Strict Polynomial-time in Simulation and Extraction[*]

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

Yehuda Lindell
Department of Computer Science
Weizmann Institute of Science, Israel
lindell@wisdom.weizmann.ac.il

September 24, 2002

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 and completely resolve the question of expected polynomial-time in constant-round zero-knowledge arguments 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 proof 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 argument (or proof) can have a strict polynomial-time *black-box* simulator. Similarly, we show that no constant-round zero-knowledge argument of knowledge (for $\mathcal{NP}$) 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.

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

Zero-Knowledge. The principle behind the definition of (computational) zero-knowledge proofs, as introduced by Goldwasser, Micali and Rackoff [24], 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 [19] 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 zero-knowledge paper [24].

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.

Given that the STRICT/STRICT definition seems to be the most natural, an immediate question to ask is why was expected polynomial-time introduced at all? The answer is that several known protocols that were not known to satisfy the more severe STRICT/STRICT definition could be proven zero-knowledge according to the more liberal STRICT/EXPECTED definition. In particular, until very recently no *constant-round* zero knowledge argument (or proof) for $\mathcal{NP}$ was known to satisfy the STRICT/STRICT definition,[2] and so in many cases one had to settle for the less natural STRICT/EXPECTED definition.

[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* [24, 30, 14]. 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 [13, Sec. 3.2], [19, Sec. 4.12.3]), the definitions that allow for expected polynomial-time computation are less than 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 ([25], see also [17], [19, 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 [26] 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 [21] demonstrates, see [26, Appendix A] for details).

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 [13, 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 seems problematic because if we regard expected polynomial-time as efficient computation, then an efficient strategy for an adversary would be to use a verifier algorithm that runs in expected polynomial-time only when interacting with the honest prover, in order to gain some information.

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 [13, Sec. 3.4] and [19, 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. Furthermore, 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.) This result therefore reopens the above-mentioned open question in the following way:

1. Is it possible to obtain an analogous result to [2] regarding extraction for proofs of knowledge?
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.*

We note that both our simulator and extractor are not black-box. In fact, we use the non-black-box zero-knowledge protocol of [2] as a central tool in our construction. (The assumption relating to the existence of collision-resistant hash functions is inherited from the construction of [3] that improved upon the construction of [2]; see Section 1.4.)

Next, 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 (for $\mathcal{NP}$) 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* $L \notin \mathcal{BPP}$.

[3] By this we mean that there exists a trapdoor permutation family $\{f_s\}_{s\in\{0,1\}^*}$ such that $f_s : \{0,1\}^{|s|} \rightarrow \{0,1\}^{|s|}$. It actually suffices to assume the existence of an enhanced family of trapdoor permutations [20, Appendix C]. Such a family can be constructed under either the RSA or factoring assumptions, see [1, Section 6.2] and [20, Appendix C].

Theorem 3 *Suppose that one-way functions exist. Then, there do not exist* constant-round *zero-knowledge efficient-prover proofs or arguments of knowledge with* strict polynomial-time black-box *knowledge extractors for any $\mathcal{NP}$-complete language L.*

We therefore conclude that the liberal definitions that allow the simulator (resp., extractor) to run in expected polynomial-time is necessary for achieving constant-round *black-box* zero-knowledge (resp., arguments of knowledge). We note that our lower bound for simulation (Theorem 2) holds for both proofs (where the prover may be all powerful) and arguments (where the prover is limited to polynomial-time), whereas our lower bound for extraction (Theorem 3) holds only for proofs or arguments with an *efficient* strategy for the honest prover (this efficiency requirement is necessary for cryptographic applications and is part of the definition for arguments). The lower bound for extraction is weaker also in the respect that it only holds for $\mathcal{NP}$-complete languages rather than any language outside of $\mathcal{BPP}$ and also assumes the existence of one-way functions. In fact, a slightly stronger (but more cumbersome) result can be proven (see Remark 5.4).

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 [19, Sec. 4.7.6]. (In actuality, [19] constructs a protocol that requires a super-logarithmic number of rounds in order to obtain negligible soundness. However, by running the [19] protocol $\log n$ times in parallel, a protocol that obtains negligible soundness for any super-constant number of rounds is obtained. Furthermore, the resulting protocol still has a strict polynomial-time simulator and extractor.)

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 [11] 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 $1/\varepsilon$, 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, the pair of transcript and witness should be compatible. By this we mean that whenever the machine's first output (the simulated transcript) is an accepting transcript, the machine's second output is a witness for the statement. (Note that if the probability that the verifier accepts is p, then both the event that the machine

[4] Such a protocol can be constructed by taking a 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.

outputs a witness, and the event that the simulated transcript is accepting should happen with probability negligibly close to p.)

This issue was addressed by Lindell in [26], who called such a machine a "witness-extended emulator". He proved that there exists such a witness extended emulator for any proof of knowledge. However, the proof of [26] is such that this emulator 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 a 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 [26]).

1.3 Failure of the naive approach and dependence on success probability

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 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 polynomial), and outputs a witness for x. Clearly, the expected running time of the extractor is polynomial (in particular, it is $p(n)$). However, if we halt this extractor before it completes $\frac{1}{\epsilon}$ steps, then with high probability the extractor will *never* output a witness. (Note that $\frac{1}{\epsilon}$ may be much larger than $p(n)$, and therefore the extractor may far exceed its expected running-time and yet still not output anything.)

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

Trading success probability for running time. The observations above also raise a security issue with respect to the use of proofs of knowledge with *expected* polynomial-time knowledge 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.

This is because 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).

1.4 Related work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [24], and were then shown to exist for all $\mathcal{NP}$ by Goldreich, Micali and Wigderson [23]. Constant-round zero-knowledge arguments and proofs were constructed by Feige and Shamir [15], Brassard, Crepeau and Yung [8] and Goldreich and Kahan [21]. All these constant-round protocols utilize expected polynomial-time simulators. Regarding zero-knowledge proofs *of knowledge*, following a discussion in [24], the first formal definitions were provided by Feige, Fiat and Shamir [14] 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 [13] and Goldreich's book [19]. Goldreich [19, 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 (polynomial-time computable) 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).

1.5 Organization.

In Section 2 we describe the basic notations and definitions that we use. Then, in Section 3 we construct a commit-with-extract commitment scheme, which is the main technical tool used to construct our zero-knowledge argument of knowledge. The construction of the zero-knowledge argument of knowledge itself is described in Section 4. 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 $\{0,1\}^n$.

We let $\mu(\cdot)$ denote an arbitrary negligible function (i.e., a function that grows slower than the inverse of any polynomial). 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 $|x|$. If M is a Turing machine, then we denote by $\mathsf{desc}(M)$ its description (or code). 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 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 theorem (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 prover, and computational soundness which needs only hold for a polynomial-time prover. In *proof* systems, unconditional soundness is guaranteed; whereas in *argument* systems only computational soundness must hold. Unless explicitly stated, when we mention proofs in discussion, we mean both proofs and arguments.

We now recall the definition of zero-knowledge proof systems [24]. Actually, we present (a slightly strengthened form of) the definition of *auxiliary-input* zero-knowledge [19, Sec. 4.3.3].[5] 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): *Let (P, V) be an interactive proof (or argument) system for a language L; denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (e.g., in the case of proof systems for $\mathcal{NP}$ languages, $P_L(x)$ denotes the set of witnesses for x). We say that (P, V) is* auxiliary-input zero-knowledge *if there exists a probabilistic polynomial-time algorithm S such that for every probabilistic polynomial-time machine V^*, and every $y_x \in P_L(x)$ and polynomial $l(\cdot)$, the following two probability ensembles are computationally indistinguishable:*

1. $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^{l(|x|)}}$
   where $\langle P(y_x), V^*(z)\rangle(x)$ denotes the output of V^* after interacting with P on common input x, prover's private input y_x and verifier's auxiliary input z.

2. $\{S(\mathsf{desc}(V^*), 1^t, x, z)\}_{x\in L, z\in\{0,1\}^{l(|x|)}}$
   where t is a bound on the running time of V^* on inputs of length $|x| + |z|$.

