contrast, since queries are random (after applying the permutation), we are guaranteed that, with high probability, the queries are (roughly) spread evenly across different blocks.

(4) Robust PCPs. Our construction is designed to work with PCPs that satisfy a stronger soundness notion, which we call *permuted robust soundness.* This notion is similar to the standard property of (strong) robust soundness of PCPs, which captures the probability of being within a particular (block-wise) *distance* from a satisfying proof. To fit our proof, we augment the standard notion to additionally consider a permutation that randomizes the proof locations so that queries are spread across blocks.

While we rely on strong soundness notions of PCPs, we show that *repeated* PCPs satisfy this stronger notion. That is, one can take a base PCP, and repeat it to amplify the soundness. What we show is that not only the soundness is amplified, but the PCP also satisfies the stronger notion of permuted robust soundness (with corresponding parameters). Intuitively, this notion lets us argue the construction's security even when the prover finds a small number of collisions or inversions. The next subsections are dedicated to the precise notion of permuted robust soundness, how to achieve it in repeated PCPs, and how we use it in our proof of security.

2.3 Permuted robust soundness

We describe *permuted robust soundness*, the PCP soundness notion that we use for our construction.

Given a block size parameter $\mathsf{b} \in \mathbb{N}$, we view a PCP string $\Pi \in \Sigma^{\mathsf{l}}$ as divided into blocks of size b, that is, as $\Pi \in (\Sigma^{\mathsf{l}/\mathsf{b}})^{\mathsf{b}}$. We denote by $\Delta_{\mathsf{b}}(\Pi, \Pi')$ the block-wise distance between two PCP strings Π and Π' (i.e., the number of blocks of symbols on which they differ); more generally, given a permutation $\mathtt{Perm}\colon [\mathsf{l}] \to [\mathsf{l}]$, we denote by $\Delta_{\mathsf{b}}^{\mathtt{Perm}}(\Pi, \Pi')$ this block-wise distance when the two PCP strings are permuted according to $\mathtt{Perm}$ (and after are divided into blocks for measuring distance).

The soundness is defined by the following game.

Game 1. The game $\mathcal{G}_{\text{per}}$ receives as input a PCP verifier $\mathbf{V}$, an instance $\mathbb{x}$, a block size parameter $\mathsf{b} \in \mathbb{N}$, an allowed distance parameter $d \in \mathbb{N}$, and a cheating prover $\tilde{\mathbf{P}}$. The game $\mathcal{G}_{\text{per}}(\mathbf{V}, \mathbb{x}, \mathsf{b}, d, \tilde{\mathbf{P}})$ works as follows:

1. Sample a random permutation $\mathtt{Perm}\colon [\mathsf{l}] \to [\mathsf{l}]$, and give it to $\tilde{\mathbf{P}}$.
2. $\tilde{\mathbf{P}}$ outputs a PCP string $\Pi \in \Sigma^{\mathsf{l}}$.
3. Sample PCP randomness $\rho \in \{0,1\}^{\mathsf{r}}$, and give it to $\tilde{\mathbf{P}}$.
4. $\tilde{\mathbf{P}}$ outputs another PCP string $\Pi' \in \Sigma^{\mathsf{l}}$.

5. The game outputs 1 if and only if $\Delta_{\mathsf{b}}^{\mathtt{Perm}}(\Pi, \Pi') \leq d$ and $\mathbf{V}^{\Pi'}(\mathbb{x}; \rho) = 1$.

Definition 1. *A PCP* $(\mathbf{P}, \mathbf{V})$ *for a relation* R *has* **permuted robust soundness error** $\varepsilon_{\text{per}}(\mathbb{x}, \mathsf{b}, d)$ *if for every instance* $\mathbb{x} \notin L(R)$*, block size* $\mathsf{b} \in \mathbb{N}$*, distance bound* $d \in \mathbb{N}$*, and malicious prover* $\tilde{\mathbf{P}}$*,*

$$\Pr\left[\mathcal{G}_{\text{per}}(\mathbf{V}, \mathbb{x}, \mathsf{b}, d, \tilde{\mathbf{P}}) = 1\right] \leq \varepsilon_{\text{per}}(\mathbb{x}, \mathsf{b}, d) \ .$$

Why we need permuted robust soundness. Before we continue with the security analysis of our construction, we give intuition for how permuted robust soundness is helpful towards security.

Consider the following strategy for a cheating prover, which captures the main ideas in our proof. The prover: selects a PCP string; permutes it according to the random permutation; commits to it via a chopped Merkle tree; and then derives PCP randomness, and thus PCP queries, from the resulting Merkle cap. Each vertex in the Merkle cap is itself a Merkle root for a subtree whose leaves are a block of the PCP string. By inverting a root in the Merkle cap, the cheating prover has complete control on the corresponding block. In particular, the cheating prover can find the minimal set of blocks to modify so to make the PCP verifier accept, and inverts the roots for these blocks. The success probability is (roughly) his probability of successfully inverting all these roots. Thus, it is important that no block has many queries, which is why we use the permutation.

In other words, the cheating prover's success probability depends on the *distance* of the PCP string to an accepting PCP string, where the distance is defined by the block-wise Hamming distance (after the permutation). This is why we need the PCP to have a robust notion of soundness where the probability that a PCP string is close to being accepting is smaller as this distance is smaller. For any constant k, we will bound the probability that the cheating prover can invert k roots, and then compare this with the probability that a proof will be of distance k from an accepting proof.

This is a high-level approach of how to handle this specific attack. However, a cheating prover has a wide range of strategies: find collisions and inversions in arbitrary locations in the chopped Merkle tree; create multiple trees from which to choose from, and derive many different PCP query sets; and try to combine all of the above. The permuted robust soundness is the notion that our proof is built on, however, we will need to somehow address all possible prover strategies.

2.4 Repeated PCPs satisfy permuted robust soundness

Our main technical lemma regarding permuted robust soundness states that any repeated PCP satisfies the (strong notion of) permuted robust soundness. Note that for constructing SNARGs, the underlying PCP must have an exponentially small soundness error. We know how to build such PCPs only by repeating some base PCP multiple times. Here, we show that in addition to improving the standard soundness, repetition improves the permuted robust soundness of the PCP. Intuitively, for a repeated PCP the distance of a PCP string to an accepting

PCP string is proportional to the number of repetitions that reject it, and different repetitions are likely to query different blocks (due to the permutation). This is a simple generic way to construct PCPs suitable for our SNARG construction, which suffices for our asymptotic result and also is useful for concrete efficiency.

In more detail, the *κ-wise repetition* of a PCP system $(\mathbf{P}, \mathbf{V})$, denoted $(\mathbf{P}_\kappa, \mathbf{V}_\kappa)$, is the PCP system obtained by setting $\mathbf{P}_\kappa := \mathbf{P}$ (the PCP string does not change) and setting $\mathbf{V}_\kappa$ to run $\mathbf{V}$ on κ independent choices of randomness. We prove the following lemma.

Lemma 1. *Let $(\mathbf{P}, \mathbf{V})$ be a PCP with soundness error $\varepsilon_{\text{base}}$, proof length l (over any alphabet), and query complexity q; moreover, suppose that each location in the PCP string is queried with probability at most p. For every $\kappa \in \mathbb{N}$ such that $\mathsf{b} \geq \kappa \cdot \mathsf{q} \cdot \varepsilon_{\text{base}}^{-1}$ and $p \leq (8\mathsf{b} \cdot \kappa)^{-1}$, $(\mathbf{P}_\kappa, \mathbf{V}_\kappa)$ has strong permuted robust soundness error*

$$\varepsilon_{\text{per}}(\mathbb{x}, \mathsf{b}, d) \leq \frac{e^{1.2 \cdot d}}{d!} \cdot \mathsf{b}^d \cdot \varepsilon_{\text{base}}(\mathbb{x})^\kappa \quad .$$

The formal statement of the lemma and its proof are in Section 5. Below we provide an overview of (a simplified version of) this lemma.

Recall that if the base PCP has soundness error $\varepsilon_{\text{base}}$ then its κ-wise repetition has soundness error $\varepsilon_{\text{base}}^\kappa$. This soundness error is for a PCP string that is fixed before the κ samples of PCP randomness are drawn. Here we are instead interested in the probability that the PCP string is d blocks away from convincing the PCP verifier. That is, the cheating prover can arbitrarily change any d blocks after learning the PCP randomness (and, in particular, derive the queried locations).

How much power does this give to the cheating prover? To understand this, we first need to see how the queries are distributed among the blocks. Since we assume that no proof location is queried with too high probability, a typical query set will have all queries distinct (or at least have only a few colliding queries). Then the random permutation will randomize query locations, as if they had been uniform random queries. In sum, queries will be mostly spread out evenly across blocks.

For even further simplicity here, let us assume that no two queries land in the same block. (This is possible as the total number of queries $\kappa \cdot \mathsf{q}$ is less than b.) In this case, the prover can change at most d blocks after seeing the queries and can affect the output of at most d out of the κ repetitions. Without assuming any additional property about the underlying PCP, changing a single query within a repetition might suffice to convince the PCP verifier. Thus we cannot expect soundness better than $\varepsilon_{\text{base}}^{\kappa-d}$. Moreover, the cheating prover chooses which d blocks to change adaptively after seeing all query locations, which grants the cheating prover additional power.

To bound the soundness error we fix in advance a choice of d repetitions among the κ repetitions that the prover controls; the remaining $\kappa - d$ iterations each contribute a multiplicative factor of soundness error $\varepsilon_{\text{base}}$. By a union bound over all choices of the d iterations, we get the expression:

$$\binom{\kappa}{d} \cdot \varepsilon_{\text{base}}^{\kappa-d} \leq \frac{\kappa^d}{d!} \cdot \varepsilon_{\text{base}}^{\kappa-d} = \frac{1}{d!} \cdot \left(\frac{\kappa}{\varepsilon_{\text{base}}}\right)^d \cdot \varepsilon_{\text{base}}^\kappa \leq \frac{\mathsf{b}^d}{d!} \cdot \varepsilon_{\text{base}}^\kappa \quad .$$

This expression is better than what we set out to prove. The additional term $e^{1.2d}$ in Lemma 1 comes from removing the simplifying assumptions used above. Without those assumptions, the cheating prover may gain an advantage when a block contains queries from more than one repetition: (i) queries might collide before the permutation is applied, and these queries will be mapped to the same (random) location by the permutation; (ii) even distinct queries might be mapped to the same block after the permutation (and this is likely to happen). The full proof must take these into account, which complicates the expressions above and introduces the additional term $e^{1.2d}$.

