How to Go Beyond the Black-Box Simulation Barrier

Boaz Barak*

December 30, 2008

Abstract

The *simulation paradigm* is central to cryptography. A *simulator* is an algorithm that tries to simulate the interaction of the adversary with an honest party, without knowing the private input of this honest party. Almost all known simulators use the adversary's algorithm as a *black-box*. We present the first constructions of non-black-box simulators. Using these new non-black-box techniques we obtain several results that were previously shown to be impossible to obtain using black-box simulators.

Specifically, assuming the existence of collision-resistent hash functions, we construct a new zero-knowledge argument system for **NP** that satisfies the following properties:

1. This system has a constant number of rounds with negligible soundness error.
2. It remains zero knowledge even when composed concurrently n times, where n is the security parameter.

   Simultaneously obtaining Properties 1 and 2 has been proven to be impossible to achieve using black-box simulators.
3. It is an Arthur-Merlin (public coins) protocol.

   Simultaneously obtaining Properties 1 and 3 has also been proven to be impossible to achieve with a black-box simulator.
4. It has a simulator that runs in *strict* polynomial time, rather than in *expected* polynomial time.

   All previously known zero-knowledge arguments satisfying Property 1 utilized *expected* polynomial-time simulators. Following this work it was shown that simultaneously obtaining Properties 1 and 4 is also impossible to achieve with a black-box simulator.

Keywords: cryptography, complexity theory, zero knowledge, black box, non-black-box, concurrent zero knowledge , CS proofs, universal arguments

*Department of Computer Science, Weizmann Institute of Science, Israel. Email: boaz@wisdom.weizmann.ac.il

Contents

List of Figures

List of Tables

1 Introduction

The *simulation paradigm* is one of the most important paradigms in the definition and design of cryptographic primitives. For example, this paradigm arises in a setting in which two parties, Alice and Bob, interact and Bob knows a secret. We want to make sure that Alice hasn't learned anything about Bob's secret as the result of this interaction, and do so by showing that Alice could have *simulated the entire interaction by herself*. Therefore, she has gained no further knowledge as the result of interacting with Bob, beyond what she could have discovered by herself.

The canonical example of the simulation paradigm is its use in the definition of *zero-knowledge proofs*, as presented by Goldwasser, Micali and Rackoff [GMR85]. Suppose that both Alice and Bob know a public graph G, and in addition Bob knows a Hamiltonian cycle C in this graph. In a zero-knowledge proof, Bob manages to *prove* to Alice that the graph G contains a Hamiltonian cycle, and yet Alice has learned nothing about the cycle C, as she could have simulated the entire interaction by herself.

A crucial point is that we do not want Alice to gain knowledge even if she *deviates arbitrarily* from the protocol when interacting with Bob. This is usually formalized in the following way: for every algorithm V^* that represents the strategy of the verifier (Alice), there exists a simulator M^* that can simulate the entire interaction of the verifier and the honest prover (Bob) without access to the prover's auxiliary information (i.e., the Hamiltonian cycle). That is, the simulator only has access to the public information (i.e., the graph) that was known to the verifier (Alice) before she interacted with the prover (Bob).

Consider the simulator's task even in the easier case in which Alice does follow her prescribed strategy. One problem that the simulator faces is that, in general, it is impossible for it to generate a convincing proof that the graph G is hamiltonian, without knowing a Hamiltonian cycle in the graph. How then can the simulator generate an interaction that is indistinguishable from the actual interaction with the prover? The answer is that the simulator has two advantages over the prover, which compensate for the serious disadvantage of (the simulator's) not knowing a Hamiltonian cycle in the graph. The first advantage is that, unlike in the true interaction, the simulator has access to the verifier's random-tape. This means that it can actually determine the next question that the verifier is going to ask. The second advantage is that, unlike in the actual interaction, the simulator has many attempts at answering the verifier's questions. This is because if it fails, it can simply choose not to output this interaction but rather retry again and output only the "take" in which it succeeds. This is in contrast to an actual proof, where if the party attempting to prove failed even once to answer a question then the proof would be rejected. The difference is similar to the difference between a live television show and a taped show. For example, if someone has a 10% probability of success in shooting a basketball, then he will probably never have 10 straight hits in his life. In contrast, using video-editing, it is very easy to create a film where this person has 10 straight hits. This second technique is called *rewinding* because the simulator that fails to answer a question posed by the verifier, simply rewinds the verifier back and tries again.

All previously known zero-knowledge protocols made use of this rewinding technique in their simulators. However this technique, despite all its usefulness, has some problems. These problems arise mainly in the context of parallel and concurrent compositions. For example, using this technique it is impossible to show that a constant-round zero-knowledge proof[1] remains zero-knowledge under concurrent composition [CKPR01]. It is also impossible to construct a constant-round zero-knowledge proof with a simulator that runs in *strict* polynomial time (rather than *expected* polyno-

[1]Here and throughout this paper, we only consider zero-knowledge proofs or arguments that have negligible soundness error.

mial running time) or a constant-round proof of knowledge with a strict polynomial time knowledge extractor (see the subsequent work [BL02]).

The reason that all the known simulators were "confined" to the rewinding technique is that it is very hard to take advantage of the knowledge of the verifier's random-tape when using the verifier's strategy as a *black-box*. Let us expand a little on what we mean by this notion. As noted above, to show that a protocol is zero knowledge, one must show that a simulator exists for *every* arbitrary algorithm V^* that represents the verifier's strategy. Almost all the known protocols simply used a *single generic* simulator that used the algorithm V^* as an oracle (i.e. as a *black-box* subroutine). Indeed it seemed very hard to do anything else, as using V^* in any other way seemed to entail some sort of "reverse-engineering" that is considered a very hard (if not impossible) thing to do.

It can be shown that for black-box simulators, the knowledge of the verifier's random-tape does not help the simulator, because a verifier can have its randomness "hardwired" into its algorithm (for instance in the form of a description of a hash/pseudorandom function). Therefore, black-box simulators are essentially restricted to using the rewinding technique, and so suffer from its consequences. Indeed, as mentioned above, several negative results have been proved about the power of black-box simulators, starting with the results of Goldreich and Krawczyk [GK90] regarding non-existence of *black-box* 3-round zero-knowledge proofs and constant-round Arthur-Merlin zero-knowledge proofs, to the recent result of Canetti, Kilian, Petrank and Rosen [CKPR01] regarding impossibility of black-box constant-round concurrent zero-knowledge.

1.1 Our Results

We show that the belief that one can not construct non-black-box simulators is false. That is, given the code of a (possibly cheating) efficient verifier as an auxiliary input, the simulator may significantly use this code in other ways than merely running it, and so obtain goals that are provably impossible to obtain when using the verifier only as a black-box. Specifically, assuming the existence of collision-resistent hash functions[2], we construct a new zero-knowledge argument (i.e., a computationally-sound proof) for any language in **NP** that satisfies the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.
2. It has a constant number of rounds and negligible soundness error.
3. It remains zero-knowledge even if executed concurrently n times, where n is the security parameter. We call a protocol that satisfies this property a *bounded concurrent zero-knowledge* protocol.[3]
4. It is an Arthur-Merlin (public coins) protocol.
5. It has a simulator that runs in *strict* probabilistic polynomial-time, rather than *expected* probabilistic polynomial-time.

The above protocol should be contrasted with the following impossibility results regarding *black-box* zero-knowledge arguments for non-trivial languages: Goldreich and Krawczyk [GK90] showed

[2]Actually in this paper we require that there exist hash functions that are collision-resistent against adversaries of size that is some fixed "nice" super-polynomial function (e.g., $n^{\log n}$ or $n^{\log \log n}$). This requirement can be relaxed to standard (i.e., secure against polynomial-sized circuits) collision-resistent hash functions as is shown in [BG01].

[3]The choice of n repetitions is quite arbitrary and could be replaced by any *fixed* polynomial (e.g. n^3) in the security parameter. This is in contrast to a standard concurrent zero-knowledge protocol [DNS98, RK99] that remains zero-knowledge when executed concurrently any polynomial number of times.

that such protocols cannot satisfy both Properties 2 and 4. Canetti, Kilian, Petrank and Rosen [CKPR01] showed that such protocols cannot satisfy both Properties 2 and 3. Subsequently to this work, Barak and Lindell [BL02] showed that such protocols cannot satisfy Properties 2 and 5.

1.2 Our approach

Our zero-knowledge argument is constructed using a technique, which we call the FLS technique, that has been used before in the design of zero-knowledge arguments (its first explicit use was by Feige, Lapidot and Shamir [FLS99]). In the FLS technique, we take an interactive proof/argument for a language L and modify it so that if the prover knows some *trapdoor information* σ, then the prover will be able to "cheat" and convince the verifier that *any* string x is in L, even without knowing a witness for x and even when x is actually not in L. Naturally, to preserve soundness, one must ensure that it is infeasible to obtain this trapdoor information σ when interacting with the honest verifier. Although at first this may seem to make the modification pointless, this modification is in fact *crucial* to obtaining the zero-knowledge property. The reason is that although the trapdoor information σ is infeasible to obtain in an actual interaction with the verifier, one can construct the protocol such that it will be *easy* to obtain by the simulator. This will allow the simulator to produce a "real-looking" (and in particular accepting) proof, even though it does not get the witness as input.

Protocols following the FLS technique are usually constructed to ensure that using black-box access to the next-message function of the verifier (or in other words, using the power to "rewind" the verifier) it would be easy to obtain the trapdoor information σ (e.g., this is the case in [FS89, RK99, KP00]). Our protocol also uses the FLS technique but with a twist. We construct our protocol in such a way that our trapdoor information σ will simply be the *description* of the verifier's *next-message function* (i.e., the verifier's *code*). Thus a *non-black-box* simulator has (trivially) access to this trapdoor information. Note that because the verifier's next-message function may be a function that is hard to learn (e.g., a pseudorandom function) it may be very hard for a *black-box* simulator to obtain the trapdoor information. Indeed, our protocol will *not* be *black-box* zero-knowledge (as mentioned above every argument system for a non-trivial language satisfying Properties 1–5 *can not* be black-box zero-knowledge).

The techniques of Feige, Lapidot and Shamir [FLS99] allow to use as trapdoor information the witness for any **NP** language. However, it turns out that for our purpose of making the trapdoor information be the verifier's code, this is not sufficient. Loosely speaking, the problem is that the running time of the verifier is not a-priori bounded by any *fixed* polynomial. This problem is similar to a problem that Canetti, Goldreich and Halevi [CGH98] encountered previously, when they tried to construct a counter-example for the Random Oracle Methodology. We solve this problem in a similar way to [CGH98], using *universal arguments*. Loosely speaking, a universal argument system is a computationally-sound proof system for **NEXP**.[4] Using universal arguments we are able to extend the technique of [FLS99] and use as trapdoor information a witness for any $\mathbf{Ntime}(T(n))$ language for some *super-polynomial* function $T(\cdot)$ (e.g., $T(n) = n^{\log\log n}$).

[4]The name and definition of universal arguments is borrowed from the subsequent work by Barak and Goldreich [BG01]. Universal arguments are a hybrid of interactive CS proofs [Mic94] and arguments [BCC88]. Their construction follows [Kil92, Kil95] which is based on the multi-prover (**PCP**) proof system for **NEXP** [BFL91]. See Section 3 for more details.

1.3 Related Work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff in [GMR85]. Goldreich, Micali and Wigderson [GMW86] gave a zero-knowledge proof for any language in **NP**, and showed the wide applicability of such proofs to solving protocol problems. *Constant-round* zero-knowledge arguments and proofs for any language in **NP** were first presented by Feige and Shamir [FS89], Brassard , Crépeau and Yung [BCY89], and Goldreich and Kahan [GK96].

All of the above protocols utilized *black-box* simulators (see also [GO87]). Goldreich and Krawczyk [GK90] showed that no language outside of **BPP** has a constant-round Arthur-Merlin zero-knowledge proof or argument system with a *black-box* simulator. They also showed that no language outside of **BPP** has a *general* (i.e., not necessarily Arthur-Merlin) *three-round* zero-knowledge proof or argument system with a *black-box* simulator.

A non-black-box zero-knowledge argument was suggested by Hada and Tanaka [HT99]. However, they used a highly non-standard assumption that in itself was of a strong "reverse-engineering" flavor.

Some of our techniques (i.e., the use of CS proofs for trapdoor information) were first used by Canetti, Goldreich and Halevi [CGH98] for the purpose of constructing cryptographic schemes that are secure in the Random Oracle Model [BR93] but are *insecure* under any implementation of this model. CS proofs were defined and constructed by Kilian [Kil92, Kil95] and Micali [Mic94](see more discussion on Section 3).

Subsequent work. Following this work, Barak and Goldreich [BG01] were able to strengthen the results of this paper, and prove that a zero-knowledge protocol satisfying all of the Properties 1–5 exists under the assumption of standard collision-resistent hash function. That is, under the assumption that there exists a hash function that is collision-resistant against polynomial-sized circuits (instead of $T(n)$-sized circuits for a "nice" super-polynomial function $T(\cdot)$ such that $T(n) = n^{\log n}$).

Barak, Goldreich, Goldwasser and Lindell [BGGL01] used the results of this paper to obtain a new results in the *resettable* model of Canetti *et al.*[CGGM00]. They constructed a *resettably-sound* zero-knowledge argument. Their protocol also has a non-black-box simulator and they show that it is impossible to obtain such an argument using a black-box simulator. Using this argument they constructed a *resettable zero-knowledge argument of knowledge*. This argument of knowledge uses a *non-black-box knowledge extractor*. It was observed by [CGGM00] that it is impossible to obtain such an argument using a black-box extractor.

Barak and Lindell [BL02] used the results of this paper to obtain a zero-knowledge argument of knowledge with a *strict* probabilistic polynomial-time knowledge extractor. They also showed that *strict* probabilistic polynomial-time extraction and simulation is impossible when restricted to black-box techniques.

We discuss additional related works in the relevant sections.

1.4 Organization

Section 2 contains some notations and definitions of the basic cryptographic primitives that we use. Section 3 describes *universal arguments* which are one of our main tools.

The construction of our zero-knowledge protocol is described in three stages. In Section 4, we construct a zero-knowledge protocol satisfying Properties 2, 4 and 5 of the introduction. That is, we construct a protocol that is constant-round Arthur-Merlin and has a strict polynomial-time simulator. However, it will not be zero-knowledge with respect to auxiliary input (i.e., non-uniform

zero-knowledge). Rather, it will only be zero-knowledge with respect to verifiers whose strategy can be implemented by a *uniform* probabilistic polynomial-time Turing machine. In Section 4.4 we present an alternative construction for a *uniform-verifier generation protocol*, which is the main component in this construction. This alternative construction is somewhat simpler and more round-efficient.