Black-box zero-knowledge. A zero-knowledge proof system is called black-box zero-knowledge 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 the ensembles $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^{l(|x|)}}$ and $\{S^{V^*(x,z,\cdot)}(x)\}_{x\in L, z\in\{0,1\}^{l(|x|)}}$ are computationally indistinguishable, 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 $V^*(x, z, r, h)$). (We note that the input 1^t is no longer needed since S gets oracle access to V^* and can invoke it at the cost of one step.)

2.2 Zero-Knowledge Arguments of Knowledge

Our definition of arguments of knowledge below differs from the standard definition of [5] in two ways:

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

Non-black-box extraction. Secondly, 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 [15] (which differs from the definition in [14]; see discussion in [5]), and suffices for all practical applications of arguments of knowledge.

[5] We deviate from the definition of auxiliary-input zero-knowledge of [19, 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 [19, Sec. 4.3.3]. Note however, that the definition of [19, Sec. 4.3.3] already implies that for any $c > 0$ there exists a universal simulator for all $\mathbf{Time}(n^c)$ verifiers.

Until recently, all known proofs of knowledge (including [15]) 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 $\mathcal{NP}$.[6] 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 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 interactive machine P such that for every $(x, y) \in R$, all possible interactions of V with P (with auxiliary input y) on common input x are accepting.*
- Validity (or knowledge soundness) with negligible error: *There exists a probabilistic polynomial-time machine K, such that for every 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 that the interactive machine V accepts, on input x, when interacting with the prover P^* upon input x, auxiliary-input y and random-tape r. Let t denote a bound on the maximum running time of P^* when its first input is x. Then, machine K, upon input $(\mathsf{desc}(P^*), 1^t, x, y, r)$, outputs a solution $s \in R(x)$ with probability at least $p(x, y, r) - \mu(|x|)$.*

An interactive pair (P, V) so 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.*

The input 1^t is provided to allow K to run in time which is (some fixed) polynomial in the running time of P^*. It may be redundant, depending on the representation chosen for $\mathsf{desc}(P^*)$ (for example it is redundant if P^* is represented as a boolean circuit).

If an argument 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 of knowledge for R.

2.3 Witness-Extended Emulation

Zero-knowledge proofs of knowledge are often used as subprotocols within larger protocols. Typically in this context, the extractor for the proof of knowledge supplies the simulator for the larger protocol with some secret information. This information then enables the simulation 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 simulator needs to not only extract a witness from the proof of knowledge, but must also obtain a transcript of the execution of the proof of knowledge itself. Unfortunately, by definition, the extractor only outputs a witness and does not provide the simulator with such a transcript. This issue was addressed in [26] where, loosely speaking, it was shown that for any zero-knowledge

[6] 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 [29].

proof of knowledge, there exists a machine who outputs both the witness (with the appropriate probability) *and* the protocol 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 [26] for a more detailed discussion). We proceed by presenting the formal definition of a witness-extended emulator, and then discuss its relevance to our work here. 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 proof (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.
- Let $\mathsf{accept}_V(\cdot)$ be a deterministic function that takes a specific view of the prover in a protocol execution, and outputs whether or not V accepts in this execution. Note that if the protocol is publicly verifiable, then this function can be efficiently computed from the prover's view.

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 $\mathsf{desc}(P^*)$, the contents of P^*'s input, auxiliary-input and random-tapes, x, y and r respectively, and 1^t, where t is a bound on P^*'s running-time when its first input is x. We denote by $E_1(\mathsf{desc}(P^*),1^t,x,y,r)$ and $E_2(\mathsf{desc}(P^*),1^t,x,y,r)$ 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 $x,y,r \in \{0,1\}^*$, every polynomial $p(\cdot)$ 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,*

$$\left\{E_1 = E_1(\mathsf{desc}(P^*),1^t,x,y,r)\right\}_{x,y,r} \overset{\mathrm{c}}{\equiv} \{\mathsf{view}_{P^*}(P^*(x,y,r),V(x))\}_{x,y,r}$$

2. *The probability that V would accept when P^*'s view is as output by E_1, and yet E_2 does not output a correct witness, is negligible. That is,*

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

There are a number of differences between Definition 2.3 and the definition provided in [26].

1. First and foremost, we require that the witness-extended emulator be strict polynomial-time, whereas in [26] it was allowed to run in expected polynomial-time.
2. In [26], the view that is output by the emulator must be distributed exactly as in a real execution. On the other hand, we only require that the view output be *indistinguishable* from in a real execution. Enabling computational indistinguishability is a natural relaxation of the

requirement and is enough for most applications. We note that we make this modification because we do not know how to achieve the result otherwise. (If E is allowed to run in expected polynomial-time, then it is easy to achieve the stronger requirement of equivalent distributions. However, given that we limit E to strict polynomial-time, we do not know how to do this.)

3. In the definition of [26], the emulator outputs the *verifier's* view of the execution. This view is output in order to enable a simulator who runs the witness-extended emulator to construct the proof transcript, and also to know whether or not the proof was accepting. We were unable to construct a witness-extended emulator who can output a witness along with a string that is indistinguishable from the verifier's view (the reason being that our extractor works very differently from a real verifier). Therefore, we output the *prover's* view instead and require, for simplicity, that the proof be publicly verifiable. Despite this significant technical difference in the definition, the same effect is obtained (this is because the proof transcript can be constructed from either party's view).[7]

We now discuss the relevance of witness-extended emulation to our work. Clearly, in order to use our zero-knowledge proof of knowledge as a subprotocol, we also need witness-extended emulation and not just extraction. However, [26] already proved that the existence of an extractor implies the existence of a witness-extended emulator.

Unfortunately, however, this is not enough. This is because [26] only requires that the emulator be *expected* polynomial-time, and not strict polynomial-time as we require here. Furthermore, the fact that the emulator of [26] runs in expected polynomial-time is *not* due to the fact that the underlying extractor runs in expected polynomial-time. That is, even if the starting point is a *strict* polynomial-time extractor (as is the case for our protocol), the methodology of [26] still results in an expected polynomial-time emulator. This is due to the design of the emulator which, loosely speaking, works in the following way. First, it verifies the proof from P^*. Then, if the proof was convincing, the emulator runs the extractor many times, until a witness is extracted (recall that the probability that the extractor obtains a witness in each "run" is at most negligibly smaller than the probability that P^* was convincing in the first stage).[8] We will not go into the analysis of the emulator here. However, it is clear that such a strategy inherently yields an expected polynomial-time machine, and not a strict polynomial-time one.

For the above reason, we are unable to provide an analogous lemma to that provided in [26] (i.e., that the existence of a strict polynomial-time extractor implies the existence of a strict polynomial-time witness-extended emulator). Rather, we directly show the existence of a (strict polynomial-time) witness-extended emulator for our proof of knowledge. Therefore, our zero-knowledge proof of knowledge can be used as a subprotocol in a larger protocol, and the simulator using it will remain strict polynomial-time.

[7] In fact, the definitions could be united by having the emulator output a witness and a transcript of the messages sent between P^* and V. This definition was not used in [26] in order to not require that the proof of knowledge be publicly verifiable. This can also be overcome, but the result is a more cumbersome definition.

[8] The exact emulator of [26] works slightly differently since they base themselves on a different definition of proofs of knowledge (that is equivalent in their setting). Here we restate the methodology to suit the definition used here. Note also that the emulation is actually more complicated because such a naive strategy may yield a super-polynomial execution; see [26] for details.

2.4 Commit with Extraction

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* (this property is called hiding). Furthermore, the commitment is binding, and thus in a later stage when the commitment is opened, it is guaranteed that the "opening" can yield only a single value determined in the committing phase. In a *perfectly binding* commitment scheme, the transcripts resulting from a commitment to 0 and a commitment to 1 are disjoint (with overwhelming probability). Thus, for every such transcript, there is a unique value that the sender can decommit to.[9] See [19, Sec. 4.4.1] for a formal definition of commitment schemes.