Remark 1 (repetition of a robust PCP). Lemma 1 shows that the repetition of *any* PCP satisfies permuted robust soundness. This will let us instantiate our SNARG construction based on the repetition of any PCP, retaining maximal freedom in choosing parameters of the PCP, without worrying about additional properties of the PCP. Nevertheless, we could also consider the repetition of a PCP that is already somewhat robust (in the standard sense), which would improve the soundness expression in the lemma. For example, suppose that in every local view of the PCP string we need to change at least two answers to make the PCP verifier accept. For this case we expect the soundness of the repeated PCP to be close to $\varepsilon_{\text{PCP}}^{\kappa-d/2}$, instead of $\varepsilon_{\text{PCP}}^{\kappa-d}$. We leave it for future work to derive the analogue of Lemma 1 for robust PCPs.

2.5 The cap soundness game

In order to obtain a security analysis of our construction, we introduce an intermediate information-theoretic game, called *cap soundness game*, that enables us to model the effects of attacks against our construction. The intermediate game then leaves us with two tasks: reduce the security of our construction to winning the cap soundness game (see Section 2.7); and reduce winning the cap soundness game to breaking the permuted robust soundness of the PCP (see further below).

The game. The cap soundness game has several inputs: a PCP verifier $\mathbf{V}$; an instance $\mathbb{x}$ (which we will usually omit from the description); an integer λ (modeling the random oracle's output size); a stop layer i^*; a malicious prover $\tilde{\mathbf{P}}$ to play the game; a query budget $t \in \mathbb{N}$; a collision budget $t_{\text{col}} \in \mathbb{N}$; and a inversion budget $t_{\text{inv}} \in \mathbb{N}$. We denote this game by $\mathcal{G}_{\text{cap}}(\mathbf{V}, \mathbb{x}, \lambda, i^*, \tilde{\mathbf{P}}, t, t_{\text{col}}, t_{\text{inv}})$.

The graph G. The game is played on a graph $G = (V, E)$ that represents the chopped Merkle trees constructed by the adversary so far. Letting d be the height of a full Merkle tree, vertices in G are the union $V := V_{i^*} \cup V_1 \cup \cdots \cup V_d$ where V_i are the vertices of level i of the tree: for every $i \in \{i^*, \ldots, d-1\}$, $V_i := \{(i, j, h) : j \in [2^i], h \in \{0,1\}^\lambda\}$ is level i; and $V_d := \{(d, j, h) : j \in [2^d], h \in \Sigma\}$ is the leaf level. The indices i and j represent the location in the tree (vertex j in level i) and the string h represents either a symbol of the PCP (if in the leaf level) or an output of the random oracle (if in any other level). Edges in G are *hyper*edges that keep track of which inputs are "hashed" together to create a given output.

That is, elements in the edge set E of G are chosen from the collection $\mathcal{E}$ below, which represents an edge between two vertices in level $i+1$ and their common parent in level i:

$$\mathcal{E} = \left\{ (u, v_0, v_1) : \begin{array}{c} u = (i, j, h) \in V_i \\ v_0 = (i+1, 2j-1, h_0) \in V_{i+1} \\ v_1 = (i+1, 2j, h_1) \in V_{i+1} \end{array} \right\} .$$

The set of valid caps consists of all possible lists of vertices in V_{i^*} that consist a full layer of vertices:

$$C := \left\{ \left((i^*, 1, h_1), \ldots, (i^*, 2^{i^*}, h_{2^{i^*}}) \right) : h_1, \ldots, h_{2^{i^*}} \in \{0,1\}^\lambda \right\} .$$

Playing the game. The game starts with the graph G empty ($E = \emptyset$), and proceeds in rounds; moreover, the game samples a random permutation $\texttt{Perm}\colon [\mathsf{l}] \to [\mathsf{l}]$ and gives it to the adversary. After that, in each round, provided there is enough query budget t left, the adversary chooses between two actions: (i) add an edge to E from the set $\mathcal{E}$, provided the edge is allowed; (ii) obtain the PCP randomness for a given Merkle cap. We discuss each in more detail.

– *Adding edges.* When the prover adds to E an edge $(u, v_0, v_1) \in \mathcal{E}$ the query budget is reduced $t \leftarrow t-1$. Moreover, the collision and inversion budgets may also be reduced if the edge creates a collision or inversion, as described below.
  - **Collisions.** If the edge (u, v_0, v_1) collides with an edge (u, v_0', v_1') that is already in E, the game charges a unit of collision budget by setting $t_{\text{col}} \leftarrow t_{\text{col}} - 1$. Note that the game charges a single unit for each collision edge, and multi-collisions are allowed. Thus, a k-wise collision costs $k-1$ units of t_{col}. This makes the collision budget versatile in that, for example, a budget of 2 can be used to create two 2-wise collisions or one 3-wise collision.
  - **Inversions.** If the edge (u, v_0, v_1) is added when u is not *free* (defined next), the game charges a unit of inversion budget by setting $t_{\text{inv}} \leftarrow t_{\text{inv}} - 1$. The vertex u is *free* if it is not already connected to a vertex in a level closer to the cap, i.e., if $u \in V_i$ then for every $w \in V_{i-1}$ and $u' \in V_i$ it holds that $(w, u, u') \notin E$. (Note that the game would not charge an inversion if these edge where added in reverse order, though, as that would not have been an inversion.)

– *Deriving randomness.* The prover submits a cap $(v_1, \ldots, v_{2^{i^*}}) \in C$, and the game samples new PCP randomness ρ. The pair $\left((v_1, \ldots, v_{2^{i^*}}), \rho\right)$ is added to a mapping Rand. (The prover is not allowed to submit a cap that already appears in the mapping Rand.) This costs a unit of the query budget, so when this happens the query budget is reduced $t \leftarrow t-1$.

Winning the game. When it decides to stop playing, the prover outputs a cap $(v_1, \ldots, v_{2^{i^*}}) \in C$ and a PCP string $\Pi \in \Sigma^{\mathsf{l}}$. The prover wins the game if the following two conditions hold.

– The PCP verifier accepts the PCP string Π when using the randomness associated to $(v_1, \ldots, v_{2^{i^*}})$. That is, $\mathbf{V}^{\Pi}(\mathbb{x}; \rho) = 1$ for $\rho := \mathsf{Rand}[(v_1, \ldots, v_{2^{i^*}})]$. (If Rand has no randomness for this cap then the prover loses.)

– The PCP string Π is consistent with the cap $(v_1, \ldots, v_{2^{i^*}})$ in the graph G. That is, if the PCP verifier queries location j of Π, then the leaf $u = (d, j, \Pi[\texttt{Perm}(j)]) \in V_d$ is connected to a vertex in the cap $(v_1, \ldots, v_{2^{i^*}})$ in G. (The collection $\mathcal{E}$ of possible edges ensures that the j-th leaf can be connected to at most one vertex in a cap, the one corresponding to the first i^* bits of the index j.)

We denote by $\varepsilon_{\text{cap}}(i^*, t, t_{\text{col}}, t_{\text{inv}})$ the maximum winning probability in the cap soundness game, with stop layer i^*, by any adversary with query budget t, collision budget t_{col}, and inversion budget t_{inv}.

From permuted robust soundness to cap soundness. We reduce the soundness of a cheating prover in the cap soundness game to the soundness in the permuted robust soundness game.

Lemma 2. *Let* $(\mathbf{P}, \mathbf{V})$ *be a PCP for a relation* R *with permuted robust soundness error* $\varepsilon_{\text{per}}(\mathsf{b}, d)$. *Then,* $(\mathbf{P}, \mathbf{V})$ *has cap soundness error*

$$\varepsilon_{\text{cap}}(i^*, t, t_{\text{col}}, t_{\text{inv}}) \leq t \cdot 2^{t_{\text{col}}} \cdot \varepsilon_{\text{per}}(\mathsf{b} = 2^{i^*}, d = t_{\text{inv}}) \ .$$

The proof of the lemma is somewhat technical and is provided in Section 6. Here we provide some intuition on the above expression. The term $\varepsilon_{\text{per}}(\mathsf{b} = 2^{i^*}, d = t_{\text{inv}})$ comes from the fact that if the attacker can make t_{inv} inversions then it suffices for the attacker to commit to a PCP string that is within a block-size distance of t_{inv} from an accepting PCP string (and the blocks have size 2^{i^*} since that is the number of leaves under a vertex in the cap). The multiplicative factor $2^{t_{\text{col}}}$ comes from the fact that if the attacker can find t_{col} collisions then the attacker can open up to $2^{t_{\text{col}}}$ PCP strings for the same cap (as each collision doubles the number of PCP strings that could be consistent with the same cap). The further multiplicative factor t comes from the fact that the attacker can re-try its strategy roughly t times.

2.6 Scoring oracle queries

The cap soundness game lets us bound the success probability of an adversary given specific budgets. But what budgets should we use when analyzing a cheating argument prover? For this, we rely on an analysis tool introduced in **[anon citation]**: a *scoring function* for the query trace of an algorithm in the random oracle model. For convenience and completeness, we review this notion below.

Intuitively, the score of a query trace "counts" the number of collisions and inversions in a way that reflects the probability of that event occurring. The lower the probability, the higher the score. This enables us to translate our claims about cheating argument provers into claims about cheating cap soundness provers, where a high score is translated to a high budget. A strategy that uses a large budget has a higher chance of winning the cap soundness game, but the probability of achieving a corresponding high score is low, and our goal is to balance these two.

The scoring function is separately defined for collisions and for inversions, as motivated below.