In Section 5, we modify the protocol of Section 4 and obtain a protocol that is zero-knowledge with respect to *non-uniform* verifiers. In Section 6 we make yet another modification to obtain a protocol that remains zero-knowledge under bounded-concurrent composition.

Section 7 contains conclusions and open problems.

2 Preliminaries

2.1 Standard Notation

In this section we state for completeness some of the notations we use. Since this subsection contains only standard notations, the reader may want to skip to Section 2.2 and return to this subsection only if necessary.

Basic notation. We use the standard O-notations $(O, \Omega, o, \omega, \Theta)$. For a finite set $S \subseteq \{0,1\}^*$, we write $x \leftarrow_{\mathrm{R}} S$ to say that x is distributed uniformly over the set S. We denote by U_n the uniform distribution over the set $\{0,1\}^n$. A function $\mu(\cdot)$, where $\mu : \mathbb{N} \to [0,1]$ is called *negligible* if $\mu(n) = n^{-\omega(1)}$ (i.e., $\mu(n) < \frac{1}{p(n)}$ for all polynomials $p(\cdot)$ and large enough n's). We say that an event happens with *overwhelming* probability if it happens with probability $1 - \mu(n)$ for some negligible function $\mu(\cdot)$.

Languages and witnesses. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We define $L(R) \stackrel{def}{=} \{x \mid \exists y \text{ s.t. } (x,y) \in R\}$. If $x \in L(R)$ and y is a string such that $(x,y) \in R$ then we say that y is a *witness* for the fact that $x \in L(R)$. We denote the set of witnesses to x by $R(x)$. That is, $R(x) = \{y \mid (x,y) \in R\}$. Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We say that $L \in \mathbf{Ntime}(T(n))$ if there exists a relation R such that $L = L(R)$ and a Turing machine M such that on input (x,y), the machine M runs for at most $T(|x|)$ steps and output 1 if $(x,y) \in R$ and 0 otherwise.

Algorithms. If A is a probabilistic algorithm, then we let $A(x;r)$ denote the output of A on input x and random-tape r. We let $A(x)$ denote the random variable that represents $A(x;r)$ for a randomly chosen r of appropriate length. We identify algorithms with their description as either Turing machines or boolean circuits. For example, $A(A)$ denotes the output of the algorithm A, when given as input the string that describes A.

Computational indistinguishability. Let X and Y be random variables over $\{0,1\}^n$ and let $s \geq n$. We say that X and Y are *indistinguishable by s-sized circuits* if for every circuit D of size s, it holds that $|\Pr[D(X)=1] - |\Pr[D(Y)=1]| < \frac{1}{s}$. A *probability ensemble* is a sequence $\{X_i\}_{i \in I}$ of random variables, where I is an infinite subset of $\{0,1\}^*$ and X_i ranges over $\{0,1\}^{p(|i|)}$ for some polynomial $p(\cdot)$. We say that two probability ensembles $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are *computationally indistinguishable*, denoted by $\{X_i\}_{i \in I} \equiv_{\mathrm{C}} \{Y_i\}_{i \in I}$, if for every polynomial $p(\cdot)$ and every sufficiently large i, X_i and Y_i are indistinguishable by $p(|i|)$-sized circuits. An equivalent formulation is that $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are computationally indistinguishable if there exists a negligible function μ :

$\mathbb{N} \to [0,1]$ such that X_i and Y_i are indistinguishable by $\frac{1}{\mu(|i|)}$-sized circuits. We will sometimes abuse notation and say that the two random variables X_i and Y_i are computationally indistinguishable, denoted by $X_i \equiv_{\mathrm{C}} Y_i$, when each of them is a part of a probability ensemble such that these ensembles $\{X_i\}_{i\in I}$ and $\{Y_i\}_{i\in I}$ are computationally indistinguishable. We will also sometimes drop the index i from a random variable if it can be inferred from the context. In most of this cases, the index i will be of the form 1^n where n is called the *security parameter*. We will use the following basic facts regarding computational indistinguishability:

Proposition 2.1. *Let M be a probabilistic polynomial-time Turing machine. If $\{X_i\}_{i\in I}$ and $\{Y_i\}_{i\in I}$ are computationally indistinguishable then so are $\{M(X_i)\}_{i\in I}$ and $\{M(Y_i)\}_{i\in I}$.*

We call a probability ensemble $\{X_i\}_{i\in I}$ *efficiently sampleable* if there exists a probabilistic polynomial-time Turing machine S and a polynomial $p(\cdot)$ such that $X_i = S(i, U_{p(|i|)})$. We have the following two facts about efficiently sampleable ensembles:

Proposition 2.2. *Let $\{X_i\}_{i\in I}$, $\{Y_i\}_{i\in I}$, $\{U_i\}_{i\in I}$ and $\{T_i\}_{i\in I}$ be four efficiently sampleable probability ensembles. If $\{X_i\}_{i\in I}$ is computaionally indistinguishable from $\{Y_i\}_{i\in I}$ and $\{U_i\}_{i\in I}$ is computationally indistinguishable from $\{T_i\}_{i\in I}$ then the ensemble $\{(X_i, U_i)\}_{i\in I}$ is computationally indistinguishable from the ensemble $\{(Y_i, T_i)\}_{i\in I}$, where (X_i, U_i) (resp. (Y_i, T_i)) represents a pair (x, u) (resp. (y, t)) such that x (resp. y) is sampled from X_i (resp. Y_i) and u (resp. t) is independently sampled from U_i (resp. T_i).*

Proposition 2.3. *Let $\{X_i\}_{i\in I}$ and $\{Y_i\}_{i\in I}$ be two efficiently sampleable and computationally indistinguishable probability ensembles. Let $p(\cdot)$ be some polynomial. Then, the ensembles $\{(X_i^{(1)}, \ldots, X_i^{(p(|i|))})\}_{i\in I}$ and $\{(Y_i^{(1)}, \ldots, Y_i^{(p(|i|))})\}_{i\in I}$ are computationally indistinguishable, where $(X_i^{(1)}, \ldots, X_i^{(p(|i|))})$ (resp. $(Y_i^{(1)}, \ldots, Y_i^{(p(|i|))})$) represents $p(|i|)$ independent copies of X_i (resp. Y_i).*

2.2 Uniform and non-uniform adversaries

Our standard way to model an efficient adversary strategy will be a family of polynomial-sized circuits. However, we also consider other models such as $T(n)$-sized circuits for a *super-polynomial* function $T : \mathbb{N} \to \mathbb{N}$ (e.g., $T(n) = n^{\log n}$). We will also consider *uniform* adversaries. That is, adversaries that are described using probabilistic polynomial-time Turing machines. Further more, we will also consider a "hybrid model" of adversaries with *bounded non-uniformity*. Such adversaries are described by a probabilistic polynomial-time Turing machine that on inputs of size n gets an advice string of length $l(n)$ where $l : \mathbb{N} \to \mathbb{N}$ is some fixed function that is polynomially related to n. We stress that the running time of such adversaries may be any polynomial and so in particular may be larger than $l(n)$.

Notes:

- It is well known that in almost all cryptographic settings, when considering *non-uniform* adversaries, without loss of generality, we can restrict ourselves to *deterministic* adversaries. This is justified by "hardwiring" into the description of the adversary a "best" random tape which exists by an averaging argument (e.g., see [Gol01b, Sec. 1.3.3]). This is not necessarily the case when talking about uniform algorithms, or algorithms with bounded non-uniformity. However, we will see in Section 4 that we can still reduce a probabilistic adversary into a deterministic adversary in the case of bounded non-uniformity. We will do so first reducing the size of the adversary's random tape using a pseudorandom generator and then"hardwiring" into the description of the adversary a "best" random tape (which will now be short enough so that the adversary will still have bounded non-uniformity).

- In most cryptographic works, a proof of security for *uniform* adversaries, can be extended to yield a proof of security for *non-uniform* adversaries (under appropriate complexity assumptions). However, this is only because these works use *black-box* reductions in their proof of security. In contrast, since in this work we will utilize *non-black-box* techniques, our proofs of security against uniform adversaries do *not* extend *automatically* into proofs of security against non-uniform adversaries. Indeed, we will need to introduce some additional ideas in order to extend the results of Section 4, which address with uniform adversaries, to the case of *non-uniform* adversaries (see Section 5).

2.3 Protocols and interaction

Next-message function, view and transcript. We consider only two-party protocols. A *view* of a party in particular step of a protocol contains the public input, the party's private input and random-tape, and the list of messages that this party received up to this step. The *next-message function* of a party is a function that maps a view in a particular step of the protocol to the party's message in the next step. An *interactive algorithm* is an algorithm that computes the next-message function of a party. If I is an interactive algorithm, and v is a view of the protocol up to a particular step s, then the *residual algorithm I with respect to the view v* is the algorithm I with the view v "hardwired in". That is, this is a function that takes a list of messages sent after the step s, and computes I's response to them, assuming that the messages sent up to step s are as described in v. A *transcript* of a protocol consists of the public input and the list of all messages exchanged in the protocol (but does not include the parties' private inputs and random-tapes). That is, the transcript of an execution is the public information seen by both parties in the execution. For any interactive algorithm I, and any view v of I, the transcript τ of the protocol can always be computed from the view v. We say in this case that the transcript τ is *contained* in the view v.

Notation. If A and B are interactive algorithms, then $\langle A(x,y), B(x,z)\rangle$ is a random variable representing the execution of A and B on public input x when A's private input is y and B's private input is z. We let $\mathsf{view}_A\langle A(x,y), B(x,z)\rangle$ denotes A's view in this execution, and let $\mathsf{out}_A\langle A(x,y), B(x,z)\rangle$ denotes A's output at the end of the execution. We define view_B and out_B in the symmetric way. We let $\mathsf{transcript}\langle A(x,y), B(x,z)\rangle$ denote the transcript of the execution.

Prescribed versus cheating parties. For a two-party protocol, the *prescribed* or *honest* strategy for a party is the strategy it should use if it follows the protocol. However, we will usually analyze the execution of the protocol also when one of the parties may *not* be following its prescribed strategy. We will sometimes call such a party "cheating". Part of the description of any protocol are conventions on who is the first party to send a message and on the length of each message. We assume, without loss of generality, that even "cheating" parties follow these conventions (e.g., if a party sends a message that is too short or too long then we assume that it is padded or truncated to the proper length).

2.4 Interactive proof and argument systems

An *interactive proof* [GMR85] is a two-party protocol, where one party is called the *prover* and the other party is called the *verifier*. We use the following definition:

Definition 2.4 (Interactive proofs)**.** An interactive protocol (P, V) is called an *interactive proof system* for a language L if the following conditions hold.

1. *(Efficiency)* The number and total length of messages exchanged between P and V are polynomially bounded and V is a probabilistic polynomial-time machine.

2. *(Perfect Completeness)* If $x \in L$, then $\Pr[(P,V)(x) = 1] = 1$.

3. *(Soundness)* If $x \notin L$, then for any P^*, $\Pr[(P^*,V)(x) = 1] \leq \mu(|x|)$ where $\mu(\cdot)$ is some negligible function.

An interactive proof system is called *Arthur-Merlin* [BM88] (a.k.a. *public-coins*) if the verifier's messages consist only of random strings and acceptance is computed as a deterministic polynomial-time function of the interaction's transcript. An interactive proof system that is not Arthur-Merlin is called *private-coins*.

The number of *rounds* in an interactive proof is the total number of messages exchanged in the interaction (that is, both prover messages and verifier messages).

Let $L \in \mathbf{NP}$, a proof system for L has an *efficient prover strategy* if the completeness property of the system can be satisfied by a probabilistic polynomial-time algorithm that when proving that $x \in L$ gets as auxiliary input a witness to this fact.

Let $L \in \mathbf{NP}$, an *interactive argument* for L [BCC88] is the following variation on the definition of an interactive proof:

- The soundness requirement is relaxed to quantify only over prover strategies P^* that can be implemented by a polynomial-sized circuit.

- The system is required to have an efficient prover strategy.

Honest verifier conventions. We say that an execution of a two-party protocol is *completed successfully* if no party aborted. For a proof (or argument) system, we assume that if a verifier rejects the proof then it aborts, and so an execution of a proof system is completed successfully only if the verifier accepts. If a proof system uses another proof system as a subprotocol, then we assume that in case the verifier of the subprotocol rejects, then execution is aborted and thus the verifier for the larger proof system will also reject. If we do not specify the verifier's acceptance condition, then it is assumed that the verifier accepts if and only if all the subprotocols were completed successfully.

2.4.1 Zero-knowledge

Informally, we say that a proof/argument system for L is *zero-knowledge* [GMR85] if after seeing a proof that $x \in L$, the verifier does not learn anything about x that it didn't know before. We require this to hold even if the verifier does not follow its prescribed strategy for the proof system, as long as its strategy can be implemented by an efficient algorithm. This is formalized by requiring that there exists an efficient algorithm called the *simulator*, that given the verifier's prior knowledge (i.e., the string x, the verifier's strategy and private inputs) can compute (or closely approximate) the verifier's state after viewing a proof that $x \in L$. The formal definition is below. Note that we define two variants of zero-knowledge: *uniform* and *non-uniform*, based on the classes of algorithms that we allow the cheating verifier to employ. Note also that in our definition we require the simulator to be *universal*. That is, the simulator is a single algorithm for all possible verifier's strategies, that gets the strategy as an additional input.

Definition 2.5 (Zero-knowledge)**.** Let $L = L(R)$ be some language and let (P,V) be an interactive argument for L. We say that (P,V) is *(non-uniform) zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial-sized circuit family $\{V_n^*\}_{n\in\mathbb{N}}$ and every

sequence $\{(x_n, y_n)\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n \cap L$ and $(x_n, y_n) \in R$ the following two probability ensembles are computationally indistinguishable:

- $\left\{\mathsf{view}_{V_n^*}\langle P(x_n, y_n), V_n^*(x_n)\rangle\right\}_{n\in\mathbb{N}}$
  and
- $\left\{S(V_n^*, x_n)\right\}_{n\in\mathbb{N}}$

We say that (P, V) is *uniform zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial $t(\cdot)$, every probabilistic $t(n)$-time Turing machine V^* every sequence $\{(x_n, y_n)\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n \cap L$ and $(x_n, y_n) \in R$ the following two probability ensembles are computationally indistinguishable:

- $\left\{\mathsf{view}_{V^*}\langle P(x_n, y_n), V^*(x_n)\rangle\right\}_{n\in\mathbb{N}}$
  and
- $\left\{S(V^*, 1^{t(n)}, x_n)\right\}_{n\in\mathbb{N}}$

In either case we say that S is a *black-box* simulator if the only use it makes of its first input (i.e., V^*) is to call it as a subroutine.

