Strict Polynomial-time in Simulation and Extraction*

Boaz Barak
Department of Computer Science
Weizmann Institute of Science
Rehovot 76403, Israel.
boaz@wisdom.weizmann.ac.il

Yehuda Lindell†
IBM T.J.Watson Research
19 Skyline Drive, Hawthorne
New York, 10532, USA.
lindell@us.ibm.com

February 2, 2004

Abstract

The notion of efficient computation is usually identified in cryptography and complexity with (strict) probabilistic polynomial time. However, until recently, in order to obtain *constant-round* zero-knowledge proofs and proofs of knowledge, one had to allow simulators and knowledge-extractors to run in time that is only polynomial *on the average* (i.e., *expected* polynomial time). Recently Barak gave the first constant-round zero-knowledge argument with a *strict* (in contrast to expected) polynomial-time simulator. The simulator in his protocol is a *non-black-box* simulator (i.e., it makes inherent use of the description of the *code* of the verifier).

In this paper, we further address the question of strict polynomial-time in constant-round zero-knowledge proofs and arguments of knowledge. First, we show that there exists a constant-round zero-knowledge *argument of knowledge* with a *strict* polynomial-time *knowledge extractor*. As in the simulator of Barak's zero-knowledge protocol, the extractor for our argument of knowledge is not black-box and makes inherent use of the code of the prover. On the negative side, we show that non-black-box techniques are *essential* for both strict polynomial-time simulation and extraction. That is, we show that no (non-trivial) constant-round zero-knowledge proof or argument can have a strict polynomial-time *black-box* simulator. Similarly, we show that no (non-trivial) constant-round zero-knowledge proof or argument of knowledge can have a strict polynomial-time *black-box* knowledge extractor.

Keywords: Zero-knowledge proof systems, proofs of knowledge, expected vs. strict polynomial-time, black-box vs. non-black-box algorithms.

*An extended abstract of this paper appeared in the *34th STOC*, 2002.
†Much of this work was carried out while at the Weizmann Institute of Science, Israel.

1 Introduction

This paper deals with the issue of *expected* versus *strict* polynomial-time with respect to simulators and extractors for zero-knowledge proofs and arguments and zero-knowledge proofs and arguments of knowledge.[1]

1.1 Expected Polynomial-Time in Zero Knowledge

The principle behind the definition of (computational) zero-knowledge proofs, as introduced by Goldwasser, Micali and Rackoff [29], is the following:

> *Anything that an* efficient *verifier can learn as a result of interacting with the prover, can be learned* without interaction *by applying an* efficient *procedure* (*i.e., simulator*) *to the public input.*

Note that there are two occurrences of the word "efficient" in this sentence. When providing a formal definition of zero knowledge, the issue of what is actually meant by "efficient computation" must be addressed. The standard interpretation in cryptography and complexity is that of probabilistic polynomial-time. However, in the context of zero knowledge, efficiency has also been taken to mean *polynomial on the average* (a.k.a. *expected polynomial-time*). That is, if we fix the input, and look at the running time of the machine in question as a random variable (depending on the machine's coins), then we only require that the *expectation* of this random variable is polynomial. Three versions of the formal definition of zero knowledge appear in the literature, differing in their interpretations of efficient computation:

1. Definition 1 – STRICT/STRICT: According to this definition both the verifier and simulator run in strict polynomial-time. This is the definition adopted by Goldreich [22, Section 4.3] and is natural in the sense that only the standard interpretation of efficiency is used.

2. Definition 2 – STRICT/EXPECTED: This more popular (and liberal) definition requires the verifier to run in strict polynomial-time while allowing the simulator to run in expected polynomial-time. This was actually the definition proposed in the original paper on zero knowledge [29].

3. Definition 3 – EXPECTED/EXPECTED: In this definition, *both* the verifier and simulator are allowed to run in expected polynomial-time. This definition is far less standard than the above two, but is nevertheless a natural one to consider. As we describe below, this definition was considered by [17], who show that (at least for one definition of expected polynomial-time) it is problematic.

As we have mentioned, the standard interpretation of efficient computation is that of (strict) polynomial-time. In light of this, Definition 1 (STRICT/STRICT) seems to be the most natural. Despite this, expected polynomial-time was introduced in the context of zero knowledge because a number of known protocols that could be proven zero-knowledge according to the more liberal "STRICT/EXPECTED definition" were not known to satisfy the more severe "STRICT/STRICT definition". In particular, until very recently no *constant-round* zero knowledge argument (or proof) for

[1] In a *proof* system, soundness holds unconditionally and with respect to all-powerful cheating provers. In contrast, in an *argument* system, soundness is only guaranteed to hold with respect to polynomial-time bounded provers. We note that lower bounds for proofs do not necessarily hold for arguments, because in arguments the soundness condition is only computational. Likewise, lower bounds for arguments do not necessarily hold for proofs, because proofs are allowed to have super-polynomial honest prover strategies, whereas arguments are not. See Section 2.1.

$\mathcal{NP}$ was known to satisfy Definition 1 (STRICT/STRICT).[2] It was therefore necessary to relax the definition and allow expected polynomial-time simulation (as in Definition 2).

Proofs of Knowledge. An analogous situation arises in proofs *of knowledge* [29, 33, 18, 6]. There, the underlying principle is that:

> *If an* efficient *prover can convince the honest verifier with some probability that $x \in L$, then this prover can apply an* efficient *procedure (i.e., extractor) to x and its private inputs and obtain a witness for x with essentially the same probability.*

Again, the word "efficient" occurs twice, and again three possible definitions can be used. In particular, the prover and extractor can be instantiated by strict polynomial-time machines, expected polynomial-time machines or a combination of both.

The different definitions - discussion. As has been observed before (e.g., see [17, Sec. 3.2], [22, Sec. 4.12.3]), the definitions that allow for expected polynomial-time computation are not fully satisfactory for several reasons:

- *Philosophical considerations:* Equating "efficient computation" with *expected polynomial-time* is more controversial than equating efficient computation with (strict) probabilistic polynomial-time. For example, Levin ([30], see also [20], [22, Sec. 4.3.1.6]) has shown that when expected polynomial-time is defined as above, the definition is too machine dependent, and is not closed under reductions. He proposed a different definition for expected polynomial-time that is closed under reductions and is less machine dependent. However, it is still unclear whether expected polynomial-time, even under Levin's definition, should be considered as efficient computation.
- *Technical considerations:* Expected polynomial-time is less understood than the more standard strict polynomial-time. This means that rigorous proofs of security of protocols that use zero-knowledge arguments with expected polynomial-time simulators (or arguments of knowledge with expected polynomial-time extractors) as components, are typically more complicated (see [31] for an example). Another technical problem that arises is that expected polynomial-time simulation is not closed under composition. Consider, for example, a protocol that uses zero-knowledge as a subprotocol. Furthermore, assume that the security of the larger protocol is proved in two stages. First, the zero-knowledge subprotocol is simulated for the adversary (using an expected polynomial-time simulator). This results in an expected polynomial-time adversary that runs the protocol with the zero-knowledge executions removed. Then, in the next stage, the rest of the protocol is simulated for this adversary. A problem arises because the simulation of the second stage must now be carried out for an expected polynomial-time adversary. However, simulation for an expected polynomial-time adversary can be highly problematic (as the protocol of [24] demonstrates, see [31, Appendix A] for details).
- *Practical considerations:* A proof of security that uses expected polynomial-time simulation does not always achieve the "expected" level of security. For example, assume that a protocol's security relies on a hard problem that would take 100 years to solve, using the best known algorithm. Then, we would like to prove that the probability that an adversary can successfully break the protocol is negligible, unless it runs for 100 years. However, when expected polynomial-time simulation is used, we cannot rule out an adversary who runs for 1 year and succeeds with probability 1/100. This is a weaker level of security and may not be acceptable. See Section 1.4 for a more detailed discussion of this issue.

[2] We note that throughout this paper we always refer to protocols with negligible soundness error.

The liberal "STRICT/EXPECTED definition" also suffers from a conceptual drawback regarding the notion of zero knowledge itself. Specifically, the idea behind the definition of zero knowledge is that anything that a verifier can learn as a result of the interaction, it can learn by just looking at its input. Therefore, it seems that the simulator should *not* be of a higher complexity class than the verifier. Rather, both the verifier and simulator should be restricted to the same complexity class (i.e., either strict or expected polynomial-time). The "EXPECTED/EXPECTED definition" has the advantage of not having any discrepancy between the computational power of the verifier and simulator. However, it still suffers from the above described drawbacks with any use of expected polynomial-time. In addition, as Feige [17, Sec. 3.3] pointed out, in order to prove that known protocols remain zero knowledge for expected polynomial-time verifiers, one needs to restrict the verifiers to run in expected polynomial-time not only when interacting with the honest prover but also when interacting with all other interactive machines. This restriction is somewhat controversial because any efficient adversarial strategy should be allowed. In particular, there seems to be no reason to disqualify an adversarial strategy that takes expected polynomial-time when attacking the honest prover, but runs longer otherwise (notice that the adversary is only interested in attacking the honest prover, and so its attack is efficient).

In contrast, the "STRICT/STRICT definition" suffers from none of the above conceptual difficulties. For this reason, it is arguably a preferred definition. However, as we have mentioned, it was not known whether this definition can be satisfied by a protocol with a *constant* number of rounds. Thus a natural open question (posed by [17, Sec. 3.4] and [22, Sec. 4.12.3]) was the following:

> *Is expected polynomial-time simulation and extraction necessary in order to obtain constant-round zero-knowledge proofs and proofs of knowledge?*

A first step in answering the above question was recently taken by Barak in [3]. Specifically, [3] presented a zero-knowledge argument system that is both *constant-round* and has a *strict* polynomial-time simulator. Interestingly, the protocol of [3] is not black-box zero knowledge. That is, the simulator utilizes the description of the code of the verifier. (This is in contrast to black-box zero knowledge where the simulator is only given oracle access to the verifier.) Given the existence of non-black-box zero-knowledge arguments with a constant number of rounds and strict polynomial-time simulation, it is natural to ask the following questions:

1. Is it possible to obtain constant-round zero-knowledge *arguments of knowledge* with strict polynomial-time extraction?
2. Is the fact that the protocol of [3] is not black-box zero knowledge coincidental, or is this an inherent property of any constant-round zero-knowledge protocol with strict polynomial-time simulation?

1.2 Our Results

In this paper we resolve both the above questions. First, we show that it is possible to obtain *strict* polynomial-time knowledge extraction in a *constant-round* protocol. In fact, we show that it is possible to obtain strict polynomial-time simulation and extraction simultaneously in a zero-knowledge protocol. That is, we prove the following theorem:

Theorem 1 *Assume the existence of collision-resistant hash functions and collections of trapdoor permutations such that the domain of each permutation is the set of all strings of a certain length.*[3] *Then, there exists a* constant-round *zero-knowledge argument of knowledge for $\mathcal{NP}$ with a* strict *polynomial-time knowledge extractor and a* strict *polynomial-time simulator.*

The definition of arguments of knowledge that we refer to in Theorem 1 differs from the standard definition of [6] in an important way. In the definition of [6], the knowledge extractor is given only black-box access to the prover. In contrast, in our definition, the knowledge extractor is given the actual description of the prover (i.e., it has non black-box access). As we will see below, this modification is actually necessary for obtaining *constant-round* arguments of knowledge with *strict* polynomial-time extraction.

In addition to the above positive result, we show that it is impossible to obtain a (non-trivial) constant-round zero-knowledge protocol that has a strict polynomial-time *black-box* simulator. Likewise, a strict polynomial-time extractor for a constant-round zero-knowledge argument of knowledge cannot be black-box. That is, we prove the following two theorems:

Theorem 2 *There do not exist* constant-round *zero-knowledge proofs or arguments with* strict polynomial-time black-box *simulators for any language $L \notin \mathcal{BPP}$.*

Theorem 3 *There do not exist* constant-round *zero-knowledge proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for any language $L \notin \mathcal{BPP}$.*

We therefore conclude that the liberal definitions that allow the simulator (resp., extractor) to run in expected polynomial-time are necessary for achieving constant-round *black-box* zero knowledge (resp., arguments of knowledge). Furthermore, our use of non-black-box techniques in order to obtain Theorem 1 is essential.

We note that Theorems 2 and 3 are tight in the sense that if any super-constant number of rounds are allowed, then zero-knowledge proofs of knowledge with strict polynomial-time black-box extraction and simulation *can* be obtained. This was shown by Goldreich in [22, Sec. 4.7.6]. (Actually, [22] shows that a *super-logarithmic* number of sequential executions of the 3-round zero-knowledge proof for Hamiltonicity [9] suffices. However, using the same ideas, it can be shown that by running $\log n$ parallel executions of the proof of Hamiltonicity, any super-constant number of sequential repetitions is actually enough.)

Zero knowledge versus ε-knowledge. Our impossibility result regarding constant-round black-box zero knowledge with strict polynomial-time simulation has an additional ramification to the question of the relation between black-box ε-knowledge [15] and black-box zero knowledge. Loosely speaking, an interactive proof is called ε-knowledge if for every ε, there exists a simulator who runs in time polynomial in the input and in $1/\varepsilon$, and outputs a distribution that can be distinguished from a real proof transcript with probability at most ε. Despite the fact that this definition seems to be a significant relaxation of zero knowledge, no separation between ε-knowledge and zero knowledge was previously known. Our lower bound indicates a separation for the black-box versions: that is, *black-box* ε-knowledge is strictly weaker than *black-box* zero-knowledge. Specifically, on the one hand, constant-round black-box ε-knowledge protocols with strict polynomial-time simulators

[3] By this we mean that there exists a trapdoor permutation family $\{f_s\}_{s\in\{0,1\}^*}$ such that $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$. It actually suffices to assume the existence of a family of enhanced trapdoor permutations [23, Appendix C.1]. Such a family can be constructed under the RSA and factoring assumptions, see [2, Section 6.2] and [23, Appendix C.1].

do exist.[4] On the other hand, as we show, analogous protocols for black-box *zero*-knowledge, do not exist.

Witness-extended emulation. Zero-knowledge proofs of knowledge are often used as subprotocols within larger protocols. Typically, in this context the mere existence of a knowledge extractor does not suffice for proving the security of the larger protocol. Loosely speaking, what is required is the existence of a machine that not only outputs a witness with the required probability (as is required from a knowledge extractor), but also outputs a corresponding simulated transcript of the interaction between the prover and the verifier. Furthermore, whenever the transcript of the interaction is such that the verifier accepts, then the witness that is obtained is valid. To explain this further, consider a case that the prover convinces the verifier in a real interaction with probability p. Then, with probability negligibly close to p, the aforementioned machine should output an accepting transcript and a valid witness. Furthermore, with probability negligibly close to $1-p$, the machine should output a rejecting transcript (and we don't care about the witness).

This issue was addressed in [31], who called such a machine a "witness-extended emulator". It was proved there that there exists such a witness extended emulator for any proof of knowledge. However, the extended emulator that is obtained runs in *expected* polynomial-time, even if the original knowledge extractor runs in *strict* polynomial-time. Unfortunately, we do not know how to prove an analogous result that, given any strict polynomial-time knowledge extractor, would provide a strict polynomial-time emulator. Instead, we directly construct a strict polynomial-time witness-extended emulator for our zero-knowledge proof of knowledge (under a slightly different definition than [31]).

