Interactive Oracle Proofs*

Eli Ben-Sasson[1], Alessandro Chiesa[2] and Nicholas Spooner[3]

[1] Technion
eli@cs.technion.ac.il
[2] UC Berkeley
alexch@berkeley.edu
[3] University of Toronto
spooner@cs.toronto.edu

Abstract. We initiate the study of a proof system model that naturally combines interactive proofs (IPs) and probabilistically-checkable proofs (PCPs), and generalizes interactive PCPs (which consist of a PCP followed by an IP). We define an *interactive oracle proof* (IOP) to be an interactive proof in which the verifier is not required to read the prover's messages in their entirety; rather, the verifier has oracle access to the prover's messages, and may probabilistically query them. IOPs retain the expressiveness of PCPs, capturing NEXP rather than only PSPACE, and also the flexibility of IPs, allowing multiple rounds of communication with the prover. IOPs have already found several applications, including unconditional zero knowledge [BCGV16], constant-rate constant-query probabilistic checking [BCG$^+$16], and doubly-efficient constant-round IPs for polynomial-time bounded-space computations [RRR16].

We offer two main technical contributions. First, we give a compiler that maps any public-coin IOP into a non-interactive proof in the random oracle model. We prove that the soundness of the resulting proof is tightly characterized by the soundness of the IOP against *state restoration attacks*, a class of rewinding attacks on the IOP verifier that is reminiscent of, but incomparable to, resetting attacks.

Second, we study the notion of state-restoration soundness of an IOP: we prove tight upper and lower bounds in terms of the IOP's (standard) soundness and round complexity; and describe a simple adversarial strategy that is optimal, in expectation, across all state restoration attacks.

Our compiler can be viewed as a generalization of the Fiat–Shamir paradigm for public-coin IPs (CRYPTO '86), and of the "CS proof" constructions of Micali (FOCS '94) and Valiant (TCC '08) for PCPs. Our analysis of the compiler gives, in particular, a unified understanding of these constructions, and also motivates the study of state restoration attacks, not only for IOPs, but also for IPs and PCPs.

When applied to known IOP constructions, our compiler implies, e.g., blackbox unconditional ZK proofs in the random oracle model with quasilinear prover and polylogarithmic verifier, improving on a result of [IMSX15].

* Parts of this paper appear in the third author's master's thesis (April 2015) in the Department of Computer Science at ETH Zurich, supervised by Alessandro Chiesa and Thomas Holenstein. Independent of our work, [RRR16] introduce the notion of *Probabilistically Checkable Interactive Proofs*, which is the same as our notion of Interactive Oracle Proofs.

1 Introduction

The notion of *proof* is central to modern cryptography and complexity theory. The class NP, for example, is the set of languages whose membership can be decided by a deterministic polynomial-time verifier by reading proof strings of polynomial length; this class captures the traditional notion of a mathematical proof. Over the last three decades, researchers have introduced and studied proof systems that generalize the above traditional notion, and investigations from these points of view have led to breakthroughs in cryptography, hardness of approximation, and other areas. In this work we introduce and study a new model of proof system.

1.1 Models of proof systems

We give some context by recalling three of the most well-known among alternative models of proof systems.

Interactive proofs (IPs). Interactive proofs were introduced by Goldwasser, Micali, and Rackoff [GMR89]: in a k-round interactive proof, a probabilistic polynomial-time verifier exchanges k messages with an all-powerful prover, and then accepts or rejects; $\mathsf{IP}[k]$ is the class of languages with a k-round interactive proof. Independently, Babai [Bab85] introduced Arthur–Merlin games: a k-round Arthur–Merlin game is a k-round *public-coin* interactive proof (i.e., the verifier messages are uniformly and independently random); $\mathsf{AM}[k]$ is the class of languages with a k-round Arthur–Merlin game. Goldwasser and Sipser [GS86] showed that the two models are equally powerful: for polynomial k, $\mathsf{IP}[k] \subseteq \mathsf{AM}[k+2]$. Shamir [Sha92], building on the "sum-check" interactive proof of Lund, Fortnow, Karloff, and Nisan [LFKN92], proved that interactive proofs correspond to languages decidable in polynomial space: $\mathsf{IP}[\mathrm{poly}(n)] = \mathsf{PSPACE}$. (Also see [Bab90].)

Multi-prover interactive proofs (MIPs). Multi-prover interactive proofs were introduced by Ben-Or, Goldwasser, Kilian, and Wigderson [BGKW88]: in a k-round p-prover interactive proof, a probabilistic polynomial-time verifier interacts k times with p non-communicating all-powerful provers, and then accepts or rejects; $\mathsf{MIP}[p,k]$ is the class of languages that have a k-round p-prover interactive proof. In [BGKW88], the authors show that two provers always suffice (i.e., $\mathsf{MIP}[p,k] = \mathsf{MIP}[2,k]$), and that all languages in NP have perfect zero knowledge proofs in this model. Fortnow, Rompel, and Sipser [FRS88] show that interaction with two provers is equivalent to interaction with one prover plus oracle access to a proof string, and from there obtain that $\mathsf{MIP}[\mathrm{poly}(n), \mathrm{poly}(n)] \subseteq \mathsf{NEXP}$; Babai, Fortnow and Lund [BFL90] show that NEXP has 1-round 2-prover interactive proofs, thus showing that $\mathsf{MIP}[2,1] = \mathsf{NEXP}$.

Probabilistically checkable proofs (PCPs). Probabilistically checkable proofs were introduced by [FRS88, BFLS91, AS98, ALM$^+$98]: in a probabilistically-checkable proof, a probabilistic polynomial-time verifier has oracle access to a proof string; $\mathsf{PCP}[r,q]$ is the class of languages for which the verifier uses at most r bits of randomness, and queries at most q locations of the proof (note that the proof length is at most 2^r). The above results on MIPs imply that $\mathsf{PCP}[\mathrm{poly}(n), \mathrm{poly}(n)] = \mathsf{NEXP}$. Later works "scaled down" this result to NP: Babai, Fortnow, Levin and Szegedy

[BFLS91] show that $\mathsf{NP} = \mathsf{PCP}[O(\log n), \mathrm{poly}(\log n)]$; Arora and Safra [AS98] show that $\mathsf{NP} = \mathsf{PCP}[O(\log n), O(\sqrt{\log n})]$; and Arora, Lund, Motwani, Sudan, and Szegedy [ALM$^+$92] show that $\mathsf{NP} = \mathsf{PCP}[O(\log n), O(1)]$. This last is known as the *PCP Theorem*.

Researchers have studied other models of proof systems, and here we name only a few: *linear IPs* [BCI$^+$13], *no-signaling MIPs* [IKM09, Ito10, KRR13, KRR14], *linear PCPs* [IKO07, Gro10, Lip12, BCI$^+$13, GGPR13, PGHR13, BCI$^+$13, SBW11, SMBW12, SVP$^+$12, SBV$^+$13], *interactive PCPs* [KR08, KR09, GIMS10].

We introduce **interactive oracle proofs** (IOPs), a model of proof system that combines aspects of IPs and PCPs, and also generalizes interactive PCPs (which consist of a PCP followed by an IP). Our work focuses on cryptographic applications of this proof system, as we discuss next.

1.2 Compiling proof systems into argument systems

The proof systems mentioned so far share a common feature: they make no assumptions on the computational resources of a (malicious) prover trying to convince the verifier. Instead, many proof systems make "structural" assumptions on the prover: MIPs assume that the prover is a collection of non-communicating strategies (each representing a "sub-prover"); PCPs assume that the prover is non-adaptive (the answer to a message does not depend on previous messages); linear IPs assume that the prover is a linear function; and so on.

In contrast, in cryptography, one often considers *argument systems* [BC86, BCC88, Kil92, Mic00]: these are proof systems where soundness holds only against provers that have a bound on computational resources (e.g., provers that run in probabilistic polynomial time). The relaxation from statistical soundness to computational soundness allows circumventing various limitations of IPs [BHZ87, GH98, GVW02, PSSV07], while also avoiding "structural" assumptions on the prover, which can be hard to enforce in applications.

Constructing argument systems. A common methodology to construct argument systems with desirable properties (e.g., sublinear communication complexity) follows these two steps: (1) give a proof system that achieves these properties in a model with structural restrictions on (all-powerful) provers; (2) use cryptographic tools to compile that proof system into an argument system, i.e., one where the only restriction on the prover is that it is an efficient algorithm. Thus, the compilation trades any structural assumptions for computational ones. This methodology has been highly productive.

Proofs in the random oracle model. An idealized model for studying computationally-bounded provers is the random oracle model [FS86, BR93], where every party has access to the same random function. A protocol proved secure in this model can potentially be instantiated in practice by replacing the random function with a concrete "random-looking" efficient function. While this intuition fails in the general case [CGH04, BBP04, GK03, BDG$^+$13], the random oracle model is nonetheless a useful testbed for cryptographic primitives. In this paper we focus on proof systems in this model for which the proof consists of a single message from the prover to the verifier. A **non-interactive random-oracle argument** (NIROA) for a relation $\mathscr{R}$ is a pair of probabilistic polynomial-time algorithms, the prover $\mathbb{P}$ and verifier $\mathbb{V}$, that satisfy the following. (1) *Completeness:* for every instance-witness pair $(\mathbb{x}, \mathbb{w})$ in the relation $\mathscr{R}$, $\Pr[\mathbb{V}^\rho(\mathbb{x}, \mathbb{P}^\rho(\mathbb{x}, \mathbb{w})) = 1] = 1$, where the probability is taken over the random oracle ρ as well as any randomness of $\mathbb{P}$ and $\mathbb{V}$. (2) *Soundness:* for every instance $\mathbb{x}$ not in the language of $\mathscr{R}$ and every malicious prover $\tilde{\mathbb{P}}$ that asks at most a polynomial number of queries to the random oracle, it holds that $\Pr[\mathbb{V}^\rho(\mathbb{x}, \tilde{\mathbb{P}}^\rho) = 1]$ is negligible in the security parameter.

Prior NIROAs and our focus. Prior work uses the above 2-step methodology to obtain NIROAs with desirable properties. For example, the Fiat–Shamir paradigm maps 3-message public-coin IPs to corresponding NIROAs [FS86, PS96]; when invoked on suitable IP constructions, this yields efficient zero knowledge non-interactive proofs. As another example, Micali's "CS proof" construction, building on [Kil92], transforms PCPs to corresponding NIROAs; Valiant [Val08] revisits Micali's construction and proves that it is a proof of knowledge; when invoked on suitable PCPs, these yield non-interactive arguments of knowledge that are short and easy to verify. In this work we study the question of how to compile IOPs (which generalize IPs and PCPs) into NIROAs;[4] our work ultimately leads to formulating and studying a game-theoretic property of IOPs, which in turn motivates similar questions for IPs and PCPs. We now discuss our results.

1.3 Results

We present three main contributions: one is definitional and the other two are technical in nature.

Interactive oracle proofs A new proof system model. We introduce a new proof system model: *interactive oracle proofs* (IOPs).[5] This model naturally combines aspects of IPs and PCPs, and also generalizes IPCPs (see comparison in Remark 1 below); namely, an IOP is a "multi-round PCP" that generalizes an interactive proof as follows: the verifier has oracle access to the prover's messages, and may probabilistically query them (rather than having to read them in full). In more detail, a k-round IOP comprises k rounds of interaction. In the i-th round of interaction: the verifier sends a message m_i to the prover, which he reads in full; then the prover replies with a message f_i to the verifier, which he can query, as an oracle proof string, in this and all later rounds. After the k rounds of interaction, the verifier either accepts or rejects.

Like the PCP model, two fundamental measures of efficiency in the IOP model are the *proof length* p, which is the total number of bits in all of the prover's messages, and the *query complexity* q, which is the total number of locations queried by the verifier

[4] We do not study the question of avoiding assuming random oracles: this is not our focus. Reducing assumptions when compiling constant-round IPs is the subject of much research, obtaining arguments with non-programmable random oracles and a common random string [Lin15, CPSV16], obfuscation [KRR16, MV16], and others. Extending such ideas to IOPs is an interesting direction.

[5] Independent of our work, [RRR16] introduce *Probabilistically Checkable Interactive Proofs*, which are equivalent to our IOPs.

across all of the prover's messages. Unlike the PCP model, another fundamental measure of efficiency is the round complexity k; the PCP model can then be viewed as a special case where $k = 1$ (and the first verifier message is empty).

