Tight Security Bounds for Micali's SNARGs

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Eylon Yogev
eylony@gmail.com
BU and TAU

August 29, 2021

Abstract

Succinct non-interactive arguments (SNARGs) in the random oracle model (ROM) have several attractive features: they are plausibly post-quantum; they can be heuristically instantiated via lightweight cryptography; and they have a transparent (public-coin) parameter setup.

The canonical construction of a SNARG in the ROM is due to Micali (FOCS 1994), who showed how to use a random oracle to compile any probabilistically checkable proof (PCP) with sufficiently-small soundness error into a corresponding SNARG. Yet, while Micali's construction is a seminal result, it has received little attention in terms of analysis in the past 25 years.

In this paper, we observe that prior analyses of the Micali construction are not tight and then present a new analysis that achieves tight security bounds. Our result enables reducing the random oracle's output size, and obtain corresponding savings in concrete argument size.

Departing from prior work, our approach relies on precisely quantifying the cost for an attacker to find several collisions and inversions in the random oracle, and proving that any PCP with small soundness error withstands attackers that succeed in finding a small number of collisions and inversions in a certain tree-based information-theoretic game.

Keywords: succinct arguments; random oracle; probabilistically checkable proofs

Contents

1 Introduction

Succinct non-interactive arguments (SNARG) are cryptographic proofs for non-deterministic languages whose size is sublinear in the witness size. In the last decade, SNARGs have become a fundamental cryptographic primitive with various applications in the real world. In this paper, we study the classical SNARG construction of Micali [Mic00], which achieves unconditional security in the random oracle model (ROM).

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 argument 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 prover uses the random oracle to derive randomness for the PCP verifier's queries; finally, the 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 argument 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.

Security of the Micali construction. A SNARG in the ROM is (t, ϵ)-*secure* if every all-powerful malicious prover that makes at most t queries to the random oracle can convince a verifier of a false statement with probability at most ϵ. The probability is taken over the choice of random oracle, whose output size λ is implicitly a function of the chosen query bound t and soundness error ϵ.

Prior work ([Mic00; Val08; BCS16], see Section 1.3) shows that, if the underlying PCP has soundness error ε_{PCP}, then the Micali construction has a soundness error that can be bounded by

$$\underbrace{t \cdot \varepsilon_{\text{PCP}}}_{\substack{\text{attacking the} \\ \text{PCP}}} + \underbrace{4 \cdot \frac{t^2}{2^\lambda}}_{\substack{\text{attacking the} \\ \text{random oracle}}} .$$

This expression can be intuitively explained (up to constants): the term $t \cdot \varepsilon_{\text{PCP}}$ bounds the probability of a cheating prover fooling the PCP after t trials, and the term $4 \cdot \frac{t^2}{2^\lambda}$ bounds the probability that prover cheats in the commitment part by either inverting the oracle on given values or finding collisions in the random oracle (the squared term comes from the birthday paradox).

This tells us how to set parameters for security: the Micali construction is (t, ϵ)-secure, e.g., if each term is bounded by $\frac{\epsilon}{2}$. This yields two requirements: (i) $\varepsilon_{\text{PCP}} \leq \frac{1}{2}\frac{\epsilon}{t}$ (the PCP has small-enough soundness error); and (ii) $\lambda \geq \log(8\frac{t^2}{\epsilon})$ (the random oracle has large-enough output size).

Our question. While it is not so hard to see that the expression $t \cdot \varepsilon_{\text{PCP}}$ is necessary in the soundness error, it is not clear if the expression $4 \cdot \frac{t^2}{2^\lambda}$ is necessary as well. (E.g., can one attack the Micali construction given any collision in the random oracle?) This leads us to ask:

Is the soundness expression for Micali's construction optimal?

A negative answer would be excellent news because improving the soundness expression for the Micali construction immediately yields improvements in the argument size (the main efficiency measure for a SNARG), and in the time complexity of the argument prover and argument verifier. In turn, a better understanding of the Micali construction is likely to lead to improved efficiency for SNARGs in the ROM of practical interest. Indeed, while the Micali construction itself is not

used in practice since PCPs with good concrete efficiency are not known, a generalization of it is. Namely, the BCS construction [BCS16] transforms public-coin interactive oracle proofs (IOPs), a multi-round extension of PCPs with much better efficiency, into SNARGs.

The Micali and BCS constructions, being SNARGs in the ROM, have attractive features: by heuristically instantiating the random oracle with a cryptographic hash function, one obtains an implementation that is 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). Moreover, they both are plausibly post-quantum secure [CMS19].

Asking the right question. The answer to our question depends, perhaps surprisingly, on fine details of the Micali construction that are generally overlooked in the literature.

For example, consider a PCP with a binary alphabet and suppose that we build the Merkle tree for this PCP by placing each bit of the PCP string in a different leaf of the Merkle tree, padding to the random oracle's input size.[1] In this case a cheating prover could conduct the following attack: find a single collision in a leaf between the symbols 0 and 1, and re-use this collision in all leaves of the tree; then answer each PCP query with 0 or 1 as needed to make the PCP verifier accept. This attack will always fool the argument verifier, and tells us that the prior analysis is tight for this flavor of the Micali construction ($t^2/2^\lambda$ is roughly the probability of finding one collision).

Nevertheless, the foregoing attack can be trivially avoided by adding an index to each leaf, preventing the re-use of one leaf collision across different leaves. This is a form of *domain separation*, whereby one derives multiple random oracles from a single random oracle by adding pre-defined prefixes to queries. Domain separation was used, e.g., by Micali in [Mic00] to separate queries for constructing the Merkle tree and queries for deriving the PCP randomness (i.e., the randomness used by the PCP verifier to select queries to the PCP string). More generally, as we wish to exclude easily preventable attacks, in this paper we consider what we call the *domain-separated* flavor of the Micali construction (see Section 3.3): (i) tree queries are domain-separated from PCP randomness queries as in [Mic00]; and, moreover, (ii) each tree query (not only for the leaf layer) is further domain-separated via a prefix specifying its location in the Merkle tree. At this point, it is not clear if the prior analysis is tight for the domain-separated flavor of the Micali construction.

The goal of this paper is to study this question.

1.1 Our contributions

In this paper, we give a negative answer to our question, by improving the soundness expression of the (domain-separated) Micali construction. In fact, we settle the security bounds for the Micali construction, up to low-order terms by giving nearly matching lower bounds. We prove the following:

Theorem 1 (informal)**.** *The Micali construction (with domain separation), when instantiated with a PCP with proof length* l *over alphabet* Σ *and soundness error* $\varepsilon_{\mathrm{PCP}}$ *and with a random oracle with output size* λ*, has soundness error*

$$t \cdot \varepsilon_{\mathrm{PCP}} + C \cdot \frac{t}{2^\lambda} \quad \textit{with} \quad C \leq 12 \cdot \mathsf{l} \cdot \log |\Sigma|$$

against t*-query adversaries, provided that* $\lambda \geq 2 \log t + 6$.

[1] Placing each symbol in a different leaf, padded with a freshly-sampled salt, leads to a zero-knowledge SNARG if the underlying PCP is honest-verifier zero-knowledge [IMSX15; BCS16].

The expression above is smaller than the previously known bound ($t \cdot \varepsilon_{\text{PCP}} + 4 \cdot \frac{t^2}{2^\lambda}$): using the same notation, previously we knew that $C \leq 4 \cdot t$, and our result reduces this bound to $C \leq 12 \cdot \mathsf{l} \cdot \log |\Sigma|$, which is significantly better for any reasonable parameter regime. This directly leads to improved security parameters: for a given query bound t and soundness error ϵ, to achieve (t, ϵ)-security we can set λ (the output size of the random oracle) to be smaller than what was previously required (i.e., $\lambda \geq \log(8\frac{t^2}{\epsilon})$), while leaving the PCP soundness error as before (i.e., $\varepsilon_{\text{PCP}} \leq \frac{1}{2}\frac{\epsilon}{t}$).

Corollary 2. *The Micali construction is (t, ϵ)-secure when instantiated with:*

- *a PCP with soundness error $\varepsilon_{\text{PCP}} \leq \frac{1}{2}\frac{\epsilon}{t}$, and*
- *a random oracle with output size $\lambda \geq \max\{2 \log t + 6\,,\ \log(t/\epsilon) + \log(\mathsf{l} \cdot \log |\Sigma|) + 5\}$.*

Matching lower bound. The security bound in Theorem 1 depends on the PCP's length l and alphabet Σ, in addition to the PCP's soundness error ε_{PCP}. We show that, perhaps surprisingly, the dependency on these parameters is inherent by proving a nearly matching lower bound on the soundness error of the Micali construction. In Section 8 we exhibit attacks showing that there exists a PCP with q queries for which the soundness error of the Micali construction is

$$\Omega\left(t \cdot \varepsilon_{\text{PCP}} + C \cdot \frac{t}{2^\lambda}\right) \quad \text{with} \quad C \geq \frac{\mathsf{l} \cdot \log |\Sigma|}{\mathsf{q}^2} \ .$$

Tighter upper bound. To ease the presentation, the expressions given in our upper bounds are simplifications of counterparts given in Theorem 7.1 and Corollary 7.2. These latter are closer to the lower bounds, and provide additional savings when setting concrete values in practice for a given security level (the desired query bound t and desired soundness error ϵ).

1.2 Concrete improvements in argument size

We have obtained a tight analysis of the Micali construction, leading to a smaller output size λ for the random oracle. This, in turn, yields corresponding savings in argument size for the Micali construction. We demonstrate this in Table 1 via argument sizes computed for an illustrative choice of PCP for different targets of security (for all $(t, \epsilon) \in \{2^{96}, 2^{128}, 2^{160}\} \times \{2^{-96}, 2^{-128}, 2^{-160}\}$).

Specifically, we consider the Micali construction applied to the amplification of an assumed "base" PCP with soundness error $1/2$, query complexity 3, and proof length 2^{30} over a binary alphabet (the Merkle tree thus has 30 levels). By repeating the PCP verifier $\log(1/\varepsilon_{\text{PCP}}) \geq \log \frac{1}{2}\frac{t}{\epsilon}$ times, the amplified PCP has soundness error ε_{PCP} and $3 \log \frac{1}{2}\frac{t}{\epsilon}$ queries. We then compute the argument size obtained by applying the Micali construction to this PCP while setting the output size of the random oracle to the value in Corollary 2 (more precisely, to its refinement in Corollary 7.2). See Section 3.3 for an overview of the Micali construction and how its argument size is computed.

Concrete improvements for IOP-based SNARGs. One might ask whether our fine-grained analysis of the Micali construction, which is based on PCPs, could be carried out for [BCS16], a SNARG construction based on IOPs, which are more efficient, that is used in practice. The answer is yes (this requires extending our techniques from PCPs to IOPs) but we leave this for future work. We deliberately limited the scope of our paper to Micali's construction because we wanted to illustrate our new ideas in a straightforward way. Looking beyond [Mic00; BCS16] (i.e., SNARGs), we believe that the ideas in our paper will help establish tight security bounds for other primitives in the ROM, such as hash-based signatures (where again Merkle trees often play a significant role).

$\log t$ \ $-\log \epsilon$	96	128	160
96	$\frac{389\,\text{KB}}{297\,\text{KB}} \approx 1.31\times$	$\frac{498\,\text{KB}}{389\,\text{KB}} \approx 1.28\times$	$\frac{618\,\text{KB}}{496\,\text{KB}} \approx 1.25\times$
128	$\frac{547\,\text{KB}}{405\,\text{KB}} \approx 1.35\times$	$\frac{675\,\text{KB}}{496\,\text{KB}} \approx 1.36\times$	$\frac{812\,\text{KB}}{615\,\text{KB}} \approx 1.32\times$
160	$\frac{730\,\text{KB}}{569\,\text{KB}} \approx 1.28\times$	$\frac{875\,\text{KB}}{635\,\text{KB}} \approx 1.38\times$	$\frac{1033\,\text{KB}}{746\,\text{KB}} \approx 1.38\times$

Table 1: Argument sizes for the Micali construction applied to an illustrative PCP for different settings of (t, ϵ)-security, comparing the size achieved by the prior analysis (in red) and our analysis (in blue).

1.3 Related work

While the Micali construction is a seminal result (it is the first SNARG construction and arguably the simplest one to intuitively understand), it has received little attention in terms of analysis.

The analysis given by Micali [Mic00] considers the special case of PCPs over a binary alphabet with soundness error $\varepsilon_{\text{PCP}} \leq 2^{-\lambda}$, and establishes (t, ϵ)-security with $t \leq 2^{\lambda/8}$ and $\epsilon \leq 2^{-\lambda/16}$.

Subsequently, Valiant [Val08] again considers the special case of a PCP with soundness error $\varepsilon_{\text{PCP}} \leq 2^{-\lambda}$, but contributes a new approach based on straightline extraction that establishes soundness (and, in fact, also proof of knowledge if the underlying PCP is a proof of knowledge) for PCPs over any alphabet. The analysis establishes (t, ϵ)-security with $t \leq 2^{\lambda/4}$ and $\epsilon \leq 2^{-\lambda/8}$.

Finally, the expression $t \cdot \varepsilon_{\text{PCP}} + 4 \cdot \frac{t^2}{2^\lambda}$ stated above, which holds for any PCP and yields relatively good security settings, is due to Ben-Sasson, Chiesa, and Spooner [BCS16], who build on the straightline extraction approach of Valiant to obtain results beyond the Micali construction. Specifically, they prove that any IOP with state-restoration soundness error $\varepsilon_{\text{sr}}(\cdot)$ can be compiled into a SNARG with soundness error $\varepsilon_{\text{sr}}(t) + 4 \cdot \frac{t^2}{2^\lambda}$ against t-query adversaries. This BCS construction, when restricted to a PCP, recovers the Micali construction, and yields the previous expression because the state-restoration soundness error of a PCP is at most $\varepsilon_{\text{sr}}(t) \leq t \cdot \varepsilon_{\text{PCP}}$.

Subsequent work. Chiesa and Yogev [CY21] gave a new SNARG construction in the random oracle model that achieves a smaller argument size than (any analysis of) the Micali construction. For (t, ϵ)-security, the CY construction has argument size $\tilde{O}(\log(t/\epsilon) \cdot \log t)$, while the Micali construction has argument size $\Omega(\log^2(t/\epsilon))$. The CY construction is achieved by identifying the limitations of the Micali construction thanks to the tight security analysis in this work, and then modifying the construction to overcome these limitations. Furthermore, the analysis in [CY21] is based on a framework inspired by the one developed in this work and adapted to the CY construction.

2 Techniques

We summarize the main ideas behind our main result (Theorem 1). In Section 2.1 we review the prior analysis of the Micali construction and in Section 2.2 explain why that analysis is not tight. Then in Sections 2.3 to 2.6 we describe our analysis, whose differences are displayed in Figure 1.

