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

October 17, 2003

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.

[†] Most of this work was carried out while the author was at the Weizmann Institute of Science, ISRAEL.

ISSN 1433-8092

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 [26], 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 [20, 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 [26].

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 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 $\mathcal{NP}$ 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).

[1] Recall that in a *proof* system, the soundness holds unconditionally and with respect to all-powerful cheating provers. In contrast, in an *argument* system, the soundness is only guaranteed to hold with respect to polynomial-time bounded provers. See Section 2.1.

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

Proofs of Knowledge. An analogous situation arises in proofs *of knowledge* [26, 30, 16]. 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 [15, Sec. 3.2], [20, 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 ([27], see also [18], [20, 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 [28] 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 [22] demonstrates, see [28, 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 at all acceptable. See Section 1.4 for a more detailed discussion of this issue.

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 [15, 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 [15, Sec. 3.4] and [20, 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 [2]. Specifically, [2] presented a zero-knowledge argument system that is both *constant-round* and has a *strict* polynomial-time simulator. Interestingly, the protocol of [2] 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 [2] 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 trapdoor permutations over* $\{0,1\}^n$ [3] *and collision-resistant hash functions. 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.*

[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 an enhanced family of trapdoor permutations [21, Appendix C.1]. Such a family can be constructed under either the RSA or factoring assumptions, see [1, Section 6.2] and [21, Appendix C.1].

In order to prove Theorem 1, we first construct what we call a "commit-with-extract" commitment scheme. Such a scheme has the property that for every sender, there exists an extractor who can obtain the committed value. This is analogous to the knowledge extractor of a proof of knowledge, but in the context of commitments. Such extractable commitment schemes have been considered before; the novelty of ours is that it is both *constant-round* and has a *strict* polynomial-time extractor. The main idea behind our scheme is to transform a standard non-interactive commitment scheme into an interactive one, where the only difference is that the random coins typically chosen by the committer are chosen jointly by both parties in a coin tossing protocol. Furthermore, this coin tossing protocol has the property that an extractor can bias the result in a way that would enable it to learn the committed value. This "biasing" capability is achieved through the extractor's ability to simulate a zero-knowledge proof that is verified by the committer. Since simulated proofs are not necessarily sound, this enables the extractor to "cheat" and gives it the necessary advantage. An important property of our construction is that we need a zero-knowledge protocol that is both constant-round and has a strict polynomial-time simulator (this is because the extractor is strict polynomial-time and must be able to run the simulator). The only known such zero-knowledge protocols [2, 3] are non-black-box. Therefore, both the simulator and extractor for our zero-knowledge argument of knowledge are also non-black-box.

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 [20, Sec. 4.7.6]. (In actuality, [20] shows that a *super-logarithmic* number of sequential executions of the 3-round zero-knowledge proof for Hamiltonicity [7] 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 [13] 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 is the first proof that black-box ε-knowledge is strictly weaker than black-box zero-knowledge. That is, on the one hand, constant-round black-box ε-knowledge protocols with strict polynomial-time simulators do exist.[4] On the other hand, as we show, analogous protocols for black-box *zero*-knowledge, do not exist. Thus we separate ε-knowledge and zero knowledge as far as *black-box* simulators are concerned.

Witness-extended emulation. Zero-knowledge proofs of knowledge are often used as subprotocols within larger protocols. In this context, the mere existence of a knowledge extractor does not always suffice for proving the security of the larger protocol. Loosely speaking, what is required in many cases 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 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 above-described 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 [28], 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 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 [28]).

1.3 Failure of the Naive Approach

A naive approach to solving the problem of expected polynomial-time in simulation and extraction, would be 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 occurs 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

[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.

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 n^{-10} then our extractor will run in time, say, n^4 and output a witness with probability at most 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 With Running Time

The observation in Section 1.3 about how expected polynomial-time extractors typically work, raises serious security issues with respect to the use of proofs of knowledge with 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 that 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 known protocols, what will actually be proven is that given an algorithm that is able to impersonate using T steps and 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 contrast, the fastest known algorithms for factoring *do not* allow such a trade-off, and so adversaries can gain from attacking the protocols.

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).

1.5 Related work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [26], and were then shown to exist for all $\mathcal{NP}$ by Goldreich, Micali and Wigderson [24]. Constant-round zero-knowledge arguments and proofs were constructed by Feige and Shamir [17], Brassard, Crepeau and Yung [9] and Goldreich and Kahan [22]. All these constant-round protocols utilize expected polynomial-time simulators. Regarding zero-knowledge proofs *of knowledge*, following a discussion in [26], the first formal definitions were provided by Feige, Fiat and Shamir [16] and by Tompa and Woll [30]. These definitions were later modified by Bellare and Goldreich [5].