1.3 On the Effect of Truncating Expected Polynomial-Time Executions

A naive approach to solving the problem of expected polynomial-time in simulation and extraction, is to simply truncate the execution of the simulator or extractor after it exceeds its expected running-time by "too much". However, this does not necessarily work. The case of knowledge-extractors is a good example. Let us fix a proof (or argument) of knowledge for some NP-language L. Let $x \in \{0,1\}^*$, and let P^* be a polynomial-time prover that, for some ϵ, aborts with probability $1-\epsilon$ and convinces the honest verifier that $x \in L$ with probability ϵ. For all previously known constant-round proofs of knowledge, the expected polynomial-time knowledge-extractor works in roughly the following way: it first verifies the proof from P^*, and if P^* was not convincing (which occurs in this case with probability $1-\epsilon$) then it aborts. On the other hand, if P^* was convincing (which happens in this case with probability ϵ), then it does expected $p(n) \cdot \frac{1}{\epsilon}$ work (where $p(\cdot)$ is some fixed polynomial), and outputs a witness for x. Clearly, the expected running time of the extractor is polynomial (in particular, it is $p(n)$ plus the time taken to honestly verify a proof). However, if we halt this extractor before it completes $\frac{1}{\epsilon}$ steps then, with high probability, the extractor will *not* output a witness. Note that $\frac{1}{\epsilon}$ may be much larger than $p(n)$, and therefore the extractor may far exceed its expected running-time and yet still not output anything.

In contrast to the above, the knowledge extractor of the argument of knowledge presented in this paper (in Section 4) runs in strict polynomial-time which is *independent* of the acceptance probability (i.e., ϵ). For example, if there exists a cheating prover P^* that runs in time n^2, but convinces the verifier that $x \in L$ with probability $\epsilon = n^{-10}$ then our extractor will always run

[4] Such a protocol can be constructed by taking any constant-round protocol with an expected polynomial-time simulator and truncating the simulator's run (outputting $\perp$), if it runs for more than $1/\varepsilon$ times its expected running-time. By Markov's inequality, the probability of this bad event happening is at most ε.

in time, say, n^4 and output a witness with probability at least (negligibly less than) n^{-10}. On the other hand, the extractors for previous protocols would do almost nothing with probability $1 - n^{-10}$, and with probability n^{-10} run for, say, n^{12} steps and output a witness.

1.4 Trading Success Probability for Running Time

The observation in Section 1.3 about how expected polynomial-time extractors typically work, raises serious security issues with respect to the application of proofs of knowledge that have such extractors. For example, suppose that we use a proof of knowledge for an identification protocol based on factoring. Suppose furthermore, that we use numbers for which the fastest known algorithms will take 100 years to factor. We claim that in this case, if we use a proof of knowledge with an *expected* polynomial-time extractor then we cannot rule out the possible existence of an adversary that will take 1 year of computation time and succeed in an impersonation attack with probability 1/100.

In order to see this, notice that the proof of security of the identification protocol works by constructing a factoring algorithm from any impersonator, using the extractor for the proof of knowledge. Thus, for typical protocols, what will actually be proven is that given an algorithm that runs for T steps and successfully impersonates with probability ϵ, we can construct an algorithm that solves the factoring problem with probability ϵ and *expected* running time T. In particular, this factoring algorithm may (and actually will) work in the following way: with probability $1-\epsilon$ it will do nothing and with probability ϵ it will run in T/ϵ steps and factor its input. Thus, the existence of an impersonator that runs for one year and succeeds with probability 1/100, only implies the existence of a factoring algorithm that runs for 100 years. Therefore, we cannot rule out such an impersonator. We conclude that the standard proofs of knowledge potentially allow adversaries to trade their success probability for running time. In the concrete example above, the impersonator lowered its running time from 100 years to one year, at the expense of succeeding with probability 1/100 instead of 1. We stress that the fastest known algorithms for factoring *do not* allow such a trade-off. That is, if the parameters are chosen so that 100 years are required to factor, then the probability of successfully factoring after one year is extremely small, and not close to 1/100.

We stress that not only is it the case that the *definition* of expected polynomial-time extraction does not allow us to rule out such an adversary, but also such adversaries cannot be ruled out by the current proofs of security for known constant-round protocols (thus, the problem lies also with the protocols and not just with the definition). In contrast, such a trade-off is not possible if the extractor runs in *strict* polynomial-time. Rather, an impersonator that runs in time T and succeeds with probability ϵ yields a factoring algorithm that runs in time (polynomially related) to T and succeeds with probability ϵ. Thus, in the above concrete example, an analogous impersonator for a protocol with a strict polynomial-time extractor would yield a factoring algorithm that runs for one year and succeeds with probability 1/100. However, such an algorithm is conjectured not to exist, and therefore such an impersonator also does not exist (unless the conjecture is wrong).

1.5 Further Discussion of Prior Work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [29], and were then shown to exist for all $\mathcal{NP}$ by Goldreich, Micali and Wigderson [26]. Constant-round zero-knowledge arguments and proofs were constructed by Feige and Shamir [19], Brassard, Crepeau and Yung [11] and Goldreich and Kahan [24]. All these constant-round protocols utilize expected polynomial-time simulators. Regarding zero-knowledge proofs *of knowledge*, following a discussion in [29], the first formal definitions were provided by Feige, Fiat and Shamir [18] and by Tompa and Woll [33]. These definitions were later modified by Bellare and Goldreich [6].

The issue of expected polynomial-time is treated in Feige's thesis [17] and Goldreich's book [22]. Goldreich [22, Sec. 4.7.6] also presents a construction for a proof of knowledge with *strict* polynomial-time extraction (and simulation) that uses any super-logarithmic number of rounds (as discussed above, a variant of this construction can be obtained that uses any super-constant number of rounds).

As we have mentioned, until a short time ago, all known constant-round zero-knowledge protocols had expected polynomial-time simulators. However, recently this barrier was broken by Barak [3], who provided the first constant-round zero-knowledge argument for $\mathcal{NP}$ with a *strict* polynomial-time simulator, assuming the existence of collision-resistant hash functions with super-polynomial hardness. Barak and Goldreich [4] later showed how to obtain the same result under the weaker assumption of the existence of standard collision-resistant hash functions (with polynomial-time hardness). The construction of [3] was also the first zero-knowledge argument to utilize a non-black-box simulator. In a similar fashion, the constant-round argument of knowledge presented in this paper utilizes a non-black-box *knowledge-extractor*. We note that [5] also utilize a non-black-box knowledge extractor. However, their extractor runs in *expected* polynomial-time, and the non-black-box access is used there for a completely different reason (specifically, to achieve a *resettable* zero-knowledge argument of knowledge).

1.6 Organization.

In Section 2 we describe the basic notations and definitions that we use. Then, in Section 3 we define and construct a commit-with-extract commitment scheme, which is the main technical tool used to construct our zero-knowledge argument of knowledge. The proof of Theorem 1 can be found in Section 4 where we present the construction of a zero-knowledge argument of knowledge with strict polynomial-time extraction. Finally, in Section 5 we prove Theorems 2 and 3. That is, we prove that it is impossible to construct strict polynomial-time black-box simulators and extractors for (non-trivial) constant-round protocols.

2 Definitions

Notation. For a binary relation R, we denote by $R(x)$ the set of all "witnesses" for x. That is, $R(x) \stackrel{\text{def}}{=} \{y \mid (x,y) \in R\}$. Furthermore, we denote by L_R the language induced by the relation R. That is, $L_R \stackrel{\text{def}}{=} \{x \mid R(x) \neq \emptyset\}$.

For a finite set $S \subseteq \{0,1\}^*$, we write $x \in_R S$ to say that x is distributed uniformly over the set S. We denote by U_n the uniform distribution over the set $\{0,1\}^n$.

A function $\mu(\cdot)$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large n's, it holds that $\mu(n) < 1/p(n)$. We let $\mu(\cdot)$ denote an arbitrary negligible function. That is, when we say that $f(n) < \mu(n)$ for some function $f(\cdot)$, we mean that *there exists* a negligible function $\mu(\cdot)$ such that for every n, $f(n) < \mu(n)$. A function $f(\cdot)$ is noticeable if there exists a positive polynomial $p(\cdot)$ such that for all sufficiently large n's, it holds that $f(n) > 1/p(n)$. We note that "noticeable" is not the complement of "negligible".

For two probability ensembles (sequences of random variables) $X = \{X_s\}_{s\in S}$ and $Y = \{Y_s\}_{s\in S}$ (where $S \subseteq \{0,1\}^*$ is a set of strings), we say that X is computationally indistinguishable from Y, denoted $X \stackrel{\text{c}}{\equiv} Y$, if for every polynomial-sized circuit family $\{D_n\}_{n\in\mathsf{N}}$ and every $s \in S$, it holds that

$|\Pr[D_{|s|}(X_s) = 1] - \Pr[D_{|s|}(Y_s) = 1]| < \mu(|s|)$. We will sometime drop the subscripts s when they can be inferred from the context. In all our protocols, we will denote the security parameter by n.

Let A be a probabilistic polynomial-time machine. We denote by $A(x, y, r)$ the output of the machine A on input x, auxiliary-input y and random-tape r. We stress that the running-time of A is polynomial in $|x|$.[5] If M is a Turing machine, then we denote by $\mathsf{desc}_n(M)$ the description of a *circuit* that computes M on inputs of size n. Note that a polynomial-time machine that receives $\mathsf{desc}_n(M)$ for input runs in time that is polynomial in the running-time of M. Let A and B be interactive machines. We denote by $\mathsf{view}_A(A(x, y, r), B(x, z, r'))$ the view of party A in an interactive execution with machine B, on public input x, where A has auxiliary-input y and random-tape r, and B has auxiliary input z and random-tape r'. The view of party B is denoted similarly. Recall that a party's view of an execution includes the contents of its input, auxiliary-input and random tape plus the transcript of messages that it receives during the execution. We will sometimes drop r or r' from this notation, which will mean that the random tape is not fixed but rather chosen at random. For example we denote by $\mathsf{view}_A(A(x, y), B(x, z))$ the random variable $\mathsf{view}_A(A(x, y, U_m), B(x, z, U'_{m'}))$ where m (resp., m') is the number of random bits that A (resp., B) uses on input of size $|x|$.

2.1 Zero Knowledge

Loosely speaking, an interactive proof system for a language L involves a prover P and a verifier V, where upon common input x, the prover P attempts to convince V that $x \in L$. We note that the prover is often given some private auxiliary-input that "helps" it to prove the statement in question to V. Such a proof system has the following two properties:

1. *Completeness:* this states that when honest P and V interact on common input $x \in L$, then V is convinced of the correctness of the statement that $x \in L$ (except with at most negligible probability).

2. *Soundness:* this states that when V interacts with any (cheating) prover P^* on common input $x \notin L$, then V will be convinced *with at most negligible probability.* (Thus V cannot be tricked into accepting a false statement.)

There are two flavors of soundness: *unconditional* (or statistical) soundness that must hold even for an all-powerful cheating prover, and *computational soundness* that needs only hold for (non-uniform) polynomial-time cheating provers. In *proof* systems [29], unconditional soundness is guaranteed; whereas in *argument* systems [10] only computational soundness must hold. We remark that a proof system is not necessarily an argument system, because the honest prover strategy in a proof system is not required to be polynomial-time (in contrast to arguments where the honest prover as well as the cheating provers must be non-uniform polynomial-time). Unless explicitly stated, when we mention "protocols" in discussion, we mean both proofs and arguments.

Throughout this paper, we will always assume that the soundness error is at most negligible. However, we will not always require this of completeness. Specifically, our lower bounds in Section 5 hold even if the completeness error is $1 - 1/p(n)$ for some polynomial $p(\cdot)$; in this case, we will call $p(n)$ the $\mathsf{completeness\ bound}$.

We now recall the definition of zero knowledge [29]. Actually, we present (a slightly strengthened form of) the definition of *auxiliary-input* zero knowledge [22, Sec. 4.3.3].[6] The main difference

[5] We assume that y and r are on different tapes. Therefore, even if y is very long (e.g., $|y| > \mathrm{poly}(|x|)$), it is still possible for A to read r.

[6] We deviate from the definition of auxiliary-input zero knowledge of [22, Sec. 4.3.3] by making the slightly stronger requirement that there exists a single universal simulator for all verifiers, rather than a different simulator for each

between our definition below and the standard definition is that we require the simulator to run in *strict*, rather than *expected*, polynomial-time. We note that in this paper, when we say zero knowledge, our intention is always auxiliary-input zero knowledge.

Definition 2.1 (auxiliary-input zero knowledge): *Let (P, V) be an interactive proof (or argument) system for a language L. Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e., when the completeness bound is $p(\cdot)$, then $P_L(x)$ is the set of strings y for which the probability that* $\mathsf{view}_V(P(x,y), V(x))$ *is accepting is at least $p(|x|)$). We say that (P, V) is* auxiliary-input zero knowledge *if there exists a strict probabilistic polynomial-time algorithm S such that for every strict probabilistic polynomial-time machine V^* it holds that*

$$\left\{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{S(\mathsf{desc}_{|x|}(V^*), x, z)\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$$ [7]

Black-box zero knowledge. A zero-knowledge proof system is called black-box zero knowledge [27] if the simulator S only uses its input $\mathsf{desc}(V^*)$ as a black-box subroutine. That is, S is an oracle algorithm such that:

$$\left\{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{S^{V^*(x,z,\cdot,\cdot)}(x)\right\}_{x\in L, y\in P_L(x), z\in\{0,1\}^*}$$

where $V^*(x, z, \cdot, \cdot)$ denotes the *next-message function* of the interactive machine V^* when the common input x and auxiliary input z are fixed (i.e., the next message function of V^* receives a random-tape r and a message history h and outputs $V^*(x, z, r, h)$).

2.2 Zero-Knowledge Arguments of Knowledge

Our definition of proofs and arguments of knowledge below differs from the standard definition of [6] in two ways:

Strict polynomial-time extraction. We require that the knowledge extractor run in *strict* polynomial-time (rather than in expected polynomial-time).

Non-black-box extraction. The knowledge extractor is given access to the *description* of the prover. This is a relaxation of the standard definition of proofs of knowledge (cf. [6, 22]) in which the knowledge extractor is given *only* oracle (or black-box) access to the prover strategy. The relaxed definition appeared originally in Feige and Shamir [19] (which differs from the definition in [18]; see discussion in [6]), and suffices for all practical applications of arguments of knowledge.

Until recently, all known proofs of knowledge (including [19]) were coupled with an extractor that used the prover algorithm only as a black-box. The extra power of non-black-box extraction (where the knowledge extractor is given the actual description of the prover) was first used in an essential way by [5] in order to obtain resettable zero-knowledge arguments of knowledge for

verifier as in [22, Sec. 4.3.3]. Note however, that the definition of [22, Sec. 4.3.3] already implies that for any $c > 0$ there exists a universal simulator for all $\mathbf{Time}(n^c)$ verifiers.