A central tool in our construction of zero-knowledge arguments of knowledge is a (constant-round) 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. 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).[10] Now, let x and v be the committed value and sender's view respectively, as output by the extractor. Then, we require that $x = \mathsf{commit\text{-}value}(v)$. 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. This is also needed for the broader notion of witness-extended emulation, see Section 2.3. We now present the formal definition:

Definition 2.4 (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 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 $(\mathsf{desc}(A^*), 1^t, x, y, r)$, where t is a bound on the running time of $A^*(x, y, r)$, machine CK outputs a pair, denoted $(CK_1, CK_2) = (CK_1(\mathsf{desc}(A^*), 1^t, x, y, r), CK_2(\mathsf{desc}(A^*), 1^t, x, y, r))$, satisfying the following conditions:*

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

Public Decommitment. We say that a commitment scheme satisfies public decommitment if in the decommitment (or reveal) phase, the receiver does not need to access its random tape from the commitment phase. In other words, the transcript from the commit phase and the sender's decommitment message are enough to efficiently obtain the commitment value. We note that this additional feature is needed for our zero-knowledge argument of knowledge.

[9] We require that the binding property holds with respect to a polynomial-time sender only. We note that perfect binding is usually stated so that it holds even for an all powerful sender.

[10] This function may not necessarily be efficiently computable. However, since the commitment scheme is perfectly binding, the function is well-defined. We note that actually the commitment scheme that we construct is only perfectly binding with overwhelming probability (like the scheme of Naor [27]). Therefore, in the negligible event that there is more than one possible committed value, we define the output of the function to be $\perp$.

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.[11] 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.

Extractable commitment schemes. The notion of a commitment scheme with the additional property that the committed value can be extracted from the sender is not new. It has been used in the context of secure multi-party computation (e.g., [18, Consruction 2.3.8]) and has been called both commit-with-knowledge [10] and non-oblivious commitment [19, Def. 4.9.3]. The implementation of these schemes has always involved proofs of knowledge, with the method described above. In contrast, our protocol (presented in Section 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 previous schemes, the committed value is determined *before* the extraction begins. Therefore, the intuition that the sender "knows" the committed value is justified because the sender can apply the extractor to itself and thereby obtain the committed value. However, in our protocol, the committed value is only determined at the *conclusion* of the protocol. Furthermore, there is no extractor that can be later applied to obtain the committed value. Thus, on one hand, it seems that the intuition that the sender knows the committed value cannot be justified here. On the other hand, for all intents and purposes, extraction of the committed value suffices whether or not the value is predetermined. At the end of Section 3, we describe a particular sender who participates in our commit-with-extract protocol and does not know the committed value (even though the extractor does).

3 A Commit-with-Extract Scheme

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

Theorem 3.1 *Assume the existence of trapdoor permutations over* $\{0,1\}^n$ [12] *and collision-resistant hash functions. Then, there exist constant-round commit-with-extract string commitment schemes satisfying public decommitment.*

To simplify the presentation, we will 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 $y = f(r)$ is chosen jointly by the sender and the receiver using a

[11] Actually, it would be enough to use a *strong witness indistinguishable* proof of knowledge [19, Sec. 4.6], rather than a *zero-knowledge* proof of knowledge.

[12] See Footnote 3.

coin-tossing protocol (based on [26]). 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 $f^{-1}(y)$. In this case, the extractor can easily obtain the commitment value σ, as desired.

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 [12]). Formally, the protocol is parameterized by a family of trapdoor permutations over $\{0,1\}^n$, with (function) sampling algorithm I. We denote a permutation from the family by f and its associated trapdoor by t. Furthermore, we denote by b a hard-core of f.

One of the components of the protocol is a constant-round zero-knowledge argument with a *strict* polynomial-time simulator. We note that such an argument exists if collision-resistant hash functions exist [3].

Protocol 3.2 (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 indeed a permutation, using a constant-round zero-knowledge argument (with a strict polynomial-time simulator).*[13] *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 (using any perfectly-binding commitment scheme and a random string 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).*[14] *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:**

[13] The use of zero-knowledge with strict polynomial-time simulation is not essential here. Rather, it simplifies our analysis. Additionally, this proof can be eliminated entirely if we use *certified* permutations [12]. We note that by replacing this argument with a proof, we also obtain perfect binding against an all-powerful prover.

[14] In contrast to step 1b, the property of strict polynomial-time simulation here is essential.

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 3.2 is indeed a secure commit-with-extract scheme:

Proposition 3.3 *Protocol* 3.2 *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 3.2 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 $\mathsf{commit\text{-}value}(t) = \sigma$.

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 indeed 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 $\sigma = v \oplus b(r)$. However, if f is not a permutation, then there may be more than one possible preimage to $r_1 \oplus r_2$. Nevertheless, this possibility is ruled out (except with negligible probability) by the soundness of the proof provided by A^* in step 1b (where A^* proves that f is indeed a permutation).

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 $\langle f(U_n), b(U_n) \rangle$ and $\langle f(U_n), b(U_n) \oplus 1 \rangle$. 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

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

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 3.2 and the commitment scheme C_n, is that in Protocol 3.2 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 3.2 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

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

(Without loss of generality, we assume that B^* always outputs its view.) 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 3.2 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 $C_n(\sigma)$. Formally, distinguisher D' works as follows (the step numbers are as in Protocol 3.2):

0. D' receives for input a permutation f and a commitment $\tilde{c} = C_n(\sigma) = \langle f(r),\ v = b(r) \oplus \sigma \rangle$, where $r \in_R \{0,1\}^n$.

   D' simulates an execution of A with B^*. Apart from where D' is explicitly instructed to output fail, if at any point A would abort (due to detecting malicious behavior from B^*), then D' also aborts and jumps to step 4 of its instructions below.

1. *Simulation of A choosing a trapdoor permutation:*

   (a) D' passes B^* the permutation f.

   (b) D' runs the zero-knowledge simulator (for the proof that f is a permutation) using the residual B^* as the verifier.

2. *First simulation of the coin-tossing protocol (information gathering):*

   (a) D' receives a commitment $c = \mathsf{Commit}(r_1)$ from B^*.

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

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

   (d) D' verifies the zero-knowledge proof given by B^*.

   If B^* does not send any of the messages that it is supposed to send during the above steps of the coin-tossing protocol, or if the verification of the proof fails, then D' outputs fail.

   Otherwise, except with negligible probability, D' has obtained r_1 where $c = \mathsf{Commit}(r_1)$.

2'. *Second simulation of the coin-tossing protocol (actual simulation):*

   D' rewinds B^* to the beginning of the coin-tossing protocol and runs it again.

   (a) D' receives a commitment $c = \mathsf{Commit}(r_1)$ from B^*.

   (b) D' passes B^* the string $r_2 = r_1 \oplus f(r)$ (recall that D' has already obtained the string r_1 from the first simulation).

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

(d) D' verifies the zero-knowledge proof given by B^*.

If $r_1'' \neq r_1'$ and the proof from B^* was accepting, then D' outputs fail. (We stress that if the verification of the proof with B^* failed, then D' does not output fail, even if $r'' \neq r'$. This is because, in this case, B^* has aborted and so D' jumps to step 4.)

3. *Simulation of 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. (We stress that this view as passed by D' to D may not be complete, as is the case if B^* fails to successfully complete the proof of step 2′d.)

First, notice that D' runs in strict polynomial-time (as we have mentioned, this is not essential but rather simplifies the proof since the security of the commitment scheme C_n is with respect to strict polynomial-time distinguishers). We now analyze D''s distinguishing probability for the commitment scheme C_n. Note that the output of D' is either 0, 1 or fail. To make a standard distinguisher (with output in $\{0,1\}$) we can always replace the output fail with any output that is independent of the input (e.g. with 0). Now, we proceed by proving two claims. First, we show that if D' does not output fail, then it successfully distinguishes commitments to 0 from commitments to 1. Formally,

Claim 3.3.1 *If D' does not output fail then D' distinguishes $C_n(0)$ from $C_n(1)$ with non-negligible advantage. That is,*

$$\left|\Pr_{y\in_R C_n(0)}[D'(y)=1|D'(y)\neq\mathsf{fail}] - \Pr_{y\in_R C_n(1)}[D'(y)=1|D'(y)\neq\mathsf{fail}]\right| > \frac{1}{p(n)}$$

for some polynomial $p(\cdot)$ and infinitely many n's.

Next, we show that with non-negligible probability, D' does not output fail. That is,

Claim 3.3.2 *The probability that D' does not output fail is noticeable. Furthermore, this probability is independent of whether the input of D' was $C_n(0)$ or $C_n(1)$. That is,*

$$\Pr_{y\in_R C_n(0)}[D'(y)\neq\mathsf{fail}] = \Pr_{y\in_R C_n(1)}[D'(y)\neq\mathsf{fail}] > \frac{1}{q(n)}$$

for some polynomial $q(\cdot)$ and infinitely many n's.