We remark that we use a somewhat stronger definition than the standard definition of uniform zero-knowledge [Gol93]: we allow both the input generation and the distinguisher to be non-uniform.

2.4.2 Witness indistinguishability

Like zero-knowledge, a *witness-indistinguishable* proof/argument system [FS90] also guarantees some secrecy property to the prover, but it is a weaker property than zero-knowledge. In a witness-indistinguishable proof system we do not require that the verifier does not learn anything about x after seeing a proof that $x \in L$. Rather, we only require that if both y and y' are witnesses that $x \in L$, then it is infeasible for the verifier to distinguish whether the prover used y or y' as auxiliary input. The formal definition is below: (we only make the definition in the non-uniform setting)

Definition 2.6. Let $L = L(R)$ be some language and let (P, V) be an interactive argument for L. We say that (P, V) is *witness-indistinguishable* if for every polynomial-sized circuit family $\{V_n^*\}_{n\in\mathbb{N}}$ and every sequence $\{(x_n, y_n, y'_n)\}_{n\in\mathbb{N}}$, where $x_n \in \{0,1\}^n$ and $(x_n, y_n), (x_n, y'_n) \in R$ the following two probability ensembles are computationally indistinguishable:

- $\left\{\mathsf{view}_{V_n^*}\langle P(x_n, y_n), V_n^*(x_n)\rangle\right\}_{n\in\mathbb{N}}$
  and
- $\left\{\mathsf{view}_{V_n^*}\langle P(x_n, y'_n), V_n^*(x_n)\rangle\right\}_{n\in\mathbb{N}}$

Witness indistinguishability is a weaker property than zero-knowledge. That is, if a protocol is zero-knowledge then it is also witness-indistinguishable [FS90]. Also, under standard assumptions, there exist protocols that are witness-indistinguishable but *not* zero-knowledge (as a trivial example, note that for any language L where each $x \in L$ has a single witness, the trivial **NP** proof system of sending the witness is witness-indistinguishable). Unlike zero-knowledge, witness indistinguishability is known to be closed under concurrent (and in particular parallel) composition [Fei90]. Using this fact, parallel repetition of the "basic protocol" of [GMW86], yields the following theorem:

Theorem 2.7 ([FS90])**.** *Suppose that one-way functions exist. Then, for every language $L \in$ **NP** there exist a constant-round Arthur-Merlin witness-indistinguishable proof system for L.*

2.4.3 Proofs of knowledge

In a proof/argument system, the prover convinces the verifier that some string x is a member of a language L. In a proof/argument *of knowledge* [FFS87, BG93, GMR85, TW87] the prover should convince the verifier that it also *knows* a witness to the fact that $x \in L$. This is formalized by requiring that if the verifier is convinced with some probability p by some (possibly cheating) prover strategy, then by applying an efficient algorithm, called the *knowledge extractor* to the cheating prover's strategy and private inputs, it is possible to obtain a witness to the fact that $x \in L$, with probability (almost equal to) p. The formal definition is below:

Definition 2.8. Let $L = L(R)$ and let (P, V) be an argument system for L. We say that (P, V) is an *argument of knowledge* for L if there exists a probabilistic polynomial-time algorithm E (called the *knowledge extractor*) such that for every polynomial-sized prover strategy P^* and for every $x \in \{0,1\}^n$

$$\Pr[E(P^*, x) \in R(x)] \geq \Pr[\mathsf{out}_V\langle P^*(x), V(x)\rangle = 1] - \mu(n)$$

where $\mu : \mathbb{N} \to [0,1]$ is a negligible function.

We will sometimes consider a generalized definition where we allow both the cheating prover P^* and the extractor E to run in time $T(n)^{O(1)}$ where $T(\cdot)$ is some *super-polynomial* function.

We say that an argument of knowledge has a *black-box* extractor if the knowledge extractor algorithm E uses its first input (i.e., P^*) as a black-box subroutine (i.e., oracle). We note that in this paper we will only use *black-box* extractors.

2.5 Cryptographic primitives

2.5.1 Pseudorandom generators and functions

We will use the standard definitions of pseudorandom generators [**?**, Yao82] and functions [GGM86]. That is, we say that a determinstic polynomial-time computable function PRG (where $|PRG(s)| = n(|s|)$ for some function $n(\cdot)$ such that $n(l) > l$) is a *pseudorandom generator* if the random variable $\mathrm{PRG}(U_l)$ is computationally indistinguishable from $U_{n(l)}$. We say that an efficiently evaluatable function ensemble $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ is a *pseudorandom function ensemble* (where $f_\alpha : \{0,1\}^{|\alpha|} \to \{0,1\}^{|\alpha|}$) if it is infeasible for a polynomial-sized circuit to distinguish between oracle access to f_{U_n} and oracle access to a function H chosen randomly of the set of functions from $\{0,1\}^n$ to $\{0,1\}^n$.

2.5.2 Commitment schemes

In this paper we will use statistically-binding commitment schemes. A non-interactive statistically binding commitment scheme can be constructed based on any one-way permutation [Blu82]. Naor [Nao89] showed a construction of an interactive (two-round) statistically-binding commitment scheme based on any one-way function . For simplicity of presentation we will define commitment schemes in this paper to be non-interactive; however, all of our results still hold if the non-interactive commitment is replaced by Naor's construction.

Definition 2.9. A polynomial-time computable function $\mathsf{Com} : \{0,1\} \times \{0,1\}^n \to \{0,1\}^{p(n)}$ (where $p(\cdot)$ is some polynomial) is a *bit commitment scheme* if it satisfies the following properties:

Computational Hiding The random variables $\mathsf{Com}(0; U_n)$ and $\mathsf{Com}(1; U_n)$ are computationally indistinguishable.

Statistical Binding The supports of the above random variables are disjoint.

Notation. We denote by $\mathsf{Com}(b)$ the random variable $\mathsf{Com}(b; U_n)$. We define $\mathsf{Com}^{-1}(y) = b$ where b is the unique bit such that $y = \mathsf{Com}(b; r)$ for some r if such b exists; otherwise, we define $\mathsf{Com}^{-1}(y) = \perp$. One can convert a bit commitment scheme into a *string* commitment scheme by concatenating independent commitments for each of the input bits. Thus for $x = x_1 \ldots x_l \in \{0,1\}^l$ and $r = r^{(1)} \cdots r^{(l)} \in \{0,1\}^{nl}$ we define $\mathsf{Com}(x; r) = \mathsf{Com}(x_1; r^{(1)}) \cdots \mathsf{Com}(x_l; r^{(l)})$. Similarly, for such a string x we will denote by $\mathsf{Com}(x)$ the random variable $\mathsf{Com}(x; U_{nl})$ and define $\mathsf{Com}^{-1}(y) = x$ where x is the unique string such that $y = \mathsf{Com}(x; r)$ for some r if such x exists; otherwise, we define $\mathsf{Com}^{-1}(y) = \perp$.

2.5.3 Collision-resistent hash functions

We will use the following definition of collision-resistent hash functions [**?**]

Definition 2.10. Let $\mathcal{H} = \{h_\alpha\}_{\alpha \in \{0,1\}^*}$ be an efficiently computable function ensemble where $h_\alpha : \{0,1\}^{2\alpha} \to \{0,1\}^\alpha$. We say that $\mathcal{H}$ is *collision-resistent against $T(n)$-sized circuits* if for every circuit family $\{A_n^*\}_{n \in \mathbb{N}}$ where $|A_n^*| \leq T(n)$, it holds that

$$\Pr_{\alpha \leftarrow_R \{0,1\}^n}[A^*(\alpha) = (x, x') \text{ s.t. } x \neq x' \wedge h_\alpha(x) = h_\alpha(x')] < \frac{1}{T(n)}$$

2.6 Computational Assumptions

Throughout this paper we will make the assumption that there exists a family of hash functions that is collision-resistent against circuits of size $n^{\log n}$ where n is the security parameter. The choice of $n^{\log n}$ is somewhat arbitrary and in fact our result holds under the weaker assumption that there exist hash functions that are collision-resistent against $f(n)$-sized circuits for some function $f(\cdot)$ that is super-polynomial (i.e., $f(n) = n^{\omega(1)}$) and polynomial-time computable. Following this work, Barak and Goldreich [BG01] showed that our result holds also under the weaker (and more standard) assumption that that there exist hash functions that are collision-resistent against all polynomial-sized circuits.

3 Universal arguments

Universal arguments are a variant of (interactive) CS proofs, as defined and constructed by Micali [Mic94] and Kilian [Kil92]. Loosely speaking, a *universal argument* is an interactive argument of knowledge for proving membership in $\mathbf{Ntime}(T(n)) \supseteq \mathbf{NP}$ for a super-polynomial function $T(\cdot)$. Note all of the languages in **NP** can be reduced to a single language in $\mathbf{Ntime}(T(n))$ via a reduction that preserves the length of the instance. Therefore, an argument system for this "universal" $\mathbf{Ntime}(T(n))$ language allows us to use a single protocol to prove membership in *all* **NP** languages, rather than use a different protocol for each language; hence the name *universal arguments*.

In a subsequent work, Barak and Goldreich [BG01] constructed universal arguments under a weaker assumption than the one we use here (namely, they constructed universal arguments under the assumption that there exist hash functions that are collision-resistent against polynomial-sized circuits). The definition we present here is slightly different than the one of [BG01]. This is due to the fact that we use a stronger assumption, which enables us to use a slightly simpler definition.

Defining universal arguments. It will be convenient to define universal arguments for a single "universal" language. We define the following relation $R_{\mathcal{U}}$. We say that $(\langle M, x, t\rangle, w) \in R_{\mathcal{U}}$, where M is a description of a Turing machine, x, w are strings and t is a number (represented in binary form), if M accepts (x, w) within t steps. We define $T_M(x, w)$ to be the number of steps made by M on input (x, w). Note that if $(\langle M, x, t\rangle, w) \in R_{\mathcal{U}}$ then $T_M(x, w) \leq t$. Note that under suitable encoding $|\langle M, x, t\rangle| = |x| + \log t + O(1)$. Let $L_{\mathcal{U}} \stackrel{def}{=} L(R_{\mathcal{U}})$. Note that $L_{\mathcal{U}} \in \mathbf{Ntime}(2^n)$.

Let $T : \mathbb{N} \to \mathbb{N}$ be a super-polynomial function (e.g. $T(n) = n^{\log\log n}$). We define the relation ${R_{\mathcal{U}}}^{T(n)}$ as follows: $(\langle M, x, t\rangle, w) \in {R_{\mathcal{U}}}^{T(n)}$ if $(\langle M, x, t\rangle, w) \in R_{\mathcal{U}}$ and $t \leq T(|\langle M, x, t\rangle|)$. We let ${L_{\mathcal{U}}}^{T(n)} \stackrel{def}{=} L({R_{\mathcal{U}}}^{T(n)})$. Note that ${L_{\mathcal{U}}}^{T(n)} \in \mathbf{Ntime}(T(n)^{O(1)})$. Also note that ${R_{\mathcal{U}}}^{2^n} = R_{\mathcal{U}}$ and ${L_{\mathcal{U}}}^{2^n} = L_{\mathcal{U}}$.

A *universal argument system for* $\mathbf{Ntime}(T(n))$ is basically an argument system of knowledge for ${L_{\mathcal{U}}}^{T(n)}$ with the following modifications:

- We make a stronger requirement on the honest prover's efficiency. Rather than only placing an upper bound on the prover's complexity on all inputs of a specified length, we require that the prover's running time on public input $\langle M, x, t\rangle$ and auxiliary input w will be polynomial in $T_M(x, w)$ and so in particular it should be polynomial in t.
- Because the length of the witness w may be $T(n)$, polynomial-time may not be sufficient to write it down. Thus, we allow the knowledge extractor to run in time $T(n)^{O(1)}$. We remark that this is a weaker condition than the one used by Barak and Goldreich [BG01]. They require the knowledge extractor to run in polynomial-time (but allow it to output an implicit representation of the witness).[5]

Below is the formal definition:

Definition 3.1 (universal argument)**.** A *universal-argument system for* $\mathbf{Ntime}(T(n))$ is a pair of strategies, denoted (P, V), that satisfies the following properties:

[5]We remark that superficially, our proof of knowledge condition appears to be stronger than the condition of [BG01] because they allow the knowledge extractor to output a witness in probability that is only polynomially related to the acceptance probability. However, once one allows the knowledge extractor to run in time $T(n)$ then it is possible to verify the witness, and amplify the acceptance probability. Thus, their condition is indeed stronger.

Efficient verification: V is a probabilistic polynomial-time interactive algorithm.

Completeness by a relatively-efficient prover: For every $y = \langle M, x, t\rangle \in {L_{\mathcal{U}}}^{T(n)}$ and $w \in {R_{\mathcal{U}}}^{T(n)}(y)$,

$$\Pr[\mathsf{out}_V\langle P(y,w), V(y)\rangle = 1] \;=\; 1$$

Furthermore, there exists a polynomial $p(\cdot)$ such that the total time spent by $P(y,w)$,is at most $p(T_M(x,w))$ (which in turn is at most $p(t)$).

Computational Soundness: For every polynomial-size circuit family $\{P_n^*\}_{n\in\mathbb{N}}$, and every $y = \langle M, x, t\rangle \in \{0,1\}^n \setminus {L_{\mathcal{U}}}^{T(n)}$,

$$\Pr[\mathsf{out}_V\langle P^*(y), V(y)\rangle = 1] < \mu(n)$$

where $\mu : \mathbb{N} \to [0,1]$ is a negligible function.

Proof of Knowledge: There exists a probabilistic $T(n)^{O(1)}$-time oracle machine E such that for every polynomial-sized circuit family $\{P_n^*\}_{n\in\mathbb{N}}$ and every string $y = \langle M, x, t\rangle \in \{0,1\}^n$,

$$\Pr[E^{P_n^*}(y) \in {R_{\mathcal{U}}}^{T(n)}(y)] \geq \Pr[\mathsf{out}_V\langle P^*(y), V(y)\rangle = 1] - \mu(n)$$

where $\mu : \mathbb{N} \to [0,1]$ is a negligible function.

We have the following theorem:

Theorem 3.2 (based on [Kil92, Mic94])**.** *Suppose that there exists a hash function ensemble that is collision-resistent against circuits of size $n^{\log n}$. Then there exists a universal argument system for* $\mathbf{Ntime}(n^{\log\log n})$*. Furthermore, this system has the following additional properties:*

1. *The system is constant-round and Arthur-Merlin.*
2. *The system is witness-indistinguishable.*

Moreover, for every $\epsilon > 0$, there exists such a system with total communication complexity of m^ϵ, where m is the instance length (i.e., $m = |\langle M, x, t\rangle|$).