We show that IOPs characterize NEXP (like PCPs); both sequential and parallel repetition of IOPs yield (perfect) exponential soundness error reduction (like IPs); and any IOP can be converted into a public-coin one (like IPs). These basic complexity-theoretic properties confirm that our definition of IOP is a natural way to combine aspects of PCPs and IPs, and to generalize IPCPs.

Motivation: efficiency. IOPs extend IPs, by treating the prover's messages as oracle strings, and PCPs, by allowing for more than 1 round. These additional degrees of freedom enable IOPs to retain the expressive power of PCP while also allowing for additional efficiency, as already demonstrated in several works.

For example, [BCGV16] obtain unconditional zero knowledge via a 2-round IOP with quasilinear proof length; such a result is not known for PCPs (or even IPCPs [KR08]). Moreover, when combined with our compiler (see next contribution) we obtain blackbox unconditional zero-knowledge with quasilinear prover and polylogarithmic verifier in the random-oracle model, improving prover runtime of [IMSX15, Sec 2.3];

As another example, [BCG$^+$16] obtain 3-round IOPs for circuit satisfiability *with linear proof length and constant query complexity*, while for PCPs prior work only achieves sublinear query complexity [BKK$^+$13]. To do so, [BCG$^+$16] show that *sumcheck* [LFKN92, Sha92] and *proof composition* [AS98] (used in many PCP constructions such as [ALM$^+$98, HS00, BGH$^+$04]) have more efficient "IOP analogues", which in turn imply a number of probabilistic checking results that are more efficient than corresponding ones that only rely on PCPs. We briefly sketch the intuition for why interactive proof composition, via IOPs, is more efficient. In a composed proof, the prover first writes a part π_0 of the proof (e.g., in [ALM$^+$98] π_0 is an evaluation of a low-degree multivariate polynomial, and in [BS08] it is an evaluation of a low-degree univariate polynomial). Then, to demonstrate that π_0 has certain good properties (e.g., it is low degree), the prover also appends a (long) sequence of sub-proofs, where each sub-proof allegedly demonstrates to the verifier that a subset of entries of π_0 is "good". Afterwards, in another invocation of the recursion, the prover appends to each sub-proof a sequence of sub-sub-proofs, and so on. A crucial observation is that the verifier typically queries locations of only a small number of such sub-proofs; moreover, once the initial proof π_0 is fixed, soundness is not harmed if the verifier randomly selects the set of sub-proofs he wants to see and tells this to the prover. In sum, in many PCP constructions (including the aforementioned ones), *the proof length can be greatly reduced via interaction between the prover and verifier*, via an IOP.

As yet another example, [RRR16] use IOPs to obtain doubly-efficient constant-round IPs for polynomial-time bounded-space computations. The result relies on an "amortization theorem" for IOPs that states that, for a so-called *unambiguous* IOPs, batch verification of multiple statements can be more efficient than simply running an independent IOP for each statement.

Remark 1 (comparison with IPCP). Kalai and Raz [KR08] introduce and study *interactive PCPs* (IPCPs), a model of proof system that also combines aspects of IPs and PCPs, but in a different way: an IPCP is a PCP followed by an IP, i.e., the prover sends to the verifier a PCP and then the prover and verifier engage in an interactive proof. An IPCP can be viewed as a special case of an IOP, i.e., it is an IOP in which the verifier has oracle access to the first prover message, but must read in full subsequent prover messages. The works of [KR08, GKR08] show that boolean formulas with n variables, size m, and depth d have IPCPs where the PCP's size is polynomial in d and n and the communication complexity of the subsequent IP is polynomial in d and $\log m$. This shows that even IPCPs give efficiency advantages over both IPs and PCPs given separately.

From interactive oracle proofs to non-interactive random-oracle arguments We give a polynomial-time transformation that maps any public-coin interactive oracle proof (IOP) to a corresponding non-interactive random-oracle argument (NIROA). We prove that the soundness of the output proof is tightly characterized by the soundness of the IOP verifier against *state restoration attacks*, a class of rewinding attacks on the verifier that we now describe.

At a high level, a state restoration attack against an IOP verifier works as follows: the malicious prover and the verifier start interacting, as they normally would in an IOP; at any moment, however, the prover can choose to set the verifier to any state at which the verifier has previously been, and the verifier then continues onwards from that point *with fresh randomness*. Of course, if the prover could restore the verifier's state an unbounded number of times, the prover would eventually succeed in making the verifier accept. We thus only consider malicious provers that interact with the verifier for at most a certain number of rounds: for $b \in \mathbb{N}$, we say a prover is *b-round* if it plays at most b rounds during any interaction with any verifier. Then, we say that an IOP has state restoration soundness $s_{\mathrm{sr}}(\mathtt{x}, b)$ if every b-round state-restoring prover cannot make the IOP verifier accept an instance $\mathtt{x}$ (not in the language) with probability greater than $s_{\mathrm{sr}}(\mathtt{x}, b)$. This notion is reminiscent of, *but incomparable to*, the notion of resettable soundness [BGGL01]; see Remark 2 below.

Informally, our result about transforming IOPs into NIROAs can be stated as follows.

Theorem 1 (IOP $\rightarrow$ NIROA). *There exists a polynomial-time transformation T such that, for every relation $\mathscr{R}$, if (P, V) is a public-coin interactive oracle proof system for $\mathscr{R}$ with state restoration soundness $s_{\mathrm{sr}}(\mathtt{x}, b)$, then $(\mathbb{P}, \mathbb{V}) := T(P, V)$ is a non-interactive random-oracle argument system for $\mathscr{R}$ with soundness*

$$s_{\mathrm{sr}}(\mathtt{x}, m) + O(m^2 2^{-\lambda}) \ ,$$

where m is an upper bound on the number of queries to the random oracle that a malicious prover can make, and λ is a security parameter. The aforementioned soundness is tight up to small factors. (Good state restoration soundness can be obtained, e.g., via parallel repetition as in Remark 4.)

Moreover, we prove that the transformation T is benign in the sense that it preserves natural properties of the IOP. Namely, (1) the runtimes of the NIROA prover and verifier are linear in those of the IOP prover and verifier (up to a polynomial factor in λ); (2) the NIROA is a proof of knowledge if the IOP is a proof of knowledge (and the extractor strategy straight-line, which has desirable properties [BW15]); and (3) the NIROA is (malicious-verifier) statistical zero knowledge if the IOP is honest-verifier statistical zero knowledge.[6] See Theorem 3 for the formal statement; the statement employs the notion of *restricted state restoration soundness* as it allows for a tighter lower bound on soundness.

An immediate application is obtained by plugging the work of [BCGV16] into our compiler, thereby achieving a variant of the black-box ZK results of [IMSX15, Sec 2.3] where the prover runs in quasilinear (rather than merely polynomial) time.

Corollary 1 (informal). *There is a blackbox non-interactive argument system for* NP*, in the random-oracle model, with unconditional zero knowledge, quasilinear-time prover, and polylogarithmic-time verifier.*

Our compiler can be viewed as a generalization of the Fiat–Shamir paradigm for public-coin IPs [FS86, PS96], and of the "CS proof" constructions of Micali [Mic00] and Valiant [Val08] for PCPs. Our analysis of the compiler gives, in particular, a *unified understanding of these constructions*, and motivates the study of state restoration attacks, not only for IOPs, but also for IPs and PCPs. (Indeed, we are not aware of works that study the security of the Fiat–Shamir paradigm, in the random oracle model, applied to a public-coin IP with arbitrary number of rounds; the analyses that we are aware of focus on the case of 2 rounds.)

Our next contribution is a first set of results about such kinds of attacks, as described in the next section.

Remark 2 (resetting, backtracking). We compare state restoration soundness with other soundness notions:

– State restoration attacks are reminiscent of, *but incomparable to*, resetting attacks [BGGL01]. In the latter, the prover invokes multiple verifier incarnations with independent randomness, and may interact multiple times with each incarnation; also, this notion does not assume that the verifier is public-coin. Instead, in a state restoration attack, the verifier must be public-coin and its randomness is not fixed at the start but, instead, a new fresh random message is sampled each time the prover restores to a previously-seen state.
– State restoration is closely related to backtracking [BD16] (independent work). The two notions differ in that: (1) backtracking "charges" more for restoring verifier states that are further in the past, and (2) backtracking also allows the verifier to restore states of the prover (as part of the completeness property of the protocol); backtracking soundness is thus polynomially related to state restoration soundness.
  Bishop and Dodis [BD16] give a compiler from a public-coin IP to an error-resilient IP, whose soundness is related to the backtracking soundness of the original IP; essentially, they use hashing techniques to limit a malicious prover impersonating

[6] Security in the random oracle model sometimes does *not* imply security when the oracle is substituted with a hash function, e.g., when applying the Fiat–Shamir paradigm to zero-knowledge proofs/arguments [HT98, DNRS03, GOSV14]. However, our transformation T only assumes that the IOP is zero knowledge against the honest verifier, seemingly avoiding the above limitations.

an adversarial channel to choosing when to backtrack the protocol. Their setting is a completely different example in which backtracking, and thus state restoration, plays a role.

Remark 3 (programmability). As in most prior works, soundness and proof of knowledge do *not* rely on programming the random oracle. As for zero knowledge, the situation is more complicated: there are several notions of zero knowledge in the random oracle model, depending on "how programmable" the random oracle is (see [Wee09]). The notion that we use is zero knowledge in the explicitly-programmable random oracle (EPRO) model; the stronger notion in the non-programmable random oracle model is not achievable for NIROAs. Such a limitation can sometimes be avoided by also using a common random string [Lin15, CPSV16], and extending such techniques to the setting of IOPs is an interesting problem.

State restoration attacks on interactive oracle proofs The analysis of our transformation from public-coin IOPs to NIROAs highlights state restoration soundness as a notion that merits further study. We provide two results in this direction. First, we prove tight upper and lower bounds on state restoration soundness in terms of the IOP's (standard) soundness and round complexity.

Theorem 2. *For any relation $\mathscr{R}$, public-coin k-round IOP for $\mathscr{R}$, and instance $\mathbb{x}$ not in the language of $\mathscr{R}$,*

$$\forall\, b \geq k(\mathbb{x}) + 1, \quad \left\lfloor \frac{b}{k(\mathbb{x}) + 1} \right\rfloor s(\mathbb{x})(1 - o(1)) \leq s_{\mathrm{sr}}(\mathbb{x}, b) \leq \binom{b}{k(\mathbb{x}) + 1} s(\mathbb{x}) \ ,^{7}$$

where $s_{\mathrm{sr}}(\mathbb{x}, b)$ is the state restoration soundness of IOP and $s(\mathbb{x})$ its (standard) soundness for the instance $\mathbb{x}$. Also, the bounds are tight: there are IOPs that meet the lower bound and IOPs that meet the upper bound.

Remark 4 (good state restoration soundness). A trivial way to obtain state restoration soundness $2^{-\lambda}$ in the general case is to apply r-fold parallel repetition to the IOP with $r = \Omega(\frac{k \log b + \lambda}{\log s(\mathbb{x})})$; note that r is polynomially bounded for natural choices of k, b, λ. This choice of r is pessimistic, because for IOPs that do not meet the upper bound (i.e., are "robust" against such attacks) a smaller r suffices. This use of parallel repetition is analogous to its use in achieving the incomparable notion of resettable soundness [PTW09, COPV13].

Second, we study the structure of optimal state restoration attacks: we prove that, for any public-coin IOP, there is a simple state restoration attack that has optimal expected cost, where cost is the number of rounds until the prover wins. This result relies on a correspondence that we establish between IOP verifiers and certain games, which we

[7] We note that [BGGL01] prove an analogous upper bound for the *incomparable* notion of resettable soundness (see Remark 2). Also, [BD16] prove an analogous, weaker upper bound on the related notion of backtracking soundness (see Remark 2). Neither of the two studies lower bounds, or tightness of bounds.

call *tree exploration games*, pitting one player against Nature. We go in more detail about this result in later sections (see Section 1.4 and full version [BCS16].). A better understanding of state restoration soundness may enable us to avoid trivial soundness amplification (see Remark 4) for IOPs of interest.

1.4 Techniques

We summarize the techniques that we use to prove our technical contributions.

The transformation. Our transformation maps any public-coin IOP to a corresponding NIROA, and it generalizes two transformations that we now recall.