Putting the above claims together, we have that D' distinguishes $C_n(0)$ and $C_n(1)$ with non-negligible probability (in particular, for infinitely many n's it distinguishes between them with probability $\frac{1}{p(n)q(n)}$).

Proof of Claim 3.3.1: Recall that in this claim, we consider the case in which B^* behaves in such a way that D' never outputs fail. Therefore, the simulation always concludes and in the case that r_1 and r_2 appear in B^*'s output view, it holds that $r_1 \oplus r_2 = f(r)$. Furthermore, the distribution of B^*'s view as output from the simulation with D' when D' receives $C_n(\sigma)$ as input, is indistinguishable from $v_n^{B^*}(\sigma)$. The only difference between the distributions is regarding the simulated 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' in the second simulation phase is uniformly distributed as in a real execution.)[15] Therefore, for $\sigma \in \{0,1\}$,

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

and by the assumption stated in Eq. (2), we have that for infinitely many n's

$$\left|\Pr[D'(C_n(0)) = 1] - \Pr[D'(C_n(1)) = 1]\right| > \frac{1}{p(n)} - 2\mu(n)$$

This completes the proof of the claim. ■

Proof of Claim 3.3.2: First of all, observe that the only step in which the honest sender A uses its input value σ is the last step. Therefore, in any execution where B^* aborts, B^* obtains *no information* about σ. This implies that the probability that B^* aborts is independent of A's input (and is the same if A is committing to 0 or 1). This also implies, as we required, that the probability that D' outputs fail is independent of whether its input came from $C_n(0)$ or from $C_n(1)$.

Now, denote by ϵ_n the probability that B^* aborts in a real execution with A, when A has 0 for input (this is equal to the probability of abort when A has 1 as input). Note that when B^* aborts, the distinguisher D also gets no information about the input bit. Therefore, it must hold that $\mathsf{adv}_n^D \leq 1 - \epsilon_n$, or in other words , $\epsilon_n \leq 1 - \mathsf{adv}_n^D$.

We proceed by showing that the probability that D' outputs fail in the above simulation is at most negligibly greater than ϵ_n. By the observation above, this will finish the proof, since it would imply that D' has a non-negligible probability ($\mathsf{adv}_n^D - \mu(n)$) of not outputting fail (recall that by the assumption, adv_n^D is non-negligible). By D''s instructions, it outputs fail in one of two cases:

1. If B^* aborts in the first simulation of the coin-tossing protocol (i.e., by refusing to send a message or failing to prove the proof of the first simulation): we claim that the probability that D' outputs fail due to this event is at most $\epsilon_n + \mu(n)$. This can be seen due to the following. Assume for a moment that the zero-knowledge proof provided by D' in step 1b is real and not simulated. Then, B^*'s view up until the end of the first simulation phase with D' is *identical* to its view in a real execution with A. By replacing the zero-knowledge proof of step 1b with a simulated one, we have that B^*'s view until the end of the first simulation phase is indistinguishable from in a real execution. Therefore, B^*'s abort probability until this point in the simulation with D' can only be negligibly far from its abort probability in a real execution.

2. If $r'' \neq r'$ (see step $2'$d): this can only happen if B^* successfully proved that $c = \mathsf{Commit}(r')$ and $c = \mathsf{Commit}(r'')$. However, the commitment scheme used is perfectly binding and therefore one of the above statements is false. Therefore, by the soundness of the zero-knowledge argument system, this event can happen with at most negligible probability $\mu(n)$.

We conclude that

$$\Pr[D'(C_n(0)) = \mathsf{fail}] = \Pr[D'(C_n(1)) = \mathsf{fail}] < \epsilon_n + 2\mu(n) \leq 1 - \mathsf{adv}_n + 2\mu(n)$$

which is what we wanted to prove. ■

[15] There's also a difference because of the test D' makes in Step 2d. However, we will see in the proof of Claim 3.3.2 that the probability that D' outputs fail in this step is negligible.

The extraction property. Having established that Protocol 3.2 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 such an aborted execution this is the 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 to A^*'s view in a real execution. In actuality, there are two differences: firstly the commitment received by A^* in step 2a is to 0^n rather than to r_1. Secondly, 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 3.2 to strings: To prove Theorem 3.1 we need to generalize Protocol 3.2 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 3.2 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 3.2, 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.

On the sender's knowledge of the commitment value. We note that although the extractor always obtains the value being committed to, there is no guarantee that the sender itself actually knows it. In fact, one can construct a sender A^* who is oblivious to this value, as follows. Let A^* be such that it chooses a one-way permutation without obtaining the corresponding trapdoor. Furthermore, in Step 3 of Protocol 3.2, A^* sends a random bit $b \in \{0,1\}$ to B. Then, since $r_1 \oplus r_2$ is uniformly distributed, A^* cannot guess the value of the bit that it committed to with probability non-negligibly greater than 1/2.

Despite this, since the commitment scheme used in step 2a of the protocol is *perfectly binding*, the transcript of a protocol execution defines a *unique* committed-to value. Thus, the commitment extractor is able to obtain this value (even if the sender itself does not know it).

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]). 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 may be constructed by running the perfectly-binding commitment scheme of [27], followed by either a constant-round zero-knowledge proof of knowledge [15] or a non-constant round strong zero-knowledge proof of knowledge, such as the one in [19].

4 A Zero-Knowledge Argument of Knowledge

Given a commit-with-extract commitment scheme, it is not hard to prove Theorem 1 by constructing a zero-knowledge argument of knowledge, with strict polynomial-time extraction. 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 $(x, w) \in R$.*[16]

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 3.1.

Proposition 4.2 *Assume that the commitment scheme of Phase 1 is a commit-with-extract string commitment scheme. Then, Protocol 4.1 is a 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.

[16] 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 $\mathcal{NP}$.

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 $\mu(\cdot)$).

To prove that Protocol 4.1 satisfies knowledge soundness, we need to show a knowledge extractor algorithm K. We now describe such an algorithm:

Algorithm 4.3 (knowledge extractor K):

- Input: $(\mathsf{desc}(P^*), 1^t, x, y, r)$

1. *Let CK be the extractor for the commit-with-extract scheme. Then, invoke CK on input $(\mathsf{desc}(P^*), 1^t, x, y, r)$, 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

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

where $\mathsf{view}_{P^*}^{CK}(x, y, r)$ 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 $\mathsf{view}_{P^*}^{CK}(x, y, r)$, 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 $p(x, y, r) - \frac{1}{p(|x|)}$. 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. Let V^* be a verifier algorithm for Protocol 4.1. Then the simulator S works as follows:

Algorithm 4.4 (simulator S):

- Input: $x \in L$

1. *S plays the honest sender for the commit-with-extract scheme with V^* 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 $V^*(\mathsf{trans})$ the residual machine who verifies the proof in Phase 2.*[17] *Then, S runs the simulator for the zero-knowledge proof of Phase 2, with $V^*(\mathsf{trans})$ as the verifier.*[18]
3. *Output whatever V^* outputs (without loss of generality, we assume that V^* always outputs its view).*

We need to prove that:

$$\{S(x)\}_{x\in L} \overset{\rm c}{\equiv} \{\mathsf{view}_{V^*}(P(x,y),V^*(x))\}_{x\in L} \text{ for any } y \text{ s.t. } (x,y)\in R$$

(where P is the honest prover algorithm). We will prove this using a standard hybrid argument. We define an intermediate distribution $H_{x,y}$ in the following way: $H_{x,y}$ 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 $(x,y)\in R$, 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 $V^*(\mathsf{trans})$ as S does (instead of really proving the statement as the honest prover would).

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

[17] 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).

[18] 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.