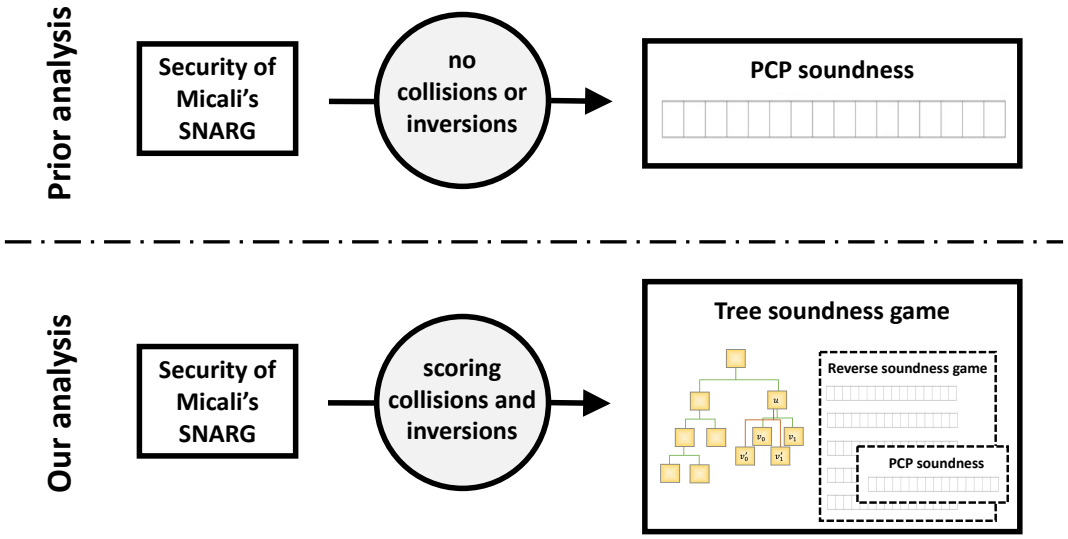

Figure 1: The prior analysis of the Micali construction conditions on the event that no collisions or inversions are found, and directly reduces soundness of the SNARG to soundness of the PCP. In contrast, we carefully assign scores to collisions and inversions, which enables us to reduce soundness of the SNARG to soundness in an information-theoretic *tree game*. Then we prove that soundness in this game reduces to soundness of the PCP, by relying on another information-theoretic game.

2.1 Prior analysis of the Micali construction

We begin by reviewing the prior security analysis of the SNARG construction of Micali. Let ε_{PCP} be the soundness error of the PCP used in the construction, and let λ be the output size of the random oracle. We wish to bound the probability that a t-query cheating prover makes the SNARG verifier accept. Intuitively, the cheating prover may try to attack the PCP or the random oracle.

First, consider a cheating prover that tries to attack the random oracle, that is, tries to find collisions or inversions. Recall that a collision consists of two or more distinct inputs that map to the same output, and an inversion is an input that maps to a target value or list of values (that were not answers to prior queries). So let E be the event that the cheating prover finds at least one collision or inversion. By the birthday bound, one can show that

$$\Pr[E] = O\left(\frac{t^2}{2^\lambda}\right) \quad .$$

Next condition on the event E not occurring, i.e., attacking the random oracle does not succeed. In this case, one can show that the cheating prover's only strategy is to attack the PCP as follows:

commit to a PCP string, derive randomness for the PCP verifier from the resulting Merkle root, and check if the PCP verifier accepts; if not, then try again with new randomness.[2] One can model this via a simple t-round game: in each round $i \in [t]$ the attacker outputs a PCP string Π_i and the game replies with fresh PCP randomness ρ_i; the attacker wins if there is $i \in [t]$ such that Π_i convinces the PCP verifier on randomness ρ_i. A union bound shows that the winning probability in this game is at most $t \cdot \varepsilon_{\text{PCP}}$. Thus, this bound is (relatively) tight (assuming ε_{PCP} is a tight bound on the soundness of the PCP).

Combining the above cases, we can bound the probability that the cheating prover makes the verifier in the Micali construction accept:

$$\Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\right] \leq \Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\;\middle|\;\overline{E}\right] + \Pr[E] \leq t \cdot \varepsilon_{\text{PCP}} + O\left(\frac{t^2}{2^\lambda}\right) \quad .$$

2.2 The prior analysis is not tight

The starting point of our work is the observation that the prior security analysis is *not* tight for the domain-separated flavor of the Micali construction (discussed in Sections 1 and 3.3). While the term $t \cdot \varepsilon_{\text{PCP}}$ in the expression is essentially tight, the term $t^2/2^\lambda$ is not. In other words, while ruling out the event E (the cheating prover finds a collision or inversion) is sufficient to show security, it is not a *necessary* condition for security. We illustrate this via a simple example about collisions.

Collisions are not too harmful. Suppose that a cheating prover finds a collision in the random oracle, for a specific location. This enables the cheating prover to commit to *two* PCP strings instead of one, derive PCP randomness from their common Merkle root, and choose which PCP string to use for answering the PCP verifier's queries. While this increases the winning probability in (a modification of) the simple t-round game described above, a union bound shows that the probability increases at most by a factor of 2. At the same time, the probability of finding a collision is much smaller than $1/2$, so overall this is not a beneficial strategy for a cheating prover.

We could alternatively consider a cheating prover that finds many collisions. For instance, suppose that the PCP is over a binary alphabet and the cheating prover has found a collision between 0 and 1 for every leaf of the Merkle tree. This enables the cheating prover to compute a Merkle root and derive corresponding PCP randomness without actually committing to any PCP string, because every leaf can be opened to a 0 or a 1. Thus the cheating prover can always provide satisfying answers to the PCP verifier. But, again, this strategy is also not problematic because the probability of finding so many collisions is very small (e.g., much smaller $\Pr[E]$).

The above considerations give intuition about why collisions are not that harmful.

What about inversions? Unlike collisions, even a single inversion lets a cheating prover win with probability one. Here is the attack: derive the PCP randomness for some arbitrary choice of Merkle root; find a satisfying PCP string for this PCP randomness; and then hope that the Merkle tree for this PCP string hashes to the chosen Merkle root. This would be considered a single "inversion".

The probability of this happening in any of the t queries by the cheating prover is at most $t \cdot 2^{-\lambda}$, which is small enough for us to afford to simply rule out and pay this term as an additive loss in the soundness error bound. Hence, it is tempting to do this, and then focus only on collisions.

That, however, would be incorrect because, perhaps surprisingly, the aforementioned attack is *not* the only way to exploit inversions, as we now discuss. That attack relies on a "strong inversion":

[2] The new randomness could be derived by changing a salt if present in the construction, or by changing just one location of the PCP string and then re-deriving a different Merkle root, leading to new PCP randomness.

the cheating prover wanted to invert a specific single Merkle root (given y, find x such that the random oracle maps x to y). However, a cheating prover could instead seek a "weak inversion": invert any one element out of a long list. For example, the cheating prover could gather a $t/2$-size list of Merkle roots (for which PCP randomness has already been derived via 1 query each), and use the remaining $t/2$ queries to try to invert any one of them. The probability that one query inverts any value in the list is roughly $t \cdot 2^{-\lambda}$, so the probability of inverting via any of the queries is roughly $t^2 \cdot 2^{-\lambda}$. This is again a term that we cannot afford to simply rule out, so we must somehow deal with weak inversions if we are to improve the security analysis of the Micali construction.

Weak inversions are not too harmful. The aforementioned considerations lead to the following non-trivial attack, which exemplifies the limits of possible strategies via inversions. The cheating prover randomly samples many Merkle roots and, for each Merkle root, derives the corresponding PCP randomness; finds a PCP string that is accepting for as many of the PCP randomness strings as possible; computes the Merkle tree for this PCP string; and tries to connect this Merkle tree to one of the Merkle roots via a weak inversion. If the cheating prover succeeds, and the PCP randomness was one of the convincing ones for this PCP string, then the cheating prover has won. If it fails, the cheating prover can try again with another PCP string (and so on).

At this point it is not clear if the success probability of this attack is something we can afford. Showing that this attack cannot succeed with high probability will require introducing new tools.

Mixing strategies. Once we do not to rule out the event E then we must consider the impact on security of strategies that may go well beyond exploiting a single collision or a single weak inversion. An adversary might be able to find many pairs of collisions or even multi-collisions of more than two elements. The cheating prover could adopt a strategy that mixes between finding collisions, finding (weak) inversions, and attacking the PCP. For example, finding some collisions leads to improving the success probability in the above inversion attack, as now the cheating prover is not committed to a single string in every trial. Moreover, the adversary may choose its strategy adaptively, based on each response from the random oracle.

In sum, the main challenge in not ruling out E is to analyze how *any combination* of these strategies will impact the security of the construction. To handle this, we define a more involved PCP soundness game, where the attacker has a budget for collisions and a budget for (weak) inversions and can perform arbitrary strategies with collisions and inversions up to those budgets. We show that any cheating prover that succeeds in fooling the argument verifier will also succeed (with similar probability) in this PCP soundness game. We describe the game next.

2.3 A tree soundness game

In order to obtain a tight security analysis of the Micali construction, we introduce an intermediate information-theoretic game, which we call *tree soundness game*, that enables us to model the effects of attacks against the Micali construction. In order to motivate the description of the game, first we describe a simplified game with no collisions or inversions (Section 2.3.1), then describe how the simplified game leads to the prior analysis of the Micali construction (Section 2.3.2), and finally describe how to augment the game with features that model collisions and inversions (Section 2.3.3). The formal description of the tree soundness game can be found in Section 5.

The intermediate game then leaves us with two tasks. First, reduce the security of the Micali construction to winning the tree soundness game (without paying for the birthday bound term). Second, reduce winning the tree soundness game to breaking the standard soundness of a PCP

(which is not clear anymore). We will discuss the second step in Section 2.4, and the first step in Sections 2.5 and 2.6.

2.3.1 The game with no collisions or inversions

The tree soundness game has several inputs: a PCP verifier $\mathbf{V}$; an instance $\mathbb{x}$; an integer λ (modeling the random oracle's output size); a malicious prover $\tilde{\mathbf{P}}$ to play the game; and a query budget $t \in \mathbb{N}$. We denote this game by $\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t)$.

The graph G. The game is played on a graph $G = (V, E)$ that represents the Merkle trees constructed by the malicious prover so far. The graph G contains all possible vertices, and actions by the malicious prover add edges to the graph.

Letting d be the height of the Merkle tree, vertices in G are the union $V := V_0 \cup V_1 \cup \cdots \cup V_d$ where V_i are the vertices of level i of the tree: for every $i \in \{0, 1, \ldots, d-1\}$, $V_i := \{(i, j, h) : j \in [2^i], h \in \{0,1\}^\lambda\}$ 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:

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

Playing the game. The game starts with the graph G empty ($E = \emptyset$), and proceeds in rounds. In each round, provided there is enough query budget left, the malicious prover 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 root. 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 updated $t \leftarrow t - 1$. However, the prover is not allowed to add any edge he wants. In particular, at least for now, he cannot add edges that correspond to collisions or inversions. Namely, if E already contains an edge of the form (u, v_0', v_1'), then (u, v_0, v_1) would form a *collision* with that edge, and so in this case (u, v_0, v_1) is not added to E. Moreover, if E already contains an edge of the form (u', v_0', v_1') with $v_0' = u$ or $v_1' = u$, then (u, v_0, v_1) would form an *inversion* with that edge, and so in this case (u, v_0, v_1) is not added to E. (Note that the game would have allowed adding these two edges in reverse order, though, as that would not have been an inversion.)

- *Deriving randomness.* A root is a vertex $v_{\mathsf{rt}} \in V_0$, which has the form $(0, 0, h)$ for some $h \in \{0,1\}^\lambda$. (The root level is 0 and has a single vertex, at position 0.) When the prover submits v_{rt}, the game samples new PCP randomness ρ, and the pair (v_{rt}, ρ) is added to a mapping Roots. (The prover is not allowed to submit a root v_{rt} that already appears in the mapping Roots.) This costs a unit of the query budget, so when this happens the game updates $t \leftarrow t - 1$.

Winning the game. When it decides to stop playing the game, the prover outputs a root vertex $v_{\mathsf{rt}} \in V_0$ 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 proof string Π when using the randomness associated to v_{rt}. That is, $\mathbf{V}^{\Pi}(\mathbb{x}; \rho) = 1$ for $\rho := \mathsf{Roots}[v_{\mathsf{rt}}]$. (If Roots has no randomness for v_{rt} then the prover loses.)
- The PCP string Π is consistent with v_{rt} in the graph G. That is, if the PCP verifier queries location j of Π, then the leaf $u = (d, j, \Pi[j]) \in V_d$ is connected to the root $v_{\mathsf{rt}} \in V_0$ in G.

We denote by $\varepsilon_{\text{tree}}(t)$ the maximum winning probability in the tree soundness game by any malicious prover with query budget t. See Figure 2 for an example of such a tree.

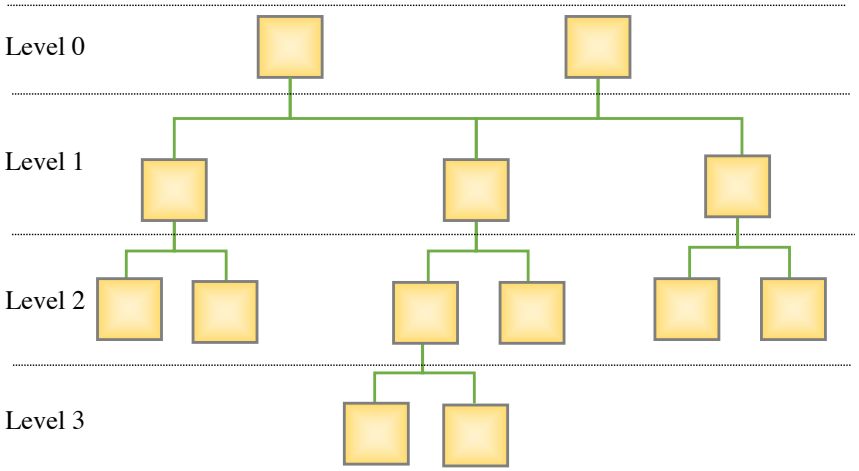

Figure 2: An example of a possible state of the graph in the tree soundness game. Level 0 happens to contain two root vertices, level 1 contains three vertices, and so on. A hyper-edge containing three vertices is drawn using the green lines that are attached to three vertices.

2.3.2 Recovering the prior analysis

The game described so far, which excludes collisions and inversions, enables us to recover the prior analysis of the Micali construction. Conditioned on not finding a collision or an inversion in the random oracle, any winning strategy of a t-query cheating argument prover can be translated into a winning strategy for a prover in the tree soundness game with query budget t. This tells us that the soundness error of the Micali construction can be bounded from above as follows:

$$\epsilon(\lambda, t) \leq \varepsilon_{\text{tree}}(t) + O(t^2/2^{\lambda}) \ .$$

In turn, if the PCP has soundness error ε_{PCP}, then the winning probability in the tree soundness game is at most $t \cdot \varepsilon_{\text{PCP}}$. This is the expression of the prior analysis of the Micali construction.

We wish, however, to analyze the Micali construction in a more fine-grained way, by studying the impact of a few collisions and inversions. That requires extending the tree soundness game.

2.3.3 Adding collisions and inversions

We describe how to extend the tree soundness game by adding *collision edges* and *inversion edges*, up to respective collision and inversion budgets. We first discuss collisions and then inversions.

Easy: supporting collision. We introduce a *collision budget* t_{col}. Whenever the prover adds to the graph G an edge that collides with an existing edge (a new edge (u, v_0, v_1) for which an edge (u, v'_0, v'_1) is already in G), the game charges the prover 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 budget versatile in that, for example, a budget of 2 can be used to create two 2-wise collisions or one 3-wise collision.

If the collision budget is large enough (as large as the proof length), then for some PCPs the prover can win with probability 1 (see the collision attack in Section 2.2). However, our analysis will say that, for this case, obtaining such a large collision budget happens with very small probability. Unlike the query budget, the collision (and inversion) budget is scarce. See Figure 3 for an illustration.