The issue of expected polynomial-time is treated in Feige's thesis [15] and Goldreich's book [20]. Goldreich [20, 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 [2], 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 [3] 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 [2] 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 [4] 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 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 polynomial $p(\cdot)$ and all sufficiently large n's, it holds that $\mu(n) < p(n)$. 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, $f(n) < \mu(n)$. 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 $|\Pr[D_{|s|}(s, X_s) = 1] - \Pr[D_{|s|}(s, Y_s) = 1]| < \mu(|s|)$. 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. Let A and B be interactive machines. We denote by $\mathsf{view}_A(A(x,y,r), B(x,z,r'))$ a random variable describing 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'. Recall that a party's view of an execution includes

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

the contents of its input, auxiliary-input and random tapes 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 $x \in L$. 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 which must hold even for an all-powerful cheating prover, and *computational soundness* which needs only hold for polynomial-time cheating provers. In *proof* systems [26], unconditional soundness is guaranteed; whereas in *argument* systems [8] only computational soundness must hold. 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 completeness is just $1/p(n)$ for some polynomial $p(\cdot)$.

We now recall the definition of zero knowledge [26]. Actually, we present (a slightly strengthened form of) the definition of *auxiliary-input* zero knowledge [20, Sec. 4.3.3].[6] The main difference 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 for $\mathcal{NP}$ languages): *Let (P,V) be an interactive proof (or argument) system for an $\mathcal{NP}$-language L, and let R_L be an $\mathcal{NP}$-relation associated with L. 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^*,*

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

[6] We deviate from the definition of auxiliary-input zero knowledge of [20, 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 verifier as in [20, Sec. 4.3.3]. Note however, that the definition of [20, 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^*.

Black-box zero knowledge. A zero-knowledge proof system is called black-box zero knowledge [25] 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:

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

where $V^*(x,z,\cdot)$ denotes the *next-message function* of the interactive machine V^* after the public 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 [5] in two ways:

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

Non-black-box extraction. Second, 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. [5]), where the knowledge extractor is given only oracle (or black-box) access to the prover strategy. The relaxed definition appeared originally in Feige and Shamir [17] (which differs from the definition in [16]; see discussion in [5]), and suffices for all practical applications of arguments of knowledge.

Until recently, all known proofs of knowledge (including [17]) used the prover algorithm only as a black-box. The extra power awarded a knowledge extractor who is given the actual description of the prover was first used in an essential way by [4] in order to obtain resettable zero-knowledge arguments of knowledge for $\mathcal{NP}$.[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 upon common input x, and 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

[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 [10].

[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.

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 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 and 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 [28] 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 a machine was termed a "witness-extended emulator", because its role is to emulate a protocol execution while also providing a witness (see [28] 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.)

  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 (strict) polynomial-time 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 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,*

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

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 [28] in a number of ways. Most notably, we require that the witness-extended emulator be 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.

We note that [28] 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 and extracts the value being committed to during the commit stage of the protocol. This idea of "extractable commitments" is not new. It has been used in the context of secure multi-party computation (e.g., [19, Construction 2.3.8]) and has been called both commit-with-knowledge [12] and non-oblivious commitment [20, Def. 4.9.3]. The main difference between our primitive and previous ones is the requirement that the extractor run in *strict* polynomial-time.

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 [20, 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). 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 note that the additional output of the view can be important in cases that a commit-with-extract protocol is to be used as a subprotocol. Indeed, this is the case in our application where we use this scheme in order to construct a zero-knowledge argument of knowledge. 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. $\left\{CK_1(\mathsf{desc}_{|x|}(A^*), x, y, r)\right\}_{x,y,r\in\{0,1\}^*} \overset{\rm c}{\equiv} \{\mathsf{view}_{A^*}(A^*(x,y,r), B)\}_{x,y,r\in\{0,1\}^*}$
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 so 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 variant of 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, using known constructions of proofs of knowledge, one would have to settle for either a non-constant number of rounds, or extraction within *expected* polynomial-time. On the other hand, we require that the commit-with-extract scheme be both *constant-round* and have a *strict polynomial-time* extractor.

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. This is analogous to "publicly verifiable" protocols, as described in Section 2.3. We use this additional feature 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 3.2 *Assume the existence of trapdoor permutations over $\{0,1\}^n$* [10] *and collision-resistant hash functions. 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.* 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 [6]: 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 defined in the same way 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 [28]). Since y is chosen uniformly, 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 so 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.

[10] 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 [14]). 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 [2, 3].