The choice of $\mathbf{Ntime}(n^{\log\log n})$ is quite arbitrary and was taken for simplicity of presentation. We can replace it by $\mathbf{Ntime}(f(n))$ for any function $f(\cdot)$ such that $f(n) = n^{o(\log n)}$. Following this work, Barak and Goldreich [BG01] proved a stronger theorem in which they showed that a universal argument for the language $L_{\mathcal{U}}$ (and hence for **NEXP**) exists under the standard assumption of hash functions that are collision-resistent against polynomial-sized circuits.

Because the proof of Theorem 3.2 closely follows the proofs in [Kil92, Mic94], and because Theorem 3.2 has been superseded by the results of [BG01], we omit its proof here. See Appendix A for a proof sketch.

We remark that it is very unlikely that it will be possible to strengthen the soundness condition of the universal argument system to *statistical soundness* that holds against even *inefficient* prover strategies. This is because if L has an interactive proof with a polynomial-time verifier then $L \in \mathbf{PSPACE}$, while it is believed that $\mathbf{Ntime}(n^{\log\log n}) \not\subseteq \mathbf{PSPACE}$.

4 A Uniform Zero-Knowledge Argument

In this section we construct a *constant-round Arthur-Merlin* argument system for **NP** that is zero-knowledge for *uniform* verifiers (i.e., verifiers whose strategy is implemented by a Turing machine without advice). The protocol of this section will utilize a *non-black-box* simulator that runs in *strict* probabilistic polynomial-time. This protocol falls short of satisfying all the properties 1–5 stated in Section 1.1 because it is only zero-knowledge against *uniform* verifiers and we do not know whether or not it remains zero-knowledge under bounded concurrent composition. However, it does illustrate the main ideas of our construction.

4.1 FLS-type protocols

In our construction, we use a general technique that has been used before in the design of zero-knowledge protocols. We call this technique the *FLS technique*, since it was introduced in a paper by Feige, Lapidot and Shamir [FLS99].

The FLS technique allows to reduce the problem of constructing a zero-knowledge proof (or argument) system to the problem of constructing two simpler objects: a *witness-indistinguishable* (WI) proof/argument system and (what we call here) a *generation protocol*. Witness-indistinguishable proof and argument systems are described in Section 2.4.2. Generation protocols are defined later on (See Definition 4.1).

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	w x ↓ ↓ P V
Steps P,V1.x (Generation protocol): Prover and verifier engage in a *generation protocol* GenProt . We denote the transcript of the execution by τ.	1^n ↓ GenProt ↓ τ
Steps P,V2.x (WI Proof): Prover proves to verifier using its auxiliary input w via a witness-indistinguishable (WI) proof/argument system that either $x \in L$ or $\tau \in \Lambda$, where Λ is a fixed language (which is part of the protocol's specification).[6]	w x, τ ↓ ↓ *WI-proof* $x \in L$ **or** $\tau \in \Lambda$ ↓ 0/1
The verifier accepts if the WI proof of the second stage is completed successfully (i.e., if the verifier algorithm for the WI proof accepts).	

The right column contains a schematic description of the protocol as defined in the left column.

Figure 1: A generic FLS-type zero-knowledge protocol

We call a zero-knowledge protocol that is constructed using the FLS technique an *FLS-type* protocol. Figure 1 describes a generic FLS-type zero-knowledge protocol. The general outline of such a protocol is that when proving some statement of the form "$x \in L$" first the prover and the verifier engage in a generation protocol. Then the prover proves to the verifier using a WI system that *either* the statement "$x \in L$" is true, *or* a different statement about the transcript of

[6] Formally, the prover proves that $\langle x, \tau \rangle \in L'$ where the language L' is defined as follows: $\langle x, \tau \rangle \in L'$ if $x \in L$ or $\tau \in \Lambda$.

the generation protocol is true. To obtain a specific protocol one needs to specify the generation protocol (GenProt) to be used in Steps P,V1.x, the language Λ, and the WI proof (or argument) to be used in Steps P,V2.x. We stress that there are also *black-box* zero-knowledge arguments that use the FLS technique (e.g., [RK99, KP00]).

Note that the generation protocol does *not* take the statement x as an input, and so honest parties do not use x in computing their strategies for the first phase. (Although cheating parties may choose to do so.) Intuitively, one may think of the generation protocol as a game that the prover and verifier play. The prover's objective in this game is to make the transcript τ of the protocol in the language Λ, while the verifier's objective is to ensure that τ will *not* be in Λ. By the way an FLS-type protocol is set up, if the prover "wins" in the generation phase, then he doesn't need to prove that $x \in L$ in the second phase. Thus, for the proof to be sound, it is important that an honest verifier will win this game with high probability, no matter what strategy the prover may use. However, we will need to require some additional properties from the generation protocol in order to make it useful in this setting.[7]

Defining generation protocols. We now turn to formally defining what is a generation protocol. Our definition is motivated by our intended application. Therefore, we make requirements from a generation protocol that will ensure that when a generation protocol is plugged into the generic construction of Figure 1, the result would be a zero-knowledge proof or argument system. We define a *uniform-verifier* generation protocol since in this section we are only interested in obtaining a protocol that is zero-knowledge against verifiers whose strategy can be implemented by a *uniform* probabilistic polynomial-time Turing machine. The formal definition follows:

Definition 4.1 (Uniform-verifier generation protocol)**.** Let GenProt be a two-party protocol where we call one party the *prover* and the other party the *verifier*. Let $\Lambda \subseteq \{0,1\}^*$ be some language in $\mathbf{Ntime}(T(n))$ for some (polynomial-time computable) function $T : \mathbb{N} \to \mathbb{N}$ (e.g., $T(n) = n^{\log\log n}$ or $T(n) = n^3$). We say that GenProt is a *(uniform-verifier) generation protocol* (with respect to the language Λ) if it satisfies the following two requirements:

Soundness (This requirement ensures that the protocol that GenProt will be plugged into will be sound.) Let τ denote the transcript of the execution of GenProt. If the verifier follows its prescribed strategy then, regardless of the prover's (efficient or inefficient) strategy, it holds that $\Pr[\tau \in \Lambda] < \mu(n)$ for some negligible function $\mu : \mathbb{N} \to [0,1]$.

Simulation of uniform verifiers (This requirement ensures that the protocol that GenProt will be plugged into will be zero-knowledge against uniform verifiers.) There exists a *simulator* S_{GenProt} that satisfies the following:

Let V^* be an interactive strategy for the verifier that runs in polynomial-time and can be described using less than $2n$ bits where n is the security parameter. Then on input the description of V^*, S_{GenProt} runs for time polynomial in the running time of V^* and outputs a pair (v, σ) such that:

1. v is computationally indistinguishable from the view of V^* in an execution of GenProt with the prescribed prover algorithm.

[7] In particular, it will have the property, useful for demonstrating that the larger protocol is zero-knowledge, that if the prover knows the strategy and random tape of the verifier then he can always win the game.

2. Let τ denote the transcript that is contained in the view v. Then it holds that $\tau \in \Lambda$ and σ is a witness to this fact. Furthermore, we require that the time to verify that σ is a witness for τ is polynomial in the running time of V^*.[8]

Note that the two requirements together imply that Λ is a hard language. This is because in a real execution with the honest verifier it is almost always the case that the transcript τ is not in Λ while in the computationally indistinguishable simulated execution it is always in Λ. Note also that the simulator S_{GENPROT} is given the description of V^* as input and so may possibly make a non-black-box use of this description.

We can now prove the main theorem that we need about the FLS technique:

Theorem 4.2. *Let* GENPROT *be a generation protocol with respect to an* $\mathbf{Ntime}(T)$ *language* Λ *(where* $T : \mathbb{N} \to \mathbb{N}$ *is a polynomial-time computable function). Let* WIPROT*be a WI proof or argument system for* $\mathbf{NP} \cup \mathbf{Ntime}(T)$ *languages. Let* L *be an* **NP** *language and let* FLSPROT *be the argument for* L *that is the result of plugging in* GENPROT *and* WIPROT *into the construction of Figure 1. Then* FLSPROT *is a uniform zero-knowledge argument for* L*.*

Note that we deliberately stated Theorem 4.2 in a way that allows to treat in a uniform way both the case that $\Lambda \in \mathbf{NP}$ (i.e., the case that $T(\cdot)$ is a polynomial) and the case that $\Lambda \in \mathbf{Ntime}(T) \backslash \mathbf{NP}$ for some super-polynomial function $T(\cdot)$. In the former case it is sufficient to use a standard WI proof system for **NP** such as the one of [FS90]. In the latter case one needs to use a WI *universal argument* (see Section 3). Note that previous FLS-type protocols used languages $\Lambda \in \mathbf{NP}$ but we will need to use $\Lambda \in \mathbf{Ntime}(T(\cdot))$ for some super-polynomial $T(\cdot)$.

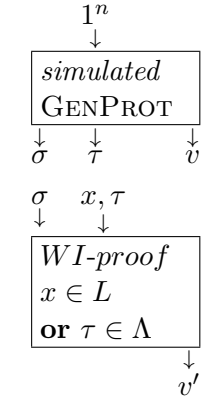
4.1.1 Proof sketch of Theorem 4.2

We only sketch the proof of Theorem 4.2 since it will be superseded by a non-uniform analogue (Theorem 5.2). To show that FLSPROT is a zero-knowledge argument one needs to show three properties: completeness, soundness, and zero-knowledge.

Completeness. Completeness follows from the fact that if the public input x is in L then the statement "$x \in L$ or $\tau \in \Lambda$" is true. Furthermore, the witness w for x can serve as a witness for this statement. Therefore completeness follows from the completeness with efficient prover condition of the WI proof/argument system. Note that since $L \in \mathbf{NP}$, if $x \in L$ then verifying that *either* $x \in L$ or $\tau \in \Lambda$ can be done in non-deterministic polynomial-time, even if deciding Λ takes non-deterministic *super-polynomial* time. This is because the witness w for x is also a witness for the combined statement.

Soundness. Suppose that $x \notin L$. Let τ denote the transcript of the first stage (Steps P,V1.x) of FLSPROT. By the soundness property of GENPROT with very high probability $\tau \notin \Lambda$. Therefore the combined statement "$x \in L$ or $\tau \in \Lambda$" will be false with very high probability and so the prover will not succeed in convincing the verifier by the soundness of the WI proof/argument system.

[8] This requirement is important when considering $\Lambda \in \mathbf{Ntime}(T(\cdot))$ for a super-polynomial function $T(\cdot)$.

Input: • $x \in \{0,1\}^n$: statement (simulate proof for "$x \in L$") • V^*: description of the Turing machine of verifier.	
Let V^{**} denote the verifier V^* with x "hardwired" into it. Note that since V^* is a Turing machine we can assume that the description of V^{**} takes at most $2n$ bits.[9]	
Simulated Steps P,V1.x (Simulated generation protocol): Let $(v, \sigma) \leftarrow S_{\text{GENPROT}}(V^{**})$ where S_{GENPROT} is the simulator for the generation protocol GENPROT. Let τ denote the transcript contained in the view v. We let V^{***} denote the *residual* verifier V^{**} with the view v hardwired in. **Simulated Steps P,V2.x (Honest WI Proof):** Run an execution of WIPROT between the verifier V^{***} and the honest prover algorithm for the WI system WIPROT the statement proved is "$x \in L$ or $\tau \in \Lambda$" using the witness σ. Let v' denote V^{***}'s view in this execution.	
Output the combined view (v, v') of the two stages	

This is a non-interactive algorithm. The right side contains a schematic description of the steps simulated in the left side.

Algorithm 4.3. A simulator for the FLS-type protocol FLSPROT.

Uniform zero-knowledge. To show that FLSPROT is zero-knowledge against uniform verifiers one should exhibit a simulator. Algorithm 4.3 is such a simulator. The simulator's operation can be summarized as follows: it uses the simulator S_{GENPROT} of the generation protocol GENPROT to obtain both a simulation v for the first stage along with a witness σ that $\tau \in \Lambda$ where τ is the transcript that v contains. Then, it uses the *honest prover algorithm* of the WI system WIPROT to prove the true statement "$x \in L$ or $\tau \in \Lambda$", while using the witness σ as auxiliary input to the prover algorithm of WIPROT. The first stage (running the simulator S_{GENPROT}) can certainly be done in time that is polynomial in the running time of V^*. The second step (running the honest prover algorithm) can be done in time that is a fixed polynomial in the size of the statement if $\Lambda \in \mathbf{NP}$. However, even if $\Lambda \notin \mathbf{NP}$, this step can still be performed in time polynomial in the time to verify that σ is a witness that $\tau \in \Lambda$ (using the completeness with efficient property of universal arguments, see Section 3). This is polynomial in the running time of V^* by Item 2 in the uniform-verifier simulation condition of GENPROT. Item 1 of the uniform-verifier simulation condition of GENPROT, along with the witness indistinguishability property of WIPROT, ensure that the output of our simulator will indeed be computationally indistinguishable from the view of the verifier in a real interaction.

[9] Note that we do not need to assume that V^* is a completely uniform Turing machine but only that its description is at most n-bits long.

Public input: 1^n: security parameter	1^n ↓ P V
Step P1 (Commitment to "junk"): Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(0^{3n})$ and sends z to the verifier.	$z = \mathsf{Com}(0^{3n})$ →
Step V2 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$r \leftarrow_{\mathrm{R}} \{0,1\}^n$ ←
The transcript of the protocol is the pair $\tau = (z, r)$.	

Protocol 4.4. A uniform-verifier generation protocol

4.2 A uniform-verifier generation protocol

Now that we have described the FLS technique we see that to describe our zero-knowledge protocol we should only specify the two components used (i.e., the WI system and generation protocol). Since we want the zero-knowledge to have a constant number of rounds and to be of the Arthur-Merlin type we must ensure that both components are indeed constant-round and Arthur-Merlin. For the WI system we will use the constant-round Arthur-Merlin WI universal argument described in Section 3. By Theorem 3.2, under our assumptions, there exists such a system for $\mathbf{Ntime}(n^{\log\log n})$. Thus, our main challenge is to construct a constant-round Arthur-Merlin generation protocol with respect to some language $\Lambda \in \mathbf{Ntime}(n^{\log\log n})$. We construct such a generation protocol now. We remark that a reader that just wants to get the flavor of our techniques may want to look at Section 4.4, where we present a somewhat simpler generation protocol. However, the current protocol generalizes more easily to the non-uniform case, which is why we choose to focus on it.