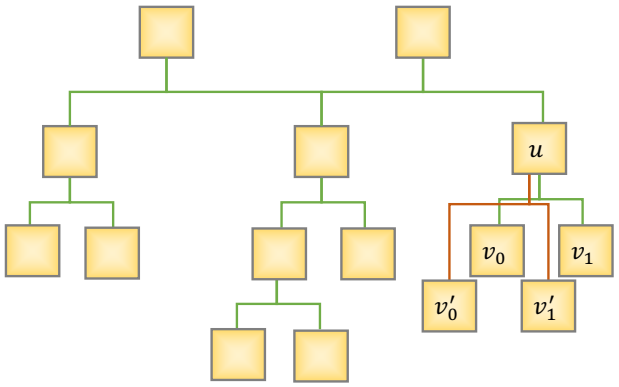

Figure 3: An example of a possible state of the graph in the tree soundness game, with a collision: the edge (u, v_0, v_1) collides with the edge (u, v'_0, v'_1).

Hard(er): supporting inversions. We introduce, similarly to the case of collisions, an *inversion budget* t_{inv}. Supporting inversions, though, is significantly more challenging. The cheating prover can win with one strong inversion (see Section 2.2), which happens with small probability. So we cannot simply let the adversary in the tree soundness game add arbitrary inversion edges to the graph, as that would enable trivially winning even with $t_{\text{inv}} = 1$. Instead, we designate t_{inv} for *weak inversions* (inversions for any value within a large set rather than for a specific value).

In the game described so far the prover adds edges to the graph by specifying the desired edge (u, v_0, v_1). In contrast, when the prover wishes to perform a weak inversion, the prover cannot fully specify the vertex $u = (i, j, h)$ of the edge: the prover provides the location i and j for u, but the game samples the string $h \in \{0,1\}^\lambda$ instead. Specifically, the game samples h at random from the set H of "possible targets for inversion" (all h such that (i', j', h) is a vertex in an edge in the graph).

How much should a weak inversion cost? Note that H might contain a single vertex, which would make this a strong inversion, and thus should be very costly for the prover. So one option would be to charge the prover according to the size of H (the smaller H is, the higher the cost). Yet we do not see how to set this cost, as a function of H, that would suffice for the proof. Hence, instead, we charge the prover differently. The cost for any (weak) inversion is always a single unit of t_{inv}, regardless of the size of H. But the game adds the edge to the graph only with probability $\frac{|H|}{2t}$. (The constant "2" appears since each edge added to the graph contributes two vertices that

could be inverted, and so after t edges the maximal size of H is $2t$.) If H is small, then there is a small probability for the edge to be added, making the expected cost to get an inversion high. The maximal size of H is $2t$, and in this case the edge is added with probability one.

2.4 PCPs are secure against collisions and inversions

We prove that *any* PCP with sufficiently small soundness error is secure against collisions and inversions, i.e., has small tree soundness error. The loss in soundness depends on the various budgets provided to the tree soundness game $(t, t_{\text{col}}, t_{\text{inv}})$, as stated in the following theorem.

Theorem 2.1. *Let $(\mathbf{P}, \mathbf{V})$ be a PCP for a relation R with soundness error ε_{PCP} and proof length l over an alphabet Σ. Then $(\mathbf{P}, \mathbf{V})$ has tree soundness error*

$$\varepsilon_{\text{tree}}(t, t_{\text{col}}, t_{\text{inv}}) = O\left(2^{t_{\text{col}}} \cdot t \cdot \varepsilon_{\text{PCP}} + 2^{t_{\text{col}}} \cdot t_{\text{inv}} \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{t}\right) \ .$$

The above formulation is a simplification of our actual theorem (Theorem 5.7), which gives a more precise bound that is useful for concrete efficiency.

The tree soundness game is relatively complicated, as an adversary may employ many different strategies that demand for different analyses. Our proof of Theorem 2.1 relies on a simpler game, which we call *reverse soundness game*, that focuses almost solely on weak inversion, and omits tree structure and collisions. (See Figure 1.) In light of this, the rest of this section is organized as follows: (i) We summarize a simplification of the reverse soundness game and discuss how reverse soundness reduces to regular PCP soundness (Lemma 2.2); technical details are in Section 4. (ii) We discuss how tree soundness reduces to reverse soundness (Lemma 2.3); technical details are in Section 5. These two steps directly lead to Theorem 2.1 above. We then conclude this section with a brief discussion of which additional features we rely on in the non-simplified reverse soundness game.

Reverse soundness. The reverse soundness game has several inputs: a PCP verifier $\mathbf{V}$; an instance $\mathbb{x}$; a parameter $K \in \mathbb{N}$; and a malicious prover $\tilde{\mathbf{P}}$ to play the game. For simplicity, here we describe a simplified version of the reverse soundness game that captures its main idea. Later on we describe how to go to from this simplification to the game that we use in the technical sections.

The simplified game works as follows: (1) the game draws K samples of PCP randomness $\rho_1, \ldots, \rho_K$ and gives them to $\tilde{\mathbf{P}}$; (2) $\tilde{\mathbf{P}}$ outputs a single PCP string Π; (3) the game samples a random index $i \in [K]$ and $\tilde{\mathbf{P}}$ wins if and only if $\mathbf{V}^{\Pi}(\mathbb{x}; \rho_i) = 1$ (Π convinces the PCP verifier with randomness ρ_i). Intuitively, the K samples are modeling randomness derived from K Merkle roots, and the subsequent choice of a PCP string to be tested with a random choice among the samples is modeling a weak inversion.

This game becomes "harder" as K increases. At one extreme where $K = 1$, the game is easy to win: the cheating prover can pick any PCP string Π that makes the single PCP randomness ρ_1 accept. At the other extreme where K equals the number of possible random strings, the (simplified) game approximately equals the regular PCP soundness game. We are interested in a regime where K is in between, which we think of as "reverse soundness": the cheating prover chooses the PCP string *after* seeing samples of PCP randomness (and is then tested via one of the samples at random).

Standard soundness implies reverse soundness. We prove that reverse soundness reduces to standard soundness of a PCP, as stated in the following lemma:

Lemma 2.2. *Let $(\mathbf{P}, \mathbf{V})$ be a PCP for a relation R with soundness error ε_{PCP} and proof length l over an alphabet Σ. If $K \cdot \varepsilon_{\text{PCP}} \leq 1/20$ then $(\mathbf{P}, \mathbf{V})$ has reverse soundness error*

$$\varepsilon_{\text{rev}}(K) = O\left(\frac{\mathsf{l} \cdot \log|\Sigma|}{K}\right) \quad .$$

The precise statement and its proof (for the non-simplified game) are given in Lemma 4.4. Next we provide a summary of the proof for the simplified reverse soundness game described above.

There is a simple optimal strategy for the reverse soundness game: given the K samples of PCP randomness, enumerate over all possible PCP strings, and choose the one that maximizes the probability of winning (over a random choice of sample from within the list of samples). This strategy is not efficient, but that is fine as we are not bounding the running time of adversaries.

Hence, to bound the reverse soundness error, we consider an infinite sum over an intermediate parameter $c \in \mathbb{N}$. For each c we bound the probability, over $\rho_1, \ldots, \rho_K$, that there exists a PCP string that wins against any c of the samples. The resulting geometric series converges to the claimed upper bound.

In more detail, winning against c specific samples happens with probability $\varepsilon_{\text{PCP}}^c$, so a union bound over all subsets of size c yields the following:

$$\Pr_{\rho_1,\ldots,\rho_K}\left[\exists\,\Pi : \Pr_{i\in[K]}[\mathbf{V}^{\Pi}(\mathbb{x};\rho_i)=1] \geq \frac{c}{K}\right] \leq |\Sigma|^{\mathsf{l}} \cdot \binom{K}{c} \cdot \varepsilon_{\text{PCP}}^c \leq 2^{\mathsf{l}\cdot\log|\Sigma| + c\cdot\log(K\cdot\varepsilon_{\text{PCP}})} \quad .$$

Next, fixing any cheating prover $\tilde{\mathbf{P}}$ and letting $\mathcal{G}_{\text{rev}}(\mathbf{V}, \mathbb{x}, K, \tilde{\mathbf{P}}) = 1$ be the output of the game for an instance $\mathbb{x}$ and parameter K, we conclude that:

$$\Pr\left[\mathcal{G}_{\text{rev}}(\mathbf{V}, \mathbb{x}, K, \tilde{\mathbf{P}}) = 1\right] \leq \sum_{c=1}^{\infty} \Pr_{\rho_1,\ldots,\rho_K}\left[\exists\,\Pi : \Pr_{i\in[K]}[\mathbf{V}^{\Pi}(\mathbb{x};\rho_i)=1] = \frac{c}{K}\right] \cdot \frac{c}{K} \leq O\left(\frac{\mathsf{l}\cdot\log|\Sigma|}{K}\right) \quad .$$

Reverse soundness implies tree soundness. We are left to prove that tree soundness reduces to reverse soundness, as stated in the following lemma:

Lemma 2.3. *If $(\mathbf{P}, \mathbf{V})$ is a PCP for a relation R with soundness error $\varepsilon_{\text{PCP}}(\mathbb{x})$ and reverse soundness error $\varepsilon_{\text{rev}}(\mathbb{x}, K)$, then it has tree soundness error*

$$\varepsilon_{\text{tree}}(t, t_{\text{col}}, t_{\text{inv}}) = O\left(2^{t_{\text{col}}} \cdot t \cdot \varepsilon_{\text{PCP}} + 2^{t_{\text{col}}} \cdot t_{\text{inv}} \cdot \varepsilon_{\text{rev}}(t)\right) \quad .$$

The precise statement and its proof (for the non-simplified game) are given in Lemma 5.6.

The expression above can be interpreted in a meaningful way.

- The first term corresponds to a cheating prover that tries to win the standard PCP soundness game over and over, aided by collisions. Given a budget of t_{col} collisions, he can commit to $2^{t_{\text{col}}}$ PCP strings before submitting a Merkle root and getting PCP randomness. Each trial's winning probability is $2^{t_{\text{col}}} \cdot \varepsilon_{\text{PCP}}$, and across t trials the winning probability becomes $O(2^{t_{\text{col}}} \cdot t \cdot \varepsilon_{\text{PCP}})$.
- The second term corresponds to a cheating prover that tries to win using an inversion. He creates t Merkle roots, commits to $2^{t_{\text{col}}}$ strings, and tries to invert. He does this t_{inv} times overall and wins with probability roughly $O(2^{t_{\text{col}}} \cdot t_{\text{inv}} \cdot \varepsilon_{\text{rev}}(t))$.

On the non-simplified game. We show that we can reduce any prover that plays the tree soundness game to one that plays the reverse soundness game (with small losses in the success probability). However, the reverse soundness game described above is too simple to capture the wide range of strategies possible in the tree soundness game, and some modifications are required.

Consider a cheating prover that performs a weak inversion on an intermediate vertex in the tree. Instead of creating a long list of isolated vertices, he can create the same list where each vertex is already connected to a partial tree, representing a partial proof string Π_i. Then, when he submits the final PCP string Π, the two strings will be *merged* in a way that depends on the vertex's location in the tree. The inversion creates a collision at the inversion vertex, which lets the prover use a combination of Π and Π_i, depending on which pre-image he chooses for the collision vertex.

To capture these strategies, we rely on a merging function that takes as input two partial PCP strings $\Pi_0, \Pi_1 \in (\Sigma \cup \{\perp\})^{\mathsf{l}}$ and outputs another partial PCP string. The function is defined as follows:

$$\mathsf{merge}(\Pi_0, \Pi_1, b)[j] := \begin{cases} \Pi_0[j] & \text{if } \Pi_1[j] = \perp \\ \Pi_1[j] & \text{if } \Pi_0[j] = \perp \\ \Pi_b[j] & \text{otherwise} \end{cases} .$$

The reverse soundness game is then extended as follows: the prover first outputs K PCP strings $\Pi_1, \ldots, \Pi_K$; then the prover receives K samples of PCP randomness; then the prover outputs a new string Π and a bit b; the game samples $i \in [K]$ and tests the merged string $\Pi^* := \mathsf{merge}(\Pi, \Pi_i, b)$ against the i-th sample. The game that we use is generous in that it allows a larger class of strategies than what is possible in the tree soundness game. This lets us use a clean a simple definition (via the merge function) whereas a tight definition capturing the set of strategies in the tree soundness game would be significantly more involved and cumbersome.

There are other modifications needed for the game to capture all possible strategies. For example, the prover can choose the K PCP strings *adaptively* instead of committing to them in advance, as above. These extensions create additional technical challenges in the full proof.

2.5 Scoring oracle queries

The tree soundness game lets us bound the success probability of a malicious prover given specific budgets in that game. But what budgets should we use when analyzing a cheating argument prover (attacking the soundness of the SNARG)? For this, we introduce a new tool: a *scoring function* for the query trace of an algorithm in the random oracle model.

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 tree 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 tree soundness game, but the probability of achieving a corresponding high score is low, and our goal is to balance these two.

We separately define scoring functions for collisions and for inversions, as motivated below.

- *Scoring collisions.* A natural idea would be to set the collision score to equal the number of pairwise collisions in a query trace. However, this choice is not useful if a query trace contains multi-collisions. For example, a k-wise collision (k inputs that all map to the same output) would have a score of $\binom{k}{2}$ because there are this many pairwise collisions in the k-wise collision. This

does not reflect well the probability of finding $\binom{k}{2}$ pairwise collisions, which is much smaller than finding one k-wise collision.

Instead, we set the score of a k-wise collision 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 it is much more likely to see two pairwise collisions than one 3-wise collision. In this case, it is fine since two pairwise collisions yield four possible proof strings, while a 3-wise collision yields only three possible proof strings.

- *Scoring inversions.* To score inversions we simply count the number of inversions in the query trace. We now elaborate on what we consider an inversion in the query trace. Recall that, in the (domain-separated) Micali construction, queries to the random oracle designated for the 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 $x \in \{0,1\}^\lambda$ 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 $x = y$.

Informally, we show the following lemma:

Lemma 2.4. *For any t-query algorithm that queries the random oracle and every $k \in \mathbb{N}$:*

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

Further details, including precise definitions of scores and the proof of the lemma, are provided in Section 6. There the lemma statement is more involved as it considers queries to the tree oracle and to the randomness oracle separately, to get tighter bounds.

Subsequent to this work, a simple variant of Lemma 2.4 has been used to analyze a SNARG construction whose argument size improves on the argument size of the Micali construction [CY21].

2.6 Concluding the proof of Theorem 1

As reviewed in Section 2.1, the prior analysis of the Micali construction separately bounds the probability that the cheating argument prover finds any collision or (weak) inversion and then conditions the cheating argument prover on the event it finds no collisions or (weak) inversions. This means that the query trace's score is 0, which in turn translates to collision and inversion budgets that equal 0 in the tree soundness game ($t_{\text{col}} = 0$ and $t_{\text{inv}} = 0$).