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 $(\mathsf{desc}(P^*), 1^t, x, y, r)$, 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 $(\mathsf{desc}(P^*), 1^t, x, y, r)$ 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^* after 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 Bound

In this section we prove Theorems 2 and 3. That is, 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. We first present a lemma that connects between strict polynomial-time *simulation* and *extraction*:

Lemma 5.1 *Suppose that there exist one-way functions and a constant-round zero-knowledge efficient-prover proof or argument of knowledge for $\mathcal{NP}$ with a strict polynomial-time black-box* extractor.[19] *Then, there exists a constant-round zero-knowledge argument for $\mathcal{NP}$ with a strict polynomial-time black-box* simulator*.*

Proof Sketch: We prove this lemma by showing how to construct a constant-round zero-knowledge argument system with a strict polynomial-time black-box simulator using the primitives assumed in the lemma (i.e., one-way functions and a zero-knowledge argument of knowledge with a strict polynomial-time black-box extractor). Our construction is based on the constant-round zero-knowledge arguments of knowledge of Feige and Shamir [15] (see Appendix A for a detailed description).

In the first phase of the Feige-Shamir argument system, a subprotocol is performed where the verifier proves to the prover that it knows an NP-witness for some hard problem (based on one-way functions) that it had generated previously. This proof is executed using a zero-knowledge (or even witness-hiding) argument of knowledge. In the second phase of the protocol, the prover proves that it knows either a witness to the statement being proved or a witness to the above-mentioned hard

[19] Actually the protocol need not be zero-knowledge and it suffices for it to be only witness hiding [16]; see Appendix A.

problem. This second proof is executed using a constant-round witness-indistinguishable proof of knowledge.

Soundness of the Feige-Shamir protocol is derived from the fact that a polynomial-time bounded prover cannot solve the hard problem (even having seen the first proof), and therefore must use a witness to the statement being proved. On the other hand, zero-knowledge is demonstrated as follows. The proof of the first phase is a proof of knowledge. Therefore, the simulator begins by running the knowledge extractor for this proof of knowledge and obtaining the witness to the hard problem. In the second phase, the simulator can then prove the proof by using this witness (rather than the witness to the statement being proved). Since the proof of the second phase is witness-indistinguishable, the resulting proof is indistinguishable from one provided by the honest prover (who uses a witness to the real statement).

Notice that the second phase of the proof can be run by a black-box simulator in strict polynomial-time. This is because it merely runs the honest prover strategy for the witness-indistinguishable proof, while using an alternative witness. Therefore, if the proof of knowledge of the first phase has a strict polynomial-time black-box extractor, it follows that the entire simulation is black-box and runs in strict polynomial-time. This completes the proof sketch. ■

Thus, assuming the existence of one-way functions, it is enough to rule out the existence of constant-round zero-knowledge protocols with strict polynomial-time black-box *simulators*, in order to also rule out the existence of analogous protocols with strict polynomial-time black-box *extractors*. That is, Theorem 3 is obtained by proving Theorem 2 and applying Lemma 5.1. We now restate and prove Theorem 2.

Theorem 5.2 (Theorem 2 – restated) *Let L be a language. If there exists a* constant-round *zero-knowledge argument or proof system for L with a* strict *polynomial-time* black-box *simulator, then $L \in \mathcal{BPP}$.*

Before presenting the proof, we motivate why it is not possible to obtain strict polynomial-time black-box simulation for constant-round protocols. First, consider a very simple (cheating) verifier V^* who at every step either aborts or sends the honest verifier message. Furthermore, the probability that it does not abort at any given step is $\epsilon = \epsilon(n)$. Then, black-box simulators for constant-round protocols would typically simulate V^*'s view using the following strategy: Invoke an execution with V^* and if V^* aborts, then also abort. However, if V^* does not abort (and thus sends a verifier message), then continually rewind V^* until another verifier message is obtained. This second case (where V^* does not abort) occurs with probability only ϵ, but then the expected number of rewind attempts by the simulator 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.

Now, let S be a strict polynomial-time black-box simulator that runs in time $t = t(n)$, and (supposedly) simulates a zero-knowledge protocol in which the verifier sends c messages, for some constant c. By carefully choosing the non-abort probability ϵ for the verifier V^* described above, we show that S cannot succeed in simulating such a verifier. Specifically, we choose ϵ so that the following two properties hold:

1. The number ϵ^c, which is the probability that V^* sends all c verifier messages in a real execution, is at least $1/p(n)$, for some polynomial $p(\cdot)$.

2. The probability that S sees *more than* c verifier messages from V^* within t steps, is noticeably less than $1/p(n)$.

(As we will see in the proof, when c is a constant, it is possible to choose ϵ in such a way.) Then, the above implies that with noticeable probability, S must successfully simulate for V^* while seeing only c (or less) verifier messages. However, this is the number of messages that are sent by the honest verifier in a real execution. Therefore, this means that with noticeable probability, S can also successfully convince the honest verifier. Intuitively, this is not possible for languages outside of $\mathcal{BPP}$ (as S does not have a witness to the statement being proved).

In summary, the impossibility result is due to the following two facts: (1) Intuitively, in order to successfully simulate, the simulator must see more than c verifier messages during the simulation. (2) For any strict polynomial-time simulator S, there exist verifiers for which the time needed to view more than c verifier messages is beyond the scope of S's running-time. We now proceed to present the formal proof.

Proof of Theorem 5.2: The general outline of our proof follows the outline of previous proofs that showed limitations on black-box simulation (e.g., [22, 9]). This outline is as follows:

1. Assume, for the sake of contradiction, that there exists a zero-knowledge protocol with a black-box simulator that satisfies the desired properties. In our case, the properties are having both a constant number of rounds and a strict polynomial-time simulator.

2. Construct a family of verifiers that will be "hard" to simulate. Typically, any verifier in the family will be deterministic, but will obtain coin tosses from an internally hardwired $t(n)$-wise independent hash function, where $t(n)$ is the running time of the simulator. Thus, when this hash function is chosen uniformly, these coin-tosses will look completely random to the simulator. (This prevents the simulator from utilizing the fact that the verifier's random-tape is fixed for the entire simulation.)

3. Argue about the behavior of the simulator when given oracle access to a verifier that is chosen randomly from the verifier family.

4. Use the simulator to construct a "cheating prover" – an interactive algorithm P^* that on input $x \in L$ manages to cause the honest verifier to accept with probability that is larger than the inverse of some polynomial. Note that, unlike the honest prover, algorithm P^* does *not* receive a witness for x as additional input.

5. The soundness of the proof system ensures us that if $x \notin L$, then any P^* will only succeed in causing the honest verifier to accept with negligible probability. On the other hand, as we have mentioned, if $x \in L$, then the above P^* convinces the honest verifier with polynomial probability. Therefore, we can use P^* and the honest verifier to obtain a decision procedure for the language L, thus proving that $L \in \mathcal{BPP}$. (Notice that both P^* and the honest verifier can be run in probabilistic polynomial-time given only the input x.)

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. 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.

Let (P, V) be the assumed zero-knowledge protocol that has a constant number of rounds (with negligible soundness-error), and let S be the probabilistic strict polynomial-time black-box simulator for this protocol. We introduce the following notation:

- Let $t = t(n)$ be the strict polynomial-time bound on the running-time of the simulator, when the length of the statement is n. We stress that S runs in time $t(n)$ regardless of the running time of the verifier it simulates.[20]
- Let $l = l(n)$ be the number of random bits that the honest verifier V uses when the length of the statement is n.
- We denote by c the (constant) number of prover messages in the protocol (P, V), and by $m = m(n)$ the length of the longest prover message in the protocol. (Thus we have that the total number of bits sent by the prover in a protocol execution is at most $c \cdot m$.) We assume, without loss of generality, that all prover messages in the protocol are of the same length (this can be obtained by padding messages with zeros and is only used to ensure the unique parsing of any sequence of prover messages).

Assumptions regarding the simulator S: Recall that a black-box simulator has oracle access to the verifier it simulates. 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 simulator S always behaves in the following way:

1. It never asks the same query twice.
2. If S 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. Recall that at the end of the simulation, the simulator should output a simulated view of the verifier. That is, S should output a transcript that contains all the prover messages in this simulated view. Let $\mathsf{trans} = (\alpha_1, \ldots, \alpha_c)$ be the transcript that the simulator outputs. Then, we assume that S has queried its oracle on all the prefixes of trans.

We can assume all of the above because any simulator can be easily modified so that it behaves in the above way, without affecting its output distribution. Furthermore, the running-time of the modified simulator is at most c times the running-time of the original one.