Protocol 4.4 is our uniform-verifier generation protocol. It consists of two rounds where in the first message the prover sends a commitment to a "junk" string (i.e., 0^{3n}) and in the second message the verifier sends a random string of length n. However, to fully specify the generation protocol one needs to specify the language Λ, which is what we do next.

Definition of the language Λ. We shall now specify the language Λ. Recall that for a string y, $\mathsf{Com}^{-1}(y)$ denotes the unique x such that y is a commitment to x or $\perp$ if no such x exists. That is $x = \mathsf{Com}^{-1}(y)$ if there exists s such that $\mathsf{Com}(x; s) = y$. We define Λ in the following way: let $\tau = (z, r)$ is in Λ iff on input z, the Turing machine described by $\mathsf{Com}^{-1}(z)$ halts and outputs r within $|r|^{\log\log|r|/5}$ steps.[10] (If $\mathsf{Com}^{-1}(z) = \perp$ or $\mathsf{Com}^{-1}(z)$ does not describe a valid Turing machine then $\tau = (z, r) \notin \Lambda$.) In other words, Λ is defined as follows:

$$(z, r) \in \Lambda \iff \Pi(z) \text{ outputs } r \text{ within } |r|^{\log\log|r|/5} \text{ steps, where } \Pi = \mathsf{Com}^{-1}(z)$$

As a first observation, note that $\Lambda \in \mathbf{Ntime}(n^{\log\log n})$. Indeed, using non-determinism it is possible to obtain $\Pi = \mathsf{Com}^{-1}(z)$ and then we have enough time to simulate the Turing machine described by Π for $n^{\log\log n/5}$ steps.

We can now present the main theorem of this section:

[10] Again, we chose $|r|^{\log\log|r|/5}$ rather arbitrarily. We just need to ensure that Λ will be in $\mathbf{Ntime}(n^{\log\log n})$.

Theorem 4.5. *Protocol 4.4 is a uniform-verifier generation protocol (as per Definition 4.1).*

To prove Theorem 4.5, one needs to prove that Protocol 4.4 satisfies both the soundness and the uniform-verifier simulation properties. We start with the soundness:

Claim 4.5.1. *Let P^* be any (possibly cheating) prover strategy for Protocol 4.4. Let τ denote the transcript of P^*'s execution with the honest verifier. Then* $\Pr[\tau \in \Lambda] \leq 2^{-n}$.

Proof. For every first prover message z, we define $f(z)$ to be the output of the Turing machine described by $\mathsf{Com}^{-1}(z)$ on input z after $n^{\log\log n/5}$ steps if $\mathsf{Com}^{-1}(z)$ is a valid Turing machine that on input z halts within this number of steps; otherwise, we define $f(z) = \perp$. For every string z, if $f(z) = \perp$ then $(z, r) \notin \Lambda$ for every r. If $f(z) \neq \perp$, then $(z, r) \in \Lambda$ iff $r = f(z)$. yet, the probability that a random $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ will be equal to $f(z)$ is at most 2^{-n}. Therefore, regardless of the prover's first message the probability that the transcript (z, r) will be in Λ is at most 2^{-n}. □

We now turn to the simulation condition:

Claim 4.5.2. *There exists a simulator S_{GENPROT} for Protocol 4.4 such that for every probabilistic polynomial-time verifier V^* whose description takes at most $2n$ bits, $S_{\text{GENPROT}}(V^*) = (v, \sigma)$ such that*

1. *v is computationally indistinguishable from V^*'s view in an execution of Protocol 4.4.*
2. *σ is a witness that the transcript τ contained in the view v is in Λ. Furthermore, it is possible to verify that σ is such a witness in time that is polynomial in the running time of V^*.*

Proof. Algorithm 4.6 is a simulator for Protocol 4.4. The output of Algorithm 4.6 is a pair (v, σ) such that $v = (s, z)$ and σ contains Π such that $\Pi(z) = V_s^*(z)$ and a witness to the fact that $z = \mathsf{Com}(\Pi)$.

The properties that we require from (v, σ) are:

1. v is computationally indistinguishable from V^*'s view in a real execution. This follows from the fact that PRG is a pseudorandom generator and so its output s is computationally indistinguishable from V^*'s random-tape in a real execution and from the fact that Com is a commitment scheme and so $\mathsf{Com}(\Pi)$ is indistinguishable from $\mathsf{Com}(0^{3n})$.
2. The transcript τ contained in v is in Λ and σ is a witness to this fact. The transcript corresponding to v is $(z, V_s^*(z) = r)$. It is indeed in Λ because $z = \mathsf{Com}(\Pi)$ such that on input z, Π outputs r within a polynomial (and therefore less than $n^{\log\log n/5}$) number of steps. The string σ contains Π and the random coins of the commitment z and so is a witness to this fact. Note that since Π is basically the verifier's strategy V^* with some inputs hardwired in, the fact that σ is a witness for τ can be verified in time that is a fixed polynomial in the running time of V^*.

Note that Algorithm 4.6 is a *non-black-box* simulator that takes the description of the verifier as input and uses it in other ways than simply as a black-box or oracle. Note also that it runs in *strict* probabilistic polynomial-time. □

Input: • 1^n: security parameter. • V^*: description of a probabilistic polynomial-time Turing machine. The length of V^* is at most $2n$.	
(Choose randomness for V^*): Let m denote the number of random bits V^* uses. Let $\mathrm{PRG} : \{0,1\}^{n/2} \to \{0,1\}^m$ be a pseudorandom generator. Choose $u \leftarrow_{\mathrm{R}} \{0,1\}^{n/2}$ and let $s = \mathrm{PRG}(u)$. We denote by V^{**} the residual verifier V^* with the randomness s hardwired into it.	
Simulated step P1 (Commitment to V^*'s program): Let Π denote the next message algorithm of V^{**}. Note that Π can be described using less than $3n$ bits (the description of V^*, the description of PRG and the seed u). Compute $z \leftarrow_{\mathrm{R}} \mathsf{Com}(\Pi)$. **Simulated Step V1 (Compute V^*'s response):** Compute the verifier V^*'s response with randomness s to the message z. That is, $r = \Pi(z)$.	$\xrightarrow{z = \mathsf{Com}(\Pi)}$ $\xleftarrow{r = \Pi(z) = V^{**}(z)}$
The output of the simulator is the pair (v, σ) where v is the view (s, z) and σ is the witness that (z, r) is in Λ (i.e., σ contains the program Π and the coins used in computing the commitment z).	

Algorithm 4.6. A simulator for Protocol 4.4.

Public input: 1^n: security parameter	1^n ↓ $\boxed{P}$ $\boxed{V}$
Step V1 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^{6n}$ and sends it.	$r \leftarrow_{\mathrm{R}} \{0,1\}^{6n}$ $\longleftarrow$
The transcript of the protocol is the string r.	

Protocol 4.7. An alternative uniform-verifier generation protocol

4.3 Summing up

When we plug into the construction of Figure 1 our generation protocol (Protocol 4.4) and the universal argument system for $\mathbf{Ntime}(n^{\log\log n})$ presented in Section 3, we obtain a zero-knowledge argument system for **NP**. This system has a constant number of rounds and is of the Arthur-Merlin type. The simulator of this zero-knowledge argument is a *non-black-box* simulator that runs in *strict* probabilistic polynomial-time.

In fact, the protocol we obtain is not just zero-knowledge against *fully uniform* verifiers but even against verifiers that have a *bounded amount of non-uniformity*. That is, verifiers that can be described in $n/2$ bits where n is the security parameter.[11] Note that although the results of Goldreich and Krawczyk [GK90] are stated for non-uniform zero-knowledge, their proofs can be extended for the case of *bounded non-uniformity*.[12] Therefore any constant-round Arthur-Merlin argument (such as ours) for a non-trivial language cannot be *black-box* zero-knowledge. This means that our simulator is *inherently* a non-black-box simulator.

Thus, the protocol of this section is sufficient for the purpose of separating black-box from non-black-box zero-knowledge. However, for other purposes, a uniform zero-knowledge protocol (or even a bounded non-uniform zero-knowledge protocol) is not completely satisfactory. For example, we don't know how to prove a *sequential composition theorem* for uniform or even bounded-non-uniformity zero-knowledge arguments [GK90]. In contrast, such a theorem *is* known to hold for *non-uniform* (a.k.a. *auxiliary input*) zero-knowledge protocols. Thus, for many application it is preferred to have such a protocol. In the next section we show how to modify our construction to obtain a *non-uniform* zero-knowledge argument system for **NP**.

4.4 An Alternative Uniform-Verifier Generation Protocol

In this section, we sketch an alternative uniform-verifier generation protocol. This alternative generation protocol, Protocol 4.7, has the advantage of being extremely simple and round efficient (consisting of only a single round, in which the verifier sends a random string). However, it has the disadvantage of being harder to generalize to the non-uniform case than Protocol 4.4. This protocol is not used in any other place in this work.

Definition of the language Λ. To fully specify Protocol 4.7, one should also specify the language Λ. Loosely speaking, we want Λ to be the language of strings with *low Kolmogorov complexity* (i.e., strings can be computed by a Turing machine with small description). However, in order to make

[11] The value $n/2$ is quite arbitrary: by "scaling" the security parameter for every polynomial $p(\cdot)$ we can obtain an argument system that is secure against verifiers that can be described using at most $p(n)$ bits.

[12] This holds also for other black-box zero-knowledge lower bounds (e.g., [CKPR01, BL02]).

Λ decidable in time $n^{\log\log n}$, we will restrict ourselves to machines that on input of size n, halt within $n^{\log\log n/5}$ steps. Formally, Λ is defined as follows:

$$r \in \Lambda \iff \exists \text{TM } M \text{ s.t. } |M| < \tfrac{|r|}{2} \text{ and } M() \text{ outputs } r \text{ within } |r|^{\log\log|r|/5} \text{ steps.}^{13}$$

We now sketch why Protocol 4.7 is indeed a uniform-verifier generation protocol:

Soundness By a simple counting argument, it can be shown that a random string r has high (and in particular higher than $\frac{|r|}{2}$) Kolmogorov complexity, and so r will not be a member of Λ with very high probability.

Simulation of a uniform-verifier Let V^* be a possibly cheating verifier whose strategy can be described in $2n$ bits, and suppose that V^* uses a random tape of size $q(n)$. To simulate the view of V^*, the simulator will use a *pseudorandom generator* $PRG : \{0,1\}^{0.1n} \to \{0,1\}^{q(n)}$, and compute $s = PRG(u)$ where $u \leftarrow_{\mathrm{R}} \{0,1\}^{0.1n}$. It will then let r be V^*'s output on input security parameter 1^n and random tape s. Because r can be computed in polynomial time from a machine whose description is at most the sum of the description of V^*, of PRG, and of u, which is less than $3n$, it follows that not only $r \in \Lambda$ but also the simulator has a *witness* to this fact. Furthermore, r is distributed in a computationally indistinguishable way from the output of V^* in a real interaction.

5 Coping with Non-Uniform Verifiers

In this section we construct a non-uniform zero-knowledge argument with the properties of the protocol of Section 4. That is, we construct an argument system for **NP** with the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.
2. It has a constant number of rounds and negligible soundness error.
3. It is an Arthur-Merlin (public coins) protocol.
4. It has a simulator that runs in *strict* polynomial-time, rather than *expected* polynomial-time.

That is, this protocol satisfies all the properties stated in Section 1.1 except for Property 3 (bounded concurrent zero-knowledge). A modification of this protocol that satisfies Property 3 is described in Section 6.

5.1 FLS'-type protocols

Like the uniform-verifier protocol of Section 4, our non-uniform protocol will also use the FLS technique. However, we will need a slight relaxation of the soundness condition of the generation protocol (Definition 4.1). This time, we will allow the possibility that the transcript τ is in Λ with non-negligible probability. However, we require that even in this case, it will be *infeasible* to come up with a *witness* that τ is indeed in Λ. Such a generation protocol is sufficient to be plugged in the construction of Figure 1, if we use a WI proof (or argument) *of knowledge* in the second

[13] We use $M()$ to denote M executed on the empty input.

stage (Steps P,V2.x), rather than just a proof of membership. In contrast, we will strengthen the simulation condition of Definition 4.1 and require simulation even of *non-uniform* verifiers. We will call a protocol that satisfies this modified definition a *non-uniform verifier* generation protocol, although we will usually drop the qualifier and simply use the name *generation protocol* for the non-uniform case.

We call a protocol that uses a generation protocol and a WI proof/argument of knowledge in this way an *FLS'-type* protocol. For completeness, we include a description of FLS'-type protocols in Figure 2.

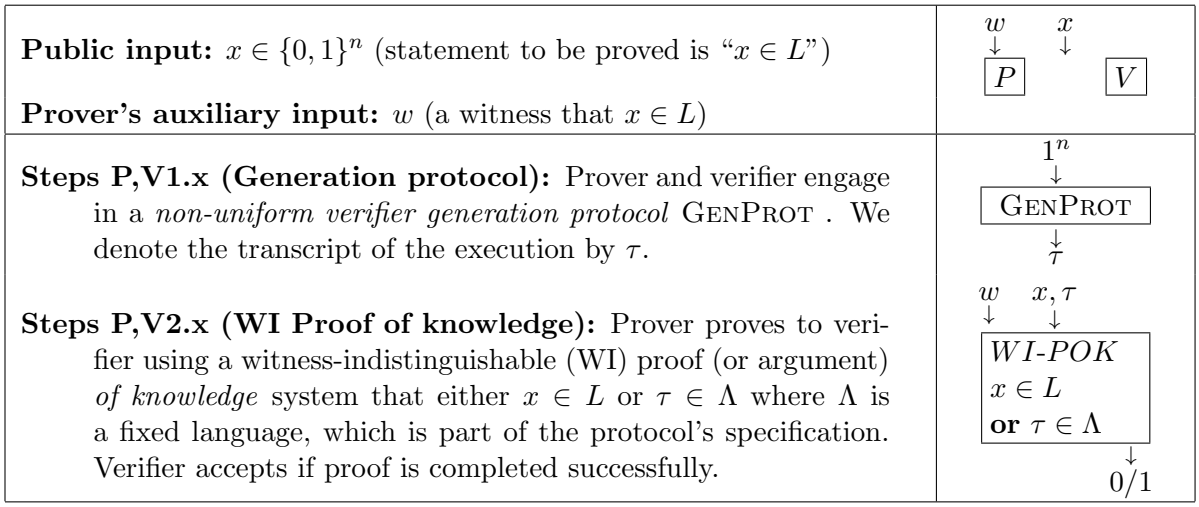


Figure 2: A generic FLS'-type zero-knowledge protocol

The formal definition of non-uniform generation protocols is as follows:

Definition 5.1 ((Non-uniform verifier) generation protocol)**.** Let GenProt be a two-party protocol where we call one party the *prover* and the other party the *verifier*. Let $\Lambda \subseteq \{0,1\}^*$ be some language in $\mathbf{Ntime}(T(n))$ for some (polynomial-time computable) function $T : \mathbb{N} \to \mathbb{N}$. We say that GenProt is a *(non-uniform) generation protocol* (with respect to the language Λ) if it satisfies the following two requirements:

Computational soundness For every $T(n)^{O(1)}$-sized (possibly cheating) prover P^* the following holds: let τ denote the transcript of the execution of GenProtbetween P^* and the prescribed verifier. The probability that P^* succeeds in outputting at the end of the interaction a witness that $\tau \in \Lambda$ is negligible.