Protocol 3.3 (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. Formally, A proves that there exists a string s_I such that f is the permutation output from $I(s_I)$. 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:**
  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 3.3 is a secure commit-with-extract scheme:

Proposition 3.4 *Protocol* 3.3 *constitutes a commit-with-extract bit commitment scheme, that satisfies public decommitment.*

Proof: It is easy to see that the protocol satisfies public decommitment. We proceed by first showing that Protocol 3.3 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).

Perfect binding. We begin with the (almost) perfect binding property. That is, we show that except with negligible property, for any transcript of messages t 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 $\mathsf{commit\text{-}value}(t) = \sigma$. 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.

Computational hiding. Intuitively, the hiding property follows from the hiding property of the non-interactive commitment scheme of [6], 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.

Formally, for a 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

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

As we have mentioned, the hiding property is derived from the hiding property of the non-interactive commitment scheme defined by

$$C_n(\sigma) \stackrel{\text{def}}{=} \langle f(U_n),\ b(U_n) \oplus \sigma \rangle \tag{1}$$

Loosely speaking, the only difference between Protocol 3.3 and the commitment scheme C_n, is that in Protocol 3.3 the value $f(U_n)$ is chosen jointly by both parties (and is not determined solely by the sender). We therefore reduce the hiding property of Protocol 3.3 to the hiding property of C_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

$$\mathsf{adv}_n^D \stackrel{\text{def}}{=} \left|\Pr[D(v_n^{B^*}(0)) = 1] - \Pr[D(v_n^{B^*}(1)) = 1]\right| \geq \frac{1}{p(n)} \tag{2}$$

We will use D and B^* to construct a distinguisher D' that contradicts the security of the commitment scheme C_n defined in Eq. (1). Intuitively, 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 obtaining an execution of Protocol 3.3 in which $r_1 \oplus r_2 = f(r)$. Then, since D has an advantage in guessing the value of σ given r_1, r_2 and $b(r) \oplus \sigma$, this translates to D' having an advantage in guessing σ given $C_n(\sigma)$. We first informally describe how D' obtains an execution with B^* where $r_1 \oplus r_2 = f(r)$. This is achieved using the following strategy. D' runs an execution of Protocol 3.3 with B^*, until B^* sends a commitment $c = \mathsf{Commit}(r_1; s)$ as part of the coin tossing protocol. D' then attempts to learn r_1 by repeatedly running the continuation of the coin tossing protocol with B^*. Once it succeeds, it sends $r_2 = f(r) \oplus r_1$, where $f(r)$ is part of its input commitment $C_n(\sigma)$. The result of the coin-tossing protocol is thus $f(r)$, as desired. The main issue to be resolved in order to implement this strategy is to bound how many times D' should repeat its attempt to learn r_1 (in order to ensure that it remains polynomial time). Fortunately, in our context, this bound is easily derived from the contradicting assumption in Eq. (2). We now proceed with the formal proof.

Before defining D', we prove the following fact:

Fact 3.5 *Let $\epsilon_n^{B^*}$ denote the probability that B^* does* not *abort during an execution with A. Then, $\epsilon_n^{B^*} \geq \mathsf{adv}_n^D$.*

Proof: The only place in which the honest sender A uses its input value σ is in the last step of Protocol 3.3. Therefore, in any execution where B^* aborts, the view of B^* is *independent* of the value σ. Therefore, the advantage of D in executions where B^* aborts is exactly zero. Thus, $\mathsf{adv}_n^D \leq \epsilon_n^{B^*}$, and the fact follows. ■

By the assumption in Eq. (2), for infinitely many n's, $\mathsf{adv}_n^D \geq 1/p(n)$. Therefore, it also holds that for all of these n's, $\epsilon_n^{B^*} \geq 1/p(n)$. In our construction of the distinguisher D', we will explicitly refer to this value $p(n)$ (note that since we only need to prove the existence of a machine D', this is fine). We are now ready to describe D'.

Distinguisher D' receives for input a one-way permutation f and a commitment $\tilde{c} = C_n(\sigma) = \langle f(r),\ v = b(r) \oplus \sigma\rangle$, where $r \in_R \{0,1\}^n$. Then, D' 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.
2. *Simulation of the coin-tossing protocol:*
   (a) D' receives a commitment $c = \mathsf{Commit}(r_1)$ from B^*.
   (b) D' repeats the following until $\mathsf{success} = 1$, but at most $2p(n)$ times (initially $\mathsf{success} = 0$):
      i. D' chooses a random string $r_2 \in_R \{0,1\}^n$ and passes it to B^*.

ii. D' obtains some string r_1 from B^* and verifies the zero knowledge argument given by B^*.

iii. If D' accepts the argument, then it sets $\mathsf{success} = 1$.