The first transformation is the Fiat–Shamir paradigm [FS86, PS96], which maps any public-coin IP to a corresponding NIROA, and it works as follows. The NIROA prover runs the interaction between the IP prover and the IP verifier "in his head", by setting the IP verifier's next message to be the output of the random oracle on the query that equals the transcript of previously exchanged messages. The NIROA prover sends a non-interactive proof that contains the final transcript of interaction; the NIROA verifier checks the proof's validity by checking that all the IP verifier's messages are computed correctly via the random oracle.

The second transformation is the "CS proof" construction of Micali [Mic00] and Valiant [Val08], which maps any PCP to a corresponding NIROA, and it works as follows. The NIROA prover first commits to the PCP via a Merkle tree [Mer89a] based on the random oracle, then queries the random oracle with the root of this tree to obtain randomness for the PCP verifier, and finally sends a non-interactive proof that contains the root as well as authentication paths for each query by the PCP verifier to the PCP; the NIROA verifier checks the proof's validity by checking that the PCP verifier's randomness is computed correctly through the random oracle, and that all authentication paths are valid. (The transformation can be viewed as a non-interactive variant of Kilian's protocol [Kil92, BG08] that uses ideas from the aforementioned Fiat–Shamir paradigm.)

Our transformation takes as input IOPs, for which both IPs and PCPs are special cases, and hence must support both (i) multiple rounds of interaction between the IOP prover and IOP verifier, as well as (ii) oracle access by the IOP verifier to the IOP prover messages. Given an instance $\mathbb{x}$, the NIROA prover thus uses the random oracle ρ to run the interaction between the IOP prover and the IOP verifier "in his head" in a way that combines the aforementioned two approaches, as follows. First, the NIROA prover computes an initial value $\sigma_0 := \rho(\mathbb{x})$. Then, for $i = 1, 2, \dots$, it simulates the i-th round by deriving the IOP verifier's i-th message m_i as $\rho(\mathbb{x}\|\sigma_{i-1})$, compressing the IOP prover's i-th message f_i via a Merkle tree to obtain the root rt_i, and computing the new value $\sigma_i := \rho(\mathsf{rt}_i\|\sigma_{i-1})$. The values $\sigma_0, \sigma_1, \dots$ are related by the Merkle–Damgård transform [Dam89, Mer89b] that, intuitively, enforces ordering between rounds. If there are $k(\mathbb{x})$ rounds of interaction, then $\rho(\mathbb{x}\|\sigma_{k(\mathbb{x})})$ is used as randomness for the queries to $f_1, \dots, f_{k(\mathbb{x})}$. The NIROA prover provides in the non-interactive proof all the roots rt_i, the final value $\sigma_{k(\mathbb{x})}$, the answers to the queries, and an authentication path for each query. This sketch omits several details; see Section 5.

Soundness analysis of the transformation. We prove that the soundness of the NIROA produced by the above transformation is tightly characterized by the state restoration soundness of the underlying IOP. This characterization comprises two arguments: an upper bound and a lower bound on the NIROA's soundness. We only discuss the upper bound here: proving that the soundness (error) of the NIROA is at most the soundness (error) of the IOP against state restoration attacks, up to small additive factors.

The upper bound essentially implies that all that a malicious prover $\tilde{\mathbb{P}}$ can do to attack the NIROA verifier is to conduct a state restoration attack against the underlying IOP verifier "in his own head": roughly, $\tilde{\mathbb{P}}$ can provide multiple inputs to the random oracle in order to induce multiple fresh samples of verifier messages for a given round so to find a lucky one, or instead go back to previous rounds and do the same there.

In more detail, the proof itself relies on a reduction: given a malicious prover $\tilde{\mathbb{P}}$ against the NIROA verifier, we show how to construct a corresponding malicious prover $\tilde{P}$ that conducts a state restoration attack against the underlying IOP verifier. We prove that the winning probability of $\tilde{P}$ is essentially the same as that of $\tilde{\mathbb{P}}$; moreover, we also prove that the reduction preserves the resources needed for the attack in the sense that if $\tilde{\mathbb{P}}$ asks at most m queries to the random oracle, then $\tilde{P}$ plays at most m rounds during the attack.

Intuitively, the construction of $\tilde{P}$ in terms of $\tilde{\mathbb{P}}$ must use some form of extraction: $\tilde{\mathbb{P}}$ outputs a non-interactive proof that contains only (i) the roots that (allegedly) are commitments to underlying IOP prover's messages, and (ii) answers to the IOP verifier's queries and corresponding authentication paths; in contrast, $\tilde{P}$ needs to actually output these IOP prover's messages. In principle, the malicious prover $\tilde{\mathbb{P}}$ may not have "in mind" any underlying IOP prover, and we must prove that, nevertheless, there is a way for $\tilde{P}$ to extract some IOP prover message for each round that convince the verifier with the claimed probability.

Our starting point is the extractor algorithm of Valiant [Val08] for the "CS proof" construction of Micali [Mic00]: Valiant proves that Micali's NIROA construction is a proof of knowledge by exhibiting an algorithm, let us call it *Valiant's extractor*, that recovers the underlying PCP whenever the NIROA prover convinces the NIROA verifier with sufficient probability. (In particular, our proof is not based on a "forking lemma" [PS96].) Our setting differs from Valiant's in that the IOP prover $\tilde{P}$ obtained from the NIROA prover $\tilde{\mathbb{P}}$ needs to be able to extract multiple times, "on the fly", while interacting with the IOP verifier; this more complex setting can potentially cause difficulties in terms of extractor size (e.g., if relying on rewinding the NIROA prover) or correlations (e.g., when extracting multiple times from the same NIROA prover). We tackle the more complex setting in two steps.

First, we prove an extractability property of Valiant's extractor and state it as a property of Merkle trees in the random oracle model (see Section A.1). Informally, we prove that, except with negligible probability, whenever an algorithm with access to a random oracle outputs multiple Merkle tree roots each accompanied with some number of (valid) authentication paths, it holds that Valiant's extractor run separately on each of these roots outputs a decommitment that is consistent with each of the values revealed in authentication paths relative to that root. We believe that distilling and proving this extractability property of Valiant's extractor is of independent interest.

Second, we show how the IOP prover $\tilde{P}$ can interact with an IOP verifier, by successively extracting messages to send, throughout the interaction, by invoking Valiant's extractor multiple times on $\tilde{\mathbb{P}}$ relative to different roots. The IOP prover $\tilde{P}$ does not rely on rewinding $\tilde{\mathbb{P}}$, and its complexity is essentially that of a single run of $\tilde{\mathbb{P}}$ plus a small amount of work.

Preserving proof of knowledge. We prove that the above soundness analysis can be adapted so that, if the underlying IOP is a proof of knowledge, then we can construct an extractor to show that the resulting NIROA is also a proof of knowledge. Moreover, the extractor algorithm only needs to inspect the queries and answers of one execution of $\tilde{\mathbb{P}}$ if the underlying IOP extractor does not use rewinding (known IOP constructions are of this type [BCGV16, BCG$^+$16]); such extractors are known as *straight line* [Pas03] or *online* [Fis05], and have very desirable properties [BW15].

Preserving zero knowledge. We prove that, if the underlying IOP is *honest-verifier* statistical zero knowledge, then the resulting NIROA is statistical zero knowledge (i.e., is a non-interactive statistical zero knowledge proof in the explicitly-programmable random oracle model). This is because the transformation uses a Merkle tree with suitable privacy guarantees (see Section A.2) to construct the NIROA. Indeed, the authentication path for a leaf in the Merkle tree reveals the sibling leaf, so one must ensure that the sibling leaf does not leak information about other values; this follows by letting leaves be commitments to the underlying values. A Merkle tree with privacy is similarly used by [IMS12, IMSX15], along with honest-verifier PCPs, to achieve zero knowledge in modifications of Kilian's [Kil92, BG08] and Micali's [Mic00] constructions. (Note that the considerations [HT98, DNRS03, GOSV14] seem to only apply to compilation of malicious-verifier IOPs, which neither [IMS12, IMSX15] nor we require.)

Understanding state restoration attacks. We prove tight upper and lower bounds to state restoration soundness in terms of the IOP's (standard) soundness and round complexity k. The upper bound takes the form of a reduction: given a b-round state-restoring malicious prover $\tilde{P}_{\mathrm{sr}}$ that makes the IOP verifier accept with probability s_{sr}, we construct a (non state-restoring) malicious prover $\tilde{P}$ that makes the IOP verifier accept with probability at least $\binom{b}{k+1}^{-1} s_{\mathrm{sr}}$. Informally, $\tilde{P}$ internally simulates $\tilde{P}_{\mathrm{sr}}$, while interacting with the "real" IOP verifier, as follows: $\tilde{P}$ first selects a random subset S of $\{1, \ldots, b\}$ with cardinality $k+1$, and lets $S[i]$ be the i-th smallest value in S; then, $\tilde{P}$ runs $\tilde{P}_{\mathrm{sr}}$ and simulates its state restoration attack on a "virtual" IOP verifier, executing round j (a) by interacting with the real verifier if $j = S[i]$ for some i; (b) by sampling fresh randomness otherwise. While this reduction appears wasteful (since it relies on S being a good guess), we show that there are IOPs for which the upper bound is tight. In other words, the sharp degradation as a function of round complexity (for large b, $\binom{b}{k+1} \approx b^{k+1}/(k+1)!$) is inherent for some choices of IOPs; this also gives a concrete answer to the intuition that compiling IOPs with large round complexity to NIROAs is "harder" (i.e., incurs in a greater soundness loss) than for IOPs with small round complexity. As for the lower bound on state restoration soundness, it takes the form of a universal state restoration attack that always achieves the lower bound; this bound is also tight.

While state restoration soundness may be far, in the worst case, from (standard) soundness for IOPs with large round complexity, it need not always be far. We thus investigate state restoration soundness for any particular IOP, and derive a simple attack strategy (which depends on the IOP) that we prove has optimal expected cost, where cost is the number of rounds until the prover wins. To do so, we "abstract away" various details of the proof system to obtain a simple game-theoretic notion, which we call *tree exploration games*, that pits a single player against Nature in reaching a node of a tree with label 1. Informally, such a game is specified by a rooted tree T and a predicate function ϕ that maps T's vertices to $\{0, 1\}$. The game proceeds in rounds: in the i-th round, a subtree $S_{i-1} \subseteq T$ is *accessible* to the player; the player picks a node $v \in S_{i-1}$, and Nature randomly samples a child u of v; the next accessible subtree is $S_i := S_{i-1} \cup \{u\}$. The initial S_0 is the set consisting of T's root vertex. The player wins in round r if there is $v \in S_r$ with $\phi(v) = 1$.

We establish a correspondence between state restoration attacks and strategies for tree exploration games, and then show a simple greedy strategy for such games with optimal expected cost. Via the correspondence, a strategy's cost determines whether the underlying IOP is strong or weak against sate restoration attacks.

2 Preliminaries

2.1 Basic notations

We denote the security parameter by λ. For $f\colon \{0,1\}^* \to \mathbb{R}$, we define $\hat{f}\colon \mathbb{N} \to \mathbb{R}$ as $\hat{f}(n) := \max_{x \in \{0,1\}^n} f(x)$.

Languages and relations. We denote by $\mathscr{R}$ a relation consisting of pairs $(\mathbb{x}, \mathbb{w})$, where $\mathbb{x}$ is the *instance* and $\mathbb{w}$ is the *witness*, and by $\mathscr{R}_n$ the restriction of $\mathscr{R}$ to instances of size n. We denote by $\mathscr{L}(\mathscr{R})$ the language corresponding to $\mathscr{R}$. For notational convenience, we define $\bar{\mathscr{L}}(\mathscr{R}_n) := \{\mathbb{x} \in \{0,1\}^n \mid \mathbb{x} \notin \mathscr{L}(\mathscr{R})\}$.

Random oracles. We denote by $\mathcal{U}(\lambda)$ the uniform distribution over all functions $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ (implicitly defined by the probabilistic algorithm that assigns, uniformly and independently at random, a λ-bit string to each new input). If ρ is sampled from $\mathcal{U}(\lambda)$, then we write $\rho \leftarrow \mathcal{U}(\lambda)$ and say that ρ is a *random oracle*. Given an oracle algorithm A, $\mathsf{NumQueries}(A, \rho)$ is the number of oracle queries that A^ρ makes. We say that A is *m-query* if $\mathsf{NumQueries}(A, \rho) \le m$ for any $\rho \in \mathcal{U}(\lambda)$ (i.e., for any ρ in $\mathcal{U}(\lambda)$'s support).