[7] Recall that $\mathsf{desc}_n(M)$ is the description of a circuit that computes M on inputs of size n. An equivalent formulation provides the simulator S with the description of the actual Turing machine M. However, in this case, it is also necessary to provide S with 1^t, where t is a bound on the running time of V^* on inputs of length $|x|$. This additional input is provided in order to allow S to run in time which is (some fixed) polynomial in the running time of V^*.

$\mathcal{NP}$.[8] We show in Section 5 that our use of non-black-box extraction is also essential, as there do not exist constant round proofs of knowledge with *black-box* strict polynomial-time extractors.

We are now ready to present the definition:

Definition 2.2 (system of proofs/arguments of knowledge): *Let R be a binary relation. We say that a probabilistic, polynomial-time interactive machine V is a* knowledge verifier for the relation R *with* negligible knowledge error *if the following two conditions hold:*

- Non-triviality: *There exists a probabilistic polynomial-time*[9] *interactive machine P such that for every $(x, y) \in R$, all possible interactions of V with P on common input x, where P has auxiliary input y, are accepting.*
- Validity (or knowledge soundness) with negligible error: *There exists a* strict *probabilistic polynomial-time machine K, such that for every strict probabilistic polynomial-time machine P^*, and every $x, y, r \in \{0,1\}^*$, machine K satisfies the following condition:*

  Denote by $p(x, y, r)$ the probability (over the random tape of V) that V accepts upon input x, when interacting with the prover P^* who has input x, auxiliary-input y and random-tape r. Then, machine K, upon input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$, outputs a solution $s \in R(x)$ with probability at least $p(x, y, r) - \mu(|x|)$.

An interactive pair (P, V) such that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called an argument of knowledge *for the relation R. If the validity condition holds with respect to any (not necessarily polynomial-time) machine P^*, then (P, V) is called a* proof of knowledge *for R.*

If an argument (resp., proof) of knowledge (P, V) is zero knowledge for the language L_R induced by R, then we say that (P, V) constitutes a system of zero-knowledge arguments (resp., proofs) of knowledge for R.

2.3 Witness-Extended Emulation

In this section, we present an extension of the notion of proofs of knowledge, called witness-extended emulation. This extension is of importance when zero-knowledge proofs or arguments of knowledge are used as subprotocols within larger protocols, as is often the case. Typically in this context, the extractor for the proof of knowledge supplies the simulator for the larger protocol with some secret information. This information is then used in the proof of security of the rest of the larger protocol.

The final output of the simulator for the larger protocol is usually a transcript of the entire simulated protocol execution (where this transcript is indistinguishable from a real execution). Thus, the extractor needs to not only extract a witness from the proof of knowledge, but must also obtain a matching transcript of the execution of the proof of knowledge itself. However, by definition, the extractor only outputs a witness and does not provide the simulator with such a transcript. This issue was addressed in [31] where, loosely speaking, it was shown that for any zero-knowledge proof of knowledge, there exists a machine who outputs both the witness (with the appropriate probability) *and* a matching transcript of messages sent in the execution. Such

[8] The use there is critical as it can be shown that if the knowledge extractor is restricted to only black-box access to the prover, then resettable zero-knowledge arguments of knowledge are possible for languages in $\mathcal{BPP}$ only, see [12].

[9] The requirement that P be polynomial-time is inherent for arguments, but not for proofs. A proof system with such a prover is called an efficient-prover proof.

a machine was termed a "witness-extended emulator", because its role is to emulate a protocol execution while also providing a witness (see [31] for a more detailed discussion). We proceed by presenting a slightly different definition of witness-extended emulation, and then discuss its relevance to our work. We begin with some notation and terminology:

- Recall that $\mathsf{view}_{P^*}(P^*(x,y,r),V(x))$ denotes a random variable describing the view of P^* in a protocol execution with the honest verifier V, where P^* has input x, auxiliary-input y and random-tape r, and the honest verifier V has input x. (This random variable depends only on the coins of V.)
- We say that a zero-knowledge protocol (P,V) is *publicly verifiable* if given the transcript of messages between any P^* and V, it is possible to efficiently determine whether or not V accepted the proof. (For example, any protocol can be made publicly verifiably by having the verifier sends the contents of its random tape at the end of the protocol execution. Note, however, that this can affect other properties of the protocol. Indeed, our proof of knowledge, Protocol 4.1, *cannot* be made publicly verifiable in this way while still preserving the witness-extended emulation property.)

  When a protocol is publicly verifiable, then V's accept/reject bit can be deduced efficiently (and deterministically) from the prover's view. We denote by $\mathsf{accept}_V(\cdot)$ the deterministic function that takes a specific view of the prover in a protocol execution, and outputs whether or not V accepts in this execution.

We are now ready to present the definition:

Definition 2.3 (witness-extended emulator): *Let R be a binary relation and let (P,V) be an interactive proof system that is* publicly verifiable*. Consider a probabilistic machine E that is given the description of a probabilistic polynomial-time prover $\mathsf{desc}_{|x|}(P^*)$, and the contents of P^*'s input, auxiliary-input and random-tapes, x, y and r respectively. We denote by $E_1(\mathsf{desc}_{|x|}(P^*),x,y,r)$ and $E_2(\mathsf{desc}_{|x|}(P^*),x,y,r)$ the random variables representing the first and second elements of the output of E, respectively. We say that E is a* witness-extended emulator for (P,V) and R *if it runs in* strict *polynomial-time and if for every probabilistic polynomial-time interactive machine P^*, every $y,r \in \{0,1\}^*$ and all sufficiently large x's,*

1. *E_1's output distribution is indistinguishable from the distribution of the view of P^* in a real execution with the honest verifier V. That is,*

$$\left\{E_1(\mathsf{desc}_{|x|}(P^*),x,y,r)\right\}_{x,y,r} \stackrel{\mathrm{c}}{\equiv} \{\mathsf{view}_{P^*}(P^*(x,y,r),V(x))\}_{x,y,r}$$

2. *The probability that V accepts in the view of P^* that is output by E_1, and yet E_2 does not output a correct witness, is negligible. That is,*

$$\Pr\left[\mathsf{accept}_V(E_1(\mathsf{desc}_{|x|}(P^*),x,y,r)) = 1 \ \& \ E_2(\mathsf{desc}_{|x|}(P^*),x,y,r) \notin R_L(x)\right] < \mu(|x|)$$

Definition 2.3 differs from the definition of [31] in a number of ways. Most notably, we provide the witness-extended emulator with the description of P^* (rather than just black-box access) and also require it to run in strict polynomial-time (rather than expected polynomial-time). The other differences are technical and are mainly due to the need to achieve strict polynomial-time emulation. Despite the differences, the definitions are the same in spirit and both achieve the desired goal of enabling a proof of knowledge to be used as a subprotocol in some other, larger protocol.

Recall that [31] proved the existence of a witness-extended emulator for any proof of knowledge. However, the emulator that is obtained runs in expected polynomial-time and thus does not achieve our goal of strict polynomial-time emulation. We will therefore directly prove the existence of a witness-extended emulator for our zero-knowledge argument of knowledge.

3 Commitment with Extraction

In order to construct a constant-round zero-knowledge argument of knowledge with strict polynomial-time extraction, we first construct a new primitive that we call commit-with-extract. Loosely speaking, a commit-with-extract scheme is a commitment scheme with the additional property that the committed value can be extracted from the sender. More precisely, there exists a (strict polynomial-time) commitment extractor who is given the description of the sender (along with the contents of its input, auxiliary input and random tape) and extracts the value being committed to during the commit stage of the protocol. This notion of "extractable commitments" is not completely new. It has been used in the context of secure multi-party computation (e.g., [21, Construction 2.3.8]) and has been called both commit-with-knowledge [14] and non-oblivious commitment [22, Def. 4.9.3]. One main technical difference between our primitive and previous ones is the requirement that the extractor run in *strict* polynomial-time. At the end of this section we discuss another significant difference that is related to the question of "knowledge" versus "extraction".

We remark that in a standard commitment scheme, the committer may not "know" the value that it committed to. This is the case, for example, in the case that the range of the commitment scheme is $\{0,1\}^*$, and so any value is a valid commitment. Clearly, such a commitment scheme does *not* have the property that the committed value can be extracted from the committer.

3.1 Definition

We begin by informally defining perfectly binding commitment schemes.

Commitment schemes. A commitment scheme is a two-party protocol that enables a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver*. A commitment scheme must be both *hiding* and *binding*. The hiding property of a commitment scheme says that the receiver's view in the case that the sender commits to 0 is computationally indistinguishable from its view in the case that the sender commits to 1. (Thus, the committed value is unknown to the receiver.) The binding property of a commitment scheme states that in a later stage when the commitment is opened, the "opening" can yield only a single value that was determined in the committing phase. (Thus, the sender cannot modify its committed value.) In a *perfectly binding* commitment scheme, the binding property states that transcripts resulting from a commitment to 0 and a commitment to 1 must be *disjoint*. Thus, for any given transcript, there is at most one commitment value that can yield that transcript. See [22, Sec. 4.4.1] for a formal definition of commitment schemes.

Commit with extract. As we have mentioned, a commit-with-extract scheme is a commitment scheme with the following additional property: there exists a (strict polynomial-time) commitment extractor who is given the description of the sender and extracts the value being committed to during the commit stage of the protocol. In addition to outputting the committed value, we also require the extractor to output the sender's view of an execution (this is similar in spirit to witness-extended emulation and is needed when the commit-with-extract is used as a subprotocol).[10] Of course, the committed value and sender's view output by the extractor must be compatible. In order to enforce this compatibility, we denote by $\mathsf{commit\text{-}value}(\cdot)$ a function that takes a sender's view and outputs the *unique* committed value implicit in this view, or $\perp$ if no such value exists. (This function is well defined for perfectly-binding commitments because in such a case every transcript can define at most one value.) Then, compatibility between the view and committed value is obtained by requiring that $x = \mathsf{commit\text{-}value}(v)$, where x and v be the committed value and sender's view respectively, as output by the extractor. We now present the formal definition:

Definition 3.1 (commit with extract): *A perfectly binding commitment scheme C (with sender A and receiver B) is a* commit-with-extract *commitment scheme if the following holds: there exists a strict probabilistic polynomial-time commitment extractor CK such that for every probabilistic polynomial-time committing party A^* and for every $x, y, r \in \{0,1\}^*$, upon input $(\mathsf{desc}_{|x|}(A^*), x, y, r)$, machine CK outputs a pair, denoted $(CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r), CK_2(\mathsf{desc}_{|x|}(A^*), x, y, r))$, satisfying the following conditions:*

1. $\left\{CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r)\right\}_{x,y,r\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \{\mathsf{view}_{A^*}(A^*(x, y, r), B)\}_{x,y,r\in\{0,1\}^*}$
2. $\Pr[CK_2(\mathsf{desc}_{|x|}(A^*), x, y, r) = \mathsf{commit\text{-}value}(CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r))] > 1 - \mu(|x|)$

We note that the requirements on the extractor CK can be relaxed such that in the case that there is no committed value (i.e., where the view v is such that $\mathsf{commit\text{-}value}(v) = \perp$), then CK can output any arbitrary value, and not just $\perp$. This relaxation suffices for our applications.

Commit-with-extract using proofs of knowledge. We note that it is possible to achieve a commit-with-extract scheme in the following straightforward way. First, the sender sends a standard perfectly-binding commitment to the receiver. Then, the sender proves knowledge of the committed value using a zero-knowledge proof or argument of knowledge. A commitment extractor can easily be constructed for this scheme by having it run the knowledge extractor from the proof of knowledge and obtain the committed value. However, as mentioned above, known constructions of proofs of knowledge with strict polynomial-time extraction have a non-constant number of rounds. In contrast, our aim is to construct a commit-with-extract scheme that has a *constant* number of rounds.

Public Decommitment. We say that a commitment scheme satisfies public decommitment if the validity of a decommitment can be ascertained by any party who holds the transcript of the messages between the sender and receiver from the commitment stage. In particular, this party need not know the random coins used by the receiver during the commitment. More formally, the specification of a commitment scheme consists of two sender protocols and two receiver protocols: one protocol for the commit phase and one for the reveal phase. In general, the input of a protocol in the reveal phase may be the *entire view* of the corresponding party from the protocol of the commit phase. We say that a commitment scheme satisfies public decommitment if the input of the *receiver* protocol in the reveal phase may merely consist of the *transcript of messages* sent by the parties in the commit phase, and nothing else. That is:

[10]The extractor outputs the view of the *sender* (and not the receiver) because the extraction procedure is used in the simulation of a corrupted sender (not receiver).

Definition 3.2 (commitment schemes with public decommitment): *A commitment scheme satisfies* public decommitment *if the receiver's decision in the reveal phase depends only on the transcript of messages sent between the parties.*

This notion is analogous to that of "publicly verifiable" protocols, as described in Section 2.3. (Again, as described in Section 2.3, any perfectly binding commitment scheme can be modified into one that provides public decommitment by having the receiver send its random coins at the conclusion of the commitment phase. However, this may affect other properties of the commitment scheme. Indeed, the extractor for our commit-with-extract scheme would fail to generate a view that is indistinguishable from the sender's view, as required in Definition 3.1, if the receiver was required to send all of its random coins when the commitment phase concludes.) We use the additional feature of public decommitment for constructing a zero-knowledge argument of knowledge from a commit-with-extract scheme.

3.2 Constant-Round Commit-with-Extract

In this section we show how to construct a constant-round commit-with-extract commitment scheme. That is, we prove the following theorem:

Theorem 3.3 *Assume the existence of collision-resistant hash functions and collections of trapdoor permutations such that the domain of each permutation is the set of all strings of a certain length.*[11] *Then, there exist constant-round commit-with-extract string commitment schemes satisfying public decommitment.*

Before presenting our scheme, we note that it suffices to present a scheme that is perfectly binding, *except with negligible probability* (where the probability is taken over the coins of the receiver). A perfectly binding scheme (with no error) can then be obtained by augmenting the commitment phase with an additional perfectly binding commitment. Specifically, the sender will commit to the same value twice; once using a perfectly binding scheme (that does not enable extraction but is perfectly binding with no error), and once using a commit-with-extract scheme (that enables extraction but has a negligible error with respect to binding). The decommitment phase is then also modified so that both commitments must be opened. The result is a commit-with-extract scheme that is perfectly binding with no error. (We note that perfect binding with negligible error usually suffices, and so no such augmentation is really necessary.)

We now present our construction. In order to simplify the presentation, we start by showing a commit-with-extract *bit* commitment scheme and then show how to generalize our construction to a *string* commitment scheme. Our protocol is based on the following well-known non-interactive commitment scheme that uses one-way permutations [8]: Let f be a one-way permutation over $\{0,1\}^n$ and let b be a hard-core predicate of f. Then, in order to commit to a bit σ, the sender chooses $r \in_R \{0,1\}^n$, lets $y = f(r)$ and sends $\langle y, b(r) \oplus \sigma \rangle$ to the receiver. Loosely speaking, our commitment scheme is similar except that the value y is chosen *jointly* by the sender and the receiver using a coin-tossing protocol (which is based on the coin tossing protocol of [31]). Since y is uniformly distributed, the hiding property remains as in the original scheme. Likewise, because f is a permutation, y defines a unique value $b(f^{-1}(y))$ and thus the scheme remains perfectly binding. The novelty of our scheme is that for every sender, there exists an extractor that can bias the coin-tossing protocol such that it concludes with a value y for which the extractor knows the preimage $r = f^{-1}(y)$. In this case, the extractor can easily obtain the commitment value σ, as desired.