The family of verifiers. We are now ready to define the family of verifiers as described in the proof outline. Let $\epsilon = \epsilon(n)$ be some function that will be determined later (it will be of the form $1/p(n)$ for some polynomial $p(\cdot)$). Furthermore, let $\mathcal{H} = \{H_n\}_{n \in \mathsf{N}}$ be a family of $t(n)$-wise independent hash functions, so that for $h \in H_n$, $h\colon \{0,1\}^{\leq c \cdot m} \rightarrow \{0,1\}^{\log(1/\epsilon)}$, where $\{0,1\}^{\leq c \cdot m}$ denotes the set of all strings of length at most $c \cdot m$.

For every $h \in \mathcal{H}$ and $r \in \{0,1\}^l$, we define a deterministic verifier $V_{h,r}$. The desired behavior of this verifier is as follows. In every round, the verifier aborts with probability $1 - \epsilon$. On the other hand, if it does not abort, then it sends an honest verifier message. Loosely speaking, this

[20] As in most black-box lower-bounds, this assumption may be relaxed if we use a pseudorandom function instead of a t-wise independent hash function.

is achieved using h and r in the following way. For every message of the form $q = (\alpha_1, \ldots, \alpha_i)$ received by $V_{h,r}$, it computes $h(q)$ and continues if and only if $h(q) = 0^{\log(1/\epsilon)}$. (Since at most t queries are asked, and h is a t-wise independent hash function, this means that for every message, the probability that $V_{h,r}$ continues is ϵ (and is independent of other messages).) Then, if $V_{h,r}$ does continue, it replies by running the honest verifier, with random-tape r, on the message q.

We now formally define the verifier (note that we define the interactive verifier in the form of its next message function):

Algorithm 5.2.1 (verifier $V_{h,r}$):

- Input: *series of prover 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, $h(\alpha_1, \ldots, \alpha_j) = 0^{\log(1/\epsilon)}$. (That is, abort unless $h(\alpha_1) = h(\alpha_1, \alpha_2) = \cdots = h(\alpha_1, \ldots, \alpha_i) = 0^{\log(1/\epsilon)}$.)

     (*Notice that since the definition of $V_{h,r}$ 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 verifier strategy:

     (a) Run the honest verifier on input $(\alpha_1, \ldots, \alpha_i)$ and with random-tape r, and obtain its response β.

     (b) Return β.

The above defines a specific verifier, for a given h and r. We define a family of verifiers by $\mathcal{V} = \{V_{h,r}\}_{h \in \mathcal{H}, r \in \{0,1\}^l}$.

Analysis of the behavior of $V_{h,r}$. We begin by analyzing the behavior of a verifier $V_{h,r}$ chosen randomly from $\mathcal{V}$, upon interacting with the honest prover P. (Notice that a verifier can be randomly chosen from the family by simply choosing h uniformly in H_n and $r \in_R \{0,1\}^l$.)

Let $x \in L$, and let P interact with $V_{h,r}$ on public-input x. Then, $V_{h,r}$ behaves exactly like the honest verifier V, except that in every round $V_{h,r}$ throws independent coins and continues only with probability ϵ. Therefore, for $h \in_R H_n$ and $r \in_R \{0,1\}^l$, the probability that $V_{h,r}$ does not abort and is convinced by the prover is ϵ^c.[21] In other words, if we denote by $p_{h,r}$ the probability that $V_{h,r}$ does not abort when interacting with P on input x (this probability is over P's coins), then we have that

$$E[p_{h,r}] = \epsilon^c$$

where the expectation is taken over $h \in H_n$ and $r \in \{0,1\}^l$.

We now proceed to analyze the analogous behavior of $V_{h,r}$, in a simulation by S. For every $h \in H_n$ and $r \in \{0,1\}^l$, denote by $\tilde{p}_{h,r}$ the probability that the simulator, on input x and with oracle access to $V_{h,r}$, outputs a convincing transcript that is not aborting. That is, $\tilde{p}_{h,r}$ equals the probability that S outputs a transcript $(\alpha_1, \ldots, \alpha_c)$, such that

[21] This assumes that if $V_{h,r}$ does not abort, then it is always convinced. This holds assuming that the interactive proof being used has perfect completeness. Throughout, we indeed do assume perfect completeness. However, our proof can be extended to hold for any noticeable completeness bound.

1. The honest verifier would accept the transcript when its random-tape equals r, and
2. For every $1 \leq i \leq c$, $h(\alpha_1 \ldots \alpha_i) = 0^{\log(1/\epsilon)}$.

By the zero-knowledge property, for *every* $h \in H_n$ and $r \in \{0,1\}^l$, it must hold that $|\tilde{p}_{h,r} - p_{h,r}| < \mu(n)$ (otherwise, we could distinguish between the output of the simulator and the view of $V_{h,r}$'s interaction with the honest prover[22]). In particular, as ϵ will be set to be the inverse of some polynomial, it follows that for *every* $h \in H_n$ and $r \in \{0,1\}^l$, $\tilde{p}_{h,r} > p_{h,r} - \epsilon^c/2$. Therefore,

$$E[\tilde{p}_{h,r}] \geq \epsilon^c/2$$

where the expectation is taken over $h \in H_n$ and $r \in \{0,1\}^l$.

We conclude that for $h \in_R H_n$ and $r \in_R \{0,1\}^l$, if we run the simulator S with oracle access to $V_{h,r}$, then with probability at least $\epsilon^c/2$, simulator S will output a transcript $\mathsf{trans} = (\alpha_1, \ldots, \alpha_c)$ such that $V_{h,r}$ returns non-$\perp$ answers on all c prefixes of trans, and such that the honest verifier accepts trans when its random-tape equals r. In the case that this happens we say that S is *successful*.

We now proceed to show that with noticeable probability, when simulator S is successful, it obtains at most c messages from $V_{h,r}$. Loosely speaking, if this is the case, then S could also convince an honest verifier with this same probability. As we will see, this will later serve as the basis for the construction of the cheating prover (as described in the outline).

Claim 5.2.2 *Let* good *denote the following event: the simulator S with oracle access to $V_{h,r}$ and input x, outputs a transcript* $\mathsf{trans} = (\alpha_1, \ldots, \alpha_c)$ *such that:*

1. *S is* successful. *That is, the honest verifier accepts* trans *when its random-tape equals r, and $V_{h,r}$ returns non-$\perp$ answers for all prefixes of* trans.
2. *S obtained exactly c non-$\perp$ answers from its oracle during the simulation.*

Then, the probability that the event good *occurs is at least $\rho \stackrel{\text{def}}{=} \epsilon^c/2 - \binom{t}{c+1}\epsilon^{c+1}$, where this probability is taken over $h \in H_n$, $r \in \{0,1\}^l$, and the random-coins of S.*

Proof: Recall that the simulator S always queries its oracle on all the prefixes of the transcript it outputs (see the assumptions regarding S above). Therefore, from item (1) of the claim, it follows that a successful S must obtain at least c non-$\perp$ answers during the simulation. Item (2) then limits the number of non-$\perp$ answers to be exactly c. The probability that good occurs is therefore equal to the probability that S is successful *and* S did not receive more than c non-$\perp$ replies (and this is greater than or equal to the probability that S is successful *minus* the probability that S does receive more than c non-$\perp$ replies).

Now, by the above analysis, we have that the probability that S is successful is at least $\epsilon^c/2$. On the other hand, we claim that the probability that S obtains more than c non-$\perp$ answers is at most $\binom{t}{c+1}\epsilon^{c+1}$. This holds because S queries the oracle at most t times, and each query is answered with non-$\perp$ with probability exactly ϵ (this immediately follows from the fact that H_n is a t-wise independent hash family). Therefore, the probability that among the t queries, $c+1$ or more non-$\perp$ replies are obtained, is at most $\binom{t}{c+1}\epsilon^{c+1}$.[23]

[22] Note that as is standard in such arguments, the distinguisher would receive the hash function h and random-coins r as auxiliary input.

[23] This follows from the formula stating that for n Bernoulli trials where each trial succeeds with probability p, the probability of having k or more successes is at most $\binom{n}{k}p^k$.