Statistical distance. The statistical distance between two discrete random variables X and Y with support V is $\Delta(X; Y) := \frac{1}{2} \sum_{v \in V} |\Pr[X = v] - \Pr[Y = v]|$. We say that X and Y are *δ-close* if $\Delta(X; Y) \le \delta$.

Remark 5. An oracle $\rho \in \mathcal{U}(\lambda)$ outputs λ bits. Occasionally we need ρ to output more than λ bits; in such cases (we point out where), we implicitly extend ρ's output via a simple strategy, e.g., we set $y := y_1 \| y_2 \| \cdots$ where $y_i := \rho(i \| x)$ and prefix 0 to all inputs that do not require an output extension.

2.2 Merkle trees

We use Merkle trees [Mer89a] based on random oracles as succinct commitments to long lists of values for which one can cheaply decommit to particular values in the list. Concretely, a *Merkle-tree scheme* is a tuple $\mathsf{MERKLE} = (\mathsf{MERKLE.GetRoot},$

$\mathsf{MERKLE.GetPath}, \mathsf{MERKLE.CheckPath})$ that uses a random oracle ρ sampled from $\mathcal{U}(\lambda)$ and works as follows.

- $\mathsf{MERKLE.GetRoot}^{\rho}(\mathbf{v}) \to \mathsf{rt}$. Given input list $\mathbf{v} = (v_i)_{i=1}^{n}$, the *root generator* $\mathsf{MERKLE.GetRoot}$ computes, in time $O_{\lambda}(n)$, a root rt of the Merkle tree over $\mathbf{v}$.
- $\mathsf{MERKLE.GetPath}^{\rho}(\mathbf{v}, i) \to \mathsf{ap}$. Given input list $\mathbf{v}$ and index i, the *authentication path generator* $\mathsf{MERKLE.GetPath}$ computes the authentication path ap for the i-th value in $\mathbf{v}$.
- $\mathsf{MERKLE.CheckPath}^{\rho}(\mathsf{rt}, i, v, \mathsf{ap}) \to b$. Given root rt, index i, input value v, and authentication path ap, the *path checker* $\mathsf{MERKLE.CheckPath}$ outputs $b = 1$ if ap is a valid path for v as the i-th value in a Merkle tree with root rt; the check can be carried out in time $O_{\lambda}(\log_2 n)$.

We assume that an authentication path ap contains the root rt, position i, and value v; accordingly, we define $\mathsf{Root}(\mathsf{ap}) := \mathsf{rt}$, $\mathsf{Position}(\mathsf{ap}) := i$, and $\mathsf{Value}(\mathsf{ap}) := v$.

Merkle trees are well known, so we do not review their construction. Less known, however, are the hiding and extractability properties of Merkle trees that we rely on in this work; we describe these in Appendix A.

2.3 Non-interactive random-oracle arguments

A *non-interactive random-oracle argument system* for a relation $\mathscr{R}$ with soundness $s\colon \{0,1\}^* \to [0,1]$ is a tuple $(\mathbb{P}, \mathbb{V})$, where $\mathbb{P}, \mathbb{V}$ are (oracle) probabilistic algorithms, that satisfies the following properties.

1. COMPLETENESS. For every $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$ and $\lambda \in \mathbb{N}$,

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

2. SOUNDNESS. For every $\mathbb{x} \notin \mathscr{L}(\mathscr{R})$, m-query $\tilde{\mathbb{P}}$, and $\lambda \in \mathbb{N}$,

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

Complexity measures. Beyond soundness, we consider other complexity measures. Given $p\colon \{0,1\}^* \to \mathbb{N}$, we say that $(\mathbb{P}, \mathbb{V})$ has proof length p if π has length $p(\mathbb{x}, \lambda)$. Given $t_{\mathrm{prv}}, t_{\mathrm{ver}}\colon \{0,1\}^* \to \mathbb{N}$, we say that $(\mathbb{P}, \mathbb{V})$ has prover time complexity t_{prv} and verifier time complexity t_{ver} if $\mathbb{P}^{\rho}(\mathbb{x}, \mathbb{w})$ runs in time $t_{\mathrm{prv}}(\mathbb{x}, \lambda)$ and $\mathbb{V}^{\rho}(\mathbb{x}, \pi)$ runs in time $t_{\mathrm{ver}}(\mathbb{x}, \lambda)$. In sum, we say that $(\mathbb{P}, \mathbb{V})$ has complexity $(s, p, t_{\mathrm{prv}}, t_{\mathrm{ver}})$ if $(\mathbb{P}, \mathbb{V})$ has soundness s, proof length p, prover time complexity t_{prv}, and verifier time complexity t_{ver}.

Proof of knowledge. Given $e\colon \{0,1\}^* \to [0,1]$, we say that $(\mathbb{P}, \mathbb{V})$ has proof of knowledge e if there exists a probabilistic polynomial-time algorithm $\mathbb{E}$ (the *extractor*) such that, for every $\mathbb{x}$, m-query $\tilde{\mathbb{P}}$, and $\lambda \in \mathbb{N}$,

$$\Pr\left[(\mathbb{x}, \mathbb{w}) \in \mathscr{R} \;\middle|\; \mathbb{w} \leftarrow \mathbb{E}^{\tilde{\mathbb{P}}}(\mathbb{x}, 1^m, 1^{\lambda})\right] \ge \Pr\left[\mathbb{V}^{\rho}(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \tilde{\mathbb{P}}^{\rho} \end{array}\right] - e(\mathbb{x}, m, \lambda) \ .$$

The notation $\mathbb{E}^{\tilde{\mathbb{P}}}(\mathbb{x}, 1^m, 1^\lambda)$ means that $\mathbb{E}$ receives as input $(\mathbb{x}, 1^m, 1^\lambda)$ and may obtain an output of $\tilde{\mathbb{P}}^\rho$ for choices of oracles ρ, as we now describe. At any time, $\mathbb{E}$ may send a λ-bit string z to $\tilde{\mathbb{P}}$; then $\tilde{\mathbb{P}}$ interprets z as the answer to its last query to ρ (if any) and then continues computing until it reaches either its next query θ or its output π; then this query or output is sent to $\mathbb{E}$ (distinguishing the two cases in some way); in the latter case, $\tilde{\mathbb{P}}$ goes back to the start of its computation (with the same randomness and any auxiliary inputs). Throughout, the code, randomness, and any auxiliary inputs of $\tilde{\mathbb{P}}$ are not available to $\mathbb{E}$.

Zero knowledge. Given $z\colon \{0,1\}^* \to [0,1]$, we say that $(\mathbb{P}, \mathbb{V})$ has z-statistical zero knowledge (in the explicitly-programmable random oracle model) if there exists a probabilistic polynomial-time algorithm $\mathbb{S}$ (the *simulator*) such that, for every $(\mathbb{x}, \mathbb{w}) \in \mathscr{R}$ and unbounded distinguisher D, the following two probabilities are $z(\mathbb{x}, \lambda)$-close:

$$\Pr\left[D^{\rho[\mu]}(\pi) = 1 \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\pi, \mu) \leftarrow \mathbb{S}^\rho(\mathbb{x}) \end{array}\right] \text{ and } \Pr\left[D^{\rho}(\pi) = 1 \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \mathbb{P}^\rho(\mathbb{x}, \mathbb{w}) \end{array}\right] .$$

Above, $\rho[\mu]$ is the function such that, given an input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise.

3 Interactive oracle proofs

We first define *interactive oracle protocols* and then *interactive oracle proof systems*.

3.1 Interactive oracle protocols

A k-round interactive oracle protocol between two parties, call them Alice and Bob, comprises k rounds of interaction. In the i-th round of interaction: Alice sends a message m_i to Bob, which he reads in full; then Bob replies with a message f_i to Alice, which she can query (via random access) in this and all later rounds. After the k rounds of interaction, Alice either accepts or rejects.

More precisely, let k be in $\mathbb{N}$ and A, B be two interactive probabilistic algorithms. A k-round interactive oracle protocol between A and B, denoted $\langle B, A\rangle$, works as follows. Let r_A, r_B denote the randomness for A, B and, for notational convenience, set $f_0 := \perp$ and $\mathsf{state}_0 := \perp$. For $i = 1, \dots, k$, in the i-th round: (i) Alice sends a message $m_i \in \{0,1\}^{u_i}$, where $(m_i, \mathsf{state}_i) := A^{f_0,\dots,f_{i-1}}(\mathsf{state}_{i-1}; r_A)$ and $u_i \in \mathbb{N}$; (ii) Bob sends a message $f_i \in \{0,1\}^{\ell_i}$, where $f_i := B(m_1, \dots, m_i; r_B)$ and $\ell_i \in \mathbb{N}$. The output of the protocol is $m_{\mathsf{fin}} := A^{f_0,\dots,f_k}(\mathsf{state}_k; r_A)$, and belongs to $\{0,1\}$.

The accepting probability of $\langle B, A\rangle$ is the probability that $m_{\mathsf{fin}} = 1$ for a random choice of r_A, r_B; this probability is denoted $\Pr[\langle B, A\rangle = 1]$ (leaving r_A, r_B implicit). The query complexity of $\langle B, A\rangle$ is the number of queries asked by A to any of the oracles during the k rounds. The proof complexity of $\langle B, A\rangle$ is the number of bits communicated by Bob to Alice (i.e., $\sum_{i=1}^{k} \ell_i$). The view of A in $\langle B, A\rangle$, denoted $\mathrm{View}_{\langle B,A\rangle}(A)$, is the random variable $(a_1, \dots, a_q, r_A)$ where a_j denotes the answer to the j-th query.

Public coins. An interactive oracle protocol is *public-coin* if Alice's messages are uniformly and independently random and Alice postpones any query to after the k-th round (i.e., all queries are asked when running $A^{f_0,\dots,f_k}(\mathsf{state}_k; r_A)$). We can thus take the randomness r_A to be of the form $(m_1,\dots,m_k,r)$, where r is additional randomness that A may use of to compute m_{fin} after the last round.

3.2 Interactive oracle proof systems

An *interactive oracle proof system* for a relation $\mathscr{R}$ with round complexity $k\colon \{0,1\}^* \to \mathbb{N}$ and soundness $s\colon \{0,1\}^* \to [0,1]$ is a tuple (P,V), where P,V are probabilistic algorithms, that satisfies the following properties.

1. COMPLETENESS. For every $(\mathbb{x},\mathbb{w}) \in \mathscr{R}$, $\langle P(\mathbb{x},\mathbb{w}), V(\mathbb{x})\rangle$ is a $k(\mathbb{x})$-round interactive oracle protocol with accepting probability 1.
2. SOUNDNESS. For every $\mathbb{x} \notin \mathscr{L}(\mathscr{R})$ and $\tilde{P}$, $\langle \tilde{P}, V(\mathbb{x})\rangle$ is a $k(\mathbb{x})$-round interactive oracle protocol with accepting probability at most $s(\mathbb{x})$.

Message lengths. We assume the existence of polynomial-time functions that determine the message lengths. Namely, for any instance $\mathbb{x}$ and malicious prover $\tilde{P}$, when considering the interactive oracle protocol $\langle \tilde{P}, V(\mathbb{x})\rangle$, the i-th messages m_i (from $V(\mathbb{x})$) and f_i (to $V(\mathbb{x})$) lie in $\{0,1\}^{u_i(\mathbb{x})}$ and $\{0,1\}^{\ell_i(\mathbb{x})}$ respectively.

Complexity measures. Beyond round complexity and soundness, we consider other complexity measures. Given $p,q\colon \{0,1\}^* \to \mathbb{N}$, we say that (P,V) has proof length p and query complexity q if the proof length and query complexity of $\langle \tilde{P}, V(\mathbb{x})\rangle$ are $p(\mathbb{x})$ and $q(\mathbb{x})$ respectively. (Note that $q(\mathbb{x}) \le p(\mathbb{x})$ and $p(\mathbb{x}) = \sum_{i=1}^{k(\mathbb{x})} \ell_i(\mathbb{x})$.) Given $t_{\mathrm{prv}}, t_{\mathrm{ver}}\colon \{0,1\}^* \to \mathbb{N}$, we say that (P,V) has prover time complexity t_{prv} and verifier time complexity t_{ver} if $P(\mathbb{x},\mathbb{w})$ runs in time $t_{\mathrm{prv}}(\mathbb{x})$ and $V(\mathbb{x})$ runs in time $t_{\mathrm{ver}}(\mathbb{x})$. In sum, we say that (P,V) has complexity $(k,s,p,q,t_{\mathrm{prv}},t_{\mathrm{ver}})$ if (P,V) has round complexity k, soundness s, proof length p, query complexity q, prover time complexity t_{prv}, and verifier time complexity t_{ver}.