In contrast, in our analysis, we consider *every possible query trace score* and also the probability that the cheating argument prover could have achieved that score (see Section 2.5). For any integer $k \in \mathbb{N}$ we consider the event of the cheating argument prover producing a query trace that has either collision score or inversion score exactly k. We show that conditioned on the cheating argument prover producing a query trace of score k, there is another cheating prover that wins the tree soundness game with the same probability and budget k (the precise statement is given in Claim 7.3). Namely,

$$\Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\;\middle|\;\text{score } k\right] \leq \varepsilon_{\text{tree}}(t,k,k) \ .$$

We consider an infinite sum over k, and for each value of k we bound the probability of the cheating argument prover getting a score of k multiplied by the maximum winning probability in the tree soundness error given budget $t_{\text{col}} = k$ and $t_{\text{inv}} = k$. This infinite sum then converges to the soundness expression stated in our main theorem, provided that $\lambda \geq 2\log t + 6$.

This approach could be over-simplified via the following equations (for simplicity here we are not careful with constants). First, using Lemma 2.4 we can conclude that the probability that either score (collisions or inversions) is bounded by the sum of the two probabilities, namely:

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

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

$$\begin{aligned}
\Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\right] &\leq \sum_{k=0}^{\infty} \Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\;\middle|\;\text{score of } k\right] \cdot \Pr[\text{score of } k] \\
&\leq \sum_{k=0}^{\infty} \varepsilon_{\text{tree}}(t,k,k) \cdot \Pr[\text{score of } k] \\
&\leq \sum_{k=0}^{\infty} O\left(\left(2^k \cdot t \cdot \varepsilon_{\text{PCP}} + 2^k \cdot k \cdot \varepsilon_{\text{rev}}(t)\right) \cdot \left(\frac{2t^2}{2^\lambda}\right)^k\right) \\
&\leq \sum_{k=0}^{\infty} O\left(\left(2^k \cdot t \cdot \varepsilon_{\text{PCP}} + 2^k \cdot k \cdot \left(\frac{\mathsf{l} \cdot \log|\Sigma|}{t}\right)\right) \cdot \left(\frac{2t^2}{2^\lambda}\right)^k\right) \\
&= O(t \cdot \varepsilon_{\text{PCP}}) \cdot \sum_{k=0}^{\infty} \left(\frac{4t^2}{2^\lambda}\right)^k + O(\mathsf{l} \cdot \log|\Sigma|) \cdot \sum_{k=0}^{\infty} \frac{2^k \cdot k}{t} \cdot \left(\frac{2t^2}{2^\lambda}\right)^k \quad .
\end{aligned}$$

Then, separately, we show that (assuming $\lambda \geq 2\log t + 6$) the two infinite sums converge:

$$\sum_{k=0}^{\infty} \left(\frac{4t^2}{2^\lambda}\right)^k = O(1) \quad \text{and} \quad \sum_{k=0}^{\infty} \frac{2^k \cdot k}{t} \cdot \left(\frac{2t^2}{2^\lambda}\right)^k = O\left(\frac{t}{2^\lambda}\right) \quad .$$

Finally, we conclude that:

$$\Pr\left[\begin{array}{c}\text{verifier}\\ \text{accepts}\end{array}\right] \leq O\left(t \cdot \varepsilon_{\text{PCP}} + C \cdot \frac{t}{2^\lambda}\right) \quad ,$$

where $C = 12 \cdot \mathsf{l} \cdot \log|\Sigma|$.

Moreover, in order to achieve concrete efficiency (e.g., the numbers reported in Section 1.2), our security analysis improves on the above expression in two ways: (1) it replaces the hidden constant (in the big-O notation) with the constant 1; and (2) it lowers the upper bound on C to $12 \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{\log \frac{1}{t \cdot \varepsilon_{\text{PCP}}}}$.

To achieve this, we count separately the queries performed to the tree and to derive PCP randomness. Thus, in the full proof, we introduce two new parameters t_{tree} and t_{rnd} such that it always holds that $t = t_{\text{tree}} + t_{\text{rnd}}$. Hence the full proof contains similar expressions as above, where in some cases, t is replaced with either t_{tree}, t_{rnd}, or their sum. See Section 7 for further details.

Adaptive soundness. The soundness notion provided by Theorem 1 is *non-adaptive*: an instance $\mathtt{x} \notin L$ is fixed, and then the random oracle is sampled. All prior works in Section 1.3 also focus on non-adaptive soundness. However, one could also consider a stronger, *adaptive*, soundness notion: the random oracle is sampled and then the cheating prover chooses an instance $\mathtt{x} \notin L$. While there is a general (black-box) method to achieve adaptive soundness from non-adaptive soundness in any SNARG, the method incurs a multiplicative factor of t in the soundness error, which is undesirable, especially so in the context of our work (aimed at a tight security analysis). Fortunately, our security analysis can be naturally extended to directly show adaptive soundness with the same soundness error. This mostly relies on using an adaptive soundness definition of the tree soundness game and the reverse soundness game. We leave working out the details for this adaptive case to future work.

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; FGLSS91; AS98; ALMSS98]. 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 $\Pi := \mathbf{P}(\mathbb{x}, \mathbb{w}) \in \Sigma^{\mathsf{l}}$, $\Pr_{\rho \in \{0,1\}^r}[\mathbf{V}^{\Pi}(\mathbb{x}; \rho) = 1] = 1$.
- **Soundness.** For every $\mathbb{x} \notin L(R)$ and malicious proof $\tilde{\Pi} \in \Sigma^{\mathsf{l}}$, $\Pr_{\rho \in \{0,1\}^r}[\mathbf{V}^{\tilde{\Pi}}(\mathbb{x}; \rho) = 1] \leq \varepsilon_{\text{PCP}}(\mathbb{x})$.

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 $\mathbb{x}$.

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 $(\mathbb{x}, \mathbb{w}) \in R$,

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

- **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.

3.3 Micali's construction

We describe Micali's construction of a (succinct) non-interactive argument from a PCP [Mic00], with the domain-separated flavor (Remark 3.1). 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 algorithms below are granted access to a random oracle $\zeta \colon \{0,1\}^* \to \{0,1\}^\lambda$, which is domain separated into two random oracles: (i) a *PCP randomness oracle* $\zeta_{\mathrm{rnd}} \colon \{0,1\}^* \to \{0,1\}^{\mathsf{r}}$, where r is the randomness complexity of the PCP verifier $\mathbf{V}$; (ii) a *tree oracle* $\zeta_{\mathrm{tree}} \colon \{0,1\}^* \to \{0,1\}^\lambda$.

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 witness $\mathbb{w}$, and computes an argument π as follows. First P runs the PCP prover $\mathbf{P}$ on $(\mathbb{x}, \mathbb{w})$ to obtain the PCP proof $\Pi \in \Sigma^{\mathsf{l}}$. Next P uses 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 $i = d-1, d-2, \ldots, 0$: for $j \in [2^i]$, compute $h_{i,j} := \zeta_{\mathrm{tree}}(i \| j \| h_{i+1,2j-1} \| h_{i+1,2j}) \in \{0,1\}^\lambda$.
- Set the root $\mathsf{rt} := h_{0,1} \in \{0,1\}^\lambda$.

Then P derives randomness $\rho := \zeta_{\mathrm{rnd}}(\mathsf{rt}) \in \{0,1\}^{\mathsf{r}}$ and simulates the PCP verifier $\mathbf{V}$ on input $(\mathbb{x}; \rho)$ and PCP string Π; this execution induces q query-answer pairs $(j_1, a_1), \ldots, (j_{\mathsf{q}}, a_{\mathsf{q}}) \in [\mathsf{l}] \times \Sigma$. Finally, P outputs

$$\pi := \Big(\mathsf{rt}, (j_1, a_1, p_1), \ldots, (j_d, a_d, p_d)\Big) \tag{1}$$

where $p_1, \ldots, p_d$ are the authentication paths for the query-answer pairs $(j_1, a_1), \ldots, (j_{\mathsf{q}}, a_{\mathsf{q}})$.

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:

- derive randomness $\rho := \zeta_{\mathrm{rnd}}(\mathsf{rt})$ for the PCP verifier;
- check that the PCP verifier $\mathbf{V}$, on input $(\mathbb{x}; \rho)$ and by answering a query to j_r with a_r, accepts;
- check that $p_1, \ldots, p_d$ are authentication paths of $(j_1, a_1), \ldots, (j_d, a_d)$ that hash into the root rt.

Argument size. The argument π contains the root $\mathsf{rt} \in \{0,1\}^\lambda$, a $(\log |\Sigma|)$-bit answer for each of q queries, and q authentication paths. This totals to an argument size that is

$$\lambda + \mathsf{q} \cdot \log |\Sigma| + \mathsf{q} \cdot \lambda \cdot \log \mathsf{l} \ . \tag{2}$$

Each of the q queries in $[\mathsf{l}]$ comes with an authentication path containing the $\log \mathsf{l}$ siblings of vertices on the path from the query to the root, which amounts to $\lambda \cdot \log \mathsf{l}$ bits. (More precisely, $\log |\Sigma| + \lambda \cdot (\log \mathsf{l} - 1)$ 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; BCRSVW19]) 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 Section 1.2 already account for this straightforward optimization.

Remark 3.1 (domain separation)**.** The reader may notice that the above description differs slightly from [Mic00]. While the random oracle ζ is split into the two oracles ζ_{rnd} and ζ_{tree} as in [Mic00], each time we use ζ_{tree} to hash two children we additionally hash the location in the tree (the level i and the vertex number j). This has no effect on argument size and only a negligible effect on the running times of the argument prover and argument verifier. However, these minor differences force an attacker to specify in its query which part of the scheme it is attacking, which we leverage.

Remark 3.2 (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 $j \in [2^i]$. This is useful for capturing the zero knowledge extension of the Micali construction (explained below), and more generally shows that our results hold against strong attacks (the attacker can obtain multiple random digests $h_{i,j}$ for any given indices i and j). Note that, when adding salts, the definition of an authentication path needs to be extended to account for salts wherever they are used, and in particular the size of π increases accordingly.

Recall that the Micali construction is (statistical) zero knowledge if the underlying PCP is honest-verifier zero knowledge and all the leaves in the Merkle tree are salted (the honest prover sets the j-th leaf $h_{d,j}$ of the tree to be $\zeta_{\text{tree}}(\Pi_j \| \sigma_{d,j})$ for a random $\sigma_{d,j} \in \{0,1\}^\lambda$ rather than just Π_j). We refer the reader to [BCS16] for an analysis of this (the cited work discusses zero knowledge for the BCS construction, which extends the Micali construction to work for public-coin IOPs). Salts in the leaves behave as hiding commitments, which indeed were similarly used in [IMSX15] to improve Kilian's zero knowledge succinct arguments to only make a black-box use of cryptography.

For simplicity, in this paper we do not explicitly add an additional layer below the tree for salting each leaf individually. Instead, we remark that the above construction is implicitly captured by having the honest prover place the honest-verifier PCP in every even leaf and setting odd leaves to a default value. The salts in the first layer of the tree then produce the same effect as salting each leaf.

4 Reverse soundness for a PCP

We define the reverse soundness game, and prove that reverse soundness for a PCP reduces to the PCP's standard soundness (Lemma 4.4).

Definition 4.1. *The function* $\mathsf{merge}\colon (\Sigma \cup \{\perp\})^{\mathsf{l}} \times (\Sigma \cup \{\perp\})^{\mathsf{l}} \times \{0,1\} \to (\Sigma \cup \{\perp\})^{\mathsf{l}}$ *maps two partial proof strings* $\Pi_0, \Pi_1 \in (\Sigma \cup \{\perp\})^{\mathsf{l}}$ *and a bit* $b \in \{0,1\}$ *to a new partial proof string whose* j*-th entry, for* $j \in [\mathsf{l}]$*, is defined as follows:*

$$\mathsf{merge}(\Pi_0, \Pi_1, b)[j] := \begin{cases} \Pi_0[j] & \text{if } \Pi_1[j] = \perp \\ \Pi_1[j] & \text{if } \Pi_0[j] = \perp \\ \Pi_b[j] & \text{otherwise} \end{cases} .$$

Game 4.2. The *reverse soundness game* is parametrized by a PCP verifier $\mathbf{V}$, an instance $\mathbb{x}$, and a positive integer K. We denote by $\mathcal{G}_{\text{rev}}(\mathbf{V}, \mathbb{x}, K, \tilde{\mathbf{P}})$ the boolean random variable denoting whether a malicious prover $\tilde{\mathbf{P}}$ wins in this game, according to the description below.

1. For $i = 1, \ldots, K$ do:
   (a) $\tilde{\mathbf{P}}$ outputs a partial proof string $\Pi_i \in (\Sigma \cup \{\perp\})^{\mathsf{l}}$.
   (b) $\tilde{\mathbf{P}}$ receives a random string $\rho_i \in \{0,1\}^{\mathsf{r}}$, which represents randomness for the PCP verifier.
2. $\tilde{\mathbf{P}}$ outputs a partial proof string $\Pi \in (\Sigma \cup \{\perp\})^{\mathsf{l}}$ and a bit $b \in \{0,1\}$.
3. The game samples $i \in [K]$, and sets $\Pi^\star := \mathsf{merge}(\Pi_i, \Pi, b)$.
4. The game outputs 1 if $\mathbf{V}^{\Pi^\star}(\mathbb{x}; \rho_i) = 1$, and 0 otherwise.

Definition 4.3. *A PCP* $(\mathbf{P}, \mathbf{V})$ *for a relation* R *has* **reverse soundness error** $\varepsilon_{\text{rev}}(\mathbb{x}, K)$ *if for every instance* $\mathbb{x} \notin L(R)$*, positive integer* K*, and malicious prover* $\tilde{\mathbf{P}}$*,*

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

Lemma 4.4. *Let* $(\mathbf{P}, \mathbf{V})$ *be a PCP for a relation* R *with soundness error* ε_{PCP} *and proof length* l *over an alphabet* Σ*. If* $K \cdot \varepsilon_{\text{PCP}}(\mathbb{x}) \leq 1/20$ *then* $(\mathbf{P}, \mathbf{V})$ *has reverse soundness error*

$$\varepsilon_{\text{rev}}(\mathbb{x}, K) \leq 1.06 \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{K \cdot \log \frac{1}{K \cdot \varepsilon_{\text{PCP}}(\mathbb{x})}} .$$

Proof. For each $i \in [K]$, define $\Pi_i^\star := \mathsf{merge}(\Pi_i, \Pi, b)$ where Π_i, Π, b are the corresponding outputs of $\tilde{\mathbf{P}}$ in the game. Let $\tilde{X}_i$ be the boolean random variable for the event that $\mathbf{V}^{\Pi_i^\star}(\mathbb{x}; \rho_i) = 1$. Set $\tilde{X} := \sum_{i \in [K]} \tilde{X}_i$, and observe that $\mathbb{E}[\tilde{X}] \leq \varepsilon_{\text{PCP}}$. Define $X := \sum_{i \in [K]} X_i$ where each X_i is defined based on $\tilde{X}_i$ as follows

$$X_i := \begin{cases} 1 & \text{if } \tilde{X} = 1 \\ 1 & \text{w.p. } \varepsilon_{\text{PCP}} - \mathbb{E}[\tilde{X}_i] \\ 0 & \text{otherwise} \end{cases} .$$

Note that it always holds that $X_i \geq \tilde{X}_i$ and thus $X \geq \tilde{X}$. Moreover, we have that $\mathbb{E}[X_i] = \varepsilon_{\text{PCP}}$, $\mathbb{E}[X] = \varepsilon_{\text{PCP}} \cdot K$, and that $X_1, \ldots, X_K$ are independent random variables.