If $\mathsf{success} = 0$ (after all $2p(n)$ attempts), then D' outputs fail and halts.

(c) D' rewinds B^* and sends it the string $r_2 = f(r) \oplus r_1$, where r_1 is the value obtained in Step 2(b)ii for which D' accepted the argument from B^*.

(d) 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 jumps to Step 4. (This is interpreted as an abort from B^*, as in a real execution.)

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 3 below.

3. *Simulation of the actual commitment:* D' passes the bit v (from its input commitment $\tilde{c}$) to B^*.

4. *Output:* D' passes the view output by B^* to D, and outputs whatever D does.[11] (We stress that this view 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.)

Informally speaking, if D' does not fail, then the view generated for B^* is indistinguishable from a real execution. Furthermore, this view is such that $r_1 \oplus r_2 = f(r)$, as desired. Of course, this is only helpful if D''s probability of failing is small. Now, D' can fail if $r'_1 \neq r_1$, as in Step 2(d)ii. However, intuitively this cannot happen with non-negligible probability, due to the binding property of commitments and the soundness of zero-knowledge protocols. In addition to the above, D' can fail if $\mathsf{success} = 0$ after all $2p(n)$ rewinding attempts. The main observation here is that by combining Fact 3.5 with Eq. (2), we have that B^*'s non-abort probability is at least $1/p(n)$. Therefore, with probability 1/2, some non-abort execution should be seen within $2p(n)$ rewinding attempts. This is what is needed because such a non-abort execution is exactly what sets $\mathsf{success} = 1$. We now formally prove a bound on D''s probability of failing.

Claim 3.5.1 *Denote the probability that D' outputs fail upon input $C_n(\sigma)$ by $\mathsf{fail}_n^{D'}(\sigma)$. Then, for every n for which $\mathsf{adv}_n^D \geq 1/p(n)$, it holds that $\mathsf{fail}_n^{D'}(0) = \mathsf{fail}_n^{D'}(1) \leq 3/4$.*

Proof: First, $\mathsf{fail}_n^{D'}(0) = \mathsf{fail}_n^{D'}(1)$ because D' only uses the value v from its input after all checks enabling fail have been passed. Since v is the only value that is dependent on σ, we have that D''s fail probability is independent of σ.

Next, we show that for $\sigma \in \{0, 1\}$, $\mathsf{fail}_n^{D'}(\sigma) \leq 1/4$. Now, machine D' outputs fail if $\mathsf{success} = 0$ after $2p(n)$ repetitions of the loop, or if $r_1 \neq r'_1$. This second event (of $r_1 \neq r'_1$) can only occur with negligible probability. This is the case because the commitment c provided by B^* in Step 2a is perfectly binding. Therefore, the value c can be a commitment to at most one of r_1 or r'_1. In other words, at least one of the statements $c = \mathsf{Commit}(r_1)$ or $c = \mathsf{Commit}(r'_1)$ is false. By the soundness of the zero knowledge argument that is verified by D' in Steps 2(b)ii and 2d, we have that it would accept the false argument with at most negligible probability $\mu(n)$.

Next, we show that the probability that $\mathsf{success} = 0$ after $2p(n)$ repetitions of the loop is at most $1/2 - \mu(n)$. The probability that $\mathsf{success} = 1$ in any given repetition of the loop is at

[11] Without loss of generality, we assume that B^* always outputs its view.

least equal to B^*'s non-abort probability in the simulation with D'. However, the only difference between B^*'s view in this simulation with D' and in a real execution is due to the fact that the proof that f is a permutation is simulated (Step 1b), rather than real. Therefore, this can alter B^*'s non-abort probability $\epsilon_n^{B^*}$ by at most a negligible amount (or it can be used to distinguish real proofs from simulated ones). Now, by Fact 3.5, $\epsilon_n^{B^*} \geq \mathsf{adv}_n^D$. Therefore, for any n where $\mathsf{adv}_n^D \geq 1/p(n)$, B^*'s non-abort probability in a real execution is at least $1/p(n)$. By the above, we have that for these n's, B^*'s non-abort probability in the simulated execution with D' is at least $1/p(n) - \mu(n) > 3/4p(n)$. Therefore, the expected number of repetitions until $\mathsf{success} = 1$ is at most $4p(n)/3$. Now, by applying Markov's inequality, we conclude that the probability that $\mathsf{success} = 0$ after $2p(n)$ repetitions is at most $2/3$.