[11] See Footnote 3.

In order to allow the sender to be implementable by an efficient algorithm, we choose f to be a *trapdoor* one-way permutation. Thus, the sender is able to efficiently compute $r = f^{-1}(y)$, where y is the output of the coin-tossing protocol (this is similar to the NIZK system constructed in [16]). Formally, the protocol is parameterized by a family of trapdoor permutations over $\{0,1\}^n$, with a function sampling algorithm I. We denote a permutation from the family by f and its associated trapdoor by t. Furthermore, we denote by b a hard-core of f.

One of the components of the protocol is a constant-round zero-knowledge argument with a *strict* polynomial-time simulator. We note that such an argument exists if collision-resistant hash functions exist [3, 4]. As we have mentioned above, another component of our commit-with-extract protocol is a coin-tossing subprotocol that is based on the protocol of [31]. We do not plug in the exact protocol of [31] (while replacing the zero-knowledge proofs with those that run in strict polynomial-time), because its proof of security uses *extraction* from a proof of knowledge, and no proof of knowledge with strict polynomial-time extraction was previously known (indeed, providing such a proof is the aim of our construction).

Protocol 3.4 (commit-with-extract bit commitment scheme):

- **Input:** *The sender has a bit* σ *to be committed to.*
- **Commit phase:**
  1. A chooses a trapdoor permutation:
     (a) *The sender* A *chooses a trapdoor permutation* f *along with its trapdoor* t (*by running the sampling algorithm* I *on a uniformly chosen string* $s_I \in_R \{0,1\}^n$), *and sends* f *to the receiver* B.
     (b) A *proves to* B *that* f *is a permutation, using any constant-round zero-knowledge proof or argument* (*even one with an expected polynomial-time simulator*). *For example, if* I *is such that it outputs a permutation with probability* 1, *then* A *may prove that there exists a string* s_I *such that* f *is the permutation output from* $I(s_I)$.[12] *If* B *does not accept the proof, then it aborts.*
  2. A and B run a coin-tossing protocol:
     (a) B *chooses a random string* $r_1 \in_R \{0,1\}^n$ *and sends* $c = \mathsf{Commit}(r_1; s)$ *to* A (*where* $\mathsf{Commit}(\cdot)$ *denotes any perfectly-binding commitment scheme, and* $\mathsf{Commit}(r_1; s)$ *denotes a commitment to value* r_1 *using random coins* s).
     (b) A *chooses a random string* $r_2 \in_R \{0,1\}^n$ *and sends* r_2 *to* B.
     (c) B *sends* r_1 *to* A (*without decommitting*).
     (d) B *proves that the string* r_1 *sent in Step 2c is indeed the value that it committed to in Step 2a, using a constant-round zero-knowledge argument with a* strict *polynomial-time simulator. Formally,* B *proves that there exists a string* s *such that* $c = \mathsf{Commit}(r_1; s)$.
     (e) *The output of the coin-tossing phase is* $r_1 \oplus r_2$.
  3. A sends the actual commitment:
     A *computes* $r = f^{-1}(r_1 \oplus r_2)$ *and sends* B *the value* $v = b(r) \oplus \sigma$.
- **Reveal phase:**

[12] We note that by using the primality testers of [28] or [1], for example, the sampling algorithms for both the RSA and Rabin families of trapdoor permutations can be made to output a permutation with probability 1. In the case that the sampling algorithm does not output a permutation with probability 1, a different method of proving that f is a permutation is needed. General methods for achieving this can be found in [7].

1. *A sends B the string r.*
2. *B checks that $f(r) = r_1 \oplus r_2$. If this is the case, then B computes $b(r) \oplus v$ obtaining σ. Otherwise, B outputs $\perp$.*

(By convention, if the commit phase of the protocol is not completed, then the committed value is defined to equal 0.) We now prove that Protocol 3.4 is a secure commit-with-extract commitment scheme. We first show that it is a secure *commitment scheme*. This involves demonstrating both the binding and hiding properties. Intuitively, these properties hold because the only difference between the above protocol and the basic commitment scheme defined by $C_n(\sigma; r) \stackrel{\text{def}}{=} \langle f(r), b(r) \oplus \sigma \rangle$ for $r \in_R \{0,1\}^n$, is that the random string r is chosen via a coin-tossing protocol (rather than being determined by the sender).

Proposition 3.5 *Protocol* 3.4 *is a secure bit commitment scheme with public decommitment.*

Proof: We first claim that Protocol 3.4 satisfies public decommitment (see Definition 3.2). This is due to the fact that the only information needed by B to verify A's decommitment is the pair of strings r_1 and r_2, that appear in the transcript between A and B from the commit phase.

Next, we prove the (almost) perfect binding property. That is, we show that except with negligible probability, for any transcript of messages trans generated by an execution between an arbitrary probabilistic polynomial-time sender A^* and the honest receiver B, there exists a *unique* value $\sigma \in \{0,1\}$ such that $\mathsf{commit\text{-}value}(\mathsf{trans}) = \sigma$. Intuitively, the perfect binding property holds as long as the function f sent by A^* is a permutation. This is the case because when f is a permutation, the values r_1 and r_2 in the transcript define a unique value $r = f^{-1}(r_1 \oplus r_2)$, which in turn uniquely defines the value $\sigma = v \oplus b(r)$. The formal argument follows.

First, note that if B does not accept the proof provided by A^* in Step 1b, then B will abort and then, by convention, σ equals 0. Likewise, if A^* does not complete the entire commit phase, σ also equals 0. Therefore, the binding property trivially holds in these cases. We continue to show that it holds when B does not abort and the commit phase is completed.

Now, assume that the function f sent by A^* is a permutation. In this case, any pair of strings r_1 and r_2 appearing in the transcript define a single preimage $r = f^{-1}(r_1 \oplus r_2)$. Therefore, any bit v sent by A^* in Step 3 defines a single value $\sigma = v \oplus b(r)$. That is, the values r_1, r_2 and v in the transcript define a single committed value σ, as required. This, however, only holds as long as f is a permutation (otherwise, there may be more than one possible preimage to $r_1 \oplus r_2$). By the soundness of the proof (or argument) of Step 1b, we have that the probability that f is not a permutation is at most negligible. We therefore conclude that, except with negligible probability, the transcripts defines a single committed value.

We now turn to the computational hiding property. Intuitively, the hiding property follows from the hiding property of the non-interactive commitment scheme of [8], and the security of the coin-tossing protocol. In particular, if $r_1 \oplus r_2$ is uniform (or pseudorandom), then distinguishing between a commitment to 0 and a commitment to 1 is essentially equivalent to distinguishing between $\{f(U_n), b(U_n)\}$ and $\{f(U_n), b(U_n) \oplus 1\}$. Since b is a hard-core of f, it is infeasible to distinguish between these distributions in polynomial time. The hiding property therefore follows from the security of the coin-tossing protocol that ensures that $r_1 \oplus r_2$ is pseudorandom.

In the above intuition, the security of the coin-tossing protocol is reduced to a single instance of a commitment. However, in the actual proof, we reduce the indistinguishability of our commitment scheme to the indistinguishability of *multiple samples* of the basic commitment scheme relative to a single permutation f. That is, we show that distinguishing between a commitment to 0 and a commitment to 1 is equivalent to distinguishing between the random variables X_0^m and X_1^m where $X_\sigma^m = \langle f, f(U_n^{(1)}), b(U_n^{(1)}) \oplus \sigma, \ldots, f(U_n^{(m)}), b(U_n^{(m)}) \oplus \sigma \rangle$, where f is a trapdoor one-way permutation, b is a hard-core bit for f and $m = \mathrm{poly}(n)$. One can use a standard hybrid argument to show that X_0^m and X_1^m are computationally indistinguishable.

Formally, for any polynomial-size receiver B^*, denote by $v_n^{B^*}(\sigma)$ the distribution over B^*'s view, when the honest sender A commits to the value σ (the distribution is over the uniform choice of random coins for A). Then, the hiding property is stated as follows: for any polynomial-size receiver B^*, it holds that

$$\left\{v_n^{B^*}(0)\right\}_{n\in\mathsf{N}} \stackrel{\mathrm{c}}{\equiv} \left\{v_n^{B^*}(1)\right\}_{n\in\mathsf{N}}$$

Assume by contradiction, that there exists a polynomial-size receiver B^*, a polynomial-time distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many n's

$$\mathsf{adv}_n^D \stackrel{\mathrm{def}}{=} \left|\Pr[D(v_n^{B^*}(0)) = 1] - \Pr[D(v_n^{B^*}(1)) = 1]\right| \geq \frac{1}{p(n)} \tag{1}$$

We will use D and B^* to construct a distinguisher D' that will distinguish between the random variables X_0^m and X_1^m mentioned above, for $m = 5p(n)$. To get an intuition for the operation of D', consider the case in which B^* doesn't behave in an ostensibly faulty way (i.e., B^* does not "abort" the computation). If this is the case, we can actually construct a distinguisher D' for the basic commitment scheme (i.e., a distinguisher between X_0^1 and X_1^1). The distinguisher D' receives a commitment $C_n(\sigma) = \langle f(r),\ b(r) \oplus \sigma \rangle$ for input (where $r \in_R \{0,1\}^n$) and works by invoking B^* and runs an execution of Protocol 3.4 with B^*, until B^* sends a commitment $c = \mathsf{Commit}(r_1; s)$ as part of the coin tossing protocol. At this point, D' learns r_1 by running the continuation of the coin tossing protocol with B^*. After learning r_1, algorithm D' rewinds B^* back to the point after B^* sent the commitment to r_1, and then D' feeds B^* with the message $r_2 = f(r) \oplus r_1$, where $f(r)$ is the first part of its input commitment $C_n(\sigma) = \langle f(r), b(r) \oplus \sigma \rangle$. The result of the coin-tossing protocol is thus $r_1 \oplus r_2 = f(r)$, which means that as the final message D' can send B^* its input $b(r) \oplus \sigma$. We see that if D distinguishes between the result of this experiment when $\sigma = 0$ and when $\sigma = 1$, then D' breaks the basic commitment scheme C_n and distinguishes between X_0^1 and X_1^1. Unfortunately, the fact that B^* may ostensibly misbehave makes the formal proof somewhat more complicated than this description. We are now ready for the formal description of the distinguisher D'.

Distinguisher D' receives for input a trapdoor one-way permutation f and a sequence

$$\langle f, f(r^{(1)}), b(r^{(1)}) \oplus \sigma, \ldots, f(r^{(m)}), b(r^{(m)}) \oplus \sigma \rangle$$

where $r^{(1)}, \ldots, r^{(m)}$ are independently and randomly distributed in $\{0,1\}^n$. Algorithm D' then simulates an execution of A with B^* as follows:

1. *Simulation of A choosing a trapdoor permutation:*
   (a) D' passes B^* the permutation f that it was given as part of its input.
   (b) D' runs the zero-knowledge simulator (for the proof that f is a permutation) using the residual B^* as the verifier.[13]

[13]Note that *any* constant-round zero-knowledge proof or argument may be used for this step in the protocol. Therefore, the simulation of this argument by D' may require expected (rather than strict) polynomial-time. We therefore obtain a distinguisher that runs in expected polynomial-time. This suffices here because in the context of distinguishing commitments (or solving a hard problem), it is possible to truncate D''s execution without lowering its success below what is needed for deriving a contradiction.

2. *Sample an execution of the coin-tossing protocol:*

   (a) D' receives a commitment c from B^* (c is supposed to equal $\mathsf{Commit}(r_1)$ for some r_1).

   (b) D' chooses a random string $r_2 \in_R \{0,1\}^n$ and passes it to B^*.

   (c) D' obtains some string r_1 from B^*.

   (d) D' verifies the zero knowledge argument given by B^*. If D' accepts the argument from B^*, then it continues. Otherwise, D' sets Z to be the partial transcript until the point that the execution aborted and jumps to Step 5.

3. *Iterate until successful simulation:* D' does the following for $i = 1, 2, \ldots, m$

   (a) D' rewinds B^* to the point after B^* sent the commitment c (i.e., Step 2a) and sends B^* the string $r_2 = f(r^{(i)}) \oplus r_1$, where r_1 is the value obtained in Step 2c, and $f(r^{(i)})$ comes from D''s input sequence.

   (b) D' obtains some string r_1' from B^* and verifies the zero knowledge argument given by B^*. There are three possibilities at this point:

      i. If D' does not accept the argument from B^*, then it lets $i \leftarrow i+1$ and continues on to the next iteration (i.e., it returns to Step 3a). If the maximum number of attempts have elapsed (i.e., if $i = m$) then D' halts and outputs fail.

      ii. If D' does accept the argument from B^* but $r_1' \neq r_1$, then D' halts and outputs fail.

      iii. If D' accepts the argument from B^* and $r_1' = r_1$, then D' proceeds to Step 4 below.

4. *Simulation of the actual commitment:* D' passes the bit $v^i = b(U_n^{(i)}) \oplus \sigma$ (from its input sequence) to B^*. Algorithm D' lets Z denote the transcript of the simulated execution, and proceeds to Step 5.

5. *Output:* D' passes the transcript Z of the simulated execution to D, and outputs whatever D does.

   (We stress that this transcript (as passed by D' to D) may not be complete, as in the case that D' does not accept the argument from B^* in Step 2d.)

To prove that D' distinguishes between X_0^m and X_1^m with non-negligible probability, it suffices to prove the following claim:

Claim 3.5.1 *Let Z_σ be the random variable that denotes the simulated transcript that D' feeds to D in Step 5, when D gets X_σ^m as input. Let Y_σ be the random variable that denotes the view of B^* in an interaction with the sender A when the sender commits to σ. Then Z_σ and Y_σ are $\frac{2}{m}$-computationally indistinguishable. That is, for every polynomial-sized circuit C,*

$$\Big|\Pr[C(Z_\sigma) = 1] - \Pr[C(Y_\sigma) = 1]\Big| < \frac{2}{m}$$

Claim 3.5.1 is sufficient to show that D' distinguishes between X_0^m and X_1^m because our contradiction hypothesis is that (for infinitely many n's) the distinguisher D distinguishes between Y_0 and Y_1 with gap $\frac{1}{p(n)} = \frac{5}{m}$. Thus, Claim 3.5.1 implies that D will also distinguish between Z_0 and Z_1 with gap at least $\frac{1}{m}$, and hence D' will distinguish between X_0^m and X_1^m with this gap (in contradiction to their computational indistinguishability). We now prove the claim.

To prove Claim 3.5.1, we will use a hybrid argument with two intermediate random variables $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ that are computationally indistinguishable from Z_σ and Y_σ, respectively. We then show that $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ are $1/m$-indistinguishable.