We conclude that the probability that the event good occurs is at least $\epsilon^c/2 - \binom{t}{c+1}\epsilon^{c+1}$, as required. ■

Intuitively, if the event good happens then the simulator succeeds in generating an accepting transcript even though it sees only c messages from the verifier. As we have mentioned, this implies that the simulator could convince the honest verifier with the same probability as the event good occurring. Before proceeding to prove this, we "modify" S into a different simulator S' for which it is easier to formally show the above-stated intuition. In order to be able to use S', we ensure that the modification is such that Claim 5.2.2 still holds with respect to S' (and thus S' could also be used to convince the honest verifier). Loosely speaking, the modified S' runs the simulator S and ensures that the only non-$\perp$ replies that S receives belong to a single transcript $(\alpha_1, \ldots, \alpha_i)$. That is, if S asks two queries for which neither one is a prefix of the other, and both of these are answered with non-$\perp$, then S' aborts the execution. Although this seems like a significant restriction of S, the point is that when the event good occurs, S anyway only receives non-$\perp$ replies for the prefixes of its output transcript. Therefore, in such a case, S' would not abort. The simulator S' is formally defined in the following way:

Algorithm 5.2.3 (simulator S'):

- Input: $x \in \{0,1\}^*$.
- Oracle access to oracle: O (*in our case* O *will be* $V_{h,r}$ *for some* $h \in H_n$, $r \in \{0,1\}^l$).

```
1. Invoke simulator $S$ on input $x$.
2. Whenever $S(x)$ makes a query $q = (\alpha_1, \ldots, \alpha_i)$ to its oracle, do the following:
    (a) Query the oracle $O$ with $q$; denote its response by $\beta$ (i.e., $\beta \leftarrow O(q)$).
    (b) If $\beta \neq \perp$, then abort if the following holds: there exists a previous query $q'$ for which
        $O(q') \neq \perp$ and $q'$ is not a prefix of $q$.
        (Note that $S'$ aborts if the above holds, regardless of the value of $\beta$, and when it aborts
        it outputs nothing.)
    (c) Pass the oracle reply $\beta$ to $S$.
3. Output the result of the execution of $S$.
```

We now claim that the probability that the event good happens for S' (with oracle access to $V_{h,r}$ where $h \in_R H_n$ and $r \in_R \{0,1\}^l$), is at least the probability that it happens for S. First, notice that the only change in the behavior of S' compared to S is that S' may sometimes abort. Yet if S' aborts in an execution in which S would have been successful, then it must be that in this execution S receives more than c non-$\perp$ replies (because one of the non-$\perp$ replies is not a prefix of the output transcript and S asks all of the c prefixes of its output), implying that good did not occur. Thus, S' does not abort in any execution in which the event good can occur. This implies that the probability that good happens with respect to S' is no less than the probability that it happens with respect to S.

Fixing the parameter ϵ. From the above analysis, we have that by Claim 5.2.2, the probability that the event good occurs for S' is at least $\rho = \epsilon^c/2 - \binom{t}{c+1}\epsilon^{c+1}$. We now fix the value of ϵ so that ρ equals the inverse of some polynomial (and thus the event good happens "often").

Let $\epsilon = (4\binom{t}{c+1})^{-1}$. First, notice that there exists a polynomial $p(\cdot)$ such that $\epsilon(n) = 1/p(n)$ (recall that t is a polynomial, and c a constant). Furthermore, for such an ϵ, it holds that $\rho = \epsilon^c/4$. We therefore have that ρ is also the inverse of some polynomial, as required. We are now ready to construct the cheating prover. (We note that it is only possible to choose ϵ in such a way since the protocol is constant-round. Thus, our lower bound holds only for constant-round zero-knowledge, and as we have mentioned, this is tight.)

Constructing the "cheating" prover. We now construct an interactive machine (prover) P^* that on input $x \in L$ (and no witness), can cause the honest verifier to accept with probability at least ρ. The prover P^* works as follows:

Algorithm 5.2.4 (prover P^*):

- Input: x

```
1. Choose $h \in_R H_n$.
2. Run the simulator $S'$ with input $x$.
3. When $S'$ asks its $j^{th}$ query $q = (\alpha_1, \ldots, \alpha_i)$ do the following:
   (a) Decide whether to return $\perp$ in the same way as $V_{h,r}$. That is, return $\perp$ to $S'$ unless
       $h(\alpha_1) = h(\alpha_1, \alpha_2) = \cdots = h(\alpha_1, \ldots, \alpha_i) = 0^{\log(1/\epsilon)}$.
   (b) If $\perp$ is not returned to $S'$ and the previous messages that were sent to the verifier were
       $\alpha_1, \ldots, \alpha_{i-1}$, then send $\alpha_i$ to the verifier and return its reply $\beta$ to $S'$.
   (c) Otherwise, choose an arbitrary non-$\perp$ answer $\beta$ and return it. (Note that it doesn't
       matter what string we choose as we know that in this case $S'$ will abort.)
```

We now claim that P^* convinces the honest verifier with probability at least ρ. That is,

Claim 5.2.5 *For any $x \in L$, the probability that $P^*(x)$ succeeds in convincing the honest verifier V that $x \in L$, is at least ρ.*

Proof: First, denote by r, the random-tape of the honest verifier in an execution with P^*. Then, we claim that P^* perfectly emulates an execution of S' with $V_{h,r}$ (when h and r are uniformly chosen). However, this follows immediately from the definition of P^* and $V_{h,r}$. (The only difference is regarding the final answer that S' receives from P^* in the case that step 3c is executed. In the emulation by P^*, S' receives garbage (and not a real reply). However, this makes no difference, because in such a case S' aborts with no output.)

Therefore, the probability that the event good happens in an execution between P^* and the honest verifier V is at least ρ. However, recall that when the event good happens, the transcript of messages output by S', and thus received by V in its execution with P^*, is convincing for V with random-tape r. Thus, the probability that V accepts the proof from P^* is at least ρ. ■

On the other hand, by the computational soundness of the argument system (P, V), we have the following claim:

Claim 5.2.6 *For any $x \notin L$, the probability that $P^*(x)$ succeeds in convincing the verifier that $x \in L$ is negligible.*

The decision procedure. Combining Claims 5.2.5 and 5.2.6 we obtain the following decision procedure for L. Upon input x, invoke $P^*(x)$ and play the honest verifier. If the proof provided by P^* is accepted by the honest verifier, then output that $x \in L$. Otherwise, output that $x \notin L$. Now, by Claim 5.2.5, we have that the decision procedure accepts $x \in L$ with probability at least ρ (where $\rho = 1/q(|x|)$ for some polynomial $q(\cdot)$). On the other hand, by Claim 5.2.6, we have that the procedure accepts $x \notin L$ with only negligible probability (and in particular, with probability less than $\rho/2$). Thus, we have constructed a probabilistic, polynomial-time decision procedure for the language L, and so $L \in \mathcal{BPP}$. This completes the proof of Theorem 5.2. ■

As noted above, by combining Lemma 5.1 and Theorem 5.2, we obtain Theorem 3 as a corollary.

Corollary 5.3 (Theorem 3 – restated) *Suppose that one-way functions exist. Then, there do not exist constant-round efficient-prover proofs or arguments of knowledge with strict polynomial-time extractors for any $\mathcal{NP}$-complete language L.*

Remark 5.4 We note that our impossibility result regarding strict polynomial-time simulation holds for any language $L \notin \mathcal{BPP}$, and for both arguments and proofs. Furthermore, we make no complexity assumptions in proving this result. On the other hand, our impossibility result for extraction is significantly weaker. Firstly, we assume the existence of one-way functions.[24] Secondly, the lower bound holds only for arguments (or proofs where the prover has an efficient implementation). Finally, due to the reduction via the Feige-Shamir protocol, we only rule out the existence of protocols with strict polynomial-time extraction for NP-complete languages. We note that in actuality, the proof of Lemma 5.1 gives a stronger (but more cumbersome) result, and rules out the existence of such protocols for any language with an invulnerable generator (see Appendix A). Thus, we do rule out such proofs of knowledge in many natural cryptographic settings. Nevertheless, a stronger lower-bound would be preferrable.

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. In *SIAM Journal on Computing*, 17(2):194–209, 1988.

[2] B. Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115, 2001.

[3] B. Barak and O. Goldreich. Universal Arguments and Their Applications. In *17th 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. Record 2001/063, Cryptology ePrint Archive, Aug. 2001. Extended abstract in *42nd FOCS*, pages 116–125, 2001.

[24] Note that this assumption is rather minimal, and in particular is implied by the existence of a zero-knowledge argument for a hard-on-the-average language [28].