Combining this with the negligible probability of outputting fail due to $r_1 \neq r_1'$, we have that for some μ, machine D' outputs fail with probability at most $2/3 + \mu(n) < 3/4$. ■

We now show that, conditioned on the fact that D' does not output fail, D' distinguishes commitments to 0 from commitments to 1 with non-negligible probability:

Claim 3.5.2 *If D' does not output* fail*, then D' distinguishes $C_n(0)$ from $C_n(1)$ with advantage that is negligibly close to* adv_n^D*. That is, for every n:*

$$\left|\Pr[D'(C_n(0)) = 1 \mid D'(C_n(0)) \neq \mathsf{fail}] - \Pr[D'(C_n(1)) = 1 \mid D'(C_n(1)) \neq \mathsf{fail}]\right| > \mathsf{adv}_n^D - \mu(n)$$

for some negligible function $\mu(\cdot)$.

Proof: In this claim, we consider the case in which B^* behaves in such a way that D' never outputs fail. Therefore, when r_1 and r_2 appear in B^*'s output view (as is the case when B^* does not abort), it holds that $r_1 \oplus r_2 = f(r)$, where $f(r)$ is party of D''s input commitment $C_n(\sigma)$. Furthermore, the distribution over B^*'s final view in the simulation is computationally indistinguishable from its view $v_n^{B^*}(\sigma)$ in a real execution with the honest sender A who commits to σ. The only difference between the views is due to the simulated, rather than real, zero-knowledge proof that f is a permutation. (Notice that since $f(r) \in_R \{0,1\}^n$ we have that the string r_2 sent by D' after the loop concludes is uniformly distributed. This is therefore exactly the same as in a real execution.) We therefore have that for $\sigma \in \{0,1\}$,

$$\left|\Pr[D'(C_n(\sigma)) = 1 \mid D'(C_n(\sigma)) \neq \mathsf{fail}] - \Pr[D(v_n^{B^*}(\sigma)) = 1]\right| < \mu'(n)$$

and so

$$\left|\Pr[D'(C_n(0)) = 1 \mid D'(C_n(0)) \neq \mathsf{fail}] - \Pr[D'(C_n(1)) = 1 \mid D'(C_n(1)) \neq \mathsf{fail}]\right| \geq \mathsf{adv}_n^D - 2\mu'(n)$$

This completes the proof of the claim. ■

Combining Claims 3.5.1 and 3.5.2, we have that for any n for which $\mathsf{adv}_n^D \geq 1/p(n)$,

$$\left|\Pr[D'(C_n(0)) = 1] - \Pr[D'(C_n(1)) = 1]\right| > \frac{\mathsf{adv}_n^D}{4} - \mu(n)$$

By the assumption in Eq. (2), there are infinitely many n's for which $\mathsf{adv}_n^D \geq 1/p(n)$. Therefore, we conclude that there are infinitely many n's for which

$$\left|\Pr[D'(C_n(0)) = 1] - \Pr[D'(C_n(1)) = 1]\right| > \frac{1}{4p(n)} - \mu(n)$$

in contradiction to the hiding property of the commitment scheme C_n. We conclude by remarking that D''s running-time is bound by some fixed polynomial number of steps, plus the time that it takes to simulate the zero-knowledge proof or argument that f is a permutation. If a zero-knowledge protocol with strict polynomial-time simulation is used, then we are done. Otherwise, we obtain a distinguisher D' that works in expected polynomial-time. By truncating its running time to twice its expected time, we obtain a strict polynomial-time distinguisher whose advantage in distinguishing commitments is at most degraded by a factor of 2, and is therefore still non-negligible.

The extraction property. Having established that Protocol 3.3 is a secure commitment scheme, we now show the existence of a *strict* polynomial-time extractor. The extractor CK works by biasing the outcome $r_1 \oplus r_2$ of the coin-tossing protocol, so 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 to do (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) Then, 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 and 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 chosen irrespective 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 since CK knows the preimage to $f(r) = f(r_1 \oplus r_2)$. 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. In actuality, there are two differences: 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. However, the only known such argument system is that of [2] (or its modified version in [3]) 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.

Extending Protocol 3.3 to strings: To prove Theorem 3.2 we need to generalize Protocol 3.3 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 3.3 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 [2, 3]). Alternatively, one can directly modify Protocol 3.3, 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 3.3 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 3.3, A^* sends a *random* bit $b \in \{0,1\}$ to B. Otherwise, A^* follows the instructions for A. 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^*). In this case, the extractor cannot be applied later to A^* in order to obtain the committed value. In particular, due to the fact that the committed value depends on the coins of B, a latter execution of the extractor applied to A^* will yield a commitment value that is *independent* of the actual value that A^* is committed to with B.

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

Sufficient assumptions. 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 [2, 3]). It is interesting to note that 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. (As we have mentioned, such a commit-with-extract scheme may be constructed by running the perfectly-binding commitment scheme of [29], followed by either a constant-round zero-knowledge proof of knowledge [17] or a non-constant round strong zero-knowledge proof of knowledge, such as the one in [20].)