The random variable $\tilde{Z}_\sigma$. The random variable $\tilde{Z}_\sigma$ is defined to be the result of the following process: Let $\tilde{D}$ be an algorithm that behaves exactly like D' except for the following two differences. First, instead of using a *simulated* zero-knowledge proof in Step 1b, it uses the honest prover algorithm (we assume that $\tilde{D}$ is given the trapdoor information for the one-way permutation). Second, if $\tilde{D}$ accepts the argument from B^* but $r_1' \neq r_1$, as in Step 3(b)ii, then it does *not* output fail. Rather, it proceeds to Step 4 and uses its knowledge of the trapdoor in order to complete the commitment like an honest committer (note that $\tilde{D}$ can derive the value of σ from its input sequence because it knows the trapdoor). We define $\tilde{Z}_\sigma$ to be the corresponding value computed by $\tilde{D}$ on input X_σ^m in Step 5.

Assuming that $\tilde{D}$ does not accept the argument from B^* when $r_1' \neq r_1$ (i.e., the case in Step 3(b)ii does not happen), the random variable $\tilde{Z}_\sigma$ is computationally indistinguishable from Z_σ because the only difference is whether the zero-knowledge protocol of Step 1b is simulated or real. It therefore suffices to show that the probability that $\tilde{D}$ accepts an argument from B^* when $r_1' \neq r_1$ is at most negligible (i.e., the case in Step 3(b)ii happens with at most negligible probability). In order to see this, notice that the case in Step 3(b)ii happens if $r_1 \neq r_1'$ and in addition, $\tilde{D}$ accepted an argument from B^* that $c = \mathsf{Commit}(r_1)$ (in Step 2d) and also an argument from B^* that $c = \mathsf{Commit}(r_1')$ (in Step 3b). However, the commitment c that $\tilde{D}$ receives from B^* in Step 2a is perfectly binding. Therefore, c defines a single decommitment value; in particular, it cannot be a commitment to both r_1 and r_1'. This means that one of the statements $c = \mathsf{Commit}(r_1)$ and $c = \mathsf{Commit}(r_1')$ is *false*. Therefore, by the soundness of the zero-knowledge argument, $\tilde{D}$ will accept both arguments from B^* (in Steps 2d and 3b) with at most negligible probability. We conclude that for large enough n, no polynomial-sized circuit can distinguish between $\tilde{Z}_\sigma$ and Z_σ with non-negligible probability.

The random variable $\hat{Z}_\sigma$. The random variable $\hat{Z}_\sigma$ is defined by considering the output of the algorithm $\hat{D}$. This algorithm behaves like $\tilde{D}$ except that it does not halt after m iterations. Rather, it continues until it accepts the argument from B^* in Step 3b (irrespective of whether or not $r_1 = r_1'$). According to the above description, in the i^{th} iteration, algorithm $\tilde{D}$ sends $r_2 = f(r^{(i)}) \oplus r_1$ to B^*. Thus, for the first $i \leq m$ iterations $\hat{D}$ works in the same way as $\tilde{D}$ and also sends $r_2 = f(r^{(i)}) \oplus r_1$ to B^*. However, if $\hat{D}$ exceeds m iterations, it cannot compute r_2 in this way (because its input includes only m commitments of the form $\langle f(r^{(i)}), b(r^{(i)}) \oplus \sigma \rangle$). Rather, it chooses r_2 uniformly at random (like the honest committer) and sends it to B^*. Then, if it accepts the argument from B^*, algorithm $\hat{D}$ proceeds to Step 4, computes $f^{-1}(r_1 \oplus r_2)$ and commits to σ like the honest committer. (Recall that, like $\tilde{D}$, machine $\hat{D}$ knows the trapdoor and so it can compute f^{-1} and can also obtain the value σ from its input sequence of commitments.) We stress that $\hat{D}$ uses fresh random choices for r_2 in every iteration $i > m$.

We claim that the statistical distance between $\hat{Z}_\sigma$ and $\tilde{Z}_\sigma$ is at most $\frac{1}{m}$. Indeed, one can see that the expected number of iterations that algorithm $\hat{D}$ runs is at most 1: let π be the partial execution of the protocol obtained by $\hat{D}$ until the point that B^* sends the commitment to r_1, and let p_π be the probability that B^* successfully proves the argument to $\hat{D}$, when continuing from the partial execution π. Then, the probability that $\hat{D}$ enters Step 3 equals p_π (because if B^* does not convince $\hat{D}$ in Step 2d, then $\hat{D}$ does not enter Step 3). Now, within the iterations of Step 3, the probability that B^* convinces $\hat{D}$ is exactly p_π. Therefore, given that $\hat{D}$ enters Step 3, the expected number of iterations is $\frac{1}{p_\pi}$. We conclude that the overall expected number of iterations made by $\hat{D}$ in Step 3 conditioned on the initial partial execution being π equals $p_\pi \cdot \frac{1}{p_\pi} = 1$. Averaging over all partial executions we obtain that the overall expected number of repetitions is also 1, and thus by Markov's inequality, we have that the probability that $\hat{D}$ will require more than m iterations is less than $\frac{1}{m}$. Notice finally that if $\hat{D}$ does not require more than m iterations, then it generates exactly the same distribution as $\tilde{D}$. Therefore, the statistical difference between $\tilde{Z}_\sigma$ and $\hat{Z}_\sigma$ is at most $\frac{1}{m}$.

Comparing $\hat{Z}_\sigma$ with Y_σ. Finally, we claim that the random variable $\hat{Z}_\sigma$ is identical to the variable Y_σ, which denotes the transcript of a real execution. In order to see this, observe that $\hat{D}$ essentially does the following. It first samples a partial execution until the point that B^* sends its commitment to r_1 (this sample is identical to a real execution). Next, it samples the remainder of the execution. If B^* does not convince $\hat{D}$ in the argument that it provides in this sample, then $\hat{D}$ outputs the transcripts and halts. In contrast, if B^* does convince $\hat{D}$ in this sample, then $\hat{D}$ continues until it obtains another sample in which B^* is convincing (all of this sampling is identical to a real execution). Assuming that $\hat{D}$ eventually halts, we have that the result is identical to Y_σ. The fact that $\hat{D}$ eventually halts follows from the fact that it only enters Step 3 if there is a non-zero probability of B^* convincing $\hat{D}$.

Combining the above, we have that the random variables Z_σ and Y_σ can be distinguished with advantage at most $\frac{1}{m} + \mu(n) < \frac{2}{m}$. With this the proof of Claim 3.5.1, and hence the proof of the computational hiding property, is completed. ■

Having proven that Protocol 3.4 is a secure commitment scheme, we proceed to show that it is also a *commit-with-extract* scheme, by demonstrating the existence of a *strict* polynomial-time extractor.

Proposition 3.6 *Protocol* 3.4 *constitutes a commit-with-extract commitment scheme.*

Proof: Intuitively, the extractor CK works by biasing the outcome $r_1 \oplus r_2$ of the coin-tossing protocol such that it knows the preimage under f. More specifically, CK chooses a random string r, computes $f(r)$ and then makes the output $r_1 \oplus r_2$ equal $f(r)$. This is clearly not possible for a real receiver (as the coin-tossing protocol ensures that $f(r)$ is pseudorandom). However, recall that CK has the description of the sender A^*, and therefore has more power than a real receiver. In particular, this gives CK the capability of running the simulator for the proof that B provides in Step 2d of the protocol. As we will see, this is enough.

Recall that CK should output a view indistinguishable from the one seen by A^* in a real interaction, as well as the unique commitment value defined by this view. Extractor CK receives the description of an arbitrary polynomial-time sender A^* and a triple (x, y, r), and works as follows:

1. A^* *chooses a trapdoor permutation:*
   (a) CK invokes $A^*(x, y, r)$ and receives the description of a permutation f from A^*.
   (b) Next, CK verifies the zero-knowledge proof from A^* attesting to the fact that f is a permutation. If the verification fails, then CK outputs A^*'s view until this point along with the value 0, and halts. (CK outputs 0 because, by our convention, in an aborted execution this is the default committed value.)
2. CK *biases the outcome of the coin-tossing protocol:*
   (a) CK passes $c = \mathsf{Commit}(0^n)$ to A^* (this is a commitment to "garbage").
   (b) CK obtains a string r_2 from A^*.

(c) CK chooses $r \in_R \{0,1\}^n$, computes $f(r)$ and passes A^* the string $r_1 = f(r) \oplus r_2$. (Notice that r_1 is distributed uniformly and independently of the initial commitment c, and that $f^{-1}(r_1 \oplus r_2) = r$.)

(d) CK invokes the zero-knowledge simulator, with the residual A^* as the verifier, for the appropriate (false) statement that there exists a string s such that $c = \mathsf{Commit}(r_1; s)$.

3. *A^* sends the actual commitment:*

   CK receives a bit v from A^*.

4. *Output:* CK outputs A^*'s view of the above execution along with $\sigma = b(r) \oplus v$.

We first claim that CK extracts the bit committed to in the execution (we focus on the case that A^* does not abort; otherwise the claim trivially holds). This is immediate because CK knows the preimage under f of $f(r_1 \oplus r_2) = f(r)$. Therefore, $b(r) \oplus v$ is exactly the unique value committed to by A^*. (The above assumes that f is indeed a permutation. However, by the soundness of the zero-knowledge argument of Step 1b, it can only occur that f is not a permutation with negligible probability.)

Next, we show that the view output by CK is computationally indistinguishable from A^*'s view in a real execution. These two distributions differ in two aspects: first, the commitment received by A^* in Step 2a is to 0^n rather than to r_1. Second, the zero-knowledge argument verified by A^* in Step 2d is simulated rather than real. Using a standard hybrid argument, computational indistinguishability can be shown. Specifically, define a hybrid experiment whereby A^* receives a commitment to r_1 (instead of to 0^n), and yet the zero-knowledge proof that it verifies is simulated. Then, by the hiding property of commitment schemes, A^*'s view in the hybrid experiment is indistinguishable from its view output by CK. On the other hand, by the indistinguishability of zero-knowledge simulation, A^*'s view in the hybrid experiment is indistinguishable from its view in a real execution. Combining the above together, we obtain that CK outputs a view that is indistinguishable from A^*'s view in a real execution.

It remains to show that CK runs in strict polynomial-time. However, this immediately follows from the above description and from the fact that the zero-knowledge simulator used by CK runs in strict polynomial-time. This completes the proof. ■

We note that any constant-round zero-knowledge argument system with a strict polynomial-time simulator suffices for Step 2d of Protocol 3.4. However, the only known such argument system is that of [3] (or its modified version in [4]) and this system utilizes a *non-black-box* simulator. Since the extractor must run this simulator, it follows that it is also non-black-box. As we will see in Section 5, this is in fact necessary for obtaining a constant-round protocol with strict polynomial-time extraction. (Recall that any constant-round zero-knowledge protocol suffices for Step 1b, even one with expected polynomial-time simulation.)

Extending Protocol 3.4 to strings: To prove Theorem 3.3 we need to generalize Protocol 3.4 to allow commitments to strings of length m (where m is polynomial in the security parameter n), instead of just allowing commitments to single bits. This extension can be obtained in two ways. Firstly, one can simply run Protocol 3.4 in parallel m times (taking care that the zero-knowledge arguments used are closed under the parallel composition of m executions, as is the case with the bounded-concurrent zero-knowledge protocol of [3, 4]). Alternatively, one can directly modify Protocol 3.4, and have A and B run m copies of the coin-tossing protocol in parallel, and then use only a single zero-knowledge argument to prove a compound statement relating to all copies.

Discussion – knowledge versus extraction. As we have mentioned, the notion of a commitment scheme with the additional property that the committed value can be extracted from the sender is not new. However, all previous constructions worked by first committing to a value, and then proving knowledge of this committed value. In contrast, Protocol 3.4 works in a completely different way: it does not consist of two distinct "commit" and "knowledge/extract" phases; rather the commitment and extraction are intertwined. Interestingly, this results in a subtle, yet important difference.

In order to exemplify this, consider the following sender A^*. Sender A^* obtains a one-way permutation and erases the corresponding trapdoor. Then, in Step 3 of Protocol 3.4, A^* sends a *random* bit $b \in \{0,1\}$ to B. Otherwise, A^* follows the instructions for A (and successfully concludes the commit stage). The important observation here is that since $r_1 \oplus r_2$ is uniformly distributed, A^* cannot guess the value of the bit that it itself committed to with probability non-negligibly greater than $1/2$. This is very strange, especially considering the fact that extraction is supposed to imply "knowledge".

Such a phenomenon cannot occur in an extractable commitment scheme that uses separate commit and extract phases. This is because the committed value is determined *before* the extraction begins. Therefore, the sender can apply the extractor to itself and thereby obtain the committed value. Thus, it makes sense to say that the sender "knows" the committed value. However, in our protocol, the committed value is only determined at the *conclusion* of the protocol. Furthermore, the value that is committed to may depend also on the random coins of the receiver B (as is indeed the case for the above-described sender A^*). Therefore, it is not possible for the sender to later "apply the extractor to itself in order to obtain the committed value".

Thus, on one hand, it seems that the intuition that the sender "knows" the committed value cannot be justified here. On the other hand, it seems that for applications that use such schemes, it suffices to extract the actual committed value, irrespective of whether or not the value is predetermined.

Open problem. Our construction of a commit-with-extract scheme requires trapdoor permutations (for finding the preimage to $r_1 \oplus r_2$), and collision-resistant hash functions (for the constant-round zero-knowledge argument with strict polynomial-time simulation [3, 4]). In contrast, when the commitment extractor may run in *expected* polynomial-time or when a *non-constant* number of rounds may be tolerated, such a scheme can be constructed based on one-way functions only. This raises an interesting question as to whether a constant-round commit-with-extract scheme can be constructed from one-way functions only.

4 A Zero-Knowledge Argument of Knowledge

Given a constant-round commit-with-extract commitment scheme, it is not hard to construct a constant-round zero-knowledge argument of knowledge with strict polynomial-time extraction, thereby proving Theorem 1. The basic idea is that the prover commits to a witness using the commit-with-extract scheme, and then proves that it has committed to a valid witness using a zero-knowledge proof of membership. Intuitively, soundness follows from the soundness of the zero-knowledge proof of membership and from the fact that the commit-with-extract scheme is perfectly binding. Zero-knowledge follows from the hiding property of the commit-with-extract scheme and from the zero-knowledge property of the proof of membership. Lastly, a knowledge extractor is immediately obtained from the extractor of the commit-with-extract scheme. Details follow.

Let R be an NP-relation. Without loss of generality, we assume that all witnesses for R are of the same length. We construct a zero knowledge argument of knowledge for R as follows:

Protocol 4.1 (zero-knowledge argument of knowledge for R):

- Common Input: x
- Auxiliary input to prover: *w such that $(x, w) \in R$.*
- Phase 1: *P and V run a commit-with-extract protocol (with public decommitment) in which P commits to the witness w.*
- Phase 2: *P proves to V, using a* constant-round *zero-knowledge proof (or argument) of membership (with a strict polynomial-time simulator) that it has committed to a valid witness w in the previous step.*

  Formally, let trans *be the transcript of the commit-with-extract execution of Phase* 1*, and let d be the decommitment message that P would send to V in the decommit phase of the commit-with-extract scheme. Then, P proves the NP-statement that there exists a value d such that* (trans, d) *defines a value w, such that $(x, w) \in R$.*[14]

In Phase 2, we use a zero-knowledge argument system with a strict polynomial-time simulator so that the resulting protocol will be a zero-knowledge argument of knowledge with both a strict polynomial-time extractor and a strict polynomial-time simulator. If the system used in Phase 2 has an *expected* polynomial-time simulator, then the resulting protocol will have a strict polynomial-time knowledge-extractor but an *expected* polynomial-time simulator.