Simulation of a non-uniform verifier There exists a probabilistic polynomial-time *simulator* S_{GenProt} that satisfies the following:

Let V^* be a polynomial-sized verifier. Then on input the description of V^*, S_{GenProt} outputs a pair (v, σ) such that:

1. v is computationally indistinguishable from the view of V^* in an execution of GenProt with the prescribed prover algorithm.
2. Let τ denote the transcript that is contained in the view v. Then it holds that $\tau \in \Lambda$ and σ is a witness to this fact. Furthermore, we require that the time to verify that σ is a witness for τ is polynomial in the running time of V^*.

Note that the computational soundness requirement refers to $T(n)^{O(1)}$-sized adversaries rather than polynomial-sized adversaries. Indeed if Λ is in $\mathbf{Ntime}(T(n))$ for a super-polynomial function $T(\cdot)$ then it may take a super-polynomial number of steps just to write down a witness. Also note that, unlike Definition 4.1, this definition does not imply that deciding Λ is hard but rather only that the search problem corresponding to Λ (of coming up with a witness) is hard.

We now state and prove the non-uniform analogue of Theorem 4.2:

Theorem 5.2. *Let* GenProt *be a* non-uniform *generation protocol with respect to the* $\mathbf{Ntime}(T)$ *language* Λ *(where* $T : \mathbb{N} \to \mathbb{N}$ *is a polynomial-time computable function). Let* WIProt *be a WI proof or argument system* of knowledge *for* $\mathbf{NP} \cup \mathbf{Ntime}(T)$ *languages. Let* L *be an* **NP** *language and let* FLSProt *be the argument for* L *that is the result of plugging in* GenProt *and* WIProt *into the construction of Figure 2. Then* FLSProt *is a* non-uniform *zero-knowledge argument for* L*.*

5.2 Proof of Theorem 5.2

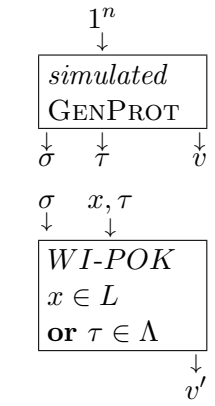
The proof of Theorem 5.2 is similar to the proof of Theorem 4.2 sketched in Section 4.1.1. To prove that FLSProt is a zero-knowledge argument one needs to prove three properties: completeness, soundness, and zero-knowledge. We will start with the soundness, since this is the main difference between this proof and the proof of Theorem 4.2.

5.2.1 Soundness

Let P^{FLSProt} be a polynomial-sized prover for FLSProt and suppose that for some $x \notin L$, the execution of P^{FLSProt} and the honest verifier is accepting with some probability ϵ. We will use P^{FLSProt} to construct a cheating prover P^{GenProt} for the generation protocol GenProt that, after interacting with the honest verifier, will be able to output a witness for the transcript with probability that is polynomially related to ϵ. Thus, if ϵ is non-negligible then this contradicts the computational soundness of GenProt.

The prover P^{GenProt} works in the following way: when interacting with the verifier of protocol GenProt, the prover P^{GenProt} will use the strategy that the prover P^{FLSProt} uses in the first stage of FLSProt on input x. Let τ denote the transcript of this interaction and let v denote the view of the prover in this interaction. After the interaction is completed, the prover P^{GenProt} will compute the *residual* prover P^{FLSProt} with state v. We denote this residual prover by P^{WIProt} since it specifies a strategy for the second stage of FLSProt: the WI proof stage. The prover P^{GenProt} then applies the *knowledge extractor* of the WI system to the P^{WIProt}. Note that this takes $T(n)^{O(1)}$ steps. If the case the extraction is successful P^{GenProt} will obtain a witness to the statement "$x \in L$ or $\tau \in \Lambda$". Since we assume that $x \notin L$ this means that in this case we obtain a witness to the fact that $\tau \in \Lambda$.

We see that to get a contradiction to the computational soundness of GenProt, all we need to show is that the extraction will be successful with probability that is polynomially related to ϵ. (Where ϵ is the overall success of P^{FLSProt} in an execution with the honest verifier.) Indeed, for at least an $\epsilon/2$ of the executions of the first stage, there is an $\epsilon/2$ probability that the second stage will finish successfully. This implies that with $\epsilon/2$ probability, the computed prover P^{WIProt} will have $\epsilon/2$ probability of convincing the verifier that "$x \in L$ or $\tau \in \Lambda$". Yet when this happens, by the proof of knowledge condition of WIProt, the knowledge extractor succeeds in extracting a witness with probability very close to $\epsilon/2$, and this finishes the proof.

Input: • $x \in \{0,1\}^n$: statement (simulate proof for "$x \in L$") • V^*: description of a polynomial-sized verifier.	
Let V^{**} denote the verifier V^* with x "hardwired" into it.	
Simulated Steps P,V1.x (Simulated generation protocol): Let $(v, \sigma) \leftarrow S_{\text{GENPROT}}(V^{**})$ where S_{GENPROT} is the simulator for the generation protocol GENPROT. Let τ denote the transcript contained in the view v. We let V^{***} denote the *residual* verifier V^{**} with the view v hardwired in. **Simulated Steps P,V2.x (Honest WI Proof of knowledge):** Run an execution of WIPROT between the verifier V^{***} and the honest prover algorithm for the WI system WIPROT the statement proved is "$x \in L$ or $\tau \in \Lambda$" using the witness σ. Let v' denote V^{***}'s view in this execution.	
Output the combined view (v, v') of the two stages	

Algorithm 5.4. A simulator for the FLS'-type protocol FLSPROT.

Remark 5.3. Note that we have actually proven that the resulting zero-knowledge system is not only sound, but actually also satisfies a weak proof of knowledge property (in the sense that if a cheating prover convinces the verifier to accept with some non-negligible probability ϵ then one can extract with probability that is polynomially related to ϵ).

5.2.2 Completeness

The proof for completeness follows the proof in the uniform case (See Section 4.1.1). Recall the description of the honest prover's algorithm on Figure 2. When given public input x and a witness w to the fact that $x \in L$, the honest prover algorithm for FLSPROT runs the honest prover algorithm for GENPROT and then runs the honest prover algorithm for the WI system to prove the combined statement "$x \in L$ or $\tau \in \Lambda$" using the witness w. Note that the witness w serves also as a witness for the combined statement, and this witness can be verifier in polynomial-time (since $L \in \mathbf{NP}$). Thus, by the completeness with efficient prover property of the WI system, the honest prover algorithm runs in probabilistic polynomial-time.

5.2.3 Zero-Knowledge

The proof for zero-knowledge also follows the proof in the uniform case (See Section 4.1.1). The simulator for GENPROT is Algorithm 5.4. The simulator's operation follows the simulator in the uniform case (Algorithm 4.3). The simulator uses the simulator S_{GENPROT} of the generation protocol GENPROT to obtain both a simulation v for the first stage along with a witness σ that is consistent with the transcript that v contains. Then, it uses the *honest prover algorithm* of the WI system WIPROT to prove the true statement "$x \in L$ or $\tau \in \Lambda$". It uses the witness σ as auxiliary input to the prover algorithm of WIPROT.

What we need to prove is the following claim:

Claim 5.4.1. *Let V^* be a polynomial-sized verifier for* FLSProt. *Let $x \in L$. Let (V_R, V'_R) be the random variable that is the view of V^* when interacting with the honest prover on input x (where V_R is the view in the first stage and V'_R is the view in the second stage, R stands for* real *as opposed to* simulated *execution.) Let (V_S, V'_S) be the random variable that is the output of the Algorithm 5.4 on input x. Then, (V_R, V'_R) and (V_S, V'_S) are computationally indistinguishable.*

Proof. The proof follows from an hybrid argument. We will prove that both distributions are computationally indistinguishable from the hybrid distribution (V_S, V'_R) where V_S represents the simulation of the first stage and V'_R represents the real execution of the second stage. That is, (V_S, V'_R) is the output of a "hybrid simulator" that uses the simulator of GenProt in the first stage but uses the witness w for x (instead of the witness σ for τ) as input to the WI prover algorithm in the second stage.

- The distribution (V_S, V'_R) is computationally indistinguishable from (V_S, V'_S) due to the WI property of the WI system WIProt.
  Indeed, if there there is an algorithm D that distinguishes between these two distributions with probability ϵ then in particular there must exist a particular view v for the first stage such that D distinguishes between (V_S, V'_R) and (V_S, V'_S) conditioned on $V_S = v$ with probability ϵ. Let τ be the transcript contained in v and let σ be the witness for τ as provided by the simulator. Let D_v be the distinguisher D with v hardwired as its first input and let V^*_v be the residual verifier V^* with the state v hardwired. Then, D_v can distinguish between an interaction of V^*_v and the honest prover of the WI system that uses σ as auxiliary input and an interaction of V^*_v and the honest prover of the WI system that uses w (the witness for x) as auxiliary input with probability ϵ. Thus, ϵ is negligible by the WI condition of WIProt.

- The distribution (V_S, V'_R) is computationally indistinguishable from (V_R, V'_R) due to the simulation condition of the generation protocol GenProt.
  Indeed, suppose that there is an algorithm D that distinguishes between these two distributions with probability ϵ. Then we can construct a distinguisher D' to contradict the simulation condition of the generation protocol in the following way. The distinguisher D' has the witness w for x hardwired in. When it gets a string v as input, it runs the honest prover algorithm of the WI proof for the statement "$x \in L$ or $\tau \in \Lambda$" (where τ is the transcript contained in v) using the witness y. It plays the part of the verifier using the residual verifier V^* with state v. Let v' denote the view of the verifier in the WI proof. The distinguisher D' returns $D(v, v')$. We see that D' distinguishes between the view of V^* in a real interaction of GenProt and the output of $S_{\text{GenProt}}(V^*)$ with probability ϵ and so ϵ is negligible by the simulation condition of GenProt.

□

5.3 A Non-Uniform Verifier Generation Protocol

Once we have Theorem 5.2, all that is left to do is to construct a *non-uniform* verifier generation protocol with respect to some $\mathbf{Ntime}(n^{\log\log n})$ language Λ. Our non-uniform verifier generation protocol will be based on the uniform-verifier generation protocol of the previous section (Protocol 4.4).

Public input: 1^n: security parameter	1^n ↓ $\boxed{P}$ $\boxed{V}$
Step V1 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover.	$\xleftarrow{h \leftarrow_{\mathrm{R}} \mathcal{H}_n}$
Step P2 (Commitment to hash of"junk"): Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z to verifier.	$\xrightarrow{z = \mathsf{Com}(h(0^n)}$
Step V3 (Send random string): The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$\xleftarrow{r \leftarrow_{\mathrm{R}} \{0,1\}^n}$
The transcript of the protocol is the pair $\tau = (h, z, r)$.	

Protocol 5.5. A non-uniform verifier generation protocol

Recall how we proved that Protocol 4.4 satisfies the uniform simulation condition. The first message of the protocol was supposed to be a commitment to 0^{3n}. However, the *simulator* (Algorithm 4.6) simulated this message by commitment z to the *next-message function* of the verifier. This may be problematic if we try to use the same protocol and simulator in the non-uniform setting. The problem is that since a commitment scheme is statistically binding, then it necessarily holds that the length of the commitment z will be longer than the length of the description of the next message function. However, once we consider *non-uniform* verifiers then we must allow for the next-message function's description to be of any polynomial length, and in particular it may be larger than the communication complexity of our protocol. The solution is quite simple. Instead of using a statistically binding commitment scheme, we will use a *computationally binding* commitment scheme. A computationally binding commitment scheme that allows to commit to messages that are longer than its output can be constructed by composing a standard, statistically binding commitment scheme, with a collision-resistent hash function. One can see why we had to relax the soundness condition of the definition of a generation protocol: once we use a computationally binding commitment scheme, we will only be able to prove that are protocol is *computationally sound*. This is the intuition that we follow in both the construction of the generation protocol and the definition of the corresponding language Λ. Protocol 5.5 is our generation protocol.

Definition of the language Λ. We define the language Λ as follows: $\tau = (h, z, r)$ is in Λ if there exists a program Π such that $z = \mathsf{Com}(h(\Pi))$ and $\Pi(z)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A *witness* that $(h, z, r) \in \Lambda$ is a couple (Π, s) such that $z = \mathsf{Com}(h(\Pi); s)$ and $\Pi(z)$ outputs r within $|r|^{\log\log|r|/5}$ steps. Note that it may be the case that Λ is easy to *decide* (in fact it may be that $\Lambda = \{0,1\}^*$) but the soundness condition of Definition 5.1 refers only to the infeasibility of coming up with a witness.

Theorem 5.6. *Protocol 5.5 is a (non-uniform verifier) generation protocol with respect to the language Λ (as per Definition 5.1).*

5.4 Proof of Theorem 5.6

To prove that Protocol 5.5 meets Definition 5.1 we need to show that it satisfies two properties: computational soundness and non-uniform simulation.

5.4.1 Computational Soundness

Let P^* be a $n^{O(\log\log n)}$-sized prover strategy for Protocol 5.5. Let τ denote the transcript of P^*'s execution with the honest verifier. We claim that the probability that P^* writes a witness that $\tau \in \Lambda$ on its auxiliary tape is negligible.