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 committed to a valid witness using a zero-knowledge proof of membership. (Recall that by zero knowledge here, we mean auxiliary-input zero knowledge.) 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. On the other hand, a knowledge extractor is immediately obtained from the extractor of the commit-with-extract scheme.

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 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 the value w, and $(x, w) \in R$.[12]

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 3.2 (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, as defined in Definition* 2.2 (*i.e., it has a strict polynomial-time simulator and 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, one must prove three properties: completeness, knowledge soundness (which implies computational soundness), and zero knowledge. The proof of completeness is immediate. We proceed to prove knowledge soundness and zero knowledge.

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. We then argue that since the view output by CK is computationally indistinguishable from P^*'s view in a real execution, and since the proof (or argument) system used in Phase 2 is sound, the probability that w will be a witness for x is at least $\epsilon - \mu(n)$ (for some negligible function $\mu(\cdot)$). We now formally describe the knowledge extractor algorithm K:

Algorithm 4.3 (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 $\mu(\cdot)$. First, recall that by the definition of a commit-with-extract scheme, it holds that

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

[12] We note that here we require that the commit-with-extract scheme satisfies public decommitment. Otherwise, the statement that P needs to prove is not guaranteed to be in $\mathcal{NP}$.

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. Now, the proof of Phase 2 of Protocol 4.1 is provided by the *residual prover* to the commit-with-extract scheme. Therefore, the probability that this residual prover provides a convincing proof when its view of the first phase is as generated by CK is negligibly close to the probability that it provides a convincing proof in a real execution. Otherwise, this residual prover (along with a procedure that outputs 1 if and only if the residual prover provides a convincing proof) constitutes a distinguisher for the above views. We therefore have that the probability that P^* provides a convincing proof in Phase 2, when its view of Phase 1 is distributed according to $\mathsf{view}^{CK}_{P^*}(x,y,r)$, is at least $p(x,y,r)-\mu(|x|)$.

Now, assume that with probability non-negligibly less than $p(x,y,r)$, the value w as output by CK is *not* a valid witness for x. That is, there exists a polynomial $q(\cdot)$ such that for infinitely many x's, the probability that CK outputs a valid witness is less than $p(x,y,r)-\frac{1}{q(|x|)}$. We now contradict the soundness of the proof (or argument) system of Phase 2. As we have shown, on the one hand, for all sufficiently large x's, the residual prover provides a convincing proof with probability at least $p(x,y,r)-\mu(|x|)$. On the other hand, for infinitely many x's, the statement being proved is true with probability less than $p(x,y,r)-\frac{1}{q(|x|)}$. We conclude that the residual prover succeeds in proving false statements with non-negligible probability. This contradicts the soundness of the proof (or argument) system. Therefore, it must be that K outputs a valid witness with probability at least $p(x,y,r)-\mu(|x|)$, as required. This completes the proof of 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. Due to the hiding property of the commit-with-extract scheme and the zero-knowledge property of the zero-knowledge proof, it holds that the output of the simulator is indistinguishable from the verifier's view in a real execution. This is shown below using a standard hybrid argument. Formally, let V^* be a verifier algorithm for Protocol 4.1. Then the simulator S works as follows:

Algorithm 4.4 (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.*[13] *Then, S runs the simulator for the zero-knowledge proof of Phase 2, with $V^*(x,z,\mathsf{trans})$ as the verifier.*[14]

[13] We note that technically, there is a difference here regarding whether the simulator has black-box or non-black-box access to V^*. If S has non-black-box access, then the definition of the residual machine is trivial. On the other hand, if the access is only black-box, then this residual machine is defined by concatenating any oracle query to the verifier with trans (i.e., instead of querying m, the string (trans,m) is queried).

[14] We note that it is for this point in the proof that we need the proof of Phase 2 to be *auxiliary-input* zero knowledge. Formally, trans is given to V^* as auxiliary-input and then the requirement is that for *every* such transcript, the proof remains zero knowledge.

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

We need to prove that:

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

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

Witness-extended emulation. We conclude by providing a proof sketch of the existence of a witness-extended emulator E for Protocol 4.1. Recall that E, upon input $(\mathsf{desc}_{|x|}(P^*),x,y,r)$, 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. This is easily accomplished as follows:

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 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 plays the honest verifier in Phase 2, where the residual machine P^* from Phase 1 plays the prover (the residual machine is defined from the view output by CK). 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 motivate why it is not possible to obtain strict polynomial-time black-box extraction for constant-round 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 protocols would 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. Obtaining two (or possibly more) different prover messages then suffices for extracting the witness. Now, this second case (where P^* does not abort) occurs with probability only ϵ. However, this means that the expected number of rewinding attempts by the extractor until a second non-abort reply is obtained equals $1/\epsilon$. Therefore, the overall expected amount of work is bounded. However, since ϵ can be any non-negligible function, we cannot provide *any* strict polynomial upper-bound on the running time of the simulator.