Let $I \subseteq [K]$ be any subset of size $c \in \mathbb{N}$. Then, we get that

$$\Pr_{\rho_1, \ldots, \rho_{|I|}}\left[\forall i \in [I] : \mathbf{V}^{\Pi_i^\star}(\mathbb{x}; \rho_i) = 1\right] \leq \varepsilon_{\text{PCP}}^{|I|} = \varepsilon_{\text{PCP}}^{c} .$$

Taking a union bound over all (complete) proofs Π in Σ^{l}, and over all subset I of size $c \in \mathbb{N}$ we obtain that

$$\Pr_{\rho_1,\ldots,\rho_K}\left[\exists\,\Pi : \Pr_{i\in[K]}[\mathbf{V}^{\mathsf{merge}(\Pi_i,\Pi,b)}(\mathbb{x};\rho_i)=1] \geq \frac{c}{K}\right] \leq |\Sigma|^{\mathsf{l}} \cdot \binom{K}{c} \cdot \varepsilon_{\mathrm{PCP}}^{c} \leq 2^{\mathsf{l}\cdot\log|\Sigma| + c\log(K\cdot\varepsilon_{\mathrm{PCP}})} \ .$$

Let $c^* = \frac{\mathsf{l}\cdot\log|\Sigma|}{\log\frac{1}{K\cdot\varepsilon_{\mathrm{PCP}}}}$. Then, we bound the success probability of the prover by:

$$\begin{aligned}
&\Pr\left[\mathcal{G}_{\mathrm{rev}}(\mathbf{V},\mathbb{x},K,\tilde{\mathbf{P}})=1\right] \\
&\leq \Pr_{\rho_1,\ldots,\rho_K}\left[\exists\,\Pi : \Pr_{i\in[K]}[\mathbf{V}^{\mathsf{merge}(\Pi_i,\Pi,b)}(\mathbb{x};\rho_i)=1] \leq \frac{c^*}{K}\right]\cdot\frac{c^*}{K} \\
&+ \sum_{c=c^*+1}^{\infty}\Pr_{\rho_1,\ldots,\rho_K}\left[\exists\,\Pi : \Pr_{i\in[K]}[\mathbf{V}^{\mathsf{merge}(\Pi_i,\Pi,b)}(\mathbb{x};\rho_i)=1] = \frac{c}{K}\right]\cdot\frac{c}{K} \\
&\leq \frac{c^*}{K} + \sum_{c=c^*+1}^{\infty} 2^{\mathsf{l}\cdot\log|\Sigma| + c\log(K\cdot\varepsilon_{\mathrm{PCP}})}\cdot\frac{c}{K} \\
&\leq \frac{c^*}{K} + \sum_{i=1}^{\infty} 2^{\mathsf{l}\cdot\log|\Sigma| + (c^*+i)\log(K\cdot\varepsilon_{\mathrm{PCP}})}\cdot\frac{c^*+i}{K} \\
&\leq \frac{c^*}{K} + \sum_{i=1}^{\infty}(K\cdot\varepsilon_{\mathrm{PCP}})^i\cdot\frac{c^*+i}{K} \\
&\leq \frac{c^*}{K} + \frac{c^*}{K}\left(\frac{K\cdot\varepsilon_{\mathrm{PCP}}}{(1-K\cdot\varepsilon_{\mathrm{PCP}})^2} + \frac{K\cdot\varepsilon_{\mathrm{PCP}}}{1-K\cdot\varepsilon_{\mathrm{PCP}}}\right) \\
&\leq \frac{c^*}{K} + \frac{c^*}{K}\left(\frac{20}{361} + \frac{1}{20}\right) \\
&\leq 1.06\cdot\frac{\mathsf{l}\cdot\log|\Sigma|}{K\cdot\log\frac{1}{K\cdot\varepsilon_{\mathrm{PCP}}}} \ .
\end{aligned}$$

□

5 Tree soundness for a PCP

We define the tree soundness game after introducing some notation and auxiliary definitions. Then we prove that tree soundness reduces to reverse soundness (Lemma 5.6) and conclude, by further invoking our lemma from Section 4, that tree soundness reduces to standard soundness (Theorem 5.7).

Definition 5.1. *Let d and λ be positive integers, and Σ a finite alphabet. The vertex set V is the union $V_0 \cup V_1 \cup \cdots \cup V_d$ where $V_d := \{(d, j, h) : j \in [2^d], h \in \Sigma\}$ and, for every $i \in \{0, 1, \ldots, d-1\}$, $V_i := \{(i, j, h) : j \in [2^i], h \in \{0,1\}^\lambda\}$. 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* **interval** *of a base vertex $u = (i, j, h)$ is $[a, b]$ where $a = 2^{d-i} \cdot j$ and $b = 2^{d-i} \cdot (j+1) - 1$;*
- *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 5.2. *Let $G = (V, E)$ be a graph over the vertex set V as in Definition 5.1.*
- *A vertex $u_d \in V_d$ is* **connected in** *G to a vertex $u_\ell \in V_\ell$ if there exist vertices $u_{d-1}, \ldots, u_{\ell+1}$ such that, for all $i \in \{d, d-1, \ldots, \ell+1\}$, $u_i \in V_i$ 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 (i.e., 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 5.3. *Let $G = (V, E)$ be a graph over the vertex set V as in Definition 5.1. Let u be a vertex and let $[a, b]$ be its associated interval. A string $s \in (\Sigma \cup \{\perp\})^{b-a}$ is* **consistent in** *G with u if for every $j \in [a, b]$ such that $s[a + j] \neq \perp$ there exists a vertex $v_j = (d, j, h) \in V_d$ such that $h = s[j - a]$ and v_j is connected to u in G. In such a case we write $\mathsf{Consistent}(G, s, u) = 1$.*

Game 5.4. The *tree 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 randomness 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 $\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}})$. The game works as follows:

- **Initialization:**
  1. Set $E := \emptyset$ to be an empty edge set for the graph $G = (V, E)$.
  2. Set Roots to be an empty mapping from V to verifier randomness.
- **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 $i \in \{0, 1, \ldots, d-1\}$ 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 free and $|e(u)| = 0$ then add $e = (u, v_0, v_1)$ to E.

3. $t_{\text{tree}} \leftarrow t_{\text{tree}} - 1$.

– **Option** COL**:** $\tilde{\mathbf{P}}$ submits a vertex $u = (i, j, h) \in V$ with $i \in \{0, 1, \ldots, d-1\}$ 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 free and $|e(u)| \geq 1$ then add $e = (u, v_0, v_1)$ to E.
3. $t_{\text{col}} \leftarrow t_{\text{col}} - 1$.

– **Option** INV**:** $\tilde{\mathbf{P}}$ submits indices i, j with $i \in \{0, 1, \ldots, d-1\}$ and $j \in [2^i]$ and strings h_0, h_1.
1. Define $H_{i,j} = \{h \in \{0,1\}^\lambda : \exists\, e \in E,\ (i, j, h) \in e\}$.
2. Sample a biased coin b such that $\Pr[b = 1] = \frac{|H_{i,j}|}{2(t_{\text{tree}} + t_{\text{rnd}})}$.
3. If $b = 1$ then choose $h \leftarrow H_{i,j}$ at random.
4. If $b = 0$ then choose $h \leftarrow \{0,1\}^\lambda \setminus H_{i,j}$ at random.
5. Add the (hyper)edge $e := (u, v_0, v_1)$ to E where $u := (i, j, h) \in V_i$, $v_0 := (i+1, 2j-1, h_0) \in V_{i+1}$, and $v_1 := (i+1, 2j, h_1) \in V_{i+1}$.
6. $t_{\text{inv}} \leftarrow t_{\text{inv}} - 1$.
7. Give h to $\tilde{\mathbf{P}}$.

– **Option** RND**:** $\tilde{\mathbf{P}}$ submits a root vertex $v_{\mathsf{rt}} \in V_0$.
1. If Roots already contains an entry for v_{rt} then set $\rho \leftarrow \mathsf{Roots}[v_{\mathsf{rt}}]$.
2. If Roots does not contain an entry for v_{rt} then sample $\rho \in \{0,1\}^{\mathsf{r}}$ at random and set $\mathsf{Roots}[v_{\mathsf{rt}}] \leftarrow \rho$.
3. $t_{\text{rnd}} \leftarrow t_{\text{rnd}} - 1$.
4. ρ is given to $\tilde{\mathbf{P}}$.

- **Output:** $\tilde{\mathbf{P}}$ outputs a root vertex $v_{\mathsf{rt}} \in V_0$ 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 $\Pi[j] := h \in \Sigma$; set $\Pi[j] := \perp$ for all other locations.
2. Retrieve PCP randomness for this root vertex: $\rho^* \leftarrow \mathsf{Roots}[v_{\mathsf{rt}}]$.
3. Check that the PCP verifier accepts: $\mathbf{V}^{\Pi}(\mathbb{x}; \rho^*) = 1$.
4. Check that Π is consistent in G with the root: $\mathsf{Consistent}(G, \Pi, v_{\mathsf{rt}}) = 1$.
5. Check that $\tilde{\mathbf{P}}$ is within budget: $t_{\text{col}} \geq 0$, $t_{\text{inv}} \geq 0$, $t_{\text{rnd}} \geq 0$, and $t_{\text{tree}} \geq 0$.

Definition 5.5. *A PCP $(\mathbf{P}, \mathbf{V})$ for a relation R has tree soundness error $\varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}})$ if for every $\mathbb{x} \notin L(R)$, output size $\lambda \in \mathbb{N}$, malicious prover $\tilde{\mathbf{P}}$, and budgets $t_{\text{rnd}}, t_{\text{col}}, t_{\text{inv}} \in \mathbb{N}$,*

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

Lemma 5.6. *If $(\mathbf{P}, \mathbf{V})$ is a PCP for a relation R with soundness error $\varepsilon_{\text{PCP}}(\mathbb{x})$ and reverse soundness error $\varepsilon_{\text{rev}}(\mathbb{x}, K)$, then it has tree soundness error*

$$\varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) \leq 2^{t_{\text{col}}} \cdot \max\left\{t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}), 2t_{\text{inv}} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, t_{\text{rnd}} + t_{\text{tree}})\right\} \ .$$

Proof. Let $\tilde{\mathbf{P}}_{\text{tree}}$ be a malicious prover that wins the tree soundness game with probability greater than $\varepsilon_{\text{tree}}$. For any root vertex v_{rt} we set $\mathsf{Consistent}(G, v_{\mathsf{rt}})$ to be all proof strings that are consistent with v_{rt} in the graph G and are also full proof strings (no entry equals $\perp$):

$$\mathsf{Consistent}(G, v_{\mathsf{rt}}) := \{\Pi \in \Sigma^{\mathsf{l}} : \mathsf{Consistent}(G, \Pi, v_{\mathsf{rt}})\} \ .$$

We define an event $\mathcal{E}$ that, intuitively, indicates that the prover $\tilde{\mathbf{P}}_{\text{tree}}$ wins the game using Option RND without using inversions. If $\mathcal{E}$ does not hold then the prover wins using an inversion submitted via Option INV. Formally, let the root vertex $v_{\mathsf{rt}} \in V_0$ and leaf vertices $v_1, \ldots, v_{\mathsf{q}} \in V_d$ be the final output of the prover. If the prover wins in the tree soundness game, then it must be

that these leaves are connected to the root: $\mathsf{Consistent}(G, \Pi, v_{\mathsf{rt}}) = 1$ where Π is the partial proof that identifies with $v_1, \ldots, v_{\mathsf{q}}$. The event $\mathcal{E}$ holds whenever there exist paths that connect every leaf $v_1, \ldots, v_{\mathsf{q}}$ to the root v_{rt} and all the edges in these paths where added using Option ADD.

We split the analysis in two cases according to the event $\mathcal{E}$, and then derive the lemma.

Case 1: $\mathcal{E}$ occurs. Conditioned on the event $\mathcal{E}$ we know that all edges connecting $v_1, \ldots, v_{\mathsf{q}}$ to v_{rt} were added without Option INV, which, in particular, means that they existed in the graph before the prover used Option RND on v_{rt}.

There are at most t_{rnd} invocations of Option RND, and in each a root is chosen. For each iteration, we wish to bound the number of consistent proof strings Π that identify with the leaves $v_1, \ldots, v_{\mathsf{q}}$. Every such proof string must be in the set $\mathsf{Consistent}(G, v_{\mathsf{rt}})$ and thus it suffices to bound the size of this set. We obtain such a bound by bounding the number of collisions (and the size of each collision): using the budget on collision t_{col} we get that

$$|\mathsf{Consistent}(G, v_{\mathsf{rt}})| \leq \prod_{u \in V} |\mathsf{edges}(u)| \leq 2^{t_{\mathrm{col}}} \ .$$

From the (standard) soundness of the PCP, we know that the probability that the verifier accepts any fixed proof string is at most $\varepsilon_{\mathrm{PCP}}$. Thus, taking a union bound over all t_{rnd} roots v_{rt} and over all (full) proof strings in $\mathsf{Consistent}(G, v_{\mathsf{rt}})$ we get that

$$\Pr\left[\mathcal{G}_{\mathrm{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\mathrm{rnd}}, t_{\mathrm{tree}}, t_{\mathrm{col}}, t_{\mathrm{inv}}) = 1 \;\middle|\; \mathcal{E}\right] \leq t_{\mathrm{rnd}} \cdot |\mathsf{Consistent}(G, v_{\mathsf{rt}})| \cdot \varepsilon_{\mathrm{PCP}} \leq t_{\mathrm{rnd}} \cdot 2^{t_{\mathrm{col}}} \cdot \varepsilon_{\mathrm{PCP}} \ .$$

Case 2: $\mathcal{E}$ does not occur. We reduce to the reverse soundness game: we describe how to obtain a malicious prover $\tilde{\mathbf{P}}_{\mathrm{rev}}$ that wins the reverse soundness game with probability greater than a certain loss times $\varepsilon_{\mathrm{rev}}(\mathbb{x}, t_{\mathrm{rnd}})$ where we set $K := t_{\mathrm{rnd}} + t_{\mathrm{tree}}$.

The cheating prover $\tilde{\mathbf{P}}_{\mathrm{rev}}$ plays in the reverse soundness game $\mathcal{G}_{\mathrm{rev}}$ by simulating a play of $\tilde{\mathbf{P}}_{\mathrm{tree}}$ and an instance of the tree soundness game $\mathcal{G}_{\mathrm{tree}}$. In particular, $\tilde{\mathbf{P}}_{\mathrm{rev}}$ maintains the internal data structures of $\mathcal{G}_{\mathrm{tree}}$ (the edge set E and roots map Roots) as they would be maintained in $\mathcal{G}_{\mathrm{tree}}$. These maintenance details are omitted from the description below. During its simulation, $\tilde{\mathbf{P}}_{\mathrm{tree}}$ may repeatedly choose one of four options available in $\mathcal{G}_{\mathrm{tree}}$. In the case of either Option INV or Option RND, $\tilde{\mathbf{P}}_{\mathrm{rev}}$ does the following for each of these cases. The prover can perform at most t_{inv} inversions (via Option INV). Thus, we guess $\ell \in [t_{\mathrm{inv}}]$ at random.