Theorem 1 is obtained by combining the following proposition with Theorem 3.3 (i.e., the existence of commit-with-extract schemes).

Proposition 4.2 *Assume that the commitment scheme of Phase* 1 *is a* constant-round *commit-with-extract string commitment scheme. Then, Protocol* 4.1 *is a* constant-round *zero-knowledge argument of knowledge for R, as defined in Definitions* 2.1 *and* 2.2 *(i.e., it has a strict polynomial-time simulator and a strict polynomial-time knowledge-extractor). Furthermore, there exists a witness-extended emulator for Protocol* 4.1*, as defined in Definition* 2.3.

Proof: In order to prove that Protocol 4.1 is a zero-knowledge argument of knowledge, we prove three properties: completeness, knowledge soundness (which implies computational soundness), and zero knowledge. The proof of completeness is immediate. We proceed to prove zero knowledge and knowledge soundness.

Zero-knowledge. The simulator S that we build to demonstrate the zero knowledge property works as follows. In Phase 1 of the protocol, S follows the honest sender strategy of the commit-with-extract scheme, but instead of committing to a real witness w (which it does not have), it commits to garbage (e.g., to all zeros). Next, in Phase 2, simulator S cannot prove that it committed to a correct witness (because it indeed did not). Rather, S runs the simulator for the zero-knowledge proof of Phase 2. The hiding property of the commit-with-extract scheme implies that the commitment to garbage is indistinguishable from a commitment to a real witness. In addition, the zero-knowledge property of the proof of membership implies that the simulation

[14] Here we use the assumption that the commit-with-extract scheme satisfies public decommitment as in Definition 3.2. Otherwise, the statement that P needs to prove is not guaranteed to be in $\mathcal{NP}$.

is indistinguishable from a real proof of membership. Combining these together (and using a standard hybrid argument), we obtain that the overall simulation by S is indistinguishable from a real execution of Protocol 4.1. Formally, let V^* be a verifier algorithm for Protocol 4.1. Then the simulator S works as follows:

Algorithm 4.3 (simulator S):

- Input: $x \in L, z \in \{0,1\}^*$

1. *S plays the honest sender for the commit-with-extract scheme with $V^*(x,z)$ as the receiver, and commits to $0^{p(|x|)}$ (where $p(n)$ is the length of all witnesses for statements of length n).*

2. *Let trans be the series of messages sent to V^* in the previous step, and denote by $V^*(x,z,\mathsf{trans})$ the residual machine who verifies the proof in Phase 2.*[15] *Then, S runs the simulator for the zero-knowledge proof of Phase 2, with $V^*(x,z,\mathsf{trans})$ as the verifier.*

3. *Output whatever V^* outputs (without loss of generality, we assume that V^* always outputs its view).*

We need to prove that:

$$\{S(x,z)\}_{x\in L, z\in\{0,1\}^*} \stackrel{\rm c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\}_{x\in L, y\in R_L(x), z\in\{0,1\}^*}$$

where P is the honest prover algorithm. This is proven using a standard hybrid argument. We define an intermediate distribution $H_{x,y,z}$ in the following way: $H_{x,y,z}$ is produced by an algorithm S' that follows the honest prover's strategy in the first phase, and the simulator's strategy in the second phase (in order to enable it to run the honest prover's strategy in the first phase, it is explicitly given a valid witness y). That is, on input (x,y,z) where $(x,y) \in R$, algorithm S' runs the commit-with-extract algorithm and commits to the value y as the honest prover does (instead of to $0^{p(|x|)}$ as the simulator S would). However, in Phase 2, algorithm S' runs the zero-knowledge simulator on $V^*(x,z,\mathsf{trans})$ exactly as S does (instead of really proving the statement as the honest prover would).

The fact that $\{S(x,z)\} \stackrel{\rm c}{\equiv} \{H_{x,y,z}\}$ follows directly from the hiding (secrecy) property of the commit-with-extract scheme. The fact that $\{H_{x,y,z}\} \stackrel{\rm c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\}$ follows directly from the (auxiliary-input) zero-knowledge property of the proof of membership used in Phase 2. These two facts together imply that $\{S(x,z)\} \stackrel{\rm c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y), V^*(x,z))\}$, as required.

We note that the simulator S inherits the properties of the underlying simulator for the proof of Phase 2. Thus, if the underlying simulator is strict polynomial-time and non-black-box, then so too is S. On the other hand, if the underlying simulator is black-box or runs in expected polynomial-time, then the same is true for S.

Knowledge soundness. Let P^* be a (possibly cheating) prover that convinces the honest verifier that $x \in L$ with probability ϵ. The extractor for the zero-knowledge argument simply uses the extractor CK of the commit-with-extract scheme in order to obtain P^*'s view of the first phase, along with a string w that is the unique value that is committed to in this phase. Intuitively, w must be a valid witness with probability at least $\epsilon - \mu(|x|)$, for some negligible function $\mu(\cdot)$. This

[15]The residual machine $V^*(x,z,\mathsf{trans})$ is formally defined as the machine who upon receiving a sequence of messages $(\alpha_1,\ldots,\alpha_i)$ replies with the message that $V^*(x,z)$ would send upon receiving the sequence of messages $(\mathsf{trans},\alpha_1,\ldots,\alpha_i)$.

holds because in Phase 2 of the protocol, P^* proves the validity of the committed witness. It then follows from the soundness of the proof of membership that if w is not a valid witness, the verifier rejects (except with negligible probability). In the formal proof of this, we also use the fact that P^* "behaves" in a similar way in its interaction with CK and in a real interaction with the honest verifier. (If this was not the case, then P^* may always convince V, but never commit to a valid witness with CK. The extractor CK would then never extract a valid witness from P^*.) We note that this fact regarding the similarity of P^*'s behavior with CK and V follows from the fact that CK also outputs a view that is computationally indistinguishable to P^*'s view in a real interaction with V. We now formally describe the knowledge extractor algorithm K:

Algorithm 4.4 (knowledge extractor K):

- Input: $(\mathsf{desc}_{|x|}(P^*), x, y, r)$

  1. *Let CK be the extractor for the commit-with-extract scheme. Then, invoke CK on input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$, and obtain the view of P^* in Phase 1, denoted v, along with a string w that is the committed value corresponding to that view (i.e., $w = \mathsf{commit\text{-}value}(v)$).*

  2. *Output w.*

Let $p(x, y, r)$ be the probability that $P^*(x, y, r)$ convinces the honest verifier on input x in a real execution. We claim that the probability that the witness output by K is valid is at least $p(x, y, r) - \mu(|x|)$, for some negligible function $\mu(\cdot)$. In order to show this, consider a modified extractor $\tilde{K}$ that has the same Step 1 as K, but in Step 2 it first verifies the proof of membership provided by P^* (in Phase 2 of the protocol) and then outputs w if and only if it accepts this proof. Formally, in Step 1, $\tilde{K}$ runs the extractor CK and obtains a pair (v, w), as described for K. Then, in Step 2, $\tilde{K}$ verifies the proof of membership given in Phase 2 by the residual prover P^* that is defined by the view v (i.e., the residual prover is the original P^*, but with its current view set to v). Extractor $\tilde{K}$ then outputs w if and only if it accepts this proof.

Now, since $\tilde{K}$ only contains additional checks, it follows that K outputs w whenever $\tilde{K}$ outputs w. Therefore, it suffices to show that $\tilde{K}$ outputs a correct witness w with probability at least $p(x, y, r) - \mu(|x|)$. In order to demonstrate that $\tilde{K}$ succeeds with this probability, first recall that by the definition of a commit-with-extract scheme, it holds that

$$\left\{\mathsf{view}^{CK}_{P^*}(x, y, r)\right\} \stackrel{c}{\equiv} \left\{\mathsf{view}_{P^*}(P^*(x, y, r), V)\right\}$$

where $\mathsf{view}^{CK}_{P^*}(x, y, r)$ is the random-variable describing the view of P^* as output by CK, and $\mathsf{view}_{P^*}(P^*(x, y, r), V)$ is the random-variable describing the view of P^* in a real execution with V. Therefore, the probability that $\tilde{K}$ accepts the proof of membership from the residual P^* is negligibly close to the probability that the honest verifier V accepts this proof of membership. (Otherwise, it is possible to distinguish between P^*'s real view and the view output by CK by emulating an execution of the proof between the residual prover P^* and the verifier $\tilde{K}$, and then outputting 1 if and only if $\tilde{K}$ accepts. This emulation is carried out by running the residual prover that is defined by the view v and seeing if the honest verifier accepts. Note that this emulation can be carried out because P^* is polynomial-time and, in addition, the description of P^* and the view v fully define the residual prover. Therefore, the residual prover can be obtained and run in polynomial-time.) Thus, $\tilde{K}$ accepts the proof of membership with probability at least $p(x, y, r) - \mu(|x|)$. By the soundness of this proof of membership, it must be that w is a valid witness with probability at least $p(x, y, r) - \mu'(|x|)$, for some negligible function μ' (otherwise, the residual prover is able to prove a false statement with non-negligible probability). This completes the proof of knowledge soundness.

Witness-extended emulation. We conclude by providing a proof of the existence of a witness-extended emulator E for Protocol 4.1. Recall that, upon input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$, E must output a view that is indistinguishable from $P^*(x, y, r)$'s view in a real execution. Furthermore, if this view contains a transcript in which the honest verifier V accepts the proof, then E must also output a valid witness (except with negligible probability). This is easily accomplished as follows:

Emulator E extracts a witness in the same way as the extractor K described above. However, E must also output P^*'s complete view of an execution. In order to do this, after CK concludes, E proceeds to verify the proof of membership of Phase 2 (like the aforementioned $\tilde{K}$). This suffices to obtain P^*'s entire view: the commit-with-extract extractor CK provides P^*'s view from Phase 1, and the remainder of P^*'s view is derived from the verification of the proof of membership of Phase 2.

More formally, the witness-extended emulator E invokes the commit-with-extract extractor CK with input $(\mathsf{desc}_{|x|}(P^*), x, y, r)$ and obtains the output. This output contains a view v that is indistinguishable from P^*'s view in a real execution of commit-with-extract. Furthermore, if this view defines a committed value, then CK also outputs this value (with overwhelming probability). Next, E verifies the proof of membership from the residual prover P^* that is defined by the view v (as described above for $\tilde{K}$). In this proof, E obtains the residual P^*'s view of Phase 2. Then, E concatenates the view output by CK in Phase 1 with P^*'s view in Phase 2, and outputs them both as P^*'s view of the entire execution. Furthermore, if E accepted the proof of Phase 2, then it outputs the witness obtained from CK in Phase 1.

It is easy to see that the view output by E is indistinguishable from P^*'s view in a real execution (P^*'s view from Phase 1 is indistinguishable from its view in a real execution and the view from Phase 2 is identical). Furthermore, if E accepts the proof of Phase 2, then with overwhelming probability the transcript of Phase 1 defines a valid witness. As we have mentioned above, by the properties of CK, it follows that with overwhelming probability it also outputs this witness. This concludes the proof. ■

5 Black-Box Lower Bounds

In this section, we show that there does not exist a constant-round zero-knowledge argument (resp., argument of knowledge), with a black-box simulator (resp., extractor) that runs in strict polynomial-time. That is, we prove Theorems 2 and 3.

Before presenting the proofs, we provide intuition as to why it is not possible to obtain a strict polynomial-time black-box extractor for constant-round zero-knowledge protocols. First, consider a very simple (cheating) prover P^* who at every step either aborts or sends the honest prover message. Furthermore, the probability that it does not abort at any given step is $\epsilon = \epsilon(n)$. Then, black-box extractors for constant-round zero-knowledge protocols typically extract a witness using the following strategy: Invoke an execution with P^* and if P^* aborts, then also abort. However, if P^* does not abort (and thus sends a prover message in some crucial round), then continually rewind P^* until another prover message is obtained for this round. It is essential that the extractor obtains at least two different prover messages, because this is what gives it additional power over an honest verifier. (Additional power is essential because an honest verifier should not learn anything from the prover, whereas the extractor must obtain the actual witness.) Now, for P^* described above, the extractor enters the "rewinding stage" with probability only ϵ. However, once it enters this stage, the expected number of rewinding attempts by the extractor until a second non-abort reply is obtained equals $1/\epsilon$ (because P^*'s non-abort probability at every step is only ϵ). Combining this together, we have that the overall expected amount of work is bounded. However, we cannot provide *any* strict polynomial upper-bound on the running time of the simulator, because for every given polynomial, it is possible to choose ϵ so that $1/\epsilon$ is greater than this polynomial.

This idea underlies our lower bounds. Specifically, we show that by carefully choosing the abort probabilities, it is possible to achieve the following effect: A strict polynomial-time black-box extractor will not have "time" to obtain two non-abort responses from the prover P^* in any given round. (By "not having time", we mean that with noticeable probability, the extractor will have to wait longer than the bound on its running-time in order to see a second non-abort response.) Essentially, this means that the extractor cannot "rewind" the prover. However, as we have mentioned above, an extractor must have additional power over a real verifier, and the only additional power awarded a *black-box* extractor is essentially the ability to rewind the prover. We therefore conclude that strict polynomial-time black-box extractors cannot exist for constant-round zero-knowledge protocols. The same argument also holds for strict polynomial-time simulation of constant-round zero-knowledge protocols. We now proceed to the proofs.

Theorem 5.1 (Theorem 2 – restated) *There do not exist* constant-round *zero-knowledge proofs or arguments with* strict polynomial-time black-box *simulators for any language* $L \notin \mathcal{BPP}$.

Theorem 5.2 (Theorem 3 – restated) *There do not exist* constant-round *zero-knowledge proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for any relation* R *such that the language* $L_R \notin \mathcal{BPP}$.

Our proofs of the above lower bounds closely follow the methodology and techniques used in previous black-box lower bounds [25, 13].

5.1 Outline of the Proofs

As we have mentioned above, the underlying idea behind the proofs of both Theorem 5.2 and Theorem 5.1 is the same. We will explain the intuition behind this idea in the context of knowledge extraction (i.e., in the context of the proof of Theorem 5.2) and then describe how this intuition generalizes also to the context of simulation (for the proof of Theorem 5.1).

Theorem 5.2 is proven by showing that if a language L has a constant-round zero-knowledge protocol (P, V) with a black-box strict polynomial-time knowledge extractor, then there exists a cheating verifier strategy V^*, such that for every $x \in L$, if V^* interacts with the honest prover P, then with noticeable probability, V^* will obtain a witness w for x from the interaction with P. Of course, the ability to do this contradicts the zero-knowledge property of the protocol, unless the verifier V^* could anyway obtain a witness by itself. Since V^* runs in probabilistic polynomial-time, it must therefore be the case that $L \in \mathcal{BPP}$ (because then, indeed, V^* could obtain a witness by itself).