Proof of knowledge. Given $e\colon \{0,1\}^* \to [0,1]$, we say that (P,V) has proof of knowledge e if there exists a probabilistic polynomial-time oracle algorithm E (the *extractor*) such that, for every $\mathbb{x}$ and $\tilde{P}$, $\Pr[(\mathbb{x}, E^{\tilde{P}}(\mathbb{x})) \in \mathscr{R}] \ge \Pr[\langle \tilde{P}, V(\mathbb{x})\rangle = 1] - e(\mathbb{x})$.[8] The notation $E^{\tilde{P}}(\mathbb{x})$ means that E receives as input $\mathbb{x}$ and may interact with $\tilde{P}$ via rewinding, as we now describe. At any time, E may send a partial prover-verifier transcript to $\tilde{P}$ and then receive $\tilde{P}$'s next message (which is empty for invalid transcripts) in the subsequent computation step; the code, randomness, and any auxiliary inputs of $\tilde{P}$ are not available to E.

Honest-verifier zero knowledge. Given $z\colon \{0,1\}^* \to [0,1]$, we say that (P,V) has z-statistical honest-verifier zero knowledge if there exists a probabilistic polynomial-time algorithm S (the *simulator*) such that, for every $(\mathbb{x},\mathbb{w}) \in \mathscr{R}$, $S(\mathbb{x})$ is $z(\mathbb{x})$-close to $\mathrm{View}_{\langle P(\mathbb{x},\mathbb{w}),V(\mathbb{x})\rangle}(V(\mathbb{x}))$.

Public coins. We say that (P,V) is *public-coin* if the underlying interactive oracle protocol is public-coin.

[8] Proof of knowledge e implies soundness $s := e$. The definition that we use is equivalent to the one in [BG93, Section 6] except that: (a) we use extractors that run in strict, rather than expected, probabilistic polynomial time; and (b) we extend the condition to hold for all $\mathbb{x}$, rather than for only those in $\mathscr{L}(\mathscr{R})$, so that proof of knowledge implies soundness.

4 State restoration attacks on interactive oracle proofs

We introduce state restoration attacks on interactive oracle proofs.

In an interactive oracle proof, a malicious prover $\tilde{P}$ works as follows: for each round i, $\tilde{P}$ receives the i-th verifier message m_i and then sends to the verifier a message f_i computed as a function of his own randomness and all the verifier messages received so far, i.e., $m_1, \dots, m_i$.

For the case of public-coin interactive oracle proof systems, we also consider a larger class of malicious provers, called *state-restoring provers*. Informally, a state-restoring prover receives in each round a verifier message as well as a *complete verifier state*, and then sends to the verifier a message and a previously-seen complete verifier state, which sets the verifier to that state; this forms a state restoration attack on the verifier.

More precisely, let (P, V) be a k-round public-coin interactive proof system (see Section 3.2) and $\mathtt{x}$ an instance. A complete verifier state cvs of $V(\mathtt{x})$ takes one of three forms: (1) the symbol null, which denotes the "empty" complete verifier state; (2) a tuple of the form $(m_1, f_1, \dots, m_i)$, with $i \in \{1, \dots, k(\mathtt{x})\}$, where each m_j is in $\{0,1\}^{u_j(\mathtt{x})}$ and each f_j is in $\{0,1\}^{\ell_j(\mathtt{x})}$; (3) a tuple of the form $(m_1, f_1, \dots, m_{k(\mathtt{x})}, f_{k(\mathtt{x})}, r)$ where each m_j and f_j is as in the previous case and r is the additional randomness of the verifier $V(\mathtt{x})$.

The interaction between a state-restoring prover $\tilde{P}$ and the verifier $V(\mathtt{x})$ is mediated through a game:

1. The game initializes the list $\mathsf{SeenStates}$ to be (null).
2. Repeat the following until the game halts and outputs:
   (a) The prover chooses a complete verifier state cvs in the list $\mathsf{SeenStates}$.
   (b) The game sets the verifier to cvs.
   (c) If $\mathsf{cvs} = \mathsf{null}$: the verifier samples a message m_1 in $\{0,1\}^{u_1(\mathtt{x})}$ and sends it to the prover; the game appends $\mathsf{cvs}' := (m_1)$ to the list $\mathsf{SeenStates}$.
   (d) If $\mathsf{cvs} = (m_1, f_1, \dots, m_{i-1})$ with $i \in \{2, \dots, k(\mathtt{x})\}$: the prover outputs a message f_{i-1} in $\{0,1\}^{\ell_{i-1}(\mathtt{x})}$; the verifier samples a message m_i in $\{0,1\}^{u_i(\mathtt{x})}$ and sends it to the prover; the game appends $\mathsf{cvs}' := \mathsf{cvs} \| f_{i-1} \| m_i$ to the list $\mathsf{SeenStates}$.
   (e) If $\mathsf{cvs} = (m_1, f_1, \dots, m_{k(\mathtt{x})})$: the prover outputs a message $f_{k(\mathtt{x})}$ in $\{0,1\}^{\ell_{k(\mathtt{x})}(\mathtt{x})}$; the verifier samples additional randomness r; the game appends $\mathsf{cvs}' := \mathsf{cvs} \| f_{k(\mathtt{x})} \| r$ to the list $\mathsf{SeenStates}$.
   (f) If $\mathsf{cvs} = (m_1, f_1, \dots, m_{k(\mathtt{x})}, f_{k(\mathtt{x})}, r)$: the verifier computes his decision $b := V^{f_0, \dots, f_{k(\mathtt{x})}}(\mathtt{x}, \mathsf{state}_{k(\mathtt{x})}; r_V)$ where $\mathsf{state}_{k(\mathtt{x})} := \emptyset$ and $r_V := (m_1, \dots, m_k, r)$; then the game halts and outputs b.

Note that there are two distinct notions of a round. *Verifier rounds* are the rounds played by the verifier within a single execution, as tracked by a complete verifier state cvs; the number of such rounds lies in the set $\{0, \dots, k(\mathtt{x})+1\}$ (the extra $(k(\mathtt{x})+1)$-th round represents the verifier V sampling r after receiving the last prover message). *Prover rounds* are all verifier rounds played by the prover across different verifier executions; the number of such rounds is the number of states in $\mathsf{SeenStates}$ above. Accordingly, for $b \in \mathbb{N}$, we say a prover is *b-round* if it plays at most b prover rounds during any interaction with any verifier.

Also note that the prover is not able to set the verifier to arbitrary states but only to previously-seen ones (starting with the empty state null); naturally, setting the verifier multiple times to the same state may yield distinct new states, because the verifier samples his message afresh each time. After being set to a state cvs, the verifier does one of three things: (i) if the number of verifier rounds in cvs is less than $k(\mathbb{x})$ (see Step 2c and Step 2d), the verifier samples a fresh next message; (ii) if the number of verifier rounds in cvs is $k(\mathbb{x})$ (see Step 2e), the verifier samples his additional randomness r; (iii) if cvs contains a full protocol execution (see Step 2f), the verifier outputs the decision corresponding to this execution. The second case means that the prover can set the verifier even *after* the conclusion of the execution (after r is sampled and known to the prover). The game halts only in the third case.

The above game between a state-restoring prover and a verifier yields corresponding notions of soundness and proof of knowledge. Below, we denote by $\Pr[\langle \tilde{P}, V(\mathbb{x})\rangle_{\mathrm{sr}} = 1]$ the probability that the state-restoring prover $\tilde{P}$ makes V accept $\mathbb{x}$ in this game.

Definition 1. *Given* $s_{\mathrm{sr}}, e_{\mathrm{sr}} \colon \{0,1\}^* \to [0,1]$*, a public-coin interactive oracle proof system* (P, V) *has*

- STATE RESTORATION SOUNDNESS s_{sr} *if, for every* $\mathbb{x} \notin \mathscr{L}(\mathscr{R})$ *and* b*-round state-restoring prover* $\tilde{P}$*,* $\Pr[\langle \tilde{P}, V(\mathbb{x})\rangle_{\mathrm{sr}} = 1] \leq s_{\mathrm{sr}}(\mathbb{x}, b)$.
- STATE RESTORATION PROOF OF KNOWLEDGE e_{sr} *if there exists a probabilistic polynomial-time algorithm* E_{sr} *(the* extractor*) such that, for every* $\mathbb{x}$ *and* b*-round state-restoring prover* $\tilde{P}$*,* $\Pr[(\mathbb{x}, E_{\mathrm{sr}}^{\tilde{P}}(\mathbb{x})) \in \mathscr{R}] \geq \Pr[\langle \tilde{P}, V(\mathbb{x})\rangle_{\mathrm{sr}} = 1] - e_{\mathrm{sr}}(\mathbb{x}, b)$.

Due to space limitations, our bounds on state restoration and our results on the corresponding tree exploration games are in the full version [BCS16].

5 From IOPs to non-interactive random-oracle arguments

We describe a transformation T such that if (P, V) is a public-coin interactive oracle proof system for a relation $\mathscr{R}$ then $(\mathbb{P}, \mathbb{V}) := T(P, V)$ is a non-interactive random-oracle argument system for $\mathscr{R}$. The transformation T runs in polynomial time: given as input code for P and V, it runs in time polynomial in the size of this code and then outputs code for $\mathbb{P}$ and $\mathbb{V}$.

Notation. For convenience, we split the random oracle ρ into two random oracles, denoted ρ_1 and ρ_2, as follows: $\rho_1(x) := \rho(1\|x)$ and $\rho_2(x) := \rho(2\|x)$. At a high level, we use ρ_1 for the verifier's randomness, and ρ_2 for Merkle trees and other hashing purposes. When counting queries, we count queries to both ρ_1 and ρ_2.

Construction of $\mathbb{P}$. The algorithm $\mathbb{P}$, given input $(\mathbb{x}, \mathbb{w})$ and oracle access to ρ:

1. Set $k := k(\mathbb{x})$, $q := q(\mathbb{x})$, $f_0 := \perp$, and $\sigma_0 := \rho_2(\mathbb{x})$.
2. Start running $P(\mathbb{x}, \mathbb{w})$ and, for $i = 1, \ldots, k$:
   (a) Compute the verifier message $m_i := \rho_1(\mathbb{x}\|\sigma_{i-1})$.
   (b) Give m_i to $P(\mathbb{x}, \mathbb{w})$ to obtain f_i.
   (c) Compute the Merkle-tree root $\mathsf{rt}_i := \mathsf{MERKLE.GetRoot}^{\rho_2}(f_i)$.
   (d) Compute the "root hash" $\sigma_i := \rho_2(\mathsf{rt}_i\|\sigma_{i-1})$.
3. Set $\mathsf{state}_k := \emptyset$ and $r_V := (m_1, \ldots, m_k, r)$, where $r := \rho_1(\mathbb{x}\|\sigma_k)$.
4. Run $V^{f_0,\ldots,f_k}(\mathbb{x}, \mathsf{state}_k; r_V)$ and compute an authentication path for each query. Namely, for $j = 1, \ldots, q$: if the j-th query is to the x_j-th bit of the y_j-th oracle, then compute

$\mathsf{ap}_j := \mathsf{MERKLE.GetPath}^{\rho_2}(f_{y_j}, x_j)$. (If MERKLE.GetRoot is probabilistic, then give the same randomness to MERKLE.GetPath as well.)

5. Set $\pi := \big((\mathsf{rt}_1, \ldots, \mathsf{rt}_k), (\mathsf{ap}_1, \ldots, \mathsf{ap}_q), \sigma_k\big)$. That is, π comprises the Merkle-tree roots, an authentication path for each query, and the final root hash.
6. Output π.