– *Scoring collisions.* The score of a k-wise collision is set to be $k-1$ (assuming k is maximal within the query trace); in particular, a 2-wise collision gets a score of 1. Note that two pairwise collisions and one 3-wise collision both get the same score of 2, even though the latter is less likely to occur. This aligns with our proof since two pairwise collisions yield four possible proof strings, while a 3-wise collision yields only three possible proof strings.
– *Scoring inversions.* Scoring inversions is done by simply counting the number of inversions in the query trace. We now elaborate on what is considered an inversion in the query trace. Recall that queries to the random oracle designated for the (chopped) Merkle tree are compressing: a query is of the form $x = (x_1, x_2) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ and an answer is $y \in \{0,1\}^\lambda$. Instead, queries to the random oracle designated for deriving PCP randomness are of the form $x \in \{0,1\}^{2^{i^*} \cdot \lambda}$ and an answer is $\rho \in \{0,1\}^{\mathsf{r}}$. For inversions we only consider tree queries, and note that a given tree query may invert one of the two components in a previous tree query or may invert (the one component of) a previous randomness query. Hence, a tree query performed at time j with answer y is an inversion if there exist a previous tree query (at time $j' < j$) of the form $x = (x_1, x_2)$ with $x_1 = y$ or $x_2 = y$, or a previous randomness query x with $x = y$.

The precise definitions of scores and the proof of the following lemma are provided in the full version:

Lemma 3. *For any t-query algorithm that queries the random oracle and every $k \in \mathbb{N}$:*

1. $\Pr\left[\textit{collision score} > k\right] \le \left(\frac{t^2}{2 \cdot 2^\lambda}\right)^k$;
2. $\Pr\left[\textit{inversion score} > k\right] \le \frac{1}{k!} \cdot \left(\frac{2t}{2^\lambda}\right)^k$.

2.7 Concluding the proof of Theorem 1

We are left with putting pieces together to derive the argument size. To this end, we first establish the soundness error of our construction, and then the argument size will follow.

Soundness of our construction. We show that our construction is sound given any PCP with permuted robust soundness. Recall that permuted robustness soundness depends on the distance d (Section 2.3). In our construction the quantity that matters is an associated worst-case ratio: we say that the PCP has permuted robustness *ratio* $\beta(\mathsf{b})$ if

$$\max_{d \in \{0,1,\dots,\mathsf{b}\}} \frac{\varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d+1)}{\varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d)} \le \beta(\mathsf{b}) \ .$$

Then, we show the following lemma.

Lemma 4. *Suppose our construction is instantiated with a random oracle with output size λ, and a PCP with soundness error ε_{PCP} and permuted robustness ratio $\beta(\mathsf{b})$ with stop layer i^*. If $\lambda \geq 2\log t + \log\beta(\mathsf{b} = 2^{i^*}) + 3$ then our construction has soundness error $\epsilon(t) \leq t \cdot \varepsilon_{\text{PCP}}$ against t-query adversaries.*

In our soundness analysis we consider *every possible query trace score* and also the probability that the cheating argument prover achieves that score (see Section 2.6). For any integer $k \in \mathbb{N}$ we consider the event of the cheating argument prover produces a query trace that has either collision score or inversion score exactly k. We show that, conditioned on the cheating prover producing a query trace of score k, there is a related adversary that wins the cap soundness game with the same probability and budget k (the precise statement is given in Claim 6). Informally,

$$\Pr\left[\begin{matrix}\text{verifier}\\ \text{accepts}\end{matrix} \,\middle|\, \text{score } k\right] \leq \varepsilon_{\text{cap}}(i^*, t, k, k) \ .$$

We consider an infinite sum over k, and for each value of k we bound the probability of the adversary getting a score of k multiplied by the maximum winning probability in the cap soundness game given budgets $t_{\text{col}} = k$ and $t_{\text{inv}} = k$. This infinite sum converges to the soundness expression stated in Lemma 4, provided that $\lambda \geq 2\log t + \log\beta + 3$.

In more detail, this approach could be over-simplified via the following equations (for simplicity here we are not careful with constants). First, using Lemma 3 we obtain that the probability that the collision or inversion score equals k is bounded by the sum of the two probabilities:

$$\Pr[\text{score of } k] \leq 2 \cdot \left(\frac{2t^2}{2^\lambda}\right)^k \ .$$

This lets us express the success probability of the cheating prover as an infinite sum conditioned on getting a score of k, for any $k \in \mathbb{N}$:

$$\begin{aligned}
\Pr\left[\begin{matrix}\text{verifier}\\ \text{accepts}\end{matrix}\right] &\leq \sum_{k=0}^{\infty} \Pr\left[\begin{matrix}\text{verifier}\\ \text{accepts}\end{matrix} \,\middle|\, \text{score of } k\right] \cdot \Pr[\text{score of } k] \\
&\leq \sum_{k=0}^{\infty} \varepsilon_{\text{cap}}(i^*, t, k, k) \cdot \Pr[\text{score of } k] \leq \sum_{k=0}^{\infty} O\left(t \cdot 2^k \cdot \varepsilon_{\text{per}}(k) \cdot \left(\frac{2t^2}{2^\lambda}\right)^k\right) \\
&\leq \sum_{k=0}^{\infty} O\left(t \cdot 2^k \cdot \beta^k \cdot \varepsilon_{\text{PCP}} \cdot \left(\frac{2t^2}{2^\lambda}\right)^k\right) = O(t \cdot \varepsilon_{\text{PCP}}) \cdot \sum_{k=0}^{\infty} \left(\frac{4\beta \cdot t^2}{2^\lambda}\right)^k = O\left(t \cdot \varepsilon_{\text{PCP}}\right) \ .
\end{aligned}$$

The last equality follows from the fact that $\sum_{k=0}^{\infty} \left(\frac{4\beta \cdot t^2}{2^\lambda}\right)^k = O(1)$ since $\lambda \geq 2\log t + \log\beta + 3$.

Argument size of our construction. We choose an appropriate PCP and obtain the argument size claimed in Theorem 1. Recall that Lemma 4 tells us that the soundness error of our construction is $t \cdot \varepsilon_{\text{PCP}}$ provided that the random oracle output size satisfies $\lambda \geq 2\log t + \log \beta + 3$; in particular, to achieve (t, ϵ) security, we need the PCP soundness error to be $\varepsilon_{\mathrm{PCP}} = \epsilon/t$.

Towards this end, we apply Lemma 1 to any constant-query constant-soundness PCP (over a small alphabet), where the probability of querying each proof location is not too high, as required by the lemma (most PCP constructions satisfy this requirement). We get that its κ-wise repetition has permuted robustness ratio $\beta(\mathsf{b}) = O\big(\frac{\mathsf{b}^{d+1}\cdot\varepsilon_{\mathrm{base}}^{\kappa}}{\mathsf{b}^{d}\cdot\varepsilon_{\mathrm{base}}^{\kappa}}\big) = O(\mathsf{b})$ for a block size b. The block size depends on the number of repetitions, the query complexity, and the soundness of the base PCP (where the last two are constant), and thus we have that $\mathsf{b} = O(\kappa)$. To achieve the desired PCP soundness error, we set the number of repetitions to be $\kappa = O(\log(t/\epsilon))$; hence the number of queries is $\mathsf{q} = O(\log(t/\epsilon))$. Finally, we set stop layer according to Lemma 4 to be $i^* = O(\log \mathsf{b})$.

The argument contains the Merkle cap (which has size $2^{i^*} \cdot \lambda$), PCP query answers (which have total size $\mathsf{q} \cdot \log|\Sigma|$), and the authentication paths (which have total size $\mathsf{q} \cdot \lambda \cdot \log(\mathsf{l}/2^{i^*})$). The argument size thus is

$$\begin{aligned}|\text{argument}| &= 2^{i^*} \cdot \lambda + \mathsf{q} \cdot \log|\Sigma| + \mathsf{q} \cdot \lambda \cdot \log(\mathsf{l}/2^{i^*}) = O(\mathsf{q} \cdot \lambda \cdot \log(\mathsf{l}/\mathsf{q})) \\ &= O\left(\log\frac{t}{\epsilon} \cdot \left(\log t + \log\log\frac{t}{\epsilon}\right) \cdot \log\frac{\mathsf{l}}{\log(t/\epsilon)}\right) = \tilde{O}\left(\log\frac{t}{\epsilon} \cdot \log t\right) \quad ,\end{aligned}$$

where the last equality hides $\log\log\frac{t}{\epsilon}$ and $\log \mathsf{l}$ factors (as we assume that $\mathsf{l} = \mathrm{poly}(n)$ where n is the input length).

Achieving concrete efficiency. In order to achieve concrete efficiency (e.g., the numbers reported in Table 1), our security analysis improves on the above expression by showing that the hidden constant (in the big-O notation) in the soundness expression can be replaced with the constant 1.

To achieve this, it does not suffice simply to pay attention to the constants in the computations, but we need to separately count the queries performed to a tree oracle and to a PCP randomness oracle. The PCP randomness oracle has a long input length (it maps $2^{i^*} \cdot \lambda$ bits to λ bits). Therefore, we count each query to it as 2^{i^*} queries. This is aligned with how one would implement such an oracle using domain extension ([Mer89; Dam89]). Thus, in the full proof, we introduce two new parameters t_{tree} and t_{rnd} such that it always holds that $t = t_{\mathrm{tree}} + 2^{i^*} \cdot t_{\mathrm{rnd}}$. Hence the full proof contains similar expressions as above, where in some cases, t is replaced with either t_{tree}, t_{rnd}, or their (weighted) sum.

3 Definitions

Relations. A relation R is a set of tuples $(\mathbb{x}, \mathbb{w})$ where $\mathbb{x}$ is the instance and $\mathbb{w}$ the witness. The corresponding language $L = L(R)$ is the set of $\mathbb{x}$ for which there exists $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$.

Random oracles. We denote by $\mathcal{U}(\lambda)$ the uniform distribution over functions $\zeta\colon \{0,1\}^* \to \{0,1\}^\lambda$ (implicitly defined by the probabilistic algorithm that assigns, uniformly and independently at random, a λ-bit string to each new input). If ζ is sampled from $\mathcal{U}(\lambda)$, we call ζ a *random oracle*.