- Option INV in $\mathcal{G}_{\mathrm{tree}}$: $\tilde{\mathbf{P}}_{\mathrm{tree}}$ submits indices i, j and strings h_0, h_1.
  1. $\ell \leftarrow \ell - 1$.
  2. If $\ell > 0$ skip the steps below and continue with the simulation.
  3. Let $v_0 = (i+1, 2j-1, h_0)$, $v_1 = (i+1, 2j, h_1)$, and $u = (i, j, 0)$, and add the edge (u, v_0, v_1) to the graph.
  4. Submit $K - |\mathsf{Roots}|$ empty proofs in the $\mathcal{G}_{\mathrm{rev}}$.
  5. Sample $\Pi^* \in \mathsf{Consistent}(G, u)$ and $b \in \{0, 1\}$ at random and submit it as the final proof in $\mathcal{G}_{\mathrm{rev}}$.
- Option RND in $\mathcal{G}_{\mathrm{tree}}$: $\tilde{\mathbf{P}}_{\mathrm{tree}}$ submits a root vertex $v_{\mathsf{rt}} \in V_0$.
  1. Sample a random $\Pi \in \mathsf{Consistent}(G, v_{\mathsf{rt}})$.
  2. Submit Π in 1.a in $\mathcal{G}_{\mathrm{rev}}$ and get ρ.
  3. Set $\mathsf{Roots}[v_{\mathsf{rt}}] \leftarrow \rho$.

We now bound the success probability of $\tilde{\mathbf{P}}_{\text{rev}}$. The final output of the cheating prover includes leaves $v_1, \ldots, v_{\mathsf{q}}$ and a root v_{rt}. Let Π' be a proof that identifies with these leaves. Then, there must exist $\Pi \in \mathsf{Consistent}(G, v_{\mathsf{rt}})$, $\Pi^* \in \mathsf{Consistent}(G, u)$, and $b \in \{0,1\}$ such that $\Pi' = \mathsf{merge}(\Pi, \Pi^*, b)$. This is since the accepting proof Π' must be consistent with v_{rt} after we added the edge (u, v_0, v_1). Thus, it must be a combination of proofs that were already consistent with v_{rt} and a (partial) proof that was consistent with u.

Let ℓ^* be the invocation of Option INV at which after this invocation there exists a proof $\Pi \in \mathsf{Consistent}(G, v_{\mathsf{rt}})$ that convinces the verifier. We guess this invocation of Option INV (i.e., $\ell = \ell^*$) with probability $1/t_{\text{inv}}$. If all our guess are correct, then the probability that we win in the reverse soundness game is at least

$$\frac{1}{t_{\text{inv}}} \cdot \frac{1}{2^{t_{\text{col}}}} \cdot \frac{1}{2} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, K) \ .$$

This yields us the bound:

$$\Pr\left[\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) = 1 \ \mid \ \neg\mathcal{E}\right] \leq 2t_{\text{inv}} \cdot 2^{t_{\text{col}}} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, K) \ .$$

Combining the cases. Combining the two cases above, we obtain the claimed bound:

$$\begin{aligned}
&\varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) \\
&= \Pr[\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) = 1 \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \\
&\quad \Pr[\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) = 1 \mid \neg\mathcal{E}] \cdot \Pr[\neg\mathcal{E}] \\
&\leq 2^{t_{\text{col}}} \cdot t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}) \cdot \Pr[\mathcal{E}] + 2t_{\text{inv}} \cdot 2^{t_{\text{col}}} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, t_{\text{rnd}} + t_{\text{tree}}) \cdot \Pr[\neg\mathcal{E}] \\
&\leq 2^{t_{\text{col}}} \cdot \max\left\{t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}), 2t_{\text{inv}} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, t_{\text{rnd}} + t_{\text{tree}})\right\} \ .
\end{aligned}$$

□

Theorem 5.7. *Let $(\mathbf{P}, \mathbf{V})$ be a PCP for a relation R with soundness error ε_{PCP} and proof length l over an alphabet Σ. Then $(\mathbf{P}, \mathbf{V})$ has tree soundness error*

$$\varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) \leq 2^{t_{\text{col}}} \cdot \max\left\{t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}), 2.12 \cdot t_{\text{inv}} \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{K \cdot \log \frac{1}{K \cdot \varepsilon_{\text{PCP}}(\mathbb{x})}}\right\} \ ,$$

where $K = t_{\text{rnd}} + t_{\text{tree}}$.

Proof. We apply Lemma 4.4 to our PCP and get that it has soundness

$$\varepsilon_{\text{rev}}(\mathbb{x}, K) \leq 1.06 \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{K \cdot \log \frac{1}{K \cdot \varepsilon_{\text{PCP}}(\mathbb{x})}} \ .$$

Then, we apply Lemma 5.6 and get that the PCP has tree soundness:

$$\begin{aligned}
\varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, t_{\text{col}}, t_{\text{inv}}) &\leq 2^{t_{\text{col}}} \cdot \max\left\{t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}), 2 \cdot t_{\text{inv}} \cdot \varepsilon_{\text{rev}}(\mathbb{x}, K)\right\} \\
&\leq 2^{t_{\text{col}}} \cdot \max\left\{t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}}(\mathbb{x}), 2.12 \cdot t_{\text{inv}} \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{K \cdot \log \frac{1}{K \cdot \varepsilon_{\text{PCP}}(\mathbb{x})}}\right\} \ .
\end{aligned}$$

□

6 Scoring oracle queries

Consider an algorithm A that has query access to a random oracle $\zeta\colon \{0,1\}^* \to \{0,1\}^\lambda$. In this section we are interested in studying the probability that A makes "notable" queries to ζ as a function of the number of queries performed. Here "notable" means one of two cases: multiple queries that map to the same answer (i.e., a collision); or a query whose answer already appeared as part of a prior query (i.e., an inversion). Below we assign *scores* to each of these cases, and bound the probability that A can succeed in making queries that in aggregate result in a high score.

We define the notion of a query trace, then how we score a query trace, and then state and prove our results about algorithms achieving query traces with high scores. (Note that in all cases we only care about A's query trace, and ignore its output.)

Definition 6.1. *A* **query trace** *with respect to* $\zeta\colon \{0,1\}^* \to \{0,1\}^\lambda$ *is a list* $\mathsf{tr} = ((x_1, y_1), \ldots, (x_t, y_t))$, *with no duplicate pairs, such that* $\zeta(x_j) = y_j$ *for every* $j \in [t]$.

In the domain-separated flavor of the Micali construction (see Section 3.3), the oracle ζ is divided via domain separation into multiple oracles, one for each vertex in the tree, and it would make sense to have separate scores for each oracle. However, here we give a simpler, and more generous, scoring definition that ignores the oracle separation for the tree queries, and only distinguishes between the tree oracle ζ_{tree} and the PCP randomness oracle ζ_{rnd}. The benefit is a more general definition that may be of interest in other contexts. This relaxation does not hurt our analysis because the tree soundness game does enforce domain separation, which prevents the adversary, e.g., from using the same collision on multiple locations.

While the collision score is well-defined for any query trace, we are interested only in collisions in the query trace of ζ_{tree}, and so use the notation $\mathsf{tr}_{\text{tree}} = \big((x_j, y_j)\big)_{j \in [t_{\text{tree}}]}$ to denote this trace.

Definition 6.2. *The* **collision score** *of a query trace* $\mathsf{tr}_{\text{tree}} = \big((x_j, y_j)\big)_{j \in [t_{\text{tree}}]}$ *with respect to* $\zeta_{\text{tree}}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *is defined as* $\mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) := \sum_{j \in [t_{\text{tree}}]} \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}, (x_j, y_j))$ *where*

$$\mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}, (x, y)) = \frac{m-1}{m} \text{ where } m = \left|\{j \in [t_{\text{tree}}] \,|\, \zeta_{\text{tree}}(x_j) = y\}\right| \ .$$

The definition of inversion score is with respect to both traces. Let t_{tree} be the length of the query trace $\mathsf{tr}_{\text{tree}}$ for ζ_{tree} and t_{rnd} the length of the query trace tr_{rnd} for ζ_{rnd}, and let $t := t_{\text{tree}} + t_{\text{rnd}}$.

As explained above, we ignore the domain separation used to distinguish different types of tree queries, and so we view a query x that is part of a pair (x, y) in the query trace $\mathsf{tr}_{\text{tree}}$ as a pair (x_0, x_1) consisting of two strings of length λ (or $\log|\Sigma|$ in the case of a query for a leaf). Then, for any query-answer pair (x, y) in $\mathsf{tr}_{\text{tree}}$, we set $\mathcal{T}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x, y))$ to be the set of possible targets for inversion in both traces: for any pair (x', y') that appears before (x, y) in either $\mathsf{tr}_{\text{tree}}$ or tr_{rnd} where $x' = (x'_0, x'_1) \in \{0,1\}^{2\lambda}$, we add x'_0, x'_1 to $\mathcal{T}_{\text{inv}}$.

Definition 6.3. *Let* $\mathsf{tr}_{\text{tree}} = \big((x_j, y_j)\big)_{j \in [t_{\text{tree}}]}$ *be a query trace for* ζ_{tree}, *and* tr_{rnd} *a query trace for* ζ_{rnd}. *The* **inversion score** $\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}})$ *is defined as* $\sum_{j \in [t_{\text{tree}}]} \mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x_j, y_j))$ *where*

$$\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x, y)) := \begin{cases} 1 & \text{if } y \in \mathcal{T}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x, y)) \\ 0 & \text{otherwise} \end{cases} \ .$$

Lemma 6.4 (trace score lemma)**.** *Let $\lambda \in \mathbb{N}$ be the oracle's output size and $t_{\text{tree}}, t_{\text{rnd}} \in \mathbb{N}$ query bounds. Then for every parameter $k \in \mathbb{N}$ and algorithm A it holds that:*

1. $$\Pr\left[\mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) > k \;\middle|\; \begin{array}{r} \zeta \leftarrow \mathcal{U}(\lambda) \\ \mathsf{tr}_{\text{tree}} \leftarrow \mathsf{queries}_{\text{tree}}(A, \zeta) \\ t_{\text{tree}} = |\mathsf{tr}_{\text{tree}}| \end{array}\right] \leq \left(\tfrac{t_{\text{tree}}^2}{2 \cdot 2^\lambda}\right)^k \;;$$

2. $$\Pr\left[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) > k \;\middle|\; \begin{array}{r} \zeta \leftarrow \mathcal{U}(\lambda) \\ \mathsf{tr}_{\text{tree}} \leftarrow \mathsf{queries}_{\text{tree}}(A, \zeta) \\ \mathsf{tr}_{\text{rnd}} \leftarrow \mathsf{queries}_{\text{rnd}}(A, \zeta) \\ t_{\text{tree}} = |\mathsf{tr}_{\text{tree}}|, t_{\text{rnd}} = |\mathsf{tr}_{\text{rnd}}| \end{array}\right] \leq \tfrac{1}{k!} \cdot \left(\tfrac{2 \cdot (t_{\text{tree}} + t_{\text{rnd}}) \cdot t_{\text{tree}}}{2^\lambda}\right)^k .$$

Proof of Item 1 of Lemma 6.4. In order to get a score of k, the trace must contain a collection of at most k subsets $(J_1, \ldots, J_k)$ that collide (namely, $\zeta(x_j) = \zeta(x_{j'})$ for all $j, j' \in J$). Let C_J be the random variable indicating that all queries in J collide. Now, we define a set $\mathcal{A}$ of all such subsets that get a score of k. The score for each J is $|J| - 1$. Thus, we define

$$\mathcal{A} := \left\{ (J_1, \ldots, J_\ell) \;:\; \begin{array}{l} J_1, \ldots, J_\ell \subseteq [t_{\text{tree}}] \text{ are distinct} \\ |J_i| \geq 2 \text{ for all } i \in [\ell] \\ \sum_{i=1}^{\ell} (|J_i| - 1) = k \end{array} \right\} .$$

Claim 6.5. *For any $t_{\text{tree}} \geq 5$ we have that $|\mathcal{A}| \leq \left(\frac{t_{\text{tree}}^2}{2}\right)^k$.*

Proof. For any ℓ we count the size of $\mathcal{A}$ with exactly ℓ subsets. We have that $\sum_{i=1}^{\ell} |J_i| = k + \ell$ and thus there are $k + \ell$ elements and $t_{\text{tree}}^{k+\ell}$ ways to choose them. For each choice of these $k + \ell$ elements, there are $\binom{k-1}{\ell-1}$ ways to divide the total score of k into ℓ groups. Finally, we observe that we counted each element in $\mathcal{A}$ multiple times. First, we order of the subsets does not matter and thus we divide by a factor of $\ell!$. Next, within each subset, the order does not matter. Each subset is of size at least 2, and there are ℓ subsets and thus we divide by 2^ℓ.

Together we get that

$$|\mathcal{A}| \leq \sum_{\ell=1}^{k} \frac{t_{\text{tree}}^{k+\ell} \cdot \binom{k-1}{\ell-1}}{\ell! \cdot 2^\ell} \leq t_{\text{tree}}^k \cdot \sum_{\ell=1}^{k} \frac{t_{\text{tree}}^\ell \cdot (k-1)^{\ell-1}}{\ell! \cdot (\ell-1)! \cdot 2^\ell} \leq t_{\text{tree}}^k \cdot \left(\frac{t_{\text{tree}}}{2}\right)^k = \left(\frac{t_{\text{tree}}^2}{2}\right)^k ,$$

where the last inequality can be easily shown using an inductive argument (and using $t_{\text{tree}} \geq 5$). □

Using this bound, we get that:

$$\begin{aligned} \Pr[\mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) \geq k] &\leq \Pr[\exists\, (J_1, \ldots, J_k) \in \mathcal{A} : C_{J_1} = 1, \ldots, C_{J_k} = 1] \\ &\leq \sum_{(J_1, \ldots, J_k) \in \mathcal{A}} 2^{-(|J_1|-1)\lambda} \cdots 2^{-(|J_k|-1)\lambda} \\ &= 2^{-\lambda \cdot k} \cdot |\mathcal{A}| \\ &\leq 2^{-\lambda \cdot k} \cdot \left(\frac{t_{\text{tree}}^2}{2}\right)^k \\ &= \left(\frac{t_{\text{tree}}^2}{2 \cdot 2^\lambda}\right)^k . \end{aligned}$$

□

Proof of Item 2 of Lemma 6.4. Recall that a query trace $\mathsf{tr}_{\text{tree}}$ is an ordered list containing all query-answer pairs to the oracle, and let $\mathsf{tr}_{\text{tree}}[j] = (x_j, y_j)$ be the j-th query pair in the list $\mathsf{tr}_{\text{tree}}$. We define random variables $\{I_j\}_{j\in[t_{\text{tree}}]}$ for which it holds that $\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = \sum_{j\in[t_{\text{tree}}]} I_j$. The variables encode the contributions to the score of individual inversions:

$$I_j := \mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, \mathsf{tr}_{\text{tree}}[j]) \ .$$

Next, we define random variables $\{\bar{I}_j\}_{j\in[t]}$ based on $\{I_j\}_{j\in[t]}$ that will help us in the analysis. Observe that the random variable I_j is not independent of the other random variables. In particular,

$$\Pr[I_j = 1] = \frac{|\mathcal{T}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x_j, y_j))|}{2^\lambda} \ ,$$

where (x_j, y_j) is the j-th query to the tree oracle. The random variable $\bar{I}_j$ below "rounds up" the probability that $I_j = 1$ to $2(j-1)/2^\lambda$ regardless of other events:

$$\bar{I}_j := \begin{cases} 1 & \text{if } I_j = 1 \\ 1 & \text{w.p. } \frac{2(j-1)-|\mathcal{T}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}, (x_j, y_j))|}{2^\lambda} \\ 0 & \text{otherwise} \end{cases} \ .$$