Indeed, suppose otherwise that P^* manages to output a witness with non-negligible probability ϵ. Then, for at least an $\epsilon/2$ fraction of the $h \in \mathcal{H}_n$, it holds that P^* manages to output a witness for the transcript starting with h with probability $\epsilon/2$. Fix such an $h \in \mathcal{H}_n$. Since P^* is a non-uniform algorithm, we can assume without loss of generality that P^* is deterministic. Thus the message z, which is the prover P^*'s response to h, is also fixed. By our assumption, if we choose $r \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\epsilon/2$ the prover will be able to output a program Π such that z is a commitment to $h(\Pi)$ and $\Pi(z) = r$ (within $|r|^{\log\log|r|/5}$ steps). This means that if we choose two independent $r, r' \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\epsilon^2/4$, we obtain two programs Π, Π' such that z is a commitment to both $h(\Pi)$ and $h(\Pi')$ and $\Pi(z) = r$, $\Pi'(z) = r'$. Since $\mathsf{Com}(\cdot)$ is a statistically binding commitment scheme, it follows that $h(\Pi) = h(\Pi')$. Yet, since we can assume that $r \neq r'$ (as this holds with $1 - 2^{-n}$ probability), it follows that $\Pi(z) \neq \Pi'(z)$ and so Π and Π' are different programs. This means that Π and Π' are a *collision* for h. This means that we have a $n^{O(\log\log n)}$-sized algorithm that, for an $\epsilon/2$ fraction of $h \in \mathcal{H}_n$, obtains a collision for h with probability $O(\epsilon^2)$. This contradicts the collision-resistence against $n^{\log n}$-sized adversaries of the family $\mathcal{H}_n$.

5.4.2 Simulation of a non-uniform verifier

The proof that Protocol 5.5 satisfies the simulation requirement is quite similar to its uniform analog (Claim 4.5.2). What we need to show is that there exists a simulator $S_{\textsc{GenProt}}$ for Protocol 5.5 such that for every polynomial-sized verifier V^*, $S_{\textsc{GenProt}}(V^*)$ outputs a pair (v, σ) such that v is computationally indistinguishable from V^*'s view and σ is a witness that the transcript τ compatible with v is in Λ.

Algorithm 5.7 is a simulator for Protocol 5.5. As we can see, when simulating an execution with transcript (h, z, r), the output of Algorithm 5.7 is a pair (z, σ) and $\sigma = (\Pi, s)$ is such that $z = \mathsf{Com}(h(\Pi); s)$. The two properties that we require from (z, σ) are:

1. z is computationally indistinguishable from V^*'s view in a real execution. In a real execution the verifier sees a single message which is $\mathsf{Com}(h(0^n))$. The message z is equal to $\mathsf{Com}(h(\Pi))$. Thus this property follows immediately from the computational hiding property of the commitment scheme Com.

2. The transcript τ corresponding to v is in Λ and σ is a witness to this fact. The transcript corresponding to v is (h, z, r) (where $h = V^*()$ and $r = H(z)$). The pair $\sigma = (\Pi, s)$ is indeed a witness that $(h, z, r) \in \Lambda$ since $z = \mathsf{Com}(h(\Pi); s)$ and $\Pi(z)$ outputs r in a polynomial (and so less than $|r|^{\log\log|r|/5}$) number of steps. Note that indeed the time to verify that σ is a witness for τ is polynomial in the running time of V^*.

Note that Algorithm 5.7, like Algorithm 4.6, is a *non-black-box* simulator. Note also that it runs in *strict* probabilistic polynomial-time.

This finishes the proof of Theorem 5.6. By using this generation protocol in Theorem 5.2 we obtain a non-uniform zero-knowledge argument with all the desired properties.

Input: • 1^n: security parameter. • V^*: a polynomial-sized circuit (without loss of generality V^* is deterministic).	
Simulated Step V1 (Choose hash-function): Compute h: the V^*'s first message. **Simulated step P2 (Commitment to V^*'s program):** Let Π denote the next message algorithm of V^*. Compute $z = \mathsf{Com}(h(\Pi); s)$ where $s \leftarrow_{\mathrm{R}} \{0,1\}^{poly(n)}$ are coins chosen for the commitment scheme. **Simulated Step V3 (Compute V^*'s response):** Compute the verifier V^*'s response to the message z. That is, $r = \Pi(z)$.	$\xleftarrow{h = V^*()}$ $\xrightarrow{z = \mathsf{Com}(h(\Pi))}$ $\xleftarrow{r = \Pi(z) = V^*(z)}$
The output of the simulator is the pair (z, σ) where z is the simulated verifier's view and $\sigma = (\Pi, s)$ is the witness that (h, z, r) is in Λ.	

Algorithm 5.7. A simulator for Protocol 5.5.

6 Achieving Bounded-Concurrent Zero-Knowledge

The condition that a proof/argument system is zero-knowledge guarantees the prover that a possibly malicious verifier will not be able to gain any new knowledge about the statement that is being proved. However, somewhat surprisingly, it turns out that this may *not* be the case if the prover is proving two or more related statements at the same time [GK90]. To guarantee security for the prover in this (quite realistic) setting, one needs a stronger form of zero-knowledge. This stronger form, called *concurrent zero-knowledge*, was introduced by Dwork, Naor and Sahai [DNS98]. Loosely speaking, a protocol is concurrent zero-knowledge if it remains zero-knowledge even when any polynomial number of possibly related statements are being proved simultaneously, with the scheduling of messages chosen (possibly in a malicious way) by the verifier.

Dwork *et al.*[DNS98] constructed a concurrent zero-knowledge argument for **NP** in the *timing* model, which is a model that assumes a some known time bounds on the delivery of messages in the communication network (See also [Gol01a]). Richardson and Kilian [RK99] were the first to construct a concurrent zero-knowledge argument for **NP** in the standard (pure asynchronous) model. Their protocol used a polynomial number of rounds. This was later improved by Kilian and Petrank to a polylogarithmic number of rounds [KP00] and further improved by Prabhakaran, Rosen and Sahai to a slightly super-logarithmic number of rounds [PRS92]. This is essentially the best one can obtain using *black-box* simulation as shown by Canetti, Kilian, Petrank and Rosen [CKPR01] (improving on [KPR98] and [Ros00]).

In this section, we show how to modify the zero-knowledge protocol of the last section as to obtain a protocol that is *bounded concurrent* zero-knowledge. A zero-knowledge protocol is bounded-concurrent zero-knowledge if it remains zero-knowledge when executed up to n times concurrently, where n is the security parameter. Since the security parameter can be "scaled", this means that for every *fixed* polynomial $p(\cdot)$, we can construct a protocol that remains zero-knowledge when executed $p(n)$ times. However, this protocol will depend on $p(\cdot)$ and in particular it will have

communication complexity greater than $p(n)$. This is in contrast with the notion of (unbounded) concurrent zero-knowledge described above, where there is a single protocol that remains zero-knowledge when executed $p(n)$ times for *every* polynomial $p(\cdot)$. We stress that the negative results of [CKPR01] regarding concurrent black-box zero-knowledge hold also for *bounded* concurrent zero-knowledge. In particular, there does not exist a constant-round bounded-concurrent zero-knowledge proof/argument (for a non-trivial language) that utilizes a *black-box* simulator.

Formally, we define bounded-concurrent zero-knowledge as follows:

Definition 6.1 (Concurrent execution)**.** Let (P, V) be a two-party protocol. Let V^* be an interactive machine. Let $\{(a_i, b_i)\}_{i=1}^t$ be a set of t inputs to the protocol (P, V). A *t-times concurrent execution* of (P, V) coordinated by V^* on inputs $\{(a_i, b_i)\}_{i=1}^t$ is the following experiment:

1. Run t independent copies of P with the i^{th} copy getting a_i as input.
2. Provide V^* with the $b_1, \ldots, b_t$.
3. On each step V^* outputs a message (i, m). The i^{th} copy of P is given with the message m. The verifier V^* is given the prover's response.

Definition 6.2 (Bounded-concurrent zero-knowledge)**.** Let (P, V) be an interactive proof *orargument* system for a language $L = L(R)$. We say that (P, V) is *bounded-concurrent zero-knowledge* if there exists a probabilistic polynomial-time algorithm S such that for every polynomial-sized V^*, and every list $\{(x_i, y_i)\}_{i=1}^n$ such that $(x_i, y_i) \in R$, the following two random variables are computationally indistinguishable:

1. The view of V^* in an n-times concurrent execution of (P, V) with inputs $\{(x_i, y_i)\}_{i=1}^n$.
2. $S(V^*, x_1, \ldots, x_n)$

Our protocol will be an FLS'-type protocol, and will use a (non-uniform verifier) generation protocol which is very similar to Protocol 5.5. In fact, the only difference between the generation protocol of this section and Protocol 5.5 will be that we will use a longer string r as the verifier's message (Step V2). That is, we will use $r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ rather than $r \leftarrow_{\mathrm{R}} \{0,1\}^n$.[14] Protocol 6.3 is the modified generation protocol.

Definition of the language Λ. We use a somewhat different language Λ that the one used in the previous section. We define Λ as follows: $\tau = (h, z, r)$ is in Λ iff there exists a program Π such that $z = \mathsf{Com}(h(\Pi))$ and *there exists a string y* such that $|y| \leq |r|/2$ and $\Pi(z, y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A witness that $(h, z, r) \in \Lambda$ is a triple (Π, s, y) such that $z = \mathsf{Com}(\Pi; s)$, $|y| \leq |r|/2$ and $\Pi(z, y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. That is, we've changed the language so that the committed program Π that outputs r can get not only z as input but is also allowed an additional input y, as long as it is not too long (i.e., as long as $|y| \leq |r|/2$).

The following theorem states that the modified protocol is still a generation protocol:

Theorem 6.4. *Protocol 6.3 is a (non-uniform verifier) generation protocol with respect to Λ*

[14] The value n^4 is also somewhat arbitrary. We have not tried to optimize the relation between the communication complexity and the number of concurrent sessions.

Public input: 1^n: security parameter	1^n ↓ $\boxed{P}$ $\boxed{V}$
Step V1 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_R \mathcal{H}_n$ and sends h to prover.	$\xleftarrow{h \leftarrow_R \mathcal{H}_n}$
Step P2 (Commitment to hash of"junk"): Prover computes $z \leftarrow_R \mathsf{Com}(h(0^n))$ and sends z to verifier.	$\xrightarrow{z = \mathsf{Com}(h(0^n)}$
Step V3 (Send long random string): The verifier selects a string $r \leftarrow_R \{0,1\}^{n^4}$ and sends it.	$\xleftarrow{r \leftarrow_R \{0,1\}^{n^4}}$
The transcript of the protocol is the pair $\tau = (h, z, r)$.	

Protocol 6.3. A generation protocol for bounded concurrent zero-knowledge.

Proof Sketch: We only sketch the proof because it is almost identical to the previous section (the proof of Theorem 5.6 in Section 5.4). The fact that the message r is longer does not change anything in the proof, and so we only need to see that the modification to Λ did no harm.

Indeed, the proof of the non-uniform simulation requirement is unchanged, since the simulator presented in the proof of Theorem 5.6 (Algorithm 5.7) is also a simulator for Protocol 6.3. The witness this simulator outputs is a valid witness also for the modified language of this section (it simply uses the empty word for the string y).

The proof of the computational soundness is slightly changed but still works. Recall that the proof there (in Section 5.4.1) relied on converting a cheating prover into an algorithm to find collision for the hash functions. We used there the following observation: if for some value z ,Π is a program such that $\Pi(z) = r$ and we choose a random $r' \leftarrow_R \{0,1\}^n$ and obtain with probability ϵ a program Π' such that $\Pi'(z) = r'$, then Π will be different from Π' with probability at least $\epsilon - 2^{-n}$. We used this observation to show that we can use a cheating prover to obtain a collision pair Π and Π' for the hash function.

The key observation we need to use now is the following: if for some value z, Π is a program such that $\exists_{y \in \{0,1\}^{m/2}} \Pi(z, y) = r$ and we choose a random $r' \leftarrow_R \{0,1\}^m$ and obtain with probability ϵ a program Π' such that $\exists_{y' \in \{0,1\}^{m/2}} \Pi'(z, y') = r'$, then Π will be different from Π' with probability at least $\epsilon - 2^{-m/2}$. This is because if $\Pi' = \Pi$ then it must hold that $r' \in \Pi(z, \{0,1\}^{m/2})$ which happens with probability at most $2^{-m/2}$. Note that in our case $m = n^4$ and so $2^{-m/2}$ is a negligible quantity.

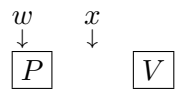
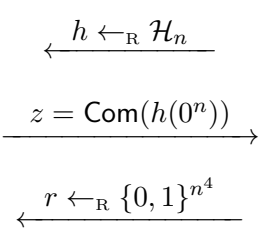
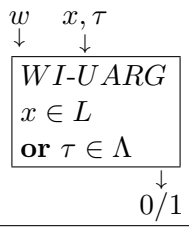
Using this observation, the proof of the soundness property follows the proof of Section 5.4.1. □

6.1 The Zero-Knowledge Argument

Our bounded-concurrent zero-knowledge protocol for **NP** is Protocol 6.5. It is constructed by plugging in the generation protocol of the previous section (Protocol 6.3) and the WI universal argument system of Theorem 3.2 using the FLS' paradigm (see Figure 2).

By the results of the previous sections, it satisfies the completeness, soundness and (standalone) non-uniform zero-knowledge properties. It is also clearly a constant-round Arthur-Merlin protocol. Thus, all that remains is to prove that it remains zero-knowledge under bounded concurrent composition. This is what we do in this section. Thus, our main theorem is the following:

Theorem 6.6. *Protocol 6.5 is bounded-concurrent zero-knowledge.*

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$)	
Steps P,V1.x: generation protocol **Step V1.1 (Choose hash-function):** Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover. **Step P1.2 (Commitment to hash of"junk"):** Prover computes $z \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z to verifier. *(Short message.)* **Step V1.3 (Send long random string):** The verifier selects a string $r \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ and sends it. The transcript of this stage is $\tau = (h, z, r)$.	
Steps P,V2.1.x (WI universal argument): Prover proves to verifier using a WI universal argument that either $x \in L$ or $\tau \in \Lambda$. *All prover's messages here are short.*	

Protocol 6.5. A bounded-concurrent zero-knowledge protocol

6.2 Proof of Theorem 6.6

Our proof that it is bounded concurrent zero-knowledge will not be as modular as our previous proofs (see also Remark 6.8). That is, we will argue about the entire zero-knowledge argument as a whole, rather than proving statements about its components.

Message lengths. We will need to consider the lengths of the messages sent by the prover. We call messages that of length less than n^2 bits "short". We will make essential use of the observation that all the prover's messages of Protocol 6.5 are "short". (Note that this is not the case for the verifier's messages since the message r of Step V1.3 is of length n^4.) In the first stage, we can assume that the prover's message in Step PV1.2 (the commitment $z = \mathsf{Com}(h(0^n))$) is short, because there are constructions of commitment schemes such that the commitment to a message of length n is at most n^2. In the second stage we will use the fact that by Theorem 3.2 for every $\epsilon > 0$ there exists a witness-indistinguishable universal argument system with communication complexity m^ϵ where m is the instance length. Therefore, we can ensure that all prover's messages in the second stage will be of length at most n^2. (Even though the length of the statement proven at this stage is more than n^4.)