Construction of $\mathbb{V}$. The algorithm $\mathbb{V}$, given input $(\mathbb{x}, \tilde{\pi})$ and oracle access to ρ:

1. Set $k := k(\mathbb{x})$, $q := q(\mathbb{x})$, $f_0 := \bot$, and $\sigma_0 := \rho_2(\mathbb{x})$.
2. Parse $\tilde{\pi}$ as a tuple $\big((\tilde{\mathsf{rt}}_1, \ldots, \tilde{\mathsf{rt}}_k), (\tilde{\mathsf{ap}}_1, \ldots, \tilde{\mathsf{ap}}_q), \tilde{\sigma}_k\big)$.
3. For $i = 1, \ldots, k$:
   (a) Compute $m_i := \rho_1(\mathbb{x} \| \sigma_{i-1})$.
   (b) Compute $\sigma_i := \rho_2(\tilde{\mathsf{rt}}_i \| \sigma_{i-1})$.
4. Set $\mathsf{state}_k := \emptyset$ and $r_V := (m_1, \ldots, m_k, r)$, where $r := \rho_1(\mathbb{x} \| \sigma_k)$.
5. Compute $m_{\mathsf{fin}} := V^{f_0, \ldots, f_k}(\mathbb{x}, \mathsf{state}_k; r_V)$, answering the j-th query with the answer a_j in the path $\tilde{\mathsf{ap}}_j$.
6. If $\sigma_k \neq \tilde{\sigma}_k$, halt and output 0.
7. For $j = 1, \ldots, q$: if the j-th query is to the x_j-th bit of the y_j-th oracle and $\mathsf{MERKLE.CheckPath}^{\rho_2}(\mathsf{rt}_{y_j}, x_j, a_j, \tilde{\mathsf{ap}}_j) \neq 1$, halt and output 0.
8. Output m_{fin}.

6 Analysis of the transformation T

The theorem below specifies guarantees of the transformation T, described in Section 5.

Theorem 3 (IOP $\to$ NIROA). *For every relation $\mathscr{R}$, if (P, V) is a public-coin interactive oracle proof system for $\mathscr{R}$ with*

$$\begin{array}{rl} \text{round complexity} & k(\mathbb{x}) \\ \text{restricted state restoration soundness} & \bar{s}_{\mathrm{sr}}(\mathbb{x}, b) \\ \text{proof length} & p(\mathbb{x}) \\ \text{prover time} & t_{\mathrm{ver}}(\mathbb{x}) \\ \text{verifier time} & t_{\mathrm{prv}}(\mathbb{x}) \end{array}$$

then $(\mathbb{P}, \mathbb{V}) := T(P, V)$ is a non-interactive random-oracle argument system for $\mathscr{R}$ with

$$\begin{array}{rl} \text{soundness} & s'(\mathbb{x}, m, \lambda) := \bar{s}_{\mathrm{sr}}(\mathbb{x}, m) + 3(m^2+1)2^{-\lambda} \\ \text{proof length} & p'(\mathbb{x}, \lambda) := \big(k(\mathbb{x}) + q(\mathbb{x}) \cdot (\lceil \log_2 p(\mathbb{x}) \rceil + 2) + 1\big) \cdot \lambda \\ \text{prover time} & t'_{\mathrm{prv}}(\mathbb{x}, \lambda) := O_\lambda(k(\mathbb{x}) + p(\mathbb{x})) + t_{\mathrm{prv}}(\mathbb{x}) + t_{\mathrm{ver}}(\mathbb{x}) \\ \text{verifier time} & t'_{\mathrm{ver}}(\mathbb{x}, \lambda) := O_\lambda(k(\mathbb{x}) + q(\mathbb{x})) + t_{\mathrm{ver}}(\mathbb{x}) \end{array}$$ [9]

By construction, if $\langle P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}) \rangle$ has accepting probability δ, then the probability that $\mathbb{V}^\rho(\mathbb{x}, \mathbb{P}^\rho(\mathbb{x}, \mathbb{w}))$ accepts is δ. The complexities $p', t'_{\mathrm{prv}}, t'_{\mathrm{ver}}$ above also directly follow from the construction. Therefore, we are left to discuss soundness. Due to space limitations, the discussion of the soundness lower bound, as well as proof of knowledge and zero knowledge, are left to the full version [BCS16].

Let $\mathbb{x} \notin \mathscr{L}(\mathscr{R})$ and let $\tilde{\mathbb{P}}$ be an m-query prover for the non-interactive random-oracle argument system $(\mathbb{P}, \mathbb{V})$. We construct a prover $\tilde{P}$ (depending on $\mathbb{x}$ and $\tilde{\mathbb{P}}$) for the interactive oracle proof system (P, V), and show that $\tilde{P}$'s ability to cheat in a (restricted) state restoration attack is closely related to $\tilde{\mathbb{P}}$'s ability to cheat.

Construction of $\tilde{P}$. Given no inputs or oracles, the prover $\tilde{P}$ works as follows.

1. Let ρ_1, ρ_2 be tables mapping $\{0,1\}^*$ to $\{0,1\}^\lambda$, and let α be a table mapping λ-bit strings to verifier states. The tables are initially empty and are later populated with suitable values, during the simulation of $\tilde{\mathbb{P}}$. Intuitively, ρ_1, ρ_2 are used to simulate $\tilde{\mathbb{P}}$'s access to a random oracle, while α is used to keep track of which verifier states $\tilde{\mathbb{P}}$ has "seen in his mind".
2. Draw $\sigma_0 \in \{0,1\}^\lambda$ at random, and define $\rho_2(\mathbb{x}) := \sigma_0$ (i.e., the oracle ρ_2 replies the query $\mathbb{x}$ with the answer σ_0). After receiving V's first message m_1, also define $\rho_1(\mathbb{x}\|\sigma_0) := m_1$ and $\alpha(\sigma_0) := (m_1)$.
3. Begin simulating $\tilde{\mathbb{P}}^\rho$ and, for $i = 1, \ldots, m$:

(a) Let θ_i be the i-th query made by $\tilde{\mathbb{P}}^\rho$.

(b) If θ_i is a query to a location of ρ_1 that is defined, respond with $\rho_1(\theta_i)$. Otherwise (if θ_i to an undefined location of ρ_1), draw a string in $\{0,1\}^\lambda$ at random and respond with it. Then go to the next iteration of Step 3.

(c) If θ_i is a query to a location of ρ_2 that is defined, respond with $\rho_2(\theta_i)$; then go to the next iteration of Step 3. Otherwise (if θ_i is to an undefined location of ρ_2), draw a string $\sigma' \in \{0,1\}^\lambda$ at random and respond with it; then continue as follows.

(d) Let rt be the first λ bits of θ_i, and σ be the second λ bits. (If the length of θ_i is not 2λ bits, go to the next iteration of Step 3.) If $\alpha(\sigma)$ is defined, let $\mathsf{cvs} := \alpha(\sigma)$ and let j be the number of verifier rounds in the state cvs. If $\alpha(\sigma)$ is not defined, go to the next iteration of Step 3.

(e) Find the query $\theta_{i^\star}$ whose result is rt. If this query is not unique, or there is no such query, then answer the verifier V with some dummy message (e.g., an all zero message of the correct length) and skip to Step 3g. Otherwise, note the index $i^\star$ and continue.

(f) Compute $f := \mathsf{VE}^{\rho_2}(\tilde{\mathbb{P}}, \ell_j(\mathbb{x}), i^\star, i)$; if VE aborts, set $f := 0^{\ell_j(\mathbb{x})}$. Recall that $\ell_j(\mathbb{x})$ is the length of the prover message in the j-th verifier round, and VE is Valiant's extractor (see Section A.1). Also note that VE does not query ρ_2 on any value outside the table, because we have already simulated the first i queries of $\tilde{\mathbb{P}}$ (see Remark 6).

(g) Send the message f to the verifier and tell the game to set the verifier to the state cvs. (Whether cvs lies in the set $\mathsf{SeenStates}$ is a matter of analysis further below.) If the game is not over, the verifier replies with a new message m'. (If $j = k(\mathbb{x}) + 1$, for the purposes of the proof, we interpret m' as the additional randomness r.) The game adds $\mathsf{cvs}' := \mathsf{cvs}\|f\|m'$ to $\mathsf{SeenStates}$. The prover defines $\rho_1(\mathbb{x}\|\sigma') := m'$ and $\alpha(\sigma') := \mathsf{cvs}'$.

Analysis of $\tilde{P}$. We now analyze $\tilde{P}$. We first prove a simple lemma, and then discuss $\tilde{P}$'s ability to cheat.

Lemma 1. *Let A be an m-query algorithm. Define:*

1. *E_1 to be the event that A^{ρ_2} outputs $\mathbb{x} \in \{0,1\}^n$, $\mathsf{rt}_1, \ldots, \mathsf{rt}_{k(\mathbb{x})} \in \{0,1\}^\lambda$, and $\sigma_{k(\mathbb{x})} \in \{0,1\}^\lambda$ that satisfy the recurrence $\sigma_0 = \rho_2(\mathbb{x})$ and $\sigma_i = \rho_2(\mathsf{rt}_i\|\sigma_{i-1})$ for all $i \in \{1, \ldots, k(\mathbb{x})\}$;*
2. *E_2 to be the event that A^{ρ_2} queries ρ_2 at $\mathbb{x}, \mathsf{rt}_1\|\sigma_0, \ldots, \mathsf{rt}_{k(\mathbb{x})}\|\sigma_{k(\mathbb{x})-1}$ (in order) and, if any rt_i is the result of a query, this query first occurs before $\mathsf{rt}_i\|\sigma_{i-1}$.*

Then

$$\Pr\left[(\neg E_1) \vee E_2 \mid \rho_2 \leftarrow \mathcal{U}(\lambda)\right] \geq 1 - (m^2+1)2^{-\lambda} \enspace .$$

Proof. Let rt_0 be $\mathbb{x}$ and σ_{-1} be the empty string. Suppose, by contradiction, that E_1 occurs and E_2 does not. Then there exists $i \in \{0, \ldots, k(\mathbb{x})\}$ for which at least one of the following holds: (i) A^{ρ_2} does not query $\mathsf{rt}_i \| \sigma_{i-1}$; (ii) A^{ρ_2} queries $\mathsf{rt}_{i+1} \| \sigma_i$ before it queries $\mathsf{rt}_i \| \sigma_{i-1}$; (iii) rt_i is the result of a query but this query first occurs after $\mathsf{rt}_i \| \sigma_{i-1}$. Consider the largest index i for which one of the above holds.

In case (i), the behavior of A^{ρ_2} is independent of $\rho_2(\mathsf{rt}_i \| \tilde{\sigma}_{i-1})$. If $i = k(\mathbb{x})$, then the output $\sigma_{k(\mathbb{x})}$ of A^{ρ_2} equals $\rho_2(\mathsf{rt}_{k(\mathbb{x})} \| \sigma_{k(\mathbb{x})-1})$ with probability $2^{-\lambda}$. If $i < k(\mathbb{x})$, then there is a sequence of queries $\mathsf{rt}_{i+1} \| \tilde{\sigma}_i, \ldots, \mathsf{rt}_{k(\mathbb{x})} \| \tilde{\sigma}_{k(\mathbb{x})-1}$ for which $\tilde{\sigma}_i = \rho_2(\mathsf{rt}_i \| \tilde{\sigma}_{i-1})$ for $i = 1, \ldots, k(\mathbb{x}) - 1$ and $\rho_2(\mathsf{rt}_{k(\mathbb{x})} \| \tilde{\sigma}_{k(\mathbb{x})-1}) = \sigma_{k(\mathbb{x})}$. If this sequence is not unique, then A^{ρ_2} has found a collision. Otherwise, the unique sequence has $\tilde{\sigma}_i = \sigma_i$ for each i, which occurs with probability at most $2^{-\lambda}$.

In cases (ii) and (iii), A^{ρ_2} has found a collision, since $\sigma_i = \rho_2(\mathsf{rt}_i \| \sigma_{i-1})$. The fraction of oracles ρ_2 for which A^{ρ_2} finds a collision is at most $m^2 2^{-\lambda}$. Overall, the probability that E_2 does not occur and E_1 does is, by the union bound, at most $(m^2+1)2^{-\lambda}$.

We now state and prove the lemma about the soundness s' as stated in Theorem 3.

Lemma 2. *Define* $\epsilon := \Pr\left[\mathbb{V}^{\rho}(\mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \pi \leftarrow \tilde{\mathbb{P}}^{\rho} \end{array}\right]$. *Then there exists* $b \in \mathbb{N}$ *with* $b \leq m$ *such that* $\tilde{P}$ *is a* b*-round state-restoring prover that makes* V *accept with probability at least* $\epsilon - 3(m^2+1)2^{-\lambda}$.