Oracle algorithms. We restrict our attention to oracle algorithms that are deterministic since, in the random oracle model, an oracle algorithm can obtain randomness from the random oracle. Given an oracle algorithm A and an oracle $\zeta \in \mathcal{U}(\lambda)$, $\mathsf{queries}(A, \zeta)$ is the set of oracle queries that A^{ζ} makes. We say that A is t*-query* if $|\mathsf{queries}(A, \zeta)| \leq t$ for every $\zeta \in \mathcal{U}(\lambda)$.

3.1 Probabilistically checkable proofs

We provide standard notations and definitions for *probabilistically checkable proofs* (PCPs) [BFLS91; FGL+91; AS98; ALM+98]. Let $\mathsf{PCP} = (\mathbf{P}, \mathbf{V})$ be a pair where $\mathbf{P}$, known as the prover, is an algorithm, and $\mathbf{V}$, known as the verifier, is an oracle algorithm. We say that PCP is a PCP for a relation R with soundness error ε_{PCP} if the following holds.

- **Completeness.**
  For every $(\mathbb{x}, \mathbb{w}) \in R$, letting $\Pi := \mathbf{P}(\mathbb{x}, \mathbb{w}) \in \Sigma^{\mathsf{l}}$, $\Pr_{\rho \in \{0,1\}^{\mathsf{r}}}[\mathbf{V}^{\Pi}(\mathbb{x}; \rho) = 1] = 1$.
- **Soundness.**
  For every $\mathbb{x} \notin L(R)$ and malicious proof $\tilde{\Pi} \in \Sigma^{\mathsf{l}}$, $\Pr_{\rho \in \{0,1\}^{\mathsf{r}}}[\mathbf{V}^{\tilde{\Pi}}(\mathbb{x}; \rho) = 1] \leq \varepsilon_{\text{PCP}}(\mathbb{x})$.

Above, Σ is a finite set that denotes the proof's alphabet, and l is an integer that denotes the proof's length. We additionally denote by q the number of queries to the proof made by the verifier. All of these complexity measures are implicitly functions of the instance $\mathbb{x}$.

Definition 2. *Let Δ be an absolute distance measure. We say that* PCP *has* **(strong) robustness soundness error** ε_{PCP} *with respect to Δ if for every instance $\mathbb{x} \notin L(R)$, proof string $\Pi \in \Sigma^{\mathsf{l}}$, and (absolute) distance parameter $d \in [\mathsf{b}]$,*

$$\Pr_{\rho \in \{0,1\}^{\mathsf{r}}}\left[\exists\, \Pi' \text{ s.t. } \mathbf{V}^{\Pi'}(\mathbb{x}; \rho) = 1 \text{ and } \Delta(\Pi, \Pi') \leq d\right] \leq \varepsilon_{\text{PCP}}(\mathbb{x}, d) \enspace .$$

Standard soundness corresponds to the case where Δ is the Hamming distance and $\varepsilon_{\text{PCP}}(\mathbb{x}, 0) = \varepsilon_{\text{PCP}}(\mathbb{x})$ for some error function $\varepsilon_{\text{PCP}}(\mathbb{x})$ and $\varepsilon_{\text{PCP}}(\mathbb{x}, d) = 1$ for any $d > 0$. The standard notion of robust soundness is a special case of Definition 2, corresponding to the case where $\varepsilon_{\text{PCP}}(\mathbb{x}, d) = \varepsilon_{\text{PCP}}(\mathbb{x})$ for d in some interval $[0, d^*]$ and $\varepsilon_{\text{PCP}}(\mathbb{x}, d) = 1$ for $d > d^*$.

3.2 Non-interactive arguments in the random oracle model

We consider non-interactive arguments in the random oracle model (ROM), where security holds against query-bounded, yet possibly computationally-unbounded, adversaries. Recall that a non-interactive argument typically consists of a prover algorithm and a verifier algorithm that prove and validate statements for a binary relation, which represents the valid instance-witness pairs.

Let $\mathsf{ARG} = (P, V)$ be a tuple of (oracle) algorithms. We say that ARG is a non-interactive argument in the ROM for a relation R with (t, ϵ)*-security* if, for a function $\lambda \colon \mathbb{N} \times (0, 1) \to \mathbb{N}$, the following holds for every query bound $t \in \mathbb{N}$ and soundness error $\epsilon \in (0, 1)$.

– **Completeness.** For every $(\mathbb{x}, \mathbb{w}) \in R$,

$$\Pr\left[V^{\zeta}(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{c} \zeta \leftarrow \mathcal{U}(\lambda(t,\epsilon)) \\ \pi \leftarrow P^{\zeta}(\mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

– **Soundness.** For every $\mathbb{x} \notin L(R)$ with $|\mathbb{x}| \leq t$ and t-query $\tilde{P}$,

$$\Pr\left[V^{\zeta}(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{r} \zeta \leftarrow \mathcal{U}(\lambda(t,\epsilon)) \\ \pi \leftarrow \tilde{P}^{\zeta} \end{array}\right] \leq \epsilon \ .$$

The argument size $\mathsf{s} := |\pi|$ is a function of the desired query bound t and soundness error ϵ. So are the running time pt of the prover P and the running time vt of the verifier V.

4 Our construction

We describe our construction of a (succinct) non-interactive argument from a PCP. Let $(\mathbf{P}, \mathbf{V})$ be a PCP system for the desired relation, with proof length l over an alphabet Σ and query complexity q; for notational convenience, we set $d := \lceil \log \mathsf{l} \rceil$. The construction is additionally parametrized by a *stop layer* $i^* \in \{0, 1, \ldots, d-1\}$, which we will set in the analysis (looking ahead, 2^{i^*} will be roughly the number of queries in the PCP).

The oracles in our construction. The algorithms below are granted access to three oracles:

1. a *tree oracle* $\zeta_{\text{tree}} \colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$, which hashes two elements to one;
2. a *PCP randomness oracle* $\zeta_{\text{rnd}} \colon \{0,1\}^{2^{i^*} \cdot \lambda} \to \{0,1\}^{\mathsf{r}}$, which hashes 2^{i^*} elements to r bits, where r is the randomness complexity of the PCP verifier $\mathbf{V}$;
3. a random permutation $\texttt{Perm} \colon [\mathsf{l}] \to [\mathsf{l}]$, over the locations of a PCP string.

In our analysis, we assume that these oracle are available; all of them can be derived from a single random oracle $\zeta \colon \{0,1\}^* \to \{0,1\}^*$. First, using domain separation one can create multiple random oracles from a single one. The second oracle has a larger domain, which is derived via domain extension (for example, using the Merkle–Damgård iterated construction [Mer89; Dam89]). The third is derived via Feistel networks, as discussed in Section 4.1.

Since the oracles have different input lengths, the cost for querying each oracle differs. We consider a query to the tree oracle as a single query to the random oracle (i.e., reducing a single unit from the cheating prover's query budget). The PCP randomness oracle hashes 2^{i^*} elements and thus its cost will be 2^{i^*} accordingly. We consider queries to the random permutation to be "free". That is, a cheating prover can completely query the permutation oracle with no change to its query budget (this makes our result stronger).

We describe the argument prover P and then the argument verifier V of the tuple $\mathsf{ARG} = (P, V)$.

Argument prover. The argument prover P takes as input an instance $\mathbb{x}$ and a witness $\mathbb{w}$, and computes an argument π as follows.

1. Run the PCP prover $\mathbf{P}$ on $(\mathbb{x}, \mathbb{w})$ to obtain a PCP string $\Pi' \in \Sigma^{\mathsf{l}}$.
2. Use the permutation $\mathtt{Perm}$ to permute this PCP string and obtain Π such that $\Pi[i] = \Pi'[\mathtt{Perm}(i)]$.
3. Use the random oracle ζ_{tree} to Merkle commit to Π, as follows:
   – For every $j \in [\mathsf{l}]$, set the j-th leaf $h_{d,j} := \Pi_j \in \Sigma$.
   – For $i = d-1, d-2, \ldots, i^*$: for $j \in [2^i]$, compute $h_{i,j} := \zeta_{\text{tree}}(i \| j \| h_{i+1,2j-1} \| h_{i+1,2j}) \in \{0,1\}^{\lambda}$.
   – Set the Merkle cap $\mathbf{h} := \{h_{i^*,j} \in \{0,1\}^{\lambda}\}_{j \in [2^{i^*}]}$.
4. Derive randomness $\rho := \zeta_{\text{rnd}}(\mathbf{h}) \in \{0,1\}^{\mathsf{r}}$ and simulate the PCP verifier $\mathbf{V}$ on input $(\mathbb{x}; \rho)$ and PCP string Π; this execution induces q query-answer pairs $(j_1, a_1), \ldots, (j_{\mathsf{q}}, a_{\mathsf{q}}) \in [\mathsf{l}] \times \Sigma$.
5. Output
$$\pi := \Big(\mathbf{h}, (j_1, a_1, p_1), \ldots, (j_d, a_d, p_d)\Big) \tag{1}$$
   where $p_1, \ldots, p_d$ are the authentication paths for the query-answer pairs $(j_1, a_1), \ldots, (j_{\mathsf{q}}, a_{\mathsf{q}})$, truncated at level i^* of the tree.

Argument verifier. The argument verifier V takes as input an instance $\mathbb{x}$ and a proof π (of the form as in Equation (1)), and computes a decision bit as follows.

1. derive randomness $\rho := \zeta_{\text{rnd}}(\mathbf{h})$ for the PCP verifier from the Merkle cap $\mathbf{h}$;
2. check that the PCP verifier $\mathbf{V}$, on input $(\mathbb{x}; \rho)$ and by answering a query to j_r with a_r, accepts;
3. check that $p_1, \ldots, p_d$ are authentication paths of $(j_1, a_1), \ldots, (j_d, a_d)$ relative to the Merkle cap $\mathbf{h}$.