We construct this verifier V^* from the black-box knowledge extractor of the system (P, V). Loosely speaking, this is done in the following way:

1. We let $x \in L$ and consider a cheating prover strategy P^* that is of the following form: P^* behaves exactly as the honest prover P behaves on input x, except that in each round of the proof, it may choose to abort the execution with some probability. We will choose these probabilities so that P^* will still have a noticeable probability to convince the honest verifier to accept x.

   The actual strategy that we use for P^* is one that causes it to abort with quite high probabilities (but still noticeably bounded away from 1). In particular, P^* will abort in each round with probability greater than $1/2$ (and even greater than $1-1/n$). Note however that since the number of rounds in the protocol is constant, this does not preclude P^* from causing the honest verifier to accept with noticeable probability.

2. Consider an execution of the knowledge extractor when it is given black-box access to P^*. Every query that the extractor makes to the black-box is a list of messages $(\alpha_1, \ldots, \alpha_i)$, and the reply received by the extractor is what P^* would send in a real execution when the first i verifier messages were $\alpha_1, \ldots, \alpha_i$. Thus, if P^* would abort on this series of messages in a real execution, then the extractor receives back an abort reply, denoted $\perp$. Note that at the end of the execution the knowledge extractor should output a witness for x with some noticeable probability.

   We show that it is possible to choose P^*'s abort probabilities in such a way, that if we run the knowledge extractor with black-box access to P^* then with very high probability, the extractor will get at most c non-abort replies, where c is the number of verifier messages in the protocol. Furthermore, these replies will correspond to i queries (where $i \leq c$) of the form (α_1), (α_1, α_2), …, $(\alpha_1, \ldots, \alpha_i)$ where $\alpha_1, \ldots, \alpha_i$ are some strings. That is, all these queries are prefixes of a single sequence $(\alpha_1, \ldots, \alpha_i)$. Notice that when this occurs, the extractor's view is like a real interaction with the honest prover P (in particular, it does not gain anything by "rewinding" P^*).

3. We then implement a verifier strategy V^* that sends these messages $\alpha_1, \ldots, \alpha_i$ when it interacts with the prescribed prover P. Basically, the verifier works by internally running the knowledge extractor. When the extractor makes a query, the verifier either answers it with $\perp$ or forwards the query to the prover, and then returns the prover's reply to the knowledge extractor. We show that the verifier has a noticeable probability of perfectly simulating P^* to the knowledge extractor. Therefore, with noticeable probability, the knowledge extractor (in conjunction with the verifier) is able to extract a witness x during this interaction with the prover P. We conclude that with noticeable probability, V^* obtains a witness w for x from the interaction with P, and so the proof system cannot be zero-knowledge (unless $L \in \mathcal{BPP}$).

As mentioned above, in the simulation case (i.e., when proving Theorem 5.1) we use a similar technique. Basically, we prove Theorem 5.1 by showing that if L has a constant-round zero-knowledge proof or argument system (P, V) with a black-box strict polynomial-time simulator, then there exists a cheating prover strategy P^* that does not have any auxiliary input (like a witness), and yet for every $x \in L$ causes the honest verifier V to accept x with noticeable probability. We stress that, unlike the honest prover, this cheating strategy P^* does *not* get a witness for x as auxiliary input. It is not hard to show that the existence of such a strategy P^* implies that $L \in \mathcal{BPP}$.

In summary, both impossibility results are due to the following two facts:

1. Intuitively, in order to successfully extract or simulate, the extractor (resp., simulator) must see at least two different continuations of the same transcript. That is, it must get meaningful replies to queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$ where $\alpha_i' \neq \alpha_i$. Otherwise the extractor (resp., simulator) does not have any advantage over the interactive verifier (resp., prover). (Informally speaking, "rewinding" is essential for black-box simulation and extraction.)

2. For any strict polynomial-time extractor or simulator, there exist provers (resp., verifiers) for which the time required to obtain non-abort responses to two queries $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$

and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i)$ where $\alpha'_i \neq \alpha_i$ is greater than the running-time of the extractor (resp., simulator).

Comparison with the black-box zero-knowledge lower bounds of [25, 13]. As we have mentioned above, we use the methodology and techniques of [25, 13] in proving our lower bounds. The lower bounds of [25, 13] and this paper all work by using a black-box simulator to construct a cheating prover P^* that convinces an honest verifier that $x \in L$, without having a witness. The prover P^* works by internally invoking the simulator and forwarding some of the messages between the simulator and the external honest verifier (the other messages are dealt with internally). The main issue to be resolved is how to carry out this execution of the simulator; the difficulty being due to the fact that the simulator is allowed to rewind the verifier during simulation, whereas the cheating prover *cannot* rewind the external honest verifier. Regarding this issue, each of the three lower bounds differ. The lower bound of [25] for constant-round public-coin proofs follows from the fact that in public-coin proofs, all verifier messages are independent of each other. Therefore, it is possible to "guess" messages that should be forwarded to the verifier, and all other messages can be internally generated by the cheating prover (independence of the verifier messages is necessary so that the internal messages generated by the cheating prover do not look inconsistent with messages that the external honest verifier generates). The lower bound of [13] for the concurrent setting is proven by showing that a polynomial-time simulator is unable to rewind the verifier in every execution (because this would require super-polynomial time). Therefore, the messages of the execution in which there is no rewinding can be forwarded to the external real verifier. Finally, in our lower bound, we show that a strict polynomial-time simulator must succeed without effectively rewinding the verifier because rewinding attempts will yield $\perp$ replies with high probability. Therefore, the messages of the simulator that do not yield $\perp$ replies can be sent to the external verifier and no rewinding will be necessary.

5.2 Proof of Theorems 5.2 and 5.1

We now proceed to the actual proofs.

Notation and Conventions: We identify an interactive program A with its *next message function* (or process, if it is randomized). That is, we consider A as a *non-interactive* algorithm that gets as input the history of the interaction (i.e., the sequence of messages that A received until this point), and outputs the next message that A would send in a protocol execution in which it sees this history.

We say that an algorithm B has *oracle access* to an interactive algorithm A, if B has oracle access to A's next message function (after fixing its random tape to some value). That is, B can query its oracle with any sequence of messages of the form $(\alpha_1, \ldots, \alpha_i)$ and it will receive back the next message that A would send in an interaction in which it received this sequence of messages.

We say that an interactive program *aborts* an execution if it sends the message $\perp$. We assume that once a party sends the message $\perp$, then (since it aborted the execution) all its future messages are also $\perp$. In the notation of the next message function, this means that if $A(\alpha_1, \ldots, \alpha_i) = \perp$ for some strings $\alpha_1, \ldots, \alpha_i$, then $A(\alpha_1, \ldots, \alpha_i, \alpha_{i+1}, \ldots, \alpha_j) = \perp$ for every choice of $\alpha_{i+1}, \ldots, \alpha_j$.

The following two lemmas (whose proofs are very similar) lie at the heart of the proofs of Theorem 5.2 and Theorem 5.1 (respectively).

Lemma 5.3 *Let (P, V) be a constant-round system of proofs or arguments of knowledge for a relation R with a black-box strict polynomial-time knowledge extractor K. Then, there exists a probabilistic polynomial-time cheating verifier algorithm V^* and a polynomial $p(\cdot)$ such that for every $x \in L_R$, the probability that V^* outputs a witness for x after interacting with the honest prover P on input x is at least $1/p(|x|)$.*

We later show that if the proof system (P, V) is zero knowledge, then the existence of such a cheating verifier V^* implies that $L \in \mathcal{BPP}$. Likewise, we prove:

Lemma 5.4 *Let (P, V) be a constant-round zero-knowledge proof or argument system for a language L, with a black-box strict polynomial-time simulator S. Then, there exists a probabilistic polynomial-time cheating prover algorithm P^* and a polynomial $p(\cdot)$ such that for every $x \in L$, the honest verifier V accepts after interacting with $P^*(x)$ with probability at least $1/p(|x|)$.*

We stress that P^*'s only input is x and, in particular, it does *not* receive a witness for x as auxiliary input. Later we will show that the existence of such a prover P^* implies that $L \in \mathcal{BPP}$ (as shown in previous black-box lower bounds for zero-knowledge; e.g., see [25, 13]).

5.2.1 Proof of Lemma 5.3

Let R be a relation and let (P, V) be a system of proofs or arguments of knowledge for R, where the number of messages sent by the verifier equals some constant c. Furthermore, let K be a black-box knowledge extractor for (P, V) that when extracting a witness for x runs for at most $t(n)$ steps, where $n = |x|$ and $t(\cdot)$ is some polynomial. Thus, the number of oracle queries made by K is strictly upper-bound by a polynomial $t(n)$.[16] We assume without loss of generality that all verifier messages in the proof system are of length $m = m(n)$, where $m(\cdot)$ is some polynomial. We also assume without loss of generality that the first message in the system is from the verifier to the prover.

Conventions regarding the extractor K: Recall that a black-box extractor has oracle access to the prover from whom it extracts. Thus it can query this oracle on sequences of messages of the form $(\alpha_1, \ldots, \alpha_i)$. For the sake of simplicity, we can assume without loss of generality, that the extractor K always behaves in the following way:

1. It never asks the same query twice.
2. If K queries the oracle with q, then prior to this query, it has queried the oracle with all the proper prefixes of q. (If $q = (\alpha_1, \ldots, \alpha_i)$ is a sequence of messages, then the *prefixes* of q are all the sequences of the form $(\alpha_1, \ldots, \alpha_j)$ for $j \leq i$.)
3. For some polynomial $t' = t'(n)$, the extractor makes exactly $t'(n)$ queries to its black-box in every execution.

[16] Note that the running time of the extractor is independent of the running time of the prover. That is, the extractor is restricted to $t(n)$-time (and thus oracle queries) even if the cheating prover runs in time that is larger than $t(n)$. This may seems like an "unfair" restriction, even for a black-box extractor. However, we can extend our lower bound to hold even if the running time of the extractor is allowed to be a fixed polynomial in the running time of the cheating prover. This extension holds under the assumption that one-way functions exist and applies to any proof system that has an efficient prover algorithm. This can be shown by considering a prover strategy P^* that uses a pseudorandom function instead of a t-wise independent hash function, see below.

We can assume all of the above because any extractor can be easily modified such that it behaves in the above way, without affecting its output distribution. Furthermore, if the upper bound on the number of queries made by the original extractor is $t(n)$, then the number of queries t' made by the modified extractor is at most $c \cdot t(n)$ (the multiplicative factor of c is used for asking all of the prefixes of a query before the query itself). From now on, we will denote the number of queries made by K by $t = t(n)$, rather than by $t'(n)$.

Our proof will follow the outline mentioned at the beginning of Section 5. That is, we will construct a prover strategy P^* that aborts with certain probability p^*, but when it does not abort, uses the same strategy as the honest prover. On the one hand, the knowledge extractor K will have to output a witness with probability close to p^* when given access to P^*. On the other hand, we will show a cheating verifier V^* that can successfully simulate the behavior of K^{P^*} with noticeable probability, even though it only gets access to one interaction with the honest prover P (and does not get black-box access to P^*).

The prover strategy P^*. We are now ready to describe the prover strategy P^*. Let $\epsilon = \epsilon(n)$ be some value that will be determined later (it will be of the form $1/q(n)$ for some polynomial $q(\cdot)$). Let $\mathcal{H} = \{H_n\}_{n \in \mathsf{N}}$ be a family of $t(n)$-wise independent hash functions,[17] such that every $h \in H_n$ is a function from $\{0,1\}^{\leq c \cdot m}$ to $\{0,1\}^n$, where $\{0,1\}^{\leq c \cdot m}$ denotes the set of all strings of length at most $c \cdot m$. (Recall that $t = t(n)$ denotes the number of queries the knowledge extractor K makes to its black-box.)

Our prover strategy P^* behaves as follows: in the i^{th} round of the protocol, with probability $\epsilon^{2^{c-i}}$ it behaves exactly as the honest prover, and otherwise (with probability $1 - \epsilon^{2^{c-i}}$) it aborts. The random coins the prover P^* uses to decide whether to abort or continue will be chosen by applying a fixed hash function $h \in H_n$ (that was initially chosen at random) to the current message history. Before proceeding to a more formal description of P^*, we explain why we choose its abort probabilities in this way. As we have described above, the main idea is to prevent an extractor from obtaining two non-abort responses from P^* for the same round. More exactly, we wish to fix the abort probabilities so that the probability that an extractor sees two non-abort responses for the same round is *significantly less* than the probability that P^* provides a full proof without aborting at all, in which case it convinces V. (Since the extractor must obtain a witness with the probability that P^* convinces V, this means that there is a significant probability that the extractor will *not* see two non-abort messages for any given round, but must still succeed in obtaining a witness.) Now, by choosing the abort probabilities so that they degrade exponentially, we achieve our aim. Specifically, for any i, the product $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^{c-i+1}} \cdot (\epsilon^{2^{c-i}})^2$ (which is the probability that two non-abort responses are obtained for the i^{th} round) is *significantly less* than the product $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^0}$ (which is the probability that P^* never aborts).

We now describe the strategy for P^*'s operation (note that we define P^* in the form of its next-message function):

Algorithm 5.5 (Prover P^*:)

- Common input: x – *the statement to be proven*
- Auxiliary input: $y \in R(x)$ – *a witness for* x

[17] Recall that a hash family is t-wise independent if for every t distinct values $x_1, \ldots, x_t$ the random variables $h(x_1), \ldots, h(x_t)$ (where h is chosen at random from H_n) are independently and uniformly distributed in the range of h. We will never make more than t queries to the function, and so one can think of it as a truly random function.

- Random tape: (h, r) – *h defines a function in H_n, and r is of the length of the random tape required by the honest prover strategy*
- Input: *series of verifier messages $q = (\alpha_1, \ldots, \alpha_i)$*

  1. Step 1 – decide whether or not to abort:

     (a) Compute $h(q')$ for every prefix q' of q. That is, for every j $(1 \leq j \leq i)$, compute $h(\alpha_1, \ldots, \alpha_j)$.

     (b) Abort (outputting a special symbol $\perp$), unless for every j, the first $2^{c-j} \log(1/\epsilon)$ bits of $h(\alpha_1, \ldots, \alpha_j)$ are equal to 0.

     (Notice that since the definition of P^* is by its next message function, we have to ensure that it replies to q only if it would not have aborted on messages sent prior to q in an interactive setting. This is carried out by checking that it would not have aborted on all prefixes of q.)

  2. Step 2 – if the decision is to not abort, then follow the honest prover strategy:

     (a) Run the honest prover P, with initial input (x, y) and random tape r, on input messages $(\alpha_1, \ldots, \alpha_i)$, and obtain its response β.

     (b) Return β.