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. This suffices for proving the lower bound because, informally speaking, black-box extractors must be able to rewind in order to extract. The same argument also holds for strict polynomial-time simulation of constant-round zero-knowledge proofs. 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 $L \notin \mathcal{BPP}$.*[15]

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 language $L \notin \mathcal{BPP}$.*

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

[15] The theorem is stated for both arguments and proofs because the existence of one does not necessarily imply the existence of the other. First, arguments do not imply proofs because proofs require unconditional soundness. Second, proofs do not imply arguments because proofs are allowed to have inefficient honest prover strategies, whereas arguments are not.

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^* which 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.[16]

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 in the real execution then the extractor receiver $\perp$. 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 this essentially means that the extractor's view is like a real interaction with the honest prover P, and that it does not obtain anything additional 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. In the case that this happens, the verifier will be able to obtain a witness for x with noticeable probability, which is what we wanted to prove.

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 $L \in \mathcal{BPP}$.

[16] We will use a strategy for P^* that causes it to abort with quite high probabilities. 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.

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

1. Intuitively, in order to successfully extract or simulate, the extractor/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 $(\alpha_1, \alpha_{i-1}, \alpha'_i)$ where $\alpha'_i \neq \alpha_i$. Otherwise the extractor/simulator does not have any advantage over the interactive verifier/prover. (Informally speaking, "rewinding" is essential for black-box simulation and extraction.)

2. For any strict polynomial-time extractor or simulator, there exist provers/verifiers for which the time needed to obtain meaningful (i.e., non-abort) replies to two different continuations is beyond the scope of the extractor's/simulator's running-time.

5.1 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 $\alpha_{i+1}, \ldots, \alpha_j$.

If A is an interactive or non-interactive algorithm, we let $\mathcal{R}_n(A)$ denote the set of possible random tapes for A on inputs of length n (i.e., $\mathcal{R}_n(A) = \{0,1\}^{r(n)}$ where $r(n)$ is the number of coins A uses on inputs of length n). We will drop the subscript n when it can be inferred from the context.

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 language L 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 $x \in L$, 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 $L \in \mathcal{BPP}$.

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 [23, 11]).

5.1.1 Proof of Lemma 5.3

Let L be a language and let (P, V) be a system of proofs or arguments of knowledge for L, 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 $t(n)$.[17] We assume without loss of generality that each verifier message in the proof system is 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 (otherwise we can always add an initial dummy message).

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. It makes exactly t queries to its black-box (and no less).

We can assume all of the above because any extractor can be easily modified so that it behaves in the above way, without affecting its output distribution. Furthermore, the number of queries made by the modified extractor is at most c times the number of queries made by the original one (this additional factor of c is used for asking all of the prefixes of a query before the query itself).

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^* then can simulate the behavior of K^{P^*} 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 $q(\cdot)$).

[17] 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.

Let $\mathcal{H} = \{H_n\}_{n \in \mathsf{N}}$ be a family of $t(n)$-wise independent hash functions,[18] 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). This will be shown more exactly below.

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^*:)

- Hardwired data: *x – statement to be proven , y – auxiliary input to the honest prover P on input x.*
- Random tape: *h – chosen at random from H_n , r - random tape for the honest prover P*
- 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 we decide not to 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.

[18] 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.

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 probability at least $1/q(n)$ for some polynomial $q(\cdot)$. 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, where $p > 1/q'(n)$ for some polynomial $q'(\cdot)$; note that for perfect completeness, we have $p = 1$). 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 result 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 greater than $1/q(n)$ for some polynomial $q(\cdot)$, as desired.

We have shown that P^* convinces V with probability at least 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).[19] 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).

Claim 5.6 *There is a choice of ϵ such that for every fixed random tape for K, the probability $\tilde{p}$ (over all choices of P^*'s random tape) that K gets 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 $\alpha_i \neq \alpha'_i$) is less than $p^*/4$. 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 $p^* = \epsilon^{2^c-1}p$). 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 following probability: 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 $\epsilon^{2^{c-1}+\ldots+2^{c-i+1}} \cdot (\epsilon^{2^{c-i}})^2$ (this 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).