Argument size. The argument π contains the Merkle cap $\mathbf{h} \in \{0,1\}^{2^{i^*} \cdot \lambda}$, a $(\log |\Sigma|)$-bit answer for each of q queries, and q authentication paths. This totals to an argument size that is

$$2^{i^*} \cdot \lambda + \mathsf{q} \cdot \log |\Sigma| + \mathsf{q} \cdot \lambda \cdot \log(\mathsf{l}/2^{i^*}) \ . \tag{2}$$

Each of the q queries in $[\mathsf{l}]$ comes with an authentication path containing the $\log(\mathsf{l}/2^{i^*})$ siblings of vertices on the path from the query to the Merkle cap, which amounts to $\lambda \cdot \log(\mathsf{l}/2^{i^*})$ bits. (More precisely, $\log |\Sigma| + \lambda \cdot (\log(\mathsf{l}/2^{i^*}) - 1)$ bits since the first sibling is a symbol in Σ rather than an output of the random oracle.)

As noted in earlier works (e.g., [BBHR19; BCR+19]) parts of the information across the q authentication paths is redundant, and the argument size can be reduced by *pruning*: the prover includes in π the minimal set of siblings to authenticate the q queries as a set. All concrete argument sizes that we report in Table 1 already account for this straightforward optimization.

Remark 2 (salts for zero knowledge and more). The security analysis that we present in this paper (see Section 7) works *even* if all the vertices in the tree are "salted", which means that an attacker may include an arbitrary string $\sigma_{i,j} \in \{0,1\}^{\lambda}$ in the query that obtains the digest $h_{i,j}$, for any $i \in \{0, 1, \ldots, d-1\}$

and $j \in [2^i]$. That is, our results hold against strong attacks (the attacker can obtain multiple random digests $h_{i,j}$ for any given indices i and j). Salts are useful for showing additional properties of SNARGs in the random oracle model, and in particular, to achieve zero-knowledge[BCS16; IMSX15].

4.1 Implementing the random permutation

We discuss how to implement the random permutation given the random oracle. We need a pseudorandom permutation over the domain $[\mathsf{l}]$, where l is the length of the PCP. There are multiple ways to do this, and here we use Feistel networks, also known as also known as Luby–Rackoff permutations [HR10; LR88; NR99]. These constructions are parameterized by a number of Feistel rounds r; each round calls the random oracle once, and the more rounds, the better the security. In particular, for any algorithm performing q queries, the advantage in distinguishing it from a truly random permutation is exponentially small. In our case, the "adversary" performing the queries is the PCP verifier, which performs non-adaptive queries. We do not need to fool the cheating prover of the argument scheme. The only goal of the permutation is to spread the PCP queries into evenly divided blocks. We use the following theorem.

Theorem 2 ([HR10, Theorem 3]). *The Feistel permutation over* $[\mathsf{l}]$ *with* r *rounds has distinguishing advantage at most* $\frac{q}{r+1}\left(\frac{4q}{\mathsf{l}}\right)^r$, *for any non-adaptive* q*-query algorithm.*

In particular, setting r to be large enough (and recalling that q is merely the number of queries in the PCP), we can set the distinguishing probability to be extremely small with negligible effect on our proof, and thus we omit these terms and perm our analysis under the assumption of a truly random permutation. For our theoretical needs, the theorem above suffices. Even setting concrete parameters, the number of rounds needed is relatively small. However, there are several other alternatives with different concrete performance. One example is to use DES (recall that we do not need to hide the key from the cheating prover) or to use other standards such as FFX [BRS10].

5 Permuted robust soundness

Definition 3 (Block distance). *Let* $\Pi, \Pi' \in \Sigma^{\mathsf{l}}$ *be two strings, and consider them divided to* b *blocks. That is, we view them as* $\Pi, \Pi' \in (\Sigma^{\mathsf{l}/\mathsf{b}})^{\mathsf{b}}$. *Define* $\Delta_{\mathsf{b}}(\Pi, \Pi')$ *to be the block-wise distance between* Π *and* Π' *(i.e., the number of blocks of symbols on which they differ).*

Moreover, for any permutation $\mathtt{Perm}$, *we define* $\Delta_{\mathsf{b}}^{\mathtt{Perm}}$ *similarly where we first permute the order according to* $\mathtt{Perm}$ *and then divide to blocks.*

Game 3. The *permuted robust soundness game* is parametrized by a PCP verifier $\mathbf{V}$, an instance $\mathbb{x}$, a positive integer b, and a non-negative integer d. We denote by $\mathcal{G}_{\text{per}}(\mathbf{V}, \mathbb{x}, \mathsf{b}, d, \tilde{\mathbf{P}})$ the boolean random variable denoting whether a malicious prover $\tilde{\mathbf{P}}$ wins in this game, according to the description below.

1. $\texttt{Perm}$ is sampled as a random permutation over $[\mathsf{l}]$.
2. $\tilde{\mathbf{P}}$ outputs a proof string $\Pi \in \Sigma^{\mathsf{l}}$.
3. $\tilde{\mathbf{P}}$ receives a random string $\rho \in \{0,1\}^{\mathsf{r}}$, which represents randomness for the PCP verifier.
4. $\tilde{\mathbf{P}}$ outputs a proof string $\Pi' \in \Sigma^{\mathsf{l}}$.
5. The game outputs 1 if and only if $\mathbf{V}^{\Pi'}(\mathbb{x};\rho) = 1$, and $\Delta_{\mathsf{b}}^{\texttt{Perm}}(\Pi, \Pi') \leq d$.

Definition 4. *A PCP $(\mathbf{P}, \mathbf{V})$ for a relation R has* **permuted robust soundness error** *$\varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d)$ if for every instance $\mathbb{x} \notin L(R)$, integers b, d, and malicious prover $\tilde{\mathbf{P}}$,*

$$\Pr\left[\mathcal{G}_{\mathrm{per}}(\mathbf{V}, \mathbb{x}, \mathsf{b}, d, \tilde{\mathbf{P}}) = 1\right] \leq \varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d) \ .$$

The PCP *has permuted robustness* **ratio** *β (with respect to b) if for any $d \in \{0, 1, \ldots, \mathsf{b}\}$ it holds that:*

$$\varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d+1) \leq \beta \cdot \varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d) \ .$$

Lemma 5 (restatement of Lemma 1). *Let $(\mathbf{P}, \mathbf{V})$ be a PCP with soundness error $\varepsilon_{\mathrm{base}}$, length l, and query complexity q, and assume each location is queried with probability at most p. Let $\kappa \in \mathbb{N}$ and let $(\mathbf{P}, \mathbf{V})_\kappa$ be the κ-repeated version of $(\mathbf{P}, \mathbf{V})$. If $\mathsf{b} \geq \kappa \cdot \mathsf{q} \cdot \varepsilon_{\mathrm{base}}^{-1}$, and $p \leq (8\mathsf{b} \cdot \kappa)^{-1}$ then $(\mathbf{P}, \mathbf{V})_\kappa$ has strong permuted robust soundness error*

$$\varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b}, d) \leq \frac{e^{1.2 \cdot d}}{d!} \cdot \mathsf{b}^d \cdot \varepsilon_{\mathrm{base}}^{\kappa} \ .$$

Moreover, the robustness ratio is $\beta \leq 2.33 \cdot \mathsf{b}$.

Using the above lemma, we can plug in a PCP with soundness error 1/2 that uses 3 queries, with proof length l and get the following corollary.

Corollary 1. *For any $\kappa \in N$, there is a PCP $(\mathbf{P}, \mathbf{V})$ that has query complexity $\mathsf{q} = 3\kappa$ and has permuted robust ratio*

$$\beta \leq 2.33 \cdot \kappa \cdot 3 \cdot 2 = 14\kappa \ .$$

The lemma is proved in the full version.

6 Cap soundness

We define a PCP soundness game that we call *cap soundness game*. The game is played on a graph $G = (V, E)$ that represents the Merkle tree in the Micali construction. The game starts out with the graph being empty ($E = \emptyset$), and the PCP adversary can iteratively choose one of several actions, with some budget limitations that constrain how many collisions and inversions the PCP adversary can create in the Merkle tree. We stress that this game is information-theoretic, and can be viewed as an abstract modeling of the effects of these attacks in the real world. The edges are in fact *hyper*edges in order to keep track of which inputs are "hashed" together to create a specific output. Below we introduce definitions for describing the game, and then relate winning this game to winning the reverse soundness game.

Definition 5. *Let d, i^*, and λ be positive integers, and Σ a finite alphabet. The vertex set V is the union $V_{i^*} \cup \cdots \cup V_d$ where $V_d := \{(d, j, h) : j \in [2^d], h \in \Sigma\}$ and, for every $i \in \{i^*, \dots, d-1\}$, $V_i := \{(i, j, h) : j \in [2^i], h \in \{0,1\}^\lambda\}$. We consider graphs of the form $G = (V, E)$ where E is a set of (hyper)edges chosen from the following collection:*

$$\mathcal{E} = \left\{ (u, v_0, v_1) : \begin{array}{c} u = (i, j, h) \in V_i \\ v_0 = (i+1, 2j-1, h_0) \in V_{i+1} \\ v_1 = (i+1, 2j, h_1) \in V_{i+1} \end{array} \right\} .$$

For an edge $e = (u, v_0, v_1)$, we call u its base vertex and v_0, v_1 its children vertices. We also define:
- *the* **edges** *of a base vertex $u = (i, j, h)$ are $\mathsf{edges}(u) := \{(u, v_0, v_1) \in E : v_0, v_1 \in V_{i+1}\}$;*
- *the* **level** *of an edge e, denoted $\mathsf{level}(e)$, is i if its base vertex has the form $u = (i, j, h)$.*

Each leaf of the graph, namely, a vertex $u = (d, j, h)$ at level d is associated to a symbol, h. A collection of leaves thus determine a string whose location j is the symbol of the j-th leaf.