Note that if the honest prover P was computationally efficient then so is the cheating prover P^*. (This is important because for the case of arguments, all provers must be efficient.) Our first step is to show that P^* convinces V with noticeable probability. Suppose that the system (P, V) has completeness bound p (i.e., the honest prover causes the honest verifier to accept with probability at least p; recall that p is assumed to be noticeable). Then, define $p^* = \epsilon^{2^c-1}p$. We claim that the probability (over P^*'s random tape) that P^* convinces the honest verifier to accept is at least p^*. Indeed, conditioned on P^* not aborting, its behavior is identical to the behavior of the honest prover P, in which case it convinces V with probability p. Now, since P^*'s probability of not aborting is equal to $\epsilon^{2^{c-1}} \cdot \ldots \cdot \epsilon^{2^0} = \epsilon^{2^c-1}$ the claim follows. As we have mentioned above, we will choose ϵ so that $\epsilon > 1/q(n)$ for some polynomial $q(\cdot)$. Then, since c is a constant, it follows that the probability p^* that P^* convinces the honest V is noticeable, as desired.

We have shown that P^* convinces V with probability at least $p^* = \epsilon^{2^c-1}p$. By the validity (or knowledge soundness) condition on the system (P, V), this means that when the knowledge extractor K is given oracle access to P^*, then it outputs a witness for the statement x with probability at least $p^* - \mu(|x|)$ (where the probability is over the random tapes of both P^* and K).[18] In particular, K outputs a witness with probability at least $p^*/2$.

We now claim that if ϵ is chosen appropriately, then with probability at least $p^*/4$, the extractor K will receive at most c non-$\perp$ answers from P^*, and all of these answers are prefixes of a single sequence $(\alpha_1, \ldots, \alpha_i)$, for some $i \leq c$. Informally speaking, this means that with probability at least $p^*/4$, the extractor K is unable to obtain any meaningful information by rewinding P^* (all attempts at rewinding P^* result in abort responses).

[18] Strictly speaking, the knowledge soundness property is defined for *deterministic* prover strategies. That is, denote by $P^*_{h,r}$ the prover strategy of P^* with its random tape set to (h, r), and denote by $p_{h,r}$ the probability (now over V's random tape only) that $P^*_{h,r}$ convinces V that $x \in L$. Then, the knowledge soundness property states that K must extract a witness from $P^*_{h,r}$ with probability at least $\tilde{p}_{h,r} = p_{h,r} - \mu(|x|)$. Now, as we have shown, the expectation of $p_{h,r}$ taken over all of P^*'s random tapes is p^* (this is the probability over both P^* and V's random tapes that P^* convinces V that $x \in L$). Therefore, the expectation of $\tilde{p}_{h,r}$ (which is the probability over both P^* and K's random tapes that K extracts a witness for x) is at least $p^* - \mu(|x|)$, as required.

Claim 5.6 *Let $\tilde{p}$ be the probability (over all choices of P^*'s random tape) that K obtains non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i)$, where $\alpha_i \neq \alpha'_i$. Then, there is a choice of ϵ such that for every fixed random tape for K, it holds that $\tilde{p} < p^*/4$. Furthermore, $\epsilon = \frac{1}{q(n)}$ for some polynomial $q(\cdot)$.*

Proof: Let $\epsilon(n) = \frac{p}{4 \cdot t^2 c}$ (recall that p is the completeness bound of the honest prover and that t is number of oracle queries made by K). First, note that for this choice of ϵ, there exists a polynomial $q(\cdot)$ such that $\epsilon = \frac{1}{q(n)}$. Next, for every $i \leq c$ and every $j, k \in [t]$, we let $p^i_{j,k}$ denote the probability that the j^{th} query of K is of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$, the k^{th} query of K is of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i)$, and K receives a non-$\perp$ response for both of these queries. (Recall that by convention K never asks the same query twice, so this means that $\alpha_i \neq \alpha'_i$.) We now bound the probability $p^i_{j,k}$. Recall that P^* decides whether or not to abort by applying a t-wise independent hash function to the query. By the t-wise independence of the function, P^*'s abort decision is independent for each query. Therefore, if K makes two queries of this form, it will obtain a reply with probability that is $\epsilon^{2^{c-1}+\ldots+2^{c-i+1}} \cdot (\epsilon^{2^{c-i}})^2$, which is the probability that P^* does not abort in the first $i-1$ rounds on history $(\alpha_1, \ldots, \alpha_{i-1})$ multiplied by the probability that P^* does not abort in the i^{th} round both when given the message α_i and when given the message α'_i. Since $2^{c-1} + \cdots + 2^{c-i+1} + 2 \cdot 2^{c-i} = 2^c$, we have

$$p^i_{j,k} = \epsilon^{2^{c-1}+\cdots+2^{c-i+1}+2 \cdot 2^{c-i}} = \epsilon^{2^c}$$

By applying the union bound over all $i \in [c]$ and $j, k \in [t]$, we have that $\tilde{p} \leq t^2 c \cdot \epsilon^{2^c}$, where $\tilde{p}$ is defined in the claim statement. Plugging in $p^* = \epsilon^{2^c - 1} p$ we have that $\tilde{p} \leq t^2 c \cdot \epsilon p^*/p$. Finally, since $\epsilon = \frac{p}{4 \cdot t^2 c}$, we have that $\tilde{p} \leq p^*/4$ as required. ■

Summing up, we have shown the following. On the one hand, when the extractor K is given oracle access to P^*, it must obtain a witness with probability at least $p^*/2$ (where the probability is over the random tapes of both P^*'s and K). On the other hand, the probability that K views two non-abort responses from P^* of the form discussed in Claim 5.6 is at most $p^*/4$. This means that with probability $p^*/2 - p^*/4 = p^*/4$, the extractor K obtains a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i)$, where $\alpha_i \neq \alpha'_i$.

We are now ready to construct the verifier V^*. Loosely speaking, the verifier V^* will manage to output a witness after interacting with the honest prover by internally running the knowledge extractor K, and referring some of its queries to the real prover. The important point is that with noticeable probability, K will work in exactly the same way as when it is given oracle access to P^*. The description of V^* is as follows:

Algorithm 5.7 (Verifier V^*)

- *Input: x (statement to be proven)*

  1. *Choose a random $h \in H_n$*
  2. *The verifier will store the history of messages that it has sent to the prover so far in the execution. Initially, this list is empty.*
  3. *Run the knowledge extractor K on input x.*
  4. *When the extractor K makes a query $(\alpha_1, \ldots, \alpha_k)$ do the following:*

(a) *Follow the same procedure as P^* in order to decide whether or not to answer the query with $\perp$ (i.e., answer the query with $\perp$ unless for every $j \in [k]$ the first $2^{c-j} log(1/\epsilon)$ bits of $h(\alpha_1, \ldots, \alpha_j)$ are equal to 0).*

(b) *If the above decision was to not answer the query with $\perp$, then check whether the history of messages sent so far to the prover consists exactly of $(\alpha_1, \ldots, \alpha_{k-1})$. If this is the case then send α_k to the prover and forward the prover's response β to the knowledge extractor K. If this is not the case, and so there was a previous query that was not answered with $\perp$ and is not a prefix of this query, then abort. (In this case we say that the verifier* failed.)

5. *At the end of the execution, if the extractor outputted a witness for x then output this witness.*

Intuitively, when V^* does not output fail, it perfectly emulates an execution of K with oracle access to P^*. Therefore, V^* outputs a valid witness whenever K would output a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$. This means that V^* outputs a valid witness with probability at least $p^*/4$.

More formally, let good be the event that K outputs a witness without obtaining non-$\perp$ replies for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i')$, where $\alpha_i \neq \alpha_i'$. Then, as we have shown, the probability that the good event occurs is at least $p^*/4$. Now, the probability space over which we computed the probability that good occurs is (h, r, r_K), where (h, r) is the random tape of P^* and r_K is the random tape of the extractor K. Recall that h was used by P^* for deciding whether or not to abort and r was used by P^* for running the honest prover strategy. Now, observe that the probability space over which we need to compute V^*'s success in outputting a valid witness is also (h, r, r_K). The only difference is that here h and r_K are randomly chosen by V^*, and r is the random-tape of the honest prover with which V^* interacts. Therefore the probability space is the same and we have that the probability that V^* outputs a witness is also at least $p^*/4$, which is noticeable. This completes the proof. ■

5.2.2 Concluding the Proof of Theorem 5.2

To prove Theorem 5.2 we need to show that if (P, V) is a constant round zero-knowledge proof of knowledge for some relation R with a strict polynomial-time knowledge extractor, then $L_R \in \mathcal{BPP}$. Indeed, if (P, V) is such a system then by Lemma 5.3, there exists a verifier V^* that for every $x \in L$, outputs a witness for x with noticeable probability after interacting with the honest prover. Now, since (P, V) is zero-knowledge, there exists a simulator S^* for V^* whose output is computationally indistinguishable from the output of V^*. Thus, for every $x \in L_R$, it holds that simulator $S^*(x)$ outputs a witness for x with noticeable probability. On the other hand, if $x \notin L_R$ then S^* will certainly not output a witness for x. Therefore, S^* can be used to obtain a probabilistic polynomial-time procedure for deciding membership in L_R. In other words, $L_R \in \mathcal{BPP}$.

5.2.3 Proof of Lemma 5.4 and Theorem 5.1

The proof of Lemma 5.4 largely follows the proof of Lemma 5.3. Given a c-round zero knowledge protocol (P, V) with a black-box strict polynomial-time simulator S, one first constructs a verifier V^* that behaves identically to the honest verifier V, except that it may choose to abort in any round. This verifier V^* corresponds to the cheating prover constructed in Algorithm 5.5 within the proof of Lemma 5.3, and uses the same procedure to decide whether or not to abort. Specifically, the probability that it does not abort in the i^{th} round of the protocol is $\epsilon^{2^{c-i}}$, in which case it follows the instructions of the honest verifier. Now, let $p^* = \epsilon^{2^c-1} \cdot p$, where p is the completeness bound of the zero-knowledge protocol. Then, as above, it follows that an interaction between the honest prover and V^* yields an accepting transcript with probability exactly p^*. Therefore, any simulator given oracle access to V^* must output an accepting transcript with probability at least $p^* - \mu(n) > p^*/2$. Next, a claim analogous to Claim 5.6 is proved. That is, we show that it is possible to choose ϵ so that the probability that the (strict polynomial-time) simulator obtains non-$\perp$ replies from V^* for two queries of the form $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i)$ and $(\alpha_1, \ldots, \alpha_{i-1}, \alpha'_i)$, where $\alpha_i \neq \alpha'_i$, is at most $p^*/4$. Furthermore, ϵ is noticeable (implying that p^* is also noticeable). Putting this together, we have that when given black-box access to V^*, the simulator S has a noticeable probability of outputting an accepting transcript, even after viewing at most c non-$\perp$ responses from its oracle, where all these responses are prefixes of the same sequence $(\alpha_1, \ldots, \alpha_i)$. Thus, S can be used to convince the honest verifier with noticeable probability that $x \in L$. That is, in a similar way to the proof of Lemma 5.3, we use this observation about the simulator S in order to construct a cheating prover algorithm P^* (the construction of P^* corresponds to the cheating verifier algorithm described in Algorithm 5.7). This prover P^* does *not* get a witness as auxiliary input, but still for every $x \in L$, it manages to convince the honest verifier to accept with noticeable probability. We omit the full details of the proof of Lemma 5.4.

To prove Theorem 5.1, we need to show that if (P, V) is a constant-round zero-knowledge protocol for L with a black-box strict polynomial-time simulator, then $L \in \mathcal{BPP}$. Now, let (P, V) be such a protocol. By Lemma 5.4, there exists a prover algorithm P^* such that on every input $x \in L$, the prover P^* with input x only (and no witness) manages to convince the honest verifier to accept x with noticeable probability. By the soundness of the system, we know that if $x \notin L$ then the honest verifier will accept x with negligible probability. Therefore, one can test whether or not $x \in L$ in probabilistic polynomial-time, by emulating an interaction between P^* and V on input x, and outputting 1 if and only if the verifier accepts in this execution. Thus, $L \in \mathcal{BPP}$.

Acknowledgements

We wish to thank Oded Goldreich for many helpful discussions and comments. We would also like to thank Moni Naor for pointing the connection between our lower bound and the separation of black-box ε-knowledge from black-box zero knowledge. Finally, we thank Jonathan Katz for pointing out an error in an earlier version of this work.

References

[1] M. Agarwal, N Kayal and N. Saxena. Primes is in P. Manuscript, 2002. Can be downloaded from `http://www.cse.iitk.ac.in/news/primality.html`.

[2] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr. RSA and RABIN Functions: Certain Parts are as Hard as the Whole. *SIAM Journal on Computing*, 17(2):194–209, 1988.

[3] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[4] B. Barak and O. Goldreich. Universal Arguments and Their Applications. In *17th IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[5] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-Sound Zero-Knowledge and its Applications. In *42nd FOCS*, pages 116–125, 2001.

[6] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *CRYPTO'92*, Springer-Verlag (LNCS 740), pages 390–420, 1992.

[7] M. Bellare and M. Yung. Certifying Permutations: Non-Interactive Zero-Knowledge Based on any Trapdoor Permutation. *Journal of Cryptology*, 9(1):149–166, 1996. (Extended abstract in *CRYPTO'92*, 1992.)

[8] M. Blum. Coin flipping by phone. In *The 24th IEEE Computer Conference (CompCon)*, pages 133–137, 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[9] M. Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, 1(2):1444–1451, 1987.

[10] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Science*, 37(2):156–189, 1988.

[11] G. Brassard, C. Crépeau, and M. Yung. Everything in NP Can be Argued in Perfect Zero-Knowledge in a Bounded Number of Rounds. In *EUROCRYPT'89*, Springer-Verlag (LNCS 434), pages 192–195, 1989.

[12] S. Goldwasser, R. Canetti, O. Goldreich and S. Micali. Resettable Zero-Knowledge. In *32nd STOC*, pages 235–244, 2000.

[13] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. *SIAM Journal on Computing*, 32(1):1–47, 2003.

[14] D. Dolev, C. Dwork and M. Naor. Non-malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[15] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[16] U. Feige, D. Lapidot, and A. Shamir. Multiple NonInteractive Zero Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing* 29(1):1–28, 1999.

[17] U. Feige. *Alternative Models for Zero Knowledge Interactive Proofs*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1990.

[18] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology* 1(2):77–94, 1988.

[19] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89*, Springer-Verlag (LNCS 435) pages 526-544, 1989.

[20] O. Goldreich. Notes on Levin's Theory of Average-Case Complexity. In *ECCC: Electronic Colloquium on Computational Complexity*, 1997.

[21] O. Goldreich. *Secure Multi-Party Computation*. Manuscript, 2002. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[22] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge University Press, 2001.

[23] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* To be published. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

[24] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology* 9(3):167–189, 1996.

[25] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[26] O. Goldreich, S. Micali, and A. Wigderson. Proofs That Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, 38(3):691–729, 1991.

[27] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[28] S. Goldwasser and J. Kilian. Primality Testing Using Elliptic Curves. *Journal of the ACM* 46(4): 450–472, 1999.

[29] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[30] L. A. Levin. Average Case Complete Problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[31] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3):143–184, 2003. (Extended abstract in *CRYPTO'01*, 2001.)

[32] M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[33] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. In *28th FOCS*, pages 472–482, 1987.