Observe that it always holds that $I_j \le \bar{I}_j$, that $I_1, \ldots, I_t$ are independent and that

$$\Pr[\bar{I}_j = 1] = \frac{2(j-1)}{2^\lambda} \le \frac{2(t_{\text{tree}} + t_{\text{rnd}})}{2^\lambda} \ .$$

If $\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) \ge k$ then there must exist distinct $j_1, \ldots, j_k$ in $[t_{\text{tree}}]$ such that $I_1 = 1, \ldots, I_k = 1$, and thus also $\bar{I}_1, \ldots, \bar{I}_k = 1$. Then, we get that:

$$\begin{aligned}
\Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) \ge k] &\le \Pr\left[\exists\,(j_1, \ldots, j_k) \in \binom{[t_{\text{tree}}]}{k} : \bar{I}_{j_1} = 1, \ldots, \bar{I}_{j_k} = 1\right] \\
&\le \Pr\left[\exists\,(j_1, \ldots, j_k) \in \binom{[t_{\text{tree}}]}{k} : \bar{I}_{j_1} = 1, \ldots, \bar{I}_{j_k} = 1\right] \\
&\le \sum_{(j_1,\ldots,j_k)\in\binom{[t_{\text{tree}}]}{k}} \Pr[\bar{I}_{j_1} = 1, \ldots, \bar{I}_{j_k} = 1] \\
&= \sum_{(j_1,\ldots,j_k)\in\binom{[t_{\text{tree}}]}{k}} \frac{2(t_{\text{tree}} + t_{\text{rnd}})}{2^\lambda} \cdots \frac{2(t_{\text{tree}} + t_{\text{rnd}})}{2^\lambda} \\
&= \frac{2^k \cdot (t_{\text{tree}} + t_{\text{rnd}})^k}{2^{k\cdot\lambda}} \cdot \binom{t_{\text{tree}}}{k} \le \frac{2^k \cdot (t_{\text{tree}} + t_{\text{rnd}})^k}{2^{k\cdot\lambda}} \cdot \frac{t_{\text{tree}}^k}{k!} \le \frac{1}{k!}\left(\frac{2 \cdot (t_{\text{tree}} + t_{\text{rnd}}) \cdot t_{\text{tree}}}{2^\lambda}\right)^k \ .
\end{aligned}$$

□

7 Upper bound on the soundness error of Micali

Theorem 7.1 (Theorem 1)**.** *Suppose that the Micali construction (described in Section 3.3) is instantiated with: (i) a PCP with soundness error $\varepsilon_{\mathrm{PCP}}$, proof length l over alphabet Σ, and query complexity q; and (ii) a random oracle with output size λ. Then, provided that $\lambda \geq 2\log t + 6$, the Micali construction has a soundness error $\epsilon(t)$ against t-query adversaries that is bounded as follows:*

$$\epsilon(t) \leq t \cdot \varepsilon_{\mathrm{PCP}} + C \cdot \frac{t}{2^{\lambda}} \quad \textit{with} \quad C := 12 \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{\log \frac{1}{t \cdot \varepsilon_{\mathrm{PCP}}}} \ .$$

The bound on the soundness error enables us to set parameters to achieve security.

Corollary 7.2 (Corollary 2)**.** *The Micali construction is (t, ϵ)-secure when instantiated with:*

- *a PCP with soundness error $\varepsilon_{\mathrm{PCP}} \leq \frac{1}{2}\frac{\epsilon}{t}$; and*
- *a random oracle with output size $\lambda \geq \max\left\{2\log t + 6\,,\ \log(t/\epsilon) + \log \frac{\mathsf{l}\cdot\log|\Sigma|}{\log \frac{1}{t\cdot\varepsilon_{\mathrm{PCP}}}} + 5\right\}$.*

Proof. We set $\varepsilon_{\mathrm{PCP}}$ and λ such that each term is bounded by $\epsilon/2$ and thus get overall soundness ϵ for the argument scheme. For the first term, we want that $t \cdot \varepsilon_{\mathrm{PCP}} \leq \epsilon/2$ which leads us to require $\varepsilon_{\mathrm{PCP}} \leq \frac{1}{2}\frac{\epsilon}{t}$. For the second term we need that $C \cdot \frac{t}{2^{\lambda}} \leq \epsilon/2$, which leads us to require that $\lambda \geq \log(t/\epsilon) + \log(2C)$. Plugging in $C = 12 \cdot \frac{\mathsf{l}\cdot\log|\Sigma|}{\log \frac{1}{t\cdot\varepsilon_{\mathrm{PCP}}}}$ we get that

$$\lambda \geq \log(t/\epsilon) + \log\left(2 \cdot 12 \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{\log \frac{1}{t\cdot\varepsilon_{\mathrm{PCP}}}}\right) \ .$$

Recall that Theorem 7.1 also requires that $\lambda \geq 2\log t + 6$. Together, we get the claimed bound. □

We prove the theorem in Section 7.1 and a technical claim in Section 7.2. Throughout this section we denote by (P, V) the argument prover and argument verifier in the Micali construction, and by $(\mathbf{P}, \mathbf{V})$ the PCP prover and PCP verifier of the PCP used in the Micali construction.

7.1 Proof of Theorem 7.1

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}. For any choice of positive integers t_{rnd} and t_{tree} such that $t_{\mathrm{rnd}} + t_{\mathrm{tree}} = t$, 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 tree 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 7.3. *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 $k \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{l|r} V^{\zeta}(\mathbb{x},\pi)=1 & \begin{array}{r}\zeta \leftarrow \mathcal{U}(\lambda)\\ \pi \leftarrow \tilde{P}^{\zeta}\\ \mathsf{tr}_{\mathrm{rnd}} \leftarrow \mathsf{queries}_{\mathrm{rnd}}(\tilde{P},\zeta)\\ \mathsf{tr}_{\mathrm{tree}} \leftarrow \mathsf{queries}_{\mathrm{tree}}(\tilde{P},\zeta)\\ |\mathsf{tr}_{\mathrm{rnd}}|=t_{\mathrm{rnd}}, |\mathsf{tr}_{\mathrm{tree}}|=t_{\mathrm{tree}}\\ \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}}) \le k\\ \mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}},\mathsf{tr}_{\mathrm{rnd}}) \le k\end{array}\end{array}\right] \le \Pr\left[\mathcal{G}_{\mathrm{tree}}(\mathbf{V},\mathbb{x},\lambda,\tilde{\mathbf{P}},t_{\mathrm{rnd}},t_{\mathrm{tree}},k,k)=1\right] \ . \tag{3}$$

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 7.3 is given in Section 7.2.

We use Lemma 6.4 to obtain two bounds that will be useful in the analysis further below; we also use the assumption that $\lambda \ge 2\log t + 6$ and the fact that $t \ge t_{\mathrm{tree}}$. Both bounds hold for any choice of a parameter $k \in \mathbb{N}$. The first bound is:

$$\begin{aligned}
&\sum_{k=0}^{\infty} \frac{k\cdot 2^k}{t} \cdot \Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}},\mathsf{tr}_{\mathrm{rnd}})=k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}})=k]\\
&\quad\le \sum_{k=0}^{\infty} \frac{k\cdot 2^k}{t}\cdot\left(\frac{1}{k!}\cdot\left(\frac{2t\cdot t_{\mathrm{tree}}}{2^\lambda}\right)^k + \left(\frac{t_{\mathrm{tree}}^2}{2\cdot 2^\lambda}\right)^k\right)\\
&\quad= \sum_{k=0}^{\infty} \frac{k\cdot 2^k}{t}\cdot\frac{1}{k!}\cdot\left(\frac{2t\cdot t_{\mathrm{tree}}}{2^\lambda}\right)^k + \sum_{k=0}^{\infty} \frac{k\cdot 2^k}{t}\cdot\left(\frac{t_{\mathrm{tree}}^2}{2\cdot 2^\lambda}\right)^k\\
&\quad\le \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot\sum_{k=0}^{\infty}\frac{2k\cdot 2^k}{k!}\cdot\left(\frac{2t\cdot t_{\mathrm{tree}}}{2^\lambda}\right)^{k-1} + \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot\sum_{k=0}^{\infty}\frac{k\cdot 2^k}{2}\cdot\left(\frac{t_{\mathrm{tree}}^2}{2\cdot 2^\lambda}\right)^{k-1} \qquad (\text{since } t\ge t_{\mathrm{tree}})\\
&\quad\le \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot\sum_{k=0}^{\infty}\frac{k\cdot 2^{2k}}{k!\cdot 2^{6(k-1)}} + \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot\sum_{k=0}^{\infty}\frac{k\cdot 2^k}{2\cdot 2^{6(k-1)}} \quad (\text{since } \lambda\ge 2\log t+6)\\
&\quad\le \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot 4.26 + \frac{t_{\mathrm{tree}}}{2^\lambda}\cdot 1.07 = 5.33\cdot\frac{t_{\mathrm{tree}}}{2^\lambda} \ .
\end{aligned}$$

The second bound is:

$$\begin{aligned}
&\sum_{k=0}^{\infty} 2^k \cdot \Pr[\mathsf{score}_{\mathrm{inv}}(\mathsf{tr}_{\mathrm{tree}},\mathsf{tr}_{\mathrm{rnd}})=k \vee \mathsf{score}_{\mathrm{col}}(\mathsf{tr}_{\mathrm{tree}})=k]\\
&\quad\le 1+\sum_{k=1}^{\infty} 2^k\cdot\left(\frac{1}{k!}\cdot\left(\frac{2t\cdot t_{\mathrm{tree}}}{2^\lambda}\right)^k + \left(\frac{t_{\mathrm{tree}}^2}{2\cdot 2^\lambda}\right)^k\right)\\
&\quad= 1+\sum_{k=1}^{\infty}\frac{1}{k!}\cdot\left(\frac{4t\cdot t_{\mathrm{tree}}}{2^\lambda}\right)^k + \sum_{k=1}^{\infty}\left(\frac{t_{\mathrm{tree}}^2}{2^\lambda}\right)^k\\
&\quad\le 1+\sum_{k=1}^{\infty}\frac{1}{k!}\cdot\left(\frac{4t_{\mathrm{tree}}}{2^6\cdot t}\right)^k + \sum_{k=1}^{\infty}\left(\frac{t_{\mathrm{tree}}}{2^6\cdot t}\right)^k \quad (\text{since } \lambda\ge 2\log t+6)
\end{aligned}$$

$$\leq 1 + \frac{8t_{\text{tree}}}{2^6 t} + \frac{2t_{\text{tree}}}{2^6 t}$$
$$\leq 1 + \frac{t_{\text{tree}}}{t} \ .$$

Using the above bounds, we conclude by establishing an upper bound on the soundness error:

$$\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}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, k, k) = 1\right] \cdot \Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = k \vee \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) = k] \\
&\leq \sum_{k=0}^{\infty} \varepsilon_{\text{tree}}(\mathbb{x}, \lambda, t_{\text{rnd}}, t_{\text{tree}}, k, k) \cdot \Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = k \vee \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) = k] \\
&\leq \sum_{k=0}^{\infty} \left(t_{\text{rnd}} \cdot 2^k \cdot \varepsilon_{\text{PCP}} + 2.12k \cdot \frac{\mathsf{l} \cdot \log|\Sigma|}{\log \frac{1}{t \cdot \varepsilon_{\text{PCP}}}} \cdot \frac{2^k}{t}\right) \cdot \Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = k \vee \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) = k] \\
&\leq t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}} \cdot \sum_{k=0}^{\infty} 2^k \cdot \Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = k \vee \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) = k] \\
&+ \frac{2.12C}{12} \cdot \sum_{k=0}^{\infty} \frac{k \cdot 2^k}{t} \cdot \Pr[\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) = k \vee \mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) = k] \\
&\leq t_{\text{rnd}} \cdot \varepsilon_{\text{PCP}} \cdot \left(1 + \frac{t_{\text{tree}}}{t}\right) + \frac{2.12C}{12} \cdot \left(5.33 \cdot \frac{t_{\text{tree}}}{2^{\lambda}}\right) \\
&\leq (t_{\text{rnd}} + t_{\text{tree}}) \cdot \varepsilon_{\text{PCP}} + C \cdot \frac{t_{\text{tree}}}{2^{\lambda}} \\
&\leq t \cdot \varepsilon_{\text{PCP}} + C \cdot \frac{t}{2^{\lambda}} \ .
\end{aligned}$$

7.2 Proof of Claim 7.3

For simplicity, we 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 3.2), 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 argument prover $\tilde{P}$ either outputs an accepting proof or an abort symbol. This can be achieved by modifying $\tilde{P}$ to run the argument verifier on its output proof and, if the proof is not accepting, outputting the abort symbol. This is not completely without loss of generality, as this might cost a few additional queries. However, this has a negligible effect on $\tilde{P}$'s query complexity (and thus our results) because the argument verifier makes few queries to the random oracle, so we ignore this effect.

We use $\tilde{P}$ to construct a PCP prover $\tilde{\mathbf{P}}$ that plays in the tree soundness game $\mathcal{G}_{\text{tree}}$ (Game 5.4). 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 $v_{\mathsf{rt}} := (0, 1, x) \in V_0$.
  2. Submit, via Option RND in $\mathcal{G}_{\text{tree}}$, the root vertex v_{rt}.
  3. Receive from $\mathcal{G}_{\text{tree}}$ 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 in Σ (depending on i) and salt $\sigma \in \{0,1\}^\lambda$.
  1. Sample a biased coin $b \in \{0,1\}$ such that $\Pr[b = 1] = \frac{2t}{2^\lambda}$.
  2. If $b = 1$ then submit i, j, h_0, h_1 via Option INV in $\mathcal{G}_{\text{tree}}$ and obtain an answer $h \in \{0,1\}^\lambda$.
  3. If $b = 0$ then:
     3.1. sample a random $h \in \{0,1\}^\lambda$ and set $u := (i, j, h) \in V_i$;
     3.2. if $|e(u)| = 0$ then submit u, h_0, h_1 via Option ADD in $\mathcal{G}_{\text{tree}}$;
     3.3. if $|e(u)| \geq 1$ then submit u, h_0, h_1 via Option COL in $\mathcal{G}_{\text{tree}}$.
  4. 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 $v_{\mathsf{rt}} := (0, 1, \mathsf{rt})$ where rt 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 (3).

The probability of the left-hand side in Equation (3) is over a random oracle ζ (with several conditions). We view the randomness in the right-hand side, which comes from $\tilde{\mathbf{P}}$ simulating $\tilde{P}$ and from the tree soundness game, also as an oracle, denoted by ζ'.

- For each root query x the output of ζ' is the string ρ sampled by the tree soundness game.
- For each tree query x the output of ζ' is the bit b sampled by $\tilde{\mathbf{P}}$, another bit b' sampled by the tree soundness game (if $b = 1$), and the string h sampled by the tree soundness game or by $\tilde{\mathbf{P}}$ (depending on the value of b).
- For all other queries the output of ζ' is the string h sampled by $\tilde{\mathbf{P}}$.

Observe that, in the second case, the output of ζ' is larger than the output of ζ.

The way that $\tilde{\mathbf{P}}$ answers $\tilde{P}$'s queries to ζ naturally induces a mapping f that maps any oracle ζ' to an oracle ζ such that:

1. f is a *regular* mapping (for every ζ_1, ζ_2 it holds that $|f^{-1}(\zeta_1)| = |f^{-1}(\zeta_2)|$);
2. if $f(\zeta') = \zeta$ and $V^{\zeta}(\mathbb{x}, \pi) = 1$ then $\mathcal{G}_{\text{tree}}(\mathbf{V}, \mathbb{x}, \lambda, \tilde{\mathbf{P}}, t_{\text{rnd}}, t_{\text{tree}}, k, k) = 1$ with the randomness ζ'.