Definition 6. *Let $G = (V, E)$ be a graph over the vertex set V as in Definition 5.*
- *A vertex $u_d \in V_d$ is* **connected in** *G to a vertex $u_\ell \in V_\ell$ if there exist vertices $u_{d-1}, \dots, u_{\ell+1}$ such that, for all $i \in \{d, d-1, \dots, \ell+1\}$, $u_i \in V_i$ and there is an edge $e \in E$ such that $u_i, u_{i-1} \in e$.*
- *A vertex $v \in V_i$ is* **free in** *G if for every $u \in V_{i-1}$ and $v' \in V_i$ it holds that $(u, v, v') \notin E$.*

Notice that the connectivity concerns only paths that begin at any leaf (a vertex at level d) and move directly towards the vertex u_ℓ. That is, at each step on the path, the level decreases by 1. Moreover, a vertex at level i is free if there is no edge that connects it to a vertex at level $i-1$.

Definition 7. *Let $G = (V, E)$ be a graph over the vertex set V as in Definition 5, and let Perm be a permutation. A string $s \in (\Sigma \cup \{\bot\})^{\mathsf{l}}$ is* **consistent with** *G with respect to Perm if for every $j \in [\mathsf{l}]$ such that $s[j] \neq \bot$ there exists a vertex $v_{\mathit{Perm}(j)} = (d, \mathit{Perm}(j), h) \in V_d$ such that $h = s[j]$ and $v_{\mathit{Perm}(j)}$ is connected some $u \in V_{i^*}$ in G. In such a case we write $\mathsf{Consistent}(G, \mathit{Perm}, s) = 1$.*

Game 4. The *cap soundness game* is parametrized by a PCP verifier $\mathbf{V}$, an instance $\mathbb{x}$, and an integer λ. The game receives as input a malicious prover $\tilde{\mathbf{P}}$, a root budget $t_{\text{rnd}} \in \mathbb{N}$, a tree budget $t_{\text{tree}} \in \mathbb{N}$, a collision budget $t_{\text{col}} \in \mathbb{N}$, and an inversion budget $t_{\text{inv}} \in \mathbb{N}$, which we denote $\mathcal{G}_{\text{cap}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, i^*, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}})$. The game works as follows:

- **Initialization:**
  1. Set $E := \emptyset$ to be an empty edge set for the graph $G = (V, E)$.
  2. Set Rand to be an empty mapping from V to verifier randomness.

   3. $\texttt{Perm}$ is a sampled as a uniformly random permutation over $[\mathsf{l}]$ and given to the prover $\tilde{\mathbf{P}}$.
- **Round:** $\tilde{\mathbf{P}}$ chooses one of the following options until it decides to exit.
  - **Option** ADD**:** $\tilde{\mathbf{P}}$ submits a vertex $u = (i, j, h) \in V$ with $i \in \{i^*, \ldots, d-1\}$ and strings h_0, h_1.
    1. Set the (hyper)edge $e := (u, v_0, v_1)$ where $v_0 := (i+1, 2j-1, h_0) \in V_{i+1}$ and $v_1 := (i+1, 2j, h_1) \in V_{i+1}$.
    2. If u is not free then $t_{\mathrm{inv}} \leftarrow t_{\mathrm{inv}} - 1$.
    3. If $|e(u)| \geq 1$ then $t_{\mathrm{col}} \leftarrow t_{\mathrm{col}} - 1$.
    4. Add $e = (u, v_0, v_1)$ to E.
    5. $t_{\mathrm{tree}} \leftarrow t_{\mathrm{tree}} - 1$.
  - **Option** RND**:** $\tilde{\mathbf{P}}$ submits a cap vertex $v_{\mathbf{h}} \in V_{i^*}$.
    1. If Rand already contains an entry for $v_{\mathbf{h}}$ then set $\rho \leftarrow \mathsf{Rand}[v_{\mathbf{h}}]$.
    2. If Rand does not contain an entry for $v_{\mathbf{h}}$ then sample $\rho \in \{0,1\}^{\mathsf{r}}$ at random and set $\mathsf{Rand}[v_{\mathbf{h}}] \leftarrow \rho$.
    3. $t_{\mathrm{rnd}} \leftarrow t_{\mathrm{rnd}} - 1$.
    4. ρ is given to $\tilde{\mathbf{P}}$.
- **Output:** $\tilde{\mathbf{P}}$ outputs a cap $v_1, \ldots, v_{2^{i^*}} \in V_{i^*}$ and leaf vertices $v_1, \ldots, v_{\mathsf{q}} \in V_d$.
- **Decision:** $\tilde{\mathbf{P}}$ wins if all checks below pass.
  1. Construct a PCP string $\Pi \in (\Sigma \cup \{\perp\})^{\mathsf{l}}$: for every $r \in [\mathsf{q}]$, parse the r-th leaf vertex as $v_r = (d, j, h)$ and set $\Pi[\texttt{Perm}(j)] := h \in \Sigma$; set $\Pi[j] := \perp$ for all other locations.
  2. Retrieve PCP randomness for this root vertex: $\rho^* \leftarrow \mathsf{Rand}[v_{\mathbf{h}}]$.
  3. Check that the PCP verifier accepts: $\mathbf{V}^{\Pi}(\mathbb{x}; \rho^*) = 1$.
  4. Check that Π is consistent in G w.r.t. $\texttt{Perm}$: $\mathsf{Consistent}(G, \texttt{Perm}, \Pi) = 1$.
  5. Check that $\tilde{\mathbf{P}}$ is within budget: $t_{\mathrm{col}} \geq 0$, $t_{\mathrm{inv}} \geq 0$, $t_{\mathrm{rnd}} \geq 0$, and $t_{\mathrm{tree}} \geq 0$.

Definition 8. *A PCP $(\mathbf{P}, \mathbf{V})$ for a relation R has cap soundness error $\varepsilon_{\mathrm{cap}}(\mathbb{x}, \lambda, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, t_{\mathrm{col}}, t_{\mathrm{inv}})$ if for every $\mathbb{x} \notin L(R)$, output size $\lambda \in \mathbb{N}$, malicious prover $\tilde{\mathbf{P}}$, stop layer i^*, and budgets $t_{\mathrm{rnd}}, t_{\mathrm{col}}, t_{\mathrm{inv}} \in \mathbb{N}$,*

$$\Pr\left[\mathcal{G}_{\mathrm{cap}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, t_{\mathrm{col}}, t_{\mathrm{inv}}) = 1\right] \leq \varepsilon_{\mathrm{cap}}(\mathbb{x}, \lambda, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, t_{\mathrm{col}}, t_{\mathrm{inv}}) \ .$$

Lemma 6 (restatement of Lemma 2). *Let $(\mathbf{P}, \mathbf{V})$ be a PCP for a relation R with permuted robust soundness error $\varepsilon_{\mathrm{PCP}}$ with respect to distance Δ_{b} for parameter b, and suppose it has uniformly random queries. Then, $(\mathbf{P}, \mathbf{V})$ has cap soundness error*

$$\varepsilon_{\mathrm{cap}}(\mathbb{x}, \lambda, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, t_{\mathrm{col}}, t_{\mathrm{inv}}) \leq t_{\mathrm{rnd}} \cdot 2^{t_{\mathrm{col}}} \cdot \varepsilon_{\mathrm{per}}(\mathbb{x}, \mathsf{b} = 2^{i^*}, d = t_{\mathrm{inv}}) \ .$$

The lemma is proved in the full version.

7 Soundness based on permuted robust soundness

Theorem 5 (restatement of Lemma 4). *Suppose that our construction (described in Section 4) is instantiated with:*

1. a PCP with soundness $\varepsilon_{\mathrm{PCP}}$ and permuted robustness ratio $\beta(\mathsf{b})$;

2. a random oracle with output size λ; and
3. stop layer i^*.

Then, provided that $\lambda \geq 2\log t + \log \beta(2^{i^*}) + 3$, the construction has a soundness error $\epsilon(t)$ against t-query adversaries that is bounded as follows:

$$\epsilon(t) \leq t \cdot \varepsilon_{\mathrm{PCP}} \ .$$

Using this soundness analysis along with concrete PCPs, we get the following corollary:

Corollary 2 (restatement of Theorem 1). *Our construction implies a SNARG for* NP *in the random oracle model that has (t, ϵ)-security with an argument size of*

$$O\Big(\log(t/\epsilon) \cdot \big(\log t + \log\log(t/\epsilon)\big) \cdot \log(\mathsf{l}/\log(t/\epsilon))\Big) \ . \tag{3}$$

We prove the theorem in Section 7.1 and the corollary in Section 7.2.

7.1 Proof of Theorem 5

Fix $t \in \mathbb{N}$. Let $\tilde{P}$ be a t-query cheating argument prover. Note that $\tilde{P}$ can make queries to the randomness oracle ζ_{rnd} and tree oracle ζ_{tree} (or the permutation but we are giving these queries to the cheating prover for free). Recall that the randomness oracle t_{rnd} has a larger domain size and thus has cost 2^{i^*}. For any choice of positive integers t_{rnd} and t_{tree} such that $2^{i^*} \cdot t_{\mathrm{rnd}} + t_{\mathrm{tree}} \leq t$, below we condition on the event that $\tilde{P}$ makes t_{rnd} queries to ζ_{rnd} and t_{tree} queries to ζ_{tree}. For any such choice, we obtain the same upper bound (independent of the choice of t_{rnd} and t_{tree}), and hence conclude that the bound holds for the distribution of t_{rnd} and t_{tree} implied by $\tilde{P}$.

We rely on the claim below, which states that a cheating argument prover can be transformed into a cheating PCP prover for the cap soundness game with a small loss, when the budgets for collisions and inversions correspond to the corresponding scores of the trace of the argument prover.