Proof. We first note that $\tilde{P}$ described plays no more than m rounds, because $\tilde{P}$ sends a message to the verifier V only in response to $\tilde{\mathbb{P}}$ making a query. Next, we define some useful notions, and use them to prove three claims which together imply the lemma.

DEFINITION 4. We say $\rho \in \mathcal{U}(\lambda)$ is *good* if

1. The verifier accepts relative to ρ, i.e., $\mathbb{V}^{\rho}(\mathbb{x}, \pi) = 1$ where $\pi \leftarrow \tilde{\mathbb{P}}^{\rho}$.
2. Parsing π as $\big((\tilde{\mathsf{rt}}_1, \ldots, \tilde{\mathsf{rt}}_{k(\mathbb{x})}), (\tilde{\mathsf{ap}}_1, \ldots, \tilde{\mathsf{ap}}_q), \tilde{\sigma}_{k(\mathbb{x})}\big)$ and setting $\sigma_0 := \mathbb{x}$, for each $i \in \{1, \ldots, k(\mathbb{x})\}$, where $\sigma_i := \rho_2(\tilde{\mathsf{rt}}_i \| \sigma_{i-1})$, there exist indices $1 \leq j_1 < \cdots < j_k \leq m$ such that:
   (a) $\tilde{\mathbb{P}}^{\rho}$'s j_i-th query is to ρ_2 at $\tilde{\mathsf{rt}}_i \| \sigma_{i-1}$;
   (b) if rt_i is the result of a query, this query first occurs before j_i;
   (c) if $\tilde{\mathbb{P}}^{\rho}$ queries ρ_1 at $\mathbb{x} \| \sigma_i$, then this query occurs *after* query j_i;
   (d) if there exists l such that $\mathsf{Root}(\tilde{\mathsf{ap}}_l) = \tilde{\mathsf{rt}}_i$, there is a unique (up to duplicate queries) $a_i \in \{0, \ldots, j_i\}$ such that $\rho_2(\theta_{a_i}) = \tilde{\mathsf{rt}}_i$ and, for every $i_{\mathsf{max}} \in \{a_i, \ldots, j_i\}$, $\mathbf{v} := \mathsf{VE}^{\rho_2}(A, \ell_i, a_i, i_{\mathsf{max}})$ is such that, for all l with $\mathsf{Root}(\tilde{\mathsf{ap}}_l) = \tilde{\mathsf{rt}}_i$, $\mathsf{Value}(\tilde{\mathsf{ap}}_l)$ equals the $\mathsf{Position}(\tilde{\mathsf{ap}}_l)$-th value in $\mathbf{v}$; we say $\mathbf{v}$ is *extracted at* i if this holds.
3. $\tilde{\sigma}_{k(\mathbb{x})} = \sigma_{k(\mathbb{x})}$.

DEFINITION 5. We say that $\tilde{\mathbb{P}}$ *chooses* $\rho \in \mathcal{U}(\lambda)$ if for every query θ made by $\tilde{\mathbb{P}}^{\rho}$ to its oracle, $\tilde{P}$ supplies it with $\rho(\theta)$ (ignoring whether this response comes from $\tilde{P}$ itself or the messages sent by V; this choice is fixed for a given ρ).

CLAIM 6. $(\tilde{P}, V)$ chooses $\rho \in \mathcal{U}(\lambda)$ uniformly at random.

Whenever the simulation of $\tilde{\mathbb{P}}$ makes a query, $\tilde{P}$ responds consistently, either with a uniformly randomly drawn string of its own, or the uniform randomness provided by V. This is equivalent in distribution to drawing ρ uniformly at random at the beginning of the protocol. ■

CLAIM 7. For any choice of randomness such that $\tilde{P}$ chooses a good ρ, $\tilde{P}$ makes $V(\mathbb{x})$ accept with a state restoration attack.

We begin by defining a property of the map α.

DEFINITION 8. For $i = 0, \ldots, k$, we say that α is *correct at* i if, immediately before $\tilde{\mathbb{P}}$'s j_{i+1}-th query is simulated (for $i = k$, at the end of the simulation), it holds that $\alpha(\sigma_i) = (\rho_1(\mathbb{x}\|\sigma_0), f_1, \ldots, \rho_1(\mathbb{x}\|\sigma_i))$, where for each $l \in \{1, \ldots, i\}$, f_l is extracted at l (see Condition 2d above), and $\alpha(\sigma_i) \in \mathsf{SeenStates}$.

We show by induction that α is correct at i for every $i \in \{0, \ldots, k\}$. First, α is correct at 0 since $\alpha(\sigma_0) = (\rho_1(\mathbb{x}\|\sigma_0))$ by construction. Suppose that α is correct at $i-1$. When $\tilde{\mathbb{P}}^\rho$ queries $\tilde{\mathsf{rt}}_i\|\sigma_{i-1}$ (i.e., query θ_{j_i}), $\tilde{P}$ restores $\alpha(\sigma_{i-1}) \in \mathsf{SeenStates}$. By Condition 2d, f_i is extracted at i. In Step 3g, $\rho_1(\mathbb{x}\|\sigma_i)$ is set to the message (or, similarly, internal randomness) sent by V in this round, which is possible by Condition 2c. The newly stored state is then $\alpha(\sigma_i) = (\rho_1(\mathbb{x}\|\sigma_0), f_1, \ldots, \rho_1(\mathbb{x}\|\sigma_{i-1}), f_i, \rho_1(\mathbb{x}\|\sigma_i)) \in \mathsf{SeenStates}$. This state is stored before query j_{i+1} by Condition 2a, and so α is correct at i.

Hence $\tilde{\mathbb{P}}$ sends a state $\alpha(\sigma_k) = (\rho_1(\mathbb{x}\|\sigma_1), f_1, \ldots, \rho_1(\mathbb{x}\|\sigma_k)) \in \mathsf{SeenStates}$. Since $\mathbb{V}$'s simulation of V accepts with this state, so does the real V when interacting with $\tilde{\mathbb{P}}$. ■

CLAIM 9. The probability that $\rho \in \mathcal{U}(\lambda)$ is good is at least $\epsilon - 3(m^2+1)2^{-\lambda}$.

By assumption, the density of oracles satisfying Condition 1 is ϵ. Lemma 1 implies that the density of oracles satisfying Condition 1 but not satisfying Condition 2a, Condition 2b, and Condition 3 is at most $(m^2+1)2^{-\lambda}$.[10] The density of oracles failing to satisfy Condition 2c is at most $m^2 2^{-\lambda}$, since this implies a 'collision' (in the sense of Lemma 3) between ρ_1 and ρ_2. Finally, the density of oracles satisfying Condition 1, Condition 2a, and Condition 2b, but not Condition 2d is at most $(m^2+1)2^{-\lambda}$, by Lemma 3 and Condition 2b (where Condition 2b allows us to restrict the possible values for a_i to $0 \le a_i < j_i$).

By the union bound, the density of good oracles ρ is at least $\epsilon - 3(m^2+1)2^{-\lambda}$. ■

Combining the claims, we deduce that $\tilde{P}$ makes V accept with probability at least $\epsilon - 3(m^2+1)2^{-\lambda}$ with a state restoration attack. Finally, note that this state restoration attack is restricted because $\tilde{P}$ never requests to set V to the empty verifier state null.

[10] More precisely, we apply Lemma 1 to an algorithm $\tilde{\mathbb{P}}$ that does not itself output $\mathbb{x}$ but this does not affect the lemma's validity because we can substitute into the definition of the event E_1 the fixed instance $\mathbb{x}$.

A Extractability and privacy of Merkle trees

We describe the specific extractability and privacy properties of Merkle trees that we rely on in this work.

A.1 Extractability

We rely on a certain extractability property of Merkle trees: there is an efficient procedure for extracting the committed list in a Merkle-tree scheme. We call the procedure *Valiant's extractor*, and denote it by VE, because it is described in [Val08]. Our presentation of the extractor and its guarantee differs from [Val08] because our use of it in this work requires "distilling" a more general property; see Lemma 3 below.

The extractor. For any oracle algorithm A, integers $\ell, i^\star, i_{\mathsf{max}} > 0$ with $i^\star \in \{1, \ldots, i_{\mathsf{max}}\}$, and ρ sampled from $\mathcal{U}(\lambda)$, the procedure VE, given input $(A, \ell, i^\star, i_{\mathsf{max}})$ and with oracle access to ρ, works as follows.

1. Run A^ρ until it has asked i_{max} unique queries to ρ (and abort if A^ρ asks fewer than i_{max}). Along the way, record the queries $\theta_1, \ldots, \theta_{i_{\mathsf{max}}}$ and answers $\rho(\theta_1), \ldots, \rho(\theta_{i_{\mathsf{max}}})$, in order and omitting duplicates.
2. Parse each query θ_i as $\theta_i^0 \| \theta_i^1$ where θ_i^0 are the first λ bits of θ_i and θ_i^1 the second λ bits. For brevity, we write $z \in \theta_i$ if $z = \theta_i^0$ or $z = \theta_i^1$. (If a query has length not equal to 2λ, then $z \notin \theta_i$ for all z.)
3. If there exist indices i, j such that $i \neq j$ and $\rho(\theta_i) = \rho(\theta_j)$, abort.
4. If there exist indices i, j such that $i \leq j$ and $\rho(\theta_j) \in \theta_i$, abort.
5. Construct a directed graph G with nodes $V = \{\theta_1, \ldots, \theta_{i_{\mathsf{max}}}\}$ and edges $E = \{(\theta_i, \theta_j) : \rho(\theta_j) \in \theta_i\}$. Note that G is acyclic, every node has out-degree ≤ 2, and $\theta_1, \ldots, \theta_{i_{\mathsf{max}}}$ is a (reverse) topological ordering.
6. Output $\mathbf{v}$, the string obtained by traversing in order the first ℓ leaf nodes of the depth-$\lceil \log_2 \ell \rceil$ binary tree rooted at $\theta_{i^\star}$ and recording the first bit of each node. If any such node does not exist, set this entry to 0.

A sample execution of the extractor is depicted in Figure 1.

Remark 6. The queries to ρ asked by $\mathsf{VE}^\rho(A, \ell, i^\star, i_{\mathsf{max}})$ equals the first i_{max} queries to ρ asked by A^ρ (provided that A does not ask fewer than i_{max} queries). Later on we use this fact.

The extractor's guarantee. We interpret A's output as containing a (possibly empty) list of tuples of the form $(\mathsf{rt}, i, v, \mathsf{ap})$, where rt is a root, i an index, v a value, and ap an authentication path.[11] We define the following events:

(i) E_1 is the event that, for each tuple $(\mathsf{rt}, i, v, \mathsf{ap})$ output by A^ρ, $\mathsf{MERKLE.CheckPath}(\mathsf{rt}, i, v, \mathsf{ap}) = 1$;

(ii) E_2 is the event that, for each $\mathsf{rt} \in \{0,1\}^\lambda$, there exists $\ell_{\mathsf{rt}} \in \mathbb{N}$ such that if A^ρ outputs a tuple of the form $(\mathsf{rt}, \cdot, \cdot, \mathsf{ap})$ then ap is an authentication path having the correct length for a ℓ_{rt}-leaf Merkle tree;

[11] Note that A's output may contain additional information not of the above form; if so, we simply ignore it for now.

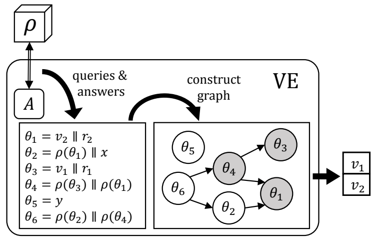

Fig. 1: A diagram of an execution of Valiant's extractor VE, with input parameters $\ell = 2$, $i^\star = 4$, and $i_{\mathsf{max}} = 6$.

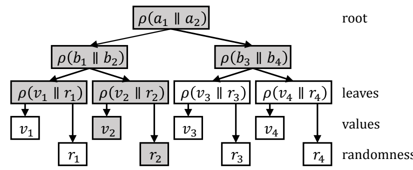

Fig. 2: A diagram of the data structure of a Merkle tree with privacy. An authentication path for v_2 is shaded; the corresponding truncated authentication path is the same minus r_2 and v_2.