[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, Vol. 1, 2 (Berkeley, Calif., 1986)*, pages 1444–1451, Providence, RI, 1987. Amer. Math. Soc.

[8] 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.

[9] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. Record 2001/051, Cryptology ePrint Archive, June 2001. An extended abstract appeared in *33^{rd} STOC*, pages 570–579, 2001.

[10] D. Dolev, C. Dwork and M. Naor. Non-malleable Cryptography. In *SIAM Journal of Computing*, Vol. 30 (20), pages 391–437, 2000.

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

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

[13] 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.

[14] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology* 1(2):77–94, 1988.

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

[16] U. Feige and A. Shamir. Witness Indistinguishable and Witness Hiding Protocols. In *22^{nd} STOC*, pages 416–426, 1990.

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

[18] O. Goldreich. *Secure Multi-Party Computation.* Manuscript. Preliminary version, 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

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

[20] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* To be published. Available from `http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/v2.ps`.

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

[22] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[23] 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 Association for Computing Machinery*, 38(3):691–729, 1991.

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

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

[26] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. To appear in the Journal of Cryptology. Extended abstract in *CRYPTO'01*, pages 171–189, 2001.

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

[28] R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. Technical Report TR-93-073, International Computer Science Institute, Berkeley, CA, Nov. 1993. Preliminary version in Proc. 2nd Israeli Symp. on Theory of Computing and Systems, 1993, pp. 3–17.

[29] S. Goldwasser, R. Canetti, O. Goldreich and S. Micali. Resettable Zero-Knowledge. Cryptology ePrint Archive, Report 1999/022, 1999. `http://eprint.iacr.org/`. Extended abstract in *32nd STOC*, pages 235–244, 2000.

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

A The Feige-Shamir Constant-Round Zero-Knowledge Argument System for $\mathcal{NP}$

In this appendix we outline the construction of the constant-round zero-knowledge arguments of knowledge of Feige and Shamir [15] (the exact construction shown here is actually according to [13]). This construction is used in Lemma 5.1 in order to show that the existence of a zero-knowledge proof (or argument) of knowledge for $\mathcal{NP}$ with strict polynomial-time *knowledge extraction*, implies the existence of a zero-knowledge argument of membership with strict polynomial-time *simulation*.

A.1 Preliminaries

In order to present the protocol we first introduce the following concepts:

Witness-indistinguishability and witness-hiding. Loosely speaking, witness indistinguishability refers to a setting in which the prover may use two (or more) different witnesses in order to prove a statement. We say that a proof is witness-indistinguishable if the verifier cannot deduce which witness the prover used in order to prove the statement. On the other hand, a witness-hiding proof is one in which the verifier is unable to reconstruct the witness from seeing the proof (i.e., the proof keeps the witness "hidden"). This is formalized by saying that if the verifier can output the (entire) witness with some probability after participating in the proof, then it could have output it with almost the same probability by itself. Notice that this does not prevent the verifier from learning part of the witness from the interaction itself, it just guarantees that it does not learn all of it.

More formally, let R be an NP-relation, and let (P, V) be a proof (or argument) system for R. We say that (P, V) is *witness indistinguishable* if for any verifier V^*, for any $x \in L$ and for any two witness w_1 and w_2 such that $(x, w_1), (x, w_2) \in R$, verifier V^*'s view when interacting with $P(x, w_1)$ is indistinguishable from its view when interacting with $P(x, w_2)$. We say that (P, V) is *witness hiding* if for every verifier V^* there exists a machine M^* such that for every x, the probability that V^* outputs w after interacting with $P(x, w)$ is negligibly close to the probability that $M^*(x)$ outputs w.[25]

Witness-indistinguishability and witness-hiding are weaker properties than zero-knowledge (in particular, every zero-knowledge protocol is also witness-indistinguishable and witness-hiding; however the other direction is not true).

Feige and Shamir show how to construct constant-round witness-indistinguishable proofs of knowledge for any NP-relation, assuming the existence of one-way functions [16]. They also show, under the same assumption, how to obtain witness-hiding proofs of knowledge for some specific relations (which suffice for their construction of a constant-round zero-knowledge protocol).

Invulnerable Generators. Loosely speaking, an invulnerable generator is an efficient machine that outputs a pair (t, s), with the property that given t, it is hard to find s. Thus, an invulnerable generator is a machine that enables us to efficiently generate "hard problems" (the hard problem being to find s given t). Formally, Let T be an NP-relation. Then, an *invulnerable generator* for T is a probabilistic polynomial-time algorithm G_T that on input 1^n outputs a pair (t, s) that satisfies:

1. $(t, s) \in T$
2. Given t, it is hard to find a witness for t. That is, for any polynomial-size circuit A, the probability that $A(t)$ outputs s such that $(t, s) \in T$ is negligible (the probability being over the choice of (t, s) by G_T). (That is, $\Pr_{(t,s) \in_R G_T(1^n)}[A(t) \in T(t)] = \mu(n)$.)

An example for a relation T and an invulnerable generator for it is the following. Let f be a one-way function. Then, define $T = \{(t, s) \mid t = f(s)\}$. An invulnerable generator for this relation chooses a random $s \in_R \{0, 1\}^n$ and outputs $(f(s), s)$. Clearly, by the properties of one-way functions, given $t = f(s)$, it is hard to find s.

A.2 The construction

We are now ready to describe the protocol of [13]. The basic idea is that the verifier V first generates a hard problem and then proves to P (using a witness-hiding proof of knowledge) that it knows the

[25] We note that in the formal definition of witness-hiding, machine M^* is allowed to run in *expected* polynomial-time. However, this does not affect the running-time of the simulator (and in particular is used only to prove soundness).

solution to this problem. Following this, P proves to V (using a witness-indistinguishable proof of knowledge) that either it knows a witness to the statement being proved, or that it knows a solution to the hard problem generated by V. Intuitively, soundness follows from the fact that P cannot know a solution to this hard problem (notice that the proof provided by V is witness-hiding and therefore it cannot help P obtain a solution to the problem). On the other hand, the zero-knowledge simulator works by first *extracting* the solution to the hard problem from the witness-hiding proof of knowledge provided by V. Then, in the next stage, it uses this solution to prove the witness-indistinguishable proof that P is supposed to provide. (Notice that since the proof provided by the prover is witness-indistinguishable, V cannot distinguish the case that a real witness is used to the case that a solution to the hard problem is used.) Of course, the hard problem is generated by V with an invulnerable generator.

Let R be an NP-relation. The zero-knowledge argument system for R is constructed as follows (let T be an NP-relation for which we have a constant-round witness-hiding proof of knowledge, and let G_T be an invulnerable generator for T):

Protocol A.1 (zero-knowledge argument for R):

- Common input: x
- Auxiliary input to prover: w *such that* $(x, w) \in R$

1. Phase 1 – V generates a "hard problem":

   (a) V *chooses* $(t, s) \in_R G_T(1^n)$ *and sends* t *to the prover.*

   (b) V *proves that it knows* s *such that* $(t, s) \in T$*, using a witness-hiding proof of knowledge.*

2. Phase 2 – P proves that either $x \in L$ or that it knows a solution to the hard problem:

   P *proves that it either knows a witness* w *such that* $(x, w) \in R$ *or that it knows a witness* s *such that* $(t, s) \in T$*, using a witness-indistinguishable proof of knowledge.*

As we have mentioned above, soundness follows from the fact that G_T is an invulnerable generator and the fact that the witness-hiding proof of Phase 1 does not help P obtain the witness s. Therefore, with overwhelming probability, P is unable to prove the proof of Phase 2 without knowledge of w. On the other hand, the simulator for the protocol works by first extracting a witness for t in Phase 1 (using the knowledge extractor for the witness-hiding proof of knowledge). Then, given this witness, the simulator is able to prove the proof of Phase 2 (without knowing w).

Notice that if the extractor for the proof of knowledge of Phase 1 is black-box and runs in strict polynomial-time, then the simulator inherits both these properties. Therefore, if there exists a witness-hiding proof of knowledge with a strict polynomial-time black-box extractor for any language with an invulnerable generator (and in particular, for all $\mathcal{NP}$), then there exists a black-box zero-knowledge argument of knowledge for $\mathcal{NP}$ with strict polynomial-time simulation.