Claim 6. *There is an efficient transformation $\mathbb{T}$ such that, for every cheating argument prover $\tilde{P}$, the cheating PCP prover $\tilde{\mathbf{P}} := \mathbb{T}(\tilde{P})$ satisfies the following condition for every $k \in \mathbb{N}$:*

$$\Pr\left[V^{\zeta}(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{r} \zeta \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \tilde{P}^{\zeta} \\ \mathsf{tr}_{\mathrm{rnd}} \leftarrow \mathsf{queries}_{\mathrm{rnd}}(\tilde{P}, \zeta) \\ \mathsf{tr}_{\mathrm{tree}} \leftarrow \mathsf{queries}_{\mathrm{tree}}(\tilde{P}, \zeta) \\ |\mathsf{tr}_{\mathrm{rnd}}| = t_{\mathrm{rnd}}, |\mathsf{tr}_{\mathrm{tree}}| = t_{\mathrm{tree}} \\ \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) \leq k \\ \mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}) \leq k \end{array} \right] \leq \Pr\left[\mathcal{G}_{\mathrm{cap}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, k, k) = 1\right] \ . \tag{4}$$

Above $\mathsf{queries}_{\mathrm{rnd}}(\tilde{P}, \zeta)$ *and* $\mathsf{queries}_{\mathrm{tree}}(\tilde{P}, \zeta)$ *respectively denote the queries by $\tilde{P}$ to the oracles ζ_{rnd} and ζ_{tree} obtained from ζ via domain separation.*

The proof of Claim 6 is given at the end of this proof.

We use the oracle scoring lemma to obtain a bound that will be useful in the analysis further below; we also use the assumption that $\lambda \geq 2\log t + \log \beta + 3$ and the fact that $t \geq t_{\mathrm{tree}}$. The bound holds for any choice of a parameter $k \in \mathbb{N}$.

$$\begin{aligned}
&\sum_{k=0}^{\infty} 2^k \cdot \beta^k \cdot \Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \\
&\quad \leq 1 + \sum_{k=1}^{\infty} (2\beta)^k \cdot \left(\frac{1}{k!} \cdot \left(\frac{2t \cdot t_{\mathrm{tree}}}{2^\lambda} \right)^k + \left(\frac{t_{\mathrm{tree}}^2}{2 \cdot 2^\lambda} \right)^k \right) \\
&\quad = 1 + \sum_{k=1}^{\infty} \frac{1}{k!} \cdot \left(\frac{4t \cdot t_{\mathrm{tree}} \cdot \beta}{2^\lambda} \right)^k + \sum_{k=1}^{\infty} \left(\frac{t_{\mathrm{tree}}^2 \cdot \beta}{2^\lambda} \right)^k \\
&\quad \leq 1 + \sum_{k=1}^{\infty} \frac{1}{k!} \cdot \left(\frac{4t_{\mathrm{tree}}}{2^3 \cdot t} \right)^k + \sum_{k=1}^{\infty} \left(\frac{t_{\mathrm{tree}}}{2^3 \cdot t} \right)^k \qquad (\text{since } \lambda \geq 2\log t + \log \beta + 3) \\
&\quad \leq 1 + \frac{t_{\mathrm{tree}}}{t} \cdot \sum_{k=1}^{\infty} \frac{1}{k!} \cdot \left(\frac{4}{2^3} \right)^k + \frac{t_{\mathrm{tree}}}{t} \cdot \sum_{k=1}^{\infty} \left(\frac{1}{2^3} \right)^k \\
&\quad \leq 1 + 0.65 \cdot \frac{t_{\mathrm{tree}}}{t} + 0.15 \cdot \frac{t_{\mathrm{tree}}}{t} \\
&\quad \leq 1 + \frac{t_{\mathrm{tree}}}{t} \quad .
\end{aligned}$$

Using the above bound, we conclude the following:

$$\begin{aligned}
&\Pr\left[V^\zeta(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{c} \zeta \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \tilde{P}^\zeta \end{array} \right] \\
&\leq \sum_{k=0}^{\infty} \Pr\left[\mathcal{G}_{\mathrm{cap}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, k, k) = 1 \right] \cdot \\
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \quad (\text{by Claim 6}) \\
&\leq \sum_{k=0}^{\infty} \varepsilon_{\mathrm{cap}}(\mathbb{x}, \lambda, i^*, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, k, k) \cdot \\
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \quad (\text{by Definition 8}) \\
&\leq \sum_{k=0}^{\infty} t_{\mathrm{rnd}} \cdot 2^k \cdot \varepsilon_{\mathrm{per}}(\mathbb{x}, 2^{i^*}, k) \cdot \\
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \quad (\text{by Lemma 6}) \\
&\leq \sum_{k=0}^{\infty} t_{\mathrm{rnd}} \cdot 2^k \cdot \beta^k \cdot \varepsilon_{\mathrm{per}}(\mathbb{x}, 2^{i^*}, 0) \cdot \\
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \quad (\text{since } \beta \text{ is the robustness ratio}) \\
&\leq \sum_{k=0}^{\infty} t_{\mathrm{rnd}} \cdot 2^k \cdot \beta^k \cdot \varepsilon_{\mathrm{PCP}}(\mathbb{x}) \cdot
\end{aligned}$$

$$\begin{aligned}
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k] \quad (\text{since } \varepsilon_{\mathrm{per}}(\mathbb{x}, 2^{i^*}, 0) = \varepsilon_{\mathrm{PCP}}(\mathbb{x}))\\
&= t_{\mathrm{rnd}} \cdot \varepsilon_{\mathrm{PCP}}(\mathbb{x}) \cdot \sum_{k=0}^{\infty} 2^k \cdot \beta^k \cdot\\
&\Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}, \mathsf{tr}_{\mathrm{rnd}}) = k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) = k]\\
&\le t_{\mathrm{rnd}} \cdot \varepsilon_{\mathrm{PCP}}(\mathbb{x}) \cdot \left(1 + \frac{t_{\mathrm{tree}}}{t}\right) \quad (\text{using the bound proved above})\\
&= t \cdot \varepsilon_{\mathrm{PCP}}(\mathbb{x}) \ .
\end{aligned}$$

This concludes the proof of the theorem, giving an upper bound on the soundness error.

We are left to prove the claim used in the proof above.

Proof of Claim 6. For the sake of simplicity of the proof, we will assume the following two conditions that are without loss of generality:

– (No duplicate queries): The cheating argument prover $\tilde{P}$ does not make duplicate queries to the random oracle. This can be achieved by having $\tilde{P}$ store the answers to prior queries, and making only new queries to the random oracle, and has no effect on the rest of the proof. Recall that we are considering the Micali construction with salts (see Remark 2), which means that the aforementioned "no duplicate query" condition implies that the prover does not make the same query with the same salt but can make the same query with a different salt (as that results in a different input to the random oracle).
– (Self-verifying): The cheating prover, before submitting his final proof, runs the verify to check that it accepts, and otherwise submits a $\perp$ symbol. This can be achieved by having $\tilde{P}$ run the verifier at the end of its execution. Admittingly, this is not completely without loss of generalization, as this might cost a few additional queries. However, this has a negligible effect on the query complexity and on our results and we omit it.

We use $\tilde{P}$ to construct a PCP prover $\tilde{\mathbf{P}}$ that plays in the cap soundness game $\mathcal{G}_{\mathrm{cap}}$ (Game 4). The PCP prover $\tilde{\mathbf{P}}$ simulates the argument prover $\tilde{P}$ and, whenever $\tilde{P}$ performs a query x to the random oracle, $\tilde{\mathbf{P}}$ performs one of the following actions depending on x.

– *Root query:* x is a query in $\{0,1\}^{\lambda}$ to the PCP randomness oracle ζ_{rnd}.
  1. Construct the root vertex $v_{\mathbf{h}} := (0, 1, x) \in V_0$.
  2. Submit, via Option RND in $\mathcal{G}_{\mathrm{cap}}$, the root vertex $v_{\mathbf{h}}$.
  3. Receive from $\mathcal{G}_{\mathrm{cap}}$ a random string $\rho \in \{0,1\}^{\mathsf{r}}$ for the PCP verifier.
  4. Send ρ to $\tilde{P}$.
– *Tree query:* x is a query (i, j, h_0, h_1, σ) to the tree oracle ζ_{tree} with indices $i \in \{0, 1, \ldots, d-1\}$ and $j \in [2^i]$, strings h_0, h_1 in $\{0,1\}^{\lambda}$ or Σ (depending on i) and salt $\sigma \in \{0,1\}^{\lambda}$.
  1. Sample a random $h \in \{0,1\}^{\lambda}$ and set $u := (i, j, h) \in V_i$;
  2. Submit u, h_0, h_1 via Option ADD in $\mathcal{G}_{\mathrm{cap}}$;
  3. Send h to $\tilde{P}$.

– *Other query:* x is a query that does not fit either case above.
  1. Sample a random $h \in \{0,1\}^{\lambda}$ and send h to $\tilde{P}$.

At the end of its simulation, $\tilde{P}$ outputs a proof π that is parsed as in Equation (1). The cheating prover $\tilde{\mathbf{P}}$ outputs the root vertex $v_{\mathbf{h}} := (0, 1, \mathbf{h})$ where $\mathbf{h}$ is the root contained in π and also outputs the leaf vertices $\{v_r\}_{r \in [\mathsf{q}]}$ where $v_r := (d, j_r, a_r)$ specifies the location $j_r \in [\mathsf{l}]$ and answer $a_r \in \Sigma$ in π for the r-th query. We now argue that the constructed PCP prover $\tilde{\mathbf{P}}$ satisfies Equation (4).

Perfect simulation. We claim that the PCP prover $\tilde{\mathbf{P}}$ performs a perfect simulation of the argument prover $\tilde{P}$, in that $\tilde{\mathbf{P}}$ gives values to $\tilde{P}$ that are identically distributed as the answers from a random oracle ζ. We argue this for (well-formed) queries to ζ_{rnd} and queries to ζ_{tree}; any other types of queries are trivially uniformly random because that is how $\tilde{\mathbf{P}}$ answers in the third bullet.

First, if $\tilde{P}$ issues a query x to the randomness oracle ζ_{rnd}, then $\tilde{P}$ replies with the randomness ρ received from the cap soundness game $\mathcal{G}_{\mathrm{cap}}$, which is uniformly distributed.

Second, suppose that $\tilde{P}$ issues a query x to the tree oracle ζ_{tree}. Since $\tilde{P}$'s queries are distinct, either of i, j, h_0, h_1 are new elements, in which case no value h has been assigned; or the salt σ is new (the salt allows $\tilde{P}$ to get new randomness for the same choice of i, j, h_0, h_1).