We are left to argue that the two properties hold, as they imply Equation (3).

Property 1. The mapping is regular since 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}; other query types 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 tree soundness game $\mathcal{G}_{\text{tree}}$, 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). Note that $\tilde{\mathbf{P}}$ replies with h that is received from Option INV in $\mathcal{G}_{\text{tree}}$ with probability $2t/2^\lambda$ and otherwise it is uniformly random. In turn, in Option INV in $\mathcal{G}_{\text{tree}}$, h is a random element from $H_{i,j}$ with probability $\frac{|H_{i,j}|}{2(t_{\text{tree}}+t_{\text{rnd}})}$ and otherwise it is uniformly random outside the set. Overall, the probability that h is chosen from $H_{i,j}$ is

$$\frac{2t}{2^\lambda} \cdot \frac{|H_{i,j}|}{2(t_{\text{tree}} + t_{\text{rnd}})} = \frac{2t}{2^\lambda} \cdot \frac{|H_{i,j}|}{2t} = \frac{|H_{i,j}|}{2^\lambda} \ ,$$

and in this case it is uniformly within the set $H_{i,j}$. Otherwise, it is uniform outside of the set $H_{i,j}$. Overall, we conclude that h is uniformly random, as required.

Property 2. We claim that the PCP prover $\tilde{\mathbf{P}}$ wins the tree 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 tree soundness game. Let $\rho^* := \mathsf{Roots}[v_{\mathsf{rt}}]$ be the PCP randomness associated to the root vertex v_{rt} output by $\tilde{\mathbf{P}}$ (recall that Roots is the table maintained by the tree 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 PCP 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_{rt} (G is the graph maintained by the tree soundness game). 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 rt in π (and the oracle ζ). Then, since we assumed that $\tilde{P}$ has queried all vertices in the authentication paths in the output proof π, all edges between the leaf vertices $\{v_r\}_{r \in [\mathsf{q}]}$ and the root vertex v_{rt} are in G.

Hence, we only need to show that when the scores of $\tilde{P}$'s query trace are bounded by k then the budgets for $\tilde{\mathbf{P}}$'s suffice to win the tree soundness game.

- If $\mathsf{score}_{\text{col}}(\mathsf{tr}_{\text{tree}}) \leq k$ then we know that $\tilde{P}$ has found at most k collisions in $\mathsf{tr}_{\text{tree}}$. In this case, $\tilde{\mathbf{P}}$ submits the same number of collisions in the tree soundness game as $\tilde{P}$ because it imitates its queries. Thus, the collision budget suffices for the simulation.

- If $\mathsf{score}_{\text{inv}}(\mathsf{tr}_{\text{tree}}, \mathsf{tr}_{\text{rnd}}) \leq k$ then we know that $\tilde{P}$ has performed at most k inversions in $\mathsf{tr}_{\text{tree}}$ and tr_{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 $\tilde{\mathbf{P}}$ will perform the same amount of queries to Option RND and Option ADD respectively.

8 Lower bounds on the soundness error of Micali

We show that the security bounds for the Micali construction proved in this paper (Theorem 1 and Corollary 2) are tight up to low-order terms. In particular, we show that the dependency in the proof length and alphabet size in our security analysis is, in general, necessary.

Lemma 8.1. *Let $M[(\mathbf{P},\mathbf{V}),\lambda]$ denote the Micali construction instantiated with the PCP $(\mathbf{P},\mathbf{V})$ and with oracle output size λ. There exists a PCP $(\mathbf{P},\mathbf{V})$ with soundness error $\varepsilon_{\mathrm{PCP}}$, proof length l over an alphabet Σ, and query complexity q such that if $M[(\mathbf{P},\mathbf{V}),\lambda]$ is (t,ϵ)-secure for $t \geq 4\mathsf{l}$ then:*[3]

1. $\epsilon(t) \geq \Omega(t \cdot \varepsilon_{\mathrm{PCP}})$;
2. $\epsilon(t) \geq \Omega(C \cdot \frac{t}{2^\lambda})$ *where* $C \geq \frac{\mathsf{l}\cdot\log|\Sigma|}{\mathsf{q}^2}$;
3. $\lambda \geq 2 \cdot (\log t - \log \mathsf{l} - \log\log \mathsf{l}) + O(1)$.

We prove each item below via different attacks on the Micali construction. Our attacks work for general PCPs that satisfy some mild assumption (each attack requires a certain assumption). Therefore, the above lemma is more robust than stated in the sense that each lower bound works for most set of parameters $(\mathsf{l},\Sigma,\mathsf{q},\varepsilon_{\mathrm{PCP}})$ for which a PCP exists.

Proof of Item 1 (resample attack). For this attack we only need the PCP to have soundness error exactly $\varepsilon_{\mathrm{PCP}}$, i.e., there exist an instance $\mathbb{x} \notin L$ and proof string $\Pi \in \Sigma^{\mathsf{l}}$ such that $\Pr_\rho[\mathbf{V}^\Pi(\mathbb{x};\rho) = 1] \geq \varepsilon_{\mathrm{PCP}}$. The cheating argument prover first commits to Π, like the honest prover would, to obtain a Merkle root rt; this costs $2\cdot\mathsf{l}$ queries to the random oracle. Then the cheating argument prover uses the remaining $t - 2\cdot\mathsf{l}$ queries to the random oracle to derive many possible random strings for the PCP verifier, by changing the salt used to derive the randomness from the same Merkle root rt. The probability that the PCP verifier accepts Π on that randomness, and thus the malicious argument prover can make the argument verifier accept, is at least $\varepsilon_{\mathrm{PCP}}$. In sum, the probability that the cheating argument prover succeeds is the probability at least one of these random strings causes the PCP verifier to accept.

We lower bound this probability via the inclusion-exclusion principle: the probability that the verifier accepts in any of the $t - 2\cdot\mathsf{l}$ query sets is at least $(t-2\cdot\mathsf{l})\cdot\varepsilon_{\mathrm{PCP}} - t^2\cdot\varepsilon_{\mathrm{PCP}}^2$. If $t\cdot\varepsilon_{\mathrm{PCP}} \leq 3/4$ then we can write this probability as $\Omega(t\cdot\varepsilon_{\mathrm{PCP}})$. Else if $t\cdot\varepsilon_{\mathrm{PCP}} > 3/4$ then we bound the probability of the attack failing by $(1-\varepsilon_{\mathrm{PCP}})^t \leq e^{-t\cdot\varepsilon_{\mathrm{PCP}}} < 1/2$ and thus $\epsilon(t) > 1/2$ which contradicts the assumption that $\epsilon(t) \leq 1/2$ in the lemma statement. □

Proof of Item 2 (inversion attack). First we describe a simple inversion attack that succeeds with probability $\Omega(C\cdot\frac{t}{2^\lambda})$ for $C = 1$ for any PCP. Then we improve the attack to obtain $C \geq \mathsf{l}/\mathsf{q}^2$ for any PCP with random queries. Finally, we show that this latter attack can be simulated over a larger alphabet, resulting in the claimed bound of $C \geq \frac{\mathsf{l}\cdot\log|\Sigma|}{\mathsf{q}^2}$.

Simple inversion attack ($C \geq 1$). The cheating argument prover selects an arbitrary Merkle root $\mathsf{rt} \in \{0,1\}^\lambda$ (e.g., $\mathsf{rt} = 0^\lambda$), without first committing to any proof string, and then uses the random oracle to derive, from the Merkle root rt, a random string $\rho \in \{0,1\}^{\mathsf{r}}$ for the PCP verifier. Then the cheating argument prover deduces a local view of q symbols that makes the PCP verifier accept for the randomness ρ (such a local view exists for non-trivial languages), and extends arbitrarily this local view into a proof string Π. Then the cheating argument prover computes a Merkle tree on Π, hoping that it will yield the same root value rt chosen earlier; this costs $2\cdot\mathsf{l}$

[3]The requirement $t \geq 4\mathsf{l}$ is minor as even the honest prover uses $2\mathsf{l}$ queries.

queries to the random oracle. If this does not happen, the cheating argument prover re-computes the last layer of the Merkle tree with new salts, over and over, hoping that the new root will collide with the exiting root rt. Each such trial costs 1 query, and so overall the cheating prover can afford $\tilde{t} := t - 2 \cdot \mathsf{l} = \Omega(t)$ trials (the one when computing the Merkle tree plus all the trials after that).

The success probability of this attack is at least $\tilde{t} \cdot 2^{-\lambda} - \tilde{t}^2 \cdot 2^{-2\lambda}$. If $\tilde{t} \cdot 2^{-\lambda} \leq 3/4$ then we can write this probability as $\Omega(\frac{t}{2^\lambda})$. Else if $\tilde{t} \cdot 2^{-\lambda} > 3/4$ then we bound the failure probability of the attack by $(1 - 2^{-\lambda})^{\tilde{t}} \leq e^{-\tilde{t} \cdot 2^{-\lambda}} \leq 1/2$ and thus $\epsilon(t) > 1/2$ which contradicts the assumption that $\epsilon(t) \leq 1/2$ in the lemma statement.

A better inversion attack ($C \geq \mathsf{l}/\mathsf{q}^2$). We increase the probability of inversion by leveraging weak inversions. The cheating argument prover arbitrarily chooses $t/2$ distinct roots $\mathsf{rt}_1, \ldots, \mathsf{rt}_{t/2}$ and derives corresponding PCP random strings $\rho_1, \ldots, \rho_{t/2}$; then it finds a PCP string Π that satisfies as many of the PCP random strings as possible; it computes the Merkle tree of this string; and finally with the remaining queries tries to invert to any of the $t/2$ roots by enumerating over salts. The probability that the prover succeeds in inverting is $\Omega(t^2/2^\lambda)$, and if Π satisfied C of the challenges then the overall success probability of this strategy is

$$\Omega\left(\frac{t^2}{2^\lambda} \cdot \frac{C}{t}\right) = \Omega\left(C \cdot \frac{t}{2^\lambda}\right) \ .$$

We are left to bound C from below. For any choice of PCP randomness, there is a proof string satisfies it. Moreover, if several PCP random strings query different parts of the proof string, then there is a proof string that satisfies all of them. Henceforth, we will assume that the queries of the PCP are uniformly random (such PCPs are known to exist). We collect distinct PCP randomness strings in a greedy manner: initially set S to be empty; then, at step i, add ρ_i to S if its queries do not collide with any queries of PCP random strings already in S; stop when $|S| = \mathsf{l}/\mathsf{q}^2$. For any $i \in [t/2]$, let X_i be an indicator variable for the event that we added ρ_i to S. Then,

$$\Pr[X_i = 1] \geq \left(1 - \frac{|S| \cdot \mathsf{q}}{\mathsf{l}}\right)^{\mathsf{q}} \geq \left(1 - \frac{1}{\mathsf{q}}\right)^{\mathsf{q}} \geq \Omega(1) \ .$$

In expectation, every $O(1)$ iterations an element is added to S. By a Chernoff bound, with high probability after $O(\mathsf{l}/\mathsf{q}^2)$ iterations the set S achieves its maximal size of $|S| = \mathsf{l}/\mathsf{q}^2$. Since we assumed that $t \geq 4\mathsf{l} > \mathsf{l}/\mathsf{q}^2$, we conclude that with high probability $C \geq \mathsf{l}/\mathsf{q}^2$, as desired.

The case of a large alphabet ($C \geq \frac{\mathsf{l} \cdot \log|\Sigma|}{\mathsf{q}^2}$). The above attack can be modified to work over a larger alphabet. Take any PCP string of length l' over the binary alphabet and divide the PCP string into blocks of length $\log|\Sigma|$, thereby obtaining a PCP string of length $\mathsf{l} = \mathsf{l}'/\log|\Sigma|$ over an alphabet Σ. (The PCP verifier is modified accordingly.) We perform the attack above on the large-alphabet PCP by exploiting the binary PCP underneath. That is, two query sets that query the same location are still considered distinct as long as they read different bits of the symbol's representation. This yields the same bound on C for the underlying binary PCP: $C \geq \frac{\mathsf{l}'}{\mathsf{q}^2} = \frac{\mathsf{l} \cdot \log|\Sigma|}{\mathsf{q}^2}$. □

Proof of Item 3 (collision attack). First we discuss an attack that works for any PCP over the binary alphabet, and then discuss the case of a PCP over a larger alphabet.

Any binary PCP. The cheating argument prover finds a collision for each leaf in the Merkle tree, thus committing to both the value 0 and the value 1 at each leaf. Then, it continues honestly computing the Merkle tree given the colliding values to obtain a Merkle root rt, and then from

rt derives a random string $\rho \in \{0,1\}^{\mathsf{r}}$ for the PCP verifier. Then, the cheating argument prover deduces a local view of q symbols that makes the PCP verifier accept for the randomness ρ (such a local view exists for non-trivial languages), and uses the correct value in each collision (0 or 1) to include in the output argument string to convince the argument verifier.

We need to bound (from below) the probability of finding l collisions. First, we perform $O(2^{\lambda/2})$ queries and find a collision with probability at least $1/2$. We repeat this process $\log(2\mathsf{l})$ times such that the probability that we did not find a collision in any of the repetitions is at most $2^{-\log(2\mathsf{l})} = 1/(2\mathsf{l})$. Taking a union bound we get that with probability at least $1/2$ we find a collision in each and every leaf as desired, and can complete the Merkle tree.

We compute the total number of queries in the above attack. The number of queries required for the attack is $t \geq O(2^{\lambda/2}) \cdot \log(2\mathsf{l}) \cdot \mathsf{l} + 2\mathsf{l} = O(2^{\lambda/2} \cdot \mathsf{l} \cdot \log \mathsf{l})$. Equivalently, we can write this as $\lambda \leq 2 \cdot (\log t - \log \mathsf{l} - \log\log \mathsf{l}) + O(1)$, and get the desired bound.

The case of a non-binary alphabet. The attack above works for any binary PCP but does not work for any PCP over any alphabet. Nevertheless, we can embed the binary alphabet into a larger alphabet by associating 0 and 1 with two arbitrary symbols of the larger alphabet. The new PCP prover then writes the same binary PCP string but over the new alphabet, and the new PCP verifier acts the same way while rejecting when seeing any query answers other than 0 or 1. One can then conduct the same collision attack on this new PCP. This establishes that, for any alphabet, there exists a PCP over the alphabet for which the collision attack works. □

Acknowledgments

We thank Adi Neuman for designing the figures in this paper. 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

[ALMSS98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (1998). Preliminary version in FOCS '92., pp. 501–555.

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998). Preliminary version in FOCS '92., pp. 70–122.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO '19. 2019, pp. 733–764.

[BCRSVW19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. 2019, pp. 103–128.

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

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. "Succinct Arguments in the Quantum Random Oracle Model". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. 2019, pp. 1–29.

[CY21] Alessandro Chiesa and Eylon Yogev. "Subquadratic SNARGs in the Random Oracle Model". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 711–741.

[FGLSS91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating clique is almost NP-complete (preliminary version)". In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. SFCS '91. 1991, pp. 2–12.

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

[IMSX15] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. *On Zero-Knowledge PCPs: Limitations, Simplifications, and Applications*. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`. 2015.

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

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

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC '08. 2008, pp. 1–18.