(iii) E_3 is the event that, for every $\mathsf{rt} \in \{0,1\}^\lambda$ such that A^ρ outputs some tuple of the form $(\mathsf{rt}, \cdot, \cdot, \cdot)$, there is a unique $j_{\mathsf{rt}} \in \{0, \dots, \mathsf{NumQueries}(A, \rho)\}$ such that $\rho(\theta_{j_{\mathsf{rt}}}) = \mathsf{rt}$ and, for every $i_{\mathsf{max}} \in \{j_{\mathsf{rt}}, \dots, \mathsf{NumQueries}(A, \rho)\}$, $\mathbf{v} := \mathsf{VE}^\rho(A, \ell_{\mathsf{rt}}, j_{\mathsf{rt}}, i_{\mathsf{max}})$ is such that $\mathbf{v}$'s i-th entry equals v_i for any tuple of the form $(\mathsf{rt}, i, v, \mathsf{ap})$ output by A^ρ.

The extractability property that we rely on is the following.

Lemma 3. *Let A^ρ be a m-query algorithm. Then*

$$\Pr\left[(\neg(E_1 \wedge E_2)) \vee E_3 \mid \rho \leftarrow \mathcal{U}(\lambda)\right] \geq 1 - (m^2 + 1)2^{-\lambda} \ .$$

Proof. Observe the following.

– By the union bound, the probability that there exist indices i, j such that $(i \neq j) \wedge (\rho(\theta_i) = \rho(\theta_j))$ or $(i \leq j) \wedge (\rho(\theta_j) \in \theta_i)$ is at most $m^2 2^{-\lambda}$. If this occurs, we say that A^ρ has found a *collision*.
– The probability that, for a tuple $(\mathsf{rt}, i, v, \mathsf{ap})$ output by A^ρ such that $\mathsf{MERKLE.CheckPath}(\mathsf{rt}, i, v, \mathsf{ap}) = 1$, the authentication path ap contains a node with no corresponding query is at most $2^{-\lambda}$, since this would mean that A^ρ has 'guessed' the answer to the query. In other words, no matter what strategy A uses to generate the result, if it does not query the oracle on this input then it can perform no better than chance.

Now suppose that $E_1 \wedge E_2$ occurs with probability δ. Then, with probability at least $\delta - (m^2 + 1)2^{-\lambda}$: (a) for each root rt output by A^ρ there is a unique query $\theta_{i^\star}$ such that $\rho(\theta_{i^\star}) = \mathsf{rt}$; (b) for each root rt output by A^ρ, if an authentication path ap claims to have root rt then ap appears in the tree rooted at $\theta_{i^\star}$ in G; and (c) the condition in the VE's Step 3 or Step 4 does not hold. In such a case we may take $j_{\mathsf{rt}} := i^\star$, and then $\mathsf{VE}^\rho(A, \ell_{\mathsf{rt}}, j_{\mathsf{rt}}, i_{\mathsf{max}})$ outputs a list $\mathbf{v}$ with the desired property. Hence, $\Pr[E_1 \wedge E_2 \wedge E_3] \geq \delta - (m^2 + 1)2^{-\lambda}$. The predicate is also satisfied if $\neg(E_1 \wedge E_2)$ occurs, which is the case with probability $1 - \delta$ and is disjoint from $E_1 \wedge E_2 \wedge E_3$. The lemma follows.

A.2 Privacy

We rely not only on the fact that the root rt of a Merkle tree is hiding, but also on the fact that an authentication path ap reveals no information about values other than the decommitted one. The latter property can be ensured via a slight tweak of the standard construction of Merkle trees: when committing to a list $\mathbf{v} = (v_i)_{i=1}^n$, the i-th leaf is not v_i but, instead, is a hiding commitment to v_i. In our case, we will store the value $\rho(v_i \| r_i)$ in the i-th leaf, where $r_i \in \{0,1\}^{2\lambda}$ is drawn uniformly at random; see Figure 2. (An authentication path for v_i then additionally includes r_i, and path verification is modified accordingly.) In what follows, we regard $\rho(v_i \| r_i)$ as a *leaf*, rather than v_i; moreover, a *truncated authentication path* ap'_i is identical to ap_i except that it does not contain r_i or v_i, and the *truncated Merkle tree* for $\mathbf{v}$ is $T'_{\mathbf{v}} := (\mathsf{ap}'_i)_{1 \le i \le n}$. Note that the *same* randomness $\mathbf{r} \in \{0,1\}^{2\lambda n}$ is used by $\mathsf{MERKLE.GetRoot}$ and $\mathsf{MERKLE.GetPath}$ (to be "in sync").

We summarize the privacy property of Merkle trees as above via the following definition and lemma.

Definition 2. *A Merkle-tree scheme has $z(n,\lambda)$-statistical privacy if there exists a probabilistic polynomial-time simulator S such that, for every list $\mathbf{v} = (v_i)_{i=1}^n$ and unbounded distinguisher D, the following two probabilities are $z(n,\lambda)$-close:*

$$\Pr_{\mathbf{r}}\left[\begin{array}{c} I \subseteq \{1,\ldots,n\} \\ D^{\rho}(\mathsf{rt}, (\mathsf{ap}_i)_{i\in I}) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ \mathsf{rt} \leftarrow \mathsf{MERKLE.GetRoot}^{\rho}(\mathbf{v};\mathbf{r}) \\ I \leftarrow D^{\rho} \\ \forall\, i \in I\,,\ \mathsf{ap}_i \leftarrow \mathsf{MERKLE.GetPath}^{\rho}(\mathbf{v}, i; \mathbf{r}) \end{array}\right]$$

and

$$\Pr\left[\begin{array}{c} I \subseteq \{1,\ldots,n\} \\ D^{\rho}(\mathsf{rt}, (\mathsf{ap}_i)_{i\in I}) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ I \leftarrow D^{\rho} \\ (\mathsf{rt}, (\mathsf{ap}_i)_{i\in I}) \leftarrow S^{\rho}(n, (i, v_i)_{i\in I}) \end{array}\right] .$$

We make no assumption on the power of the distinguisher D in the definition above. In particular, D may query the random oracle ρ at every input, and use the information to attempt to learn v_i for some $i \notin I$. For example, for some ρ, it is the case that $\Pr_r[v = 1 \mid \rho(v\|r) = x] \gg \Pr_r[v = 0 \mid \rho(v\|r) = x]$ for $x = \rho(v_2\|r_2)$, in which case D can determine v_2 from ap_1 with good accuracy. The next (easy to prove) lemma shows that the probability that D gains a significant statistical advantage in this way (or otherwise) is negligible in λ.

Lemma 4. *There exists a Merkle-tree scheme having $z(n,\lambda)$-statistical privacy with $z(n,\lambda) := n2^{-\lambda/4+2}$.*

References

ALM$^+$92. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. 1992.

ALM$^+$98. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *JACM*, 1998.

AS98. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *JACM*, 1998.

Bab85. László Babai. Trading group theory for randomness. In *STOC '85*, 1985.

Bab90. Laszlo Babai. E-mail and the Unexpected Power of Interaction. Technical report, University of Chicago, Chicago, IL, USA, 1990.

BBP04. Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *EUROCRYPT '04*, 2004.

BC86. Gilles Brassard and Claude Crépeau. Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond. In *FOCS '86*, 1986.

BCC88. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 1988.

BCG$^+$16. Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Short interactive oracle proofs with constant query complexity, via composition and sumcheck, 2016. Crypto ePrint 2016/324.

BCGV16. Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *TCC '16-A*, 2016.

BCI$^+$13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC '13*, 2013.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs, 2016. Crypto ePrint 2016/116.

BD16. Allison Bishop and Yevgeniy Dodis. Interactive coding for interactive proofs. In *TCC '16-A*, 2016.

BDG$^+$13. Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. Why "Fiat-Shamir for proofs" lacks a proof. In *TCC '13*, 2013.

BFL90. László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *SFCS '90*, 1990.

BFLS91. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC '91*, 1991.

BG93. Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, 1993.

BG08. Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comp.*, 2008.

BGGL01. Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS '01*, 2001.

BGH$^+$04. Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In *STOC '04*, 2004.

BGKW88. Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *STOC '88*, 1988.

BHZ87. Ravi B. Boppana, Johan Håstad, and Stathis Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 1987.

BKK$^+$13. Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate PCPs for Circuit-SAT with sublinear query complexity. In *FOCS '13*, 2013.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93*, 1993.

BS08. Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comp.*, 2008.

BW15. David Bernhard and Bogdan Warinschi. On limitations of the Fiat–Shamir transformation. ePrint 2015/712, 2015.

CGH04. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *JACM*, 2004.

COPV13. Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, and Ivan Visconti. Simultaneous resettability from one-way functions. In *FOCS '13*, 2013.

CPSV16. Michele Ciampi, Giuseppe Persiano, Luisa Siniscalchi, and Ivan Visconti. A transform for NIZK almost as efficient and general as the Fiat-Shamir transform without programmable random oracles. In *TCC '16-A*, 2016.

Dam89. Ivan Damgård. A design principle for hash functions. In *CRYPTO '89*, 1989.

DNRS03. Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. *JACM*, 2003.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO '05*, 2005.

FRS88. Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. 1988.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO '86*, 1986.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT '13*, 2013.

GH98. Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Information Processing Letters*, 1998.

GIMS10. Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *CRYPTO'10*, 2010.

GK03. Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *FOCS '03*, 2003.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for Muggles. In *STOC '08*, 2008.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comp.*, 1989.

GOSV14. Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In *STOC '14*, 2014.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT '10*, 2010.

GS86. Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC '86*, 1986.

GVW02. Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 2002.

HS00. Prahladh Harsha and Madhu Sudan. Small PCPs with low query complexity. *Computational Complexity*, 2000.

HT98. Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *CRYPTO '98*, 1998.

IKM09. Tsuyoshi Ito, Hirotada Kobayashi, and Keiji Matsumoto. Oracularization and two-prover one-round interactive proofs against nonlocal strategies. In *CCC '09*, 2009.

IKO07. Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *CCC '07*, 2007.

IMS12. Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. On efficient zero-knowledge PCPs. In *TCC '12*, 2012.

IMSX15. Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. On zero-knowledge PCPs: Limitations, simplifications, and applications, 2015. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`.

Ito10. Tsuyoshi Ito. Polynomial-space approximation of no-signaling provers. In *ICALP '10*, 2010.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC '92*, 1992.

KR08. Yael Kalai and Ran Raz. Interactive PCP. In *ICALP '08*, 2008.

KR09. Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO '09*, 2009.

KRR13. Yael Kalai, Ran Raz, and Ron Rothblum. Delegation for bounded space. In *STOC '13*, 2013.

KRR14. Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC '14*, 2014.

KRR16. Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. ePrint 2016/303, 2016.

LFKN92. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *JACM*, 1992.

Lin15. Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In *TCC '15*, 2015.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC '12*, 2012.

Mer89a. Ralph C. Merkle. A certified digital signature. In *CRYPTO '89*, 1989.

Mer89b. Ralph C. Merkle. One way hash functions and DES. In *CRYPTO '89*, 1989.

Mic00. Silvio Micali. Computationally sound proofs. *SIAM J. Comp.*, 2000.

MV16. Arno Mittelbach and Daniele Venturi. Fiat-shamir for highly sound protocols is instantiable. ePrint 2016/313, 2016.

Pas03. Rafael Pass. On deniability in the common reference string and random oracle model. In *CRYPTO '03*, 2003.

PGHR13. Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Oakland '13*, 2013.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT '96*, 1996.

PSSV07. Aduri Pavan, Alan L. Selman, Samik Sengupta, and Vinodchandranm N. V. Polylogarithmic-round interactive proofs for coNP collapse the exponential hierarchy. *Theoretical Computer Science*, 2007.

PTW09. Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. In *CRYPTO '09*, 2009.

RRR16. Omer Reingold, Ron Rothblum, and Guy Rothblum. Constant-round interactive proofs for delegating computation. In *STOC '16*, 2016.

SBV^{+}13. Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys '13*, 2013.

SBW11. Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Toward practical and unconditional verification of remote computations. In *HotOS '11*, 2011.

Sha92. Adi Shamir. IP = PSPACE. *JACM*, 1992.

SMBW12. Srinath Setty, Michael McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS '12*, 2012.

SVP+12. Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Security '12*, 2012.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC '08*, 2008.

Wee09. Hoeteck Wee. Zero knowledge in the random oracle model, revisited. In *ASIACRYPT '09*, 2009.