When $\tilde{P}$ wins then $\tilde{\mathbf{P}}$ wins. We claim that the PCP prover $\tilde{\mathbf{P}}$ wins the cap soundness game whenever the argument prover $\tilde{P}$ convinces the argument verifier V.

Suppose that $V^{\zeta}(\mathbb{x}, \pi) = 1$ for the proof π output by $\tilde{P}$ and the (partial) random oracle ζ implied by the randomness of the simulation and cap soundness game. Let $\rho^* \leftarrow \mathsf{Rand}[v_{\mathbf{h}}]$ where $v_{\mathbf{h}}$ is the root vertex output by $\tilde{\mathbf{P}}$ and Rand is the table maintained by the cap soundness game.

We can deduce the following two items, which mean that $\tilde{\mathbf{P}}$ wins up to budget constraints.

– $\mathbf{V}^{\Pi}(\mathbb{x}; \rho^*) = 1$ where Π is the PCP proof with value $a_r \in \Sigma$ at location $j_r \in [\mathsf{l}]$ for every $r \in [\mathsf{q}]$, and the value $\perp$ at all other locations. This is because if the argument verifier V accepts then the underlying PCP verifier $\mathbf{V}$ also accepts: $\mathbf{V}$ on instance $\mathbb{x}$ and randomness ρ^* accepts when, for every $r \in [\mathsf{q}]$, the answer to query j_r is the value a_r.
– For every $r \in [\mathsf{q}]$, the leaf vertex v_r is connected in G to the root vertex $v_{\mathbf{h}}$ (G is the graph maintained by the cap soundness game), provided that $\tilde{P}$ has queried all vertices in the authentication paths in the final proof π (which we assumed is the case). This is because $p_1, \ldots, p_{\mathsf{q}}$ in π are valid authentication paths for the query-answer pairs $(j_1, a_1), \ldots, (j_{\mathsf{q}}, a_{\mathsf{q}})$ with respect to the root $\mathbf{h}$ in π (and the oracle ζ), and thus all the edges between the leaf vertices $\{v_r\}_{r \in [\mathsf{q}]}$ and the root vertex $v_{\mathbf{h}}$ are in the graph.

The budget of $\tilde{\mathbf{P}}$ suffices. We left to argue that the budgets given to $\tilde{\mathbf{P}}$ suffices for the cap soundness game to accept. Let ϵ_k be the success probability of the argument prover $\tilde{P}$ conditioned on $\mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) \leq k$ and $\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}) \leq k$

where $\mathsf{tr}_{\mathrm{rnd}}$ is $\tilde{P}$'s trace of queries to the randomness oracle ζ_{rnd} and $\mathsf{tr}_{\mathrm{tree}}$ is $\tilde{P}$'s trace of queries to the tree oracle ζ_{tree} (with $|\mathsf{tr}_{\mathrm{rnd}}| = t_{\mathrm{rnd}}$ and $|\mathsf{tr}_{\mathrm{tree}}| = t_{\mathrm{tree}}$).

Consider any fixed oracle ζ that contributes to ϵ_k, i.e., $\tilde{P}$ wins and the scores are at most k. There is a one-to-one mapping of the values of the oracle ζ to the random values in the simulation that make $\tilde{\mathbf{P}}$ win within the required budget of k. The mapping is done in a natural way because $\tilde{\mathbf{P}}$ does a perfect simulation of $\tilde{P}$: the randomness used by the simulation for query x (either tree of root query) corresponds to the value of the random oracle on query x.

Hence, we only need to show that when the scores are bounded by k then the budgets for the cap soundness game suffice to win the game.

- If $\mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) \leq k$ then we know that $\tilde{P}$ has found at most k collisions in $\mathsf{tr}_{\mathrm{tree}}$. In this case, $\tilde{\mathbf{P}}$ submits the same number of collisions as $\tilde{P}$ because it imitates its queries. Thus, the collision budget will suffice for the simulation.
- If $\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}}) \leq k$ then we know that $\tilde{P}$ has performed at most k "almost inversions" in $\mathsf{tr}_{\mathrm{tree}}$ and $\mathsf{tr}_{\mathrm{rnd}}$ together. In this case, $\tilde{\mathbf{P}}$ simulates the same queries and will use Option INV at most k times. Thus, the inversion budget will suffice for the simulation.
- Since $\tilde{P}$ performs at most t_{rnd} queries to the randomness oracle ζ_{rnd} and at most t_{tree} queries to the tree oracle ζ_{tree}, then $\tilde{\Pi}$ will perform the same amount of queries to Option RND and Option ADD respectively.

□

7.2 The argument size

We prove Corollary 2. Fix t and ϵ. As a base PCP, we take any constant query, constant soundness PCP over a binary (or small) alphabet of polynomial length l. We amplify the soundness by repeating $\kappa = O(\log(t/\epsilon))$ times, and get soundness $\varepsilon_{\mathrm{PCP}} = O(\epsilon/t)$, and query complexity $\mathsf{q} = O(\log(t/\epsilon))$. By Corollary 1, we get that the repeated PCP has permuted robustness ratio $\beta = O(\kappa) = O(\log(t/\epsilon))$, with $\mathsf{b} = O(\kappa)$, and $2^{i^*} = O(\mathsf{q})$. Thus, we need to set

$$\lambda = 2\log t + \log\beta + 3 = O(\log t + \log\log(t/\epsilon)) \ .$$

Plugging this in the argument size formula given in Equation (2), we get that the argument size is:

$$\begin{aligned} &2^{i^*} \cdot \lambda + \mathsf{q} \cdot \log|\Sigma| + \mathsf{q} \cdot \lambda \cdot \log(\mathsf{l}/2^{i^*}) = \\ &O\left(\log(t/\epsilon) \cdot (\log t + \log\log(t/\epsilon)) \cdot \log(\mathsf{l}/\log(t/\epsilon))\right) \ . \end{aligned}$$

Acknowledgments

Alessandro Chiesa is funded by the Ethereum Foundation and Eylon Yogev is funded by the ISF grants 484/18, 1789/19, Len Blavatnik and the Blavatnik Foundation, The Blavatnik Interdisciplinary Cyber Research Center at Tel Aviv University, and The Raymond and Beverly Sackler Post-Doctoral Scholarship. This work was done (in part) while the second author was visiting the Simons Institute for the Theory of Computing.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. “Proof verification and the hardness of approximation problems”. In: *Journal of the ACM* (1998). Preliminary version in FOCS ’92.

[AS98] S. Arora and S. Safra. “Probabilistic checking of proofs: a new characterization of NP”. In: *Journal of the ACM* (1998). Preliminary version in FOCS ’92.

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: S&P ’18.

[BBC+18] C. Baum, J. Bootle, A. Cerulli, R. d. Pino, J. Groth, and V. Lyubashevsky. “Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits”. In: CRYPTO ’18.

[BBHR19] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. “Scalable Zero Knowledge with No Trusted Setup”. In: CRYPTO ’19.

[BCC+16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. “Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting”. In: EUROCRYPT ’16.

[BCG+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: SP ’14.

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

[BCR+19] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. “Aurora: Transparent Succinct Arguments for R1CS”. In: EUROCRYPT ’19.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. “Interactive Oracle Proofs”. In: TCC ’16-B.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. “Checking computations in polylogarithmic time”. In: STOC ’91.

[BFS20] B. Bünz, B. Fisch, and A. Szepieniec. “Transparent SNARKs from DARK Compilers”. In: EUROCRYPT ’20.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. “Efficient Probabilistically Checkable Proofs and Applications to Approximations”. In: STOC ?93.

[BISW17] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. “Lattice-Based SNARGs and Their Application to More Efficient Obfuscation”. In: EUROCRYPT ’17.

[BISW18] D. Boneh, Y. Ishai, A. Sahai, and D. J. Wu. “Quasi-Optimal SNARGs via Linear Multi-Prover Interactive Proofs”. In: EUROCRYPT ’18.

[BRS10] M. Bellare, P. Rogaway, and T. Spies. “The FFX mode of operation for format-preserving encryption”. In: *NIST submission* (2010).

[CMS19] A. Chiesa, P. Manohar, and N. Spooner. “Succinct Arguments in the Quantum Random Oracle Model”. In: TCC ’19.

[CY20] A. Chiesa and E. Yogev. “Barriers for Succinct Arguments in the Random Oracle Model”. In: TCC ’20.

[Dam89] I. Damgård. “A Design Principle for Hash Functions”. In: CRYPTO ’89.

[DHK15] I. Dinur, P. Harsha, and G. Kindler. “Polynomially Low Error PCPs with polyloglog n Queries via Modular Composition”. In: STOC ’15.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. “Approximating clique is almost NP-complete (preliminary version)”. In: SFCS ’91.

[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.

[GH98] O. Goldreich and J. Håstad. “On the complexity of interactive proofs with bounded communication”. In: *Information Processing Letters* (1998).

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

[HR10] V. T. Hoang and P. Rogaway. “On Generalized Feistel Networks”. In: CRYPTO ’10.

[IMSX15] Y. Ishai, M. Mahmoody, A. Sahai, and D. Xiao. “On Zero-Knowledge PCPs: Limitations, Simplifications, and Applications”. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`.

[Kil92] J. Kilian. “A note on efficient zero-knowledge proofs and arguments”. In: STOC ’92.

[LR88] M. Luby and C. Rackoff. “How to Construct Pseudorandom Permutations from Pseudorandom Functions”. In: *SIAM Journal on Computing* (1988).

[Mer89] R. C. Merkle. “One Way Hash Functions and DES”. In: CRYPTO ’89.

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

[NR99] M. Naor and O. Reingold. “On the Construction of Pseudorandom Permutations: Luby–Rackoff Revisited”. In: *Journal of Cryptology* (1999).

[Zc14] Electric Coin Company. “Zcash Cryptocurrency”. `https://z.cash/`.

[zkr] Ethereum. “ZK-Rollups”. `https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/`.