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

We see that since $2^{c-i} = 2^{c-i-1} + \ldots + 2^0 + 1$,

$$p^i_{j,k} = \epsilon^{2^{c-1}+\ldots+2^{c-i+1}+2^{c-i}+2^{c-i}} = \epsilon^{2^{c-1}+\ldots+2^{c-i}+2^{c-i-1}+\ldots+2^0+1} = \epsilon^{2^c}$$

By applying the union bound over all $i \in [c]$ and $j, k \in [t]$, we have that $\tilde{p} \leq t^2 c \cdot \epsilon^{2^c}$. 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 shown in Claim 5.6 is at most $p^*/4$. This means that with probability $p^*/2 - p^*/4 = p^*/4$, the extractor K must obtain a witness without any successful "rewinding".

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. 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 its* j^{th} *query, denoted by* $(\alpha_1, \ldots, \alpha_k)$ *do the following:*
   (a) *Decide whether to answer the query with* $\perp$ *in the same way as* P^* *(i.e., answer 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 decided not to answer query with* $\perp$*, then check whether the history of messages sent so far to the prover consists exactly of* $(\alpha_1, \ldots, \alpha_{i-1})$*. If this is the case then send* α_i *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.*

Note that the random tape of V^* *consists of a hash function in* H_n *and a random tape* $r_K \in \mathcal{R}(K)$ *for the knowledge extractor* K.

To finish the proof we need to show that with noticeable probability, the verifier V^* outputs a witness after interacting with the honest prover. Let $r_K \in \mathcal{R}(K)$ and $r_{P^*} \in \mathcal{R}(P^*)$ be random tapes for K and P^*, respectively. We say that these random tapes are *good* if when we execute K with access to P^* and both machines use the random tapes r_K and r_{P^*}, respectively, then:

(a) At the end of the execution K outputs a witness for x.

(b) Throughout the execution K receives at most c non-$\perp$ responses from P^* and all these responses are for prefixes of the *same* sequence $\alpha_1, \ldots, \alpha_i$.

We have shown above that at least a $\frac{p^*}{4}$ fraction of the random tapes r_K, r_{P^*} are good. Note that r_{P^*} is equal to (r_P, h) where $r_P \in \mathcal{R}(P)$ is a random tape for the honest prover and h is a hash function in H_n. Now fix $r_{P^*} = (r_P, h)$ and r_K to be good random tapes and consider an interaction between the verifier V^* and the honest prover P where P uses the tape r_P and V^* uses the tape (h, R_K). We can see that since the tapes are good, the verifier V^* will not fail and will output a witness at the end of the interaction. Since the good tapes consist of at least a $p^*/4$ fraction of all possible random tapes for such interactions, we conclude that V^* outputs a witness with at least this probability. The proof is concluding by noting that there exists a polynomial $p(\cdot)$ such that $p^*/4 > 1/p(n)$, as required in the lemma. ■

5.1.2 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 language L with a strict polynomial-time knowledge extractor, then $L \in \mathcal{BPP}$. 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, based on the above, there exists a polynomial $p(\cdot)$ so that for every $x \in L$, $S^*(x)$ outputs a witness for x with probability at least $1/p(|x|)$. On the other hand, if $x \notin L$ then S^* will certainly not output a witness for x. Therefore, S^* can be used to obtain a probabilistic decision procedure to decide membership in L. In other words, $L \in \mathcal{BPP}$.

5.1.3 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 system (P, V) with a black-box strict polynomial-time simulator, 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 algorithm that we constructed in the proof of Lemma 5.3 and will use the same procedure to decide whether to abort or not. We then show that when given black-box access to V^*, the simulator S has a noticeable probability of outputting a transcript that the original verifier accepts, even after viewing at most c non-$\perp$ responses from its oracle, where all these responses are prefixes of the same sequence $(\alpha_1, \ldots, \alpha_i)$. In a similar way to the proof of Lemma 5.3, we use this observation to construct a cheating prover algorithm P^* from the simulator S (this corresponds to the cheating verifier algorithm constructed in the proof of Lemma 5.3). The prover P^* that we construct 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 argument for L with a black-box strict polynomial-time simulator, then $L \in \mathcal{BPP}$. Now, let (P, V) be such a proof system. 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 only with negligible probability. Therefore, one can test whether $x \in L$ in probabilistic polynomial-time, by executing an interaction between P^* and V on input x,[20] and outputting 1 if and only if the verifier accepts in this execution.

[20] Note that it is crucial here that the prover P^* does *not* need to obtain a witness for x as an auxiliary input.

Acknowledgements

We wish to thank Oded Goldreich for helpful discussions. 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.

References

[1] 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.

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

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

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

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

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

[7] 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.

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

[9] 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.

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

[11] 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.

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

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

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

[15] 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.

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

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

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

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

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

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

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

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

[24] 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.

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

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

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

[28] 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.)

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

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