6.2.1 Overview of the simulator

To prove that Protocol 6.5 is bounded-concurrent zero-knowledge, one needs to describe a *simulator* for the protocol, and then analyze its output. We will start with an overview of the simulator's operation, and then provide a detailed description and analysis.

Let V^* be a polynomial-sized algorithm describing the strategy of a verifier in an n-times

concurrent execution of Protocol 6.5. Recall than an n-time concurrent execution of a protocol involves an execution of n sessions, which are interleaved in a way chosen by the verifier. We denote the j^{th} overall *prover message* (in the entire execution) by m_j. Each message m_j belongs to some session i (where $1 \leq i \leq n$). Because the execution is concurrent it does not necessarily hold that the messages of a particular session are consecutive, but they are always ordered (e.g., the first message of session i always comes before the second message of session i). The *view* of the verifier in the execution is the sequence $(m_1, \ldots, m_{cn})$, where c is the (constant) number of prover messages in Protocol 6.5. The object of our simulator is to generate a sequence $(m_1, \ldots, m_{cn})$ that is indistinguishable from the view of the verifier in a real execution.

Our simulator will generate this sequence incrementally. That is, we will generate the message m_j only after we generated $(m_1, \ldots, m_{j-1})$ and once we generated a message we will not "go back" and change it. Therefore, it is possible (and useful) to think of our simulator as interacting with the verifier V^*. Of course, the difference between this interaction and a real interaction is that our simulator has the "unfair" advantage of knowing V^*'s code.

Notation. Throughout this section we will use i to index a session (i.e., $1 \leq i \leq n$) and j and k to index an overall prover message (i.e., $1 \leq j, k \leq cn$). We will use subscript to denote the overall index of a message (e.g., m_j) and use parenthesized superscript to denote the session that a message belongs to (e.g., $r^{(i)}$). We will sometimes drop the session number when it is clear from the context. We will sometime identify a prover or verifier message not by its overall index, but rather by its session number and step number in Protocol 6.5. Thus we will say statements like "let $r = r^{(i)}$ denote the verifier message of Step V.1.3 of the i^{th} session".

The naive simulator. The naive first attempt at a simulator would be to try to invoke the standalone simulator of Protocol 6.5 n times independently. Let us see exactly where this naive attempt fails. Suppose that we have to compute the k^{th} prover message which is Step P1.2 of some session i. On the previous step (Step V1.1), the verifier sent a hash function $h = h^{(i)}$ and on Step P1.2 we need to compute a commitment $z = z^{(i)}$ to $h(\Pi)$ where $\Pi = \Pi^{(i)}$ is some program. If we follow the instruction of the standalone simulator, then we will let Π be simply the program of the residual verifier V^* at this point. Suppose now that the verifier decides to schedule at this point some other sessions (i.e., different than i). That is, the next messages $m_{k+1}, \ldots, m_{j-1}$ the verifier receives are part of sessions other than i (where m_j is the next prover message after m_k that belongs to the i^{th} session). Suppose that when the verifier sends the string $r = r^{(i)}$ corresponding to Step V1.3 of the i^{th} session, the verifier computes r as a function of the messages $m_{k+1}, \ldots, m_{j-1}$ it received. Since Π was the residual verifier at point k, it is *not* the case that $\Pi(z) = r$. Thus it will not hold that $(h, z, r) \in \Lambda$ and we see that simply running the independent simulator will fail.

However, not all is lost: indeed it *is* the case that $r = \Pi(z, m_{k+1}, \ldots, m_{j-1})$. That is, r is V^*'s response after it receives the messages $m_1, \ldots, m_{j-1}$ where $m_1, \ldots, m_{k-1}$ were already part of Π's description, and $m_k = z$. The crucial observation here is that since all prover's messages are "short" (of length at most n^2), and since there are at most cn messages (where c is the constant number of rounds in Protocol 6.5) it holds that

$$|m_{k+1}| + \ldots + |m_{j-1}| \leq O(n^3) < \frac{n^4}{2} = \frac{|r|}{2}$$

Yet this means that we have a witness to the fact that the transcript (h, z, r) is in Λ. This is because, under the definition of Λ in this section, we don't need to show that $\Pi(z) = r$ but rather only to show that $\Pi(z, y) = r$ for *some short string* y (i.e. for y such that $|y| < |r|/2$).

Therefore, we can continue the simulation at this point, and simply use in the session the honest prover strategy for the WI universal argument, with Π, $(m_{k+1}, \ldots, m_{j-1})$ and the coins used in computing the commitment z.

6.2.2 Actual description of the simulator

Our simulator's operation follows the above description. We now turn to formally describing the simulator algorithm:

Algorithm $\mathcal{S}im$:

Input:

- $x_1, \ldots, x_n$: the statement to be proved in the i^{th} session is that $x_i \in L$.
- V^*: description of polynomial-sized verifier coordinating an n-time concurrent execution.

Initialization: The simulator will construct the a table $\mathcal{A}$ of length n. Initially the table $\mathcal{A}$ will be empty. We will maintain the invariant that after we simulated Step P1.2 of the i^{th} session and computed a message $z = z^{(i)}$, $\mathcal{A}[i]$ will contain a pair $(\Pi, s) = (\Pi^{(i)}, s^{(i)})$ such that $z = \mathsf{Com}(h(\Pi; s)$ where $h = h^{(i)}$ is the hash function chosen by the verifier in Step V1.1 of the i^{th} session. After the simulator obtains the message $r = r^{(i)}$ of Step V1.3 from the verifier, it will add to $\mathcal{A}[i]$ a string $y = y^{(i)}$ of length less than $n^4/2$ such that $\Pi(z, y) = r$. That is, at this point $\mathcal{A}[i]$ contains a witness (Π, s, y) for the fact that $(h, z, r) \in \Lambda$ where $h = h^{(i)}$,$z = z^{(i)}$ and $r = r^{(i)}$ are respectively the messages of Steps V1.1, P1.2, and V1.3 of the simulated i^{th} session.

Simulating each step: For $j = 1, \ldots, cn$ the simulator computes the j^{th} simulated prover message m_j in the following way:

Feed the previously computed messages $(m_1, \ldots, m_{j-1})$ to V^* and obtain the j^{th} verifier message (i, m) (where i is the session the verifier's message is intended to). Compute the message m_j according to the current step in the simulated proof of the i^{th} session:

Step P1.2 - Commitment to program If the verifier's message is for Step V1.1 of the i^{th} session, do the following:

- Let $h = h^{(i)}$ denote the verifier's message.
- Compute the description of the following program Π:

  $\mathbf{\Pi}(\mathbf{z}, \mathbf{y})$ returns $V^*(m_1, \ldots, m_{j-1}, z, y)$.

  (Note that the values $m_1, \ldots, m_{j-1}$ were previously computed.)
- $z = \mathsf{Com}(h(\Pi); s)$ where s is the randomness for the commitment.
- Store Π and s in $\mathcal{A}[i]$.
- The j^{th} message m_j will be z

Receiving message of Step V1.3 If the verifier's message was for Step V1.3 of the i^{th} message then do as follows:

- Let $r = r^{(i)}$ denote the verifier's message. Note that $r = V^*(m_1, \ldots, m_{j-1})$.

- Let k denote the overall index of prover's message in Step P1.2 of the same session. That is, m_k was the message z of the same session.
- Let $y = (y_1, \ldots, y_{j-k-1})$ denote the sequence $(m_{k+1}, \ldots, m_{j-1})$. Note that since $j \leq cn$ we have that $|y| \leq cn \cdot 10n^2$ and so $|y| < n^4/2$ (for sufficiently large n).
- Add y to the cell $\mathcal{A}[i]$. Note that $\mathcal{A}[i]$ already contains (Π, s) such that $z = \mathsf{Com}(\Pi; s)$ and $\Pi(z, y) = r$.

Steps P2.x - WI UARGS In these steps we simply follow the honest prover strategy of the WI universal arguments to prove that either $x_i \in L$ or the transcript $\tau = \tau^{(i)}$ of the first stage of the i^{th} session is in Λ. Note that we can use $\mathcal{A}[i]$ as a witness that indeed the transcript of the first stage is in Λ.

6.2.3 Analysis

The theorem we need to prove is the following:

Theorem 6.7. *For every polynomial sized verifier v^* and sequence $\{(x_i, y_i)\}_{i=1}^n$ such that y_i is a witness that $x_i \in L$, the following two random variables are computationally indistinguishable:*

- *The view of V^* in an n-times concurrent execution of Protocol 6.5 on inputs $\{(x_i, y_i)\}_{i=1}^n$. We denote this random variable by X.*
- *The output of Algorithm $\mathcal{S}im$ (of Section 6.2.2) on input V^* and $(x_1, \ldots, x_n)$. We denote this random variable by Y.*

Proof. The proof is actually not complicated. We will use the hybrid argument to show that X and Y are computationally indistinguishable. Let $\mathcal{S}im'$ be an algorithm that on input $V^*, \{(x_i, y_i)\}_{i=1}^n$ follows the same strategy as the simulator $\mathcal{S}im$ on input V^* and $(x_1, \ldots, x_n)$, except that when simulating the steps of the WI universal argument (Steps P2.x) in the i^{th} session it will provide y_i as input the honest prover algorithm. Let Z denote the output of $\mathcal{S}im'$ on input V^* and $V^*, \{(x_i, y_i)\}_{i=1}^n$. We will prove the theorem by showing that Z is computationally indistinguishable from both X and Y. That is, we make the following two claims:

1. Z is computationally indistinguishable from Y:

   Note that the only difference between Z and Y is the witness that is used as input to the WI universal argument prover. Note also that the randomness used in running the WI prover is never referred to or used again in any other part of the simulation. Thus, this claim basically follows from the fact that WI is closed under concurrent composition. Still, as the protocol contains also messages that do not belong to the WI universal arguments, we prove the claim below.

   Let us order the sessions according to the scheduling of the first step in their second stage (the WI universal argument stage). Let Z_i denote a distribution where in the first i sessions we use follow the strategy for $\mathcal{S}im$ and in the last $n - i$ sessions we follow the strategy for $\mathcal{S}im'$. Note that $Z_0 = Z$ and $Z_n = Y$. It will be sufficient to prove that Z_i is computationally indistinguishable from Z_{i+1} for all $0 \leq i < n$.

   Indeed, suppose that we have a distinguisher D that distinguishes between Z_i and Z_{i+1} with probability ϵ. Let us fix a choice of coins used in all sessions before the $i + 1^{st}$ such that D distinguishes between Z_i and Z_{i+1} conditioned on this choice with probability ϵ. Note that

the statement to be proved is now identical in Y and Z. Since the execution of all sessions other than the i^{th} is the same in Y and Z, and since the strategy for this execution is a function of the inputs, the coins, and possibly the public messages of the i^{th} session, the distinguisher D can be turned into a distinguisher for the WI universal argument system.

2. Z is computationally indistinguishable from X:

   The only difference between Z and X is that the commitment in Step P1.2 is for $h(\Pi)$ instead of $h(0^n)$. Note also that in both Z and X, the randomness used for this commitment is never referred to or used again in any other part of the simulation (because the WI stage uses y_i as a witness). Again, we prove the claim below, even though it is basically implied by the multiple-sample security of the commitment scheme.

   We now order the sessions according to the order of the message in Step P1.2. We define X_i to be a distribution where in the first i sessions we use $\mathsf{Com}(h(0^n))$ and in the last $n-i$ sessions we follow the strategy of $\mathcal{S}im$ (i.e., use $\mathsf{Com}(h(\Pi))$). Note that $X_0 = X$ and $X_n = Z$.

   Suppose that there exists a distinguisher D that distinguishes between X_i and X_{i+1} with probability ϵ. Suppose we fix a choice of coins for all sessions except the i^{th} such that D distinguishes between X_i and X_{i+1} conditioned on this choice with probability ϵ. The distinguisher D can be converted to a distinguisher between a commitment to $h(\Pi)$ and a commitment to $h(0^n)$ by hardwiring all messages before the message of Step P1.2 of the i^{th} session, and since the later messages are a function of the input and the previous messages.

□

We now mention some remarks regarding the proof and the protocol.

Remark 6.8. It is possible to define a "bounded-concurrent generation protocol" and to prove that Protocol 6.3 satisfies this definition. This, combined with the fact that WI is closed under concurrent composition, may be used to give more a modular proof of Theorem 6.6.

Remark 6.9. As observed by Yehuda Lindell, the proof of Theorem 6.6 actually yields a stronger statement than the theorem. For the purposes of the proof it does not matter if the message history consists of messages from an execution of our protocol, or from other executions of arbitrary protocol, as long as the history is short enough. Therefore, our protocol does not only compose concurrently with itself, but also with other protocols, as long as we have a bound on the total communication complexity of the other protocols. This property of our protocol was used in a recent work by Lindell [Lin03].

Also note that for the proof it is not necessary for the messages themselves to be short. Rather, it is enough that they have a *short description*. To be more specific, for the proof to hold it is not necessary to have a the entire message history h be of length shorter than $n^4/2$. Rather, it is sufficient that there will be a polynomial-time computable function $F : \{0,1\}^{n^4/2} \to \{0,1\}^*$ to which the simulator can commit in advance such that there will exist an input x that satisfies $h = F(x)$. This input x will be the *short explanation* for h.

7 Conclusions and future directions

We have shown that (under standard complexity assumptions) there exists a protocol that can be shown to be zero-knowledge using a *non-black-box* simulator, but cannot be shown to be zero-knowledge using a *black-box* simulator. Moreover, we have shown that this protocol satisfies some desirable properties that are impossible to obtain when restricted to black-box simulation.

7.1 Reverse-engineering

Arguably, our simulator does not "reverse-engineer" the verifier's code, although it applies some non-trivial transformations (such as a **PCP** reduction and a Cook-Levin reduction) to this code. Yet, we see that even without doing "true" reverse-engineering, one can achieve results that are impossible in a black-box model. This is in a similar vein to [BGI+01], where the impossibility of code obfuscation is shown without doing "true" reverse-engineering. One may hope to be able to define a model for an "enhanced black-box" simulator that would be strong enough to allow all the techniques we used, but weak enough to prove impossibility results that explain difficulties in constructing certain objects. We are not optimistic about such attempts.

7.2 Non-black-box proofs of security

A statement of security of a cryptographic scheme is usually underlined by a proof of the following form

> "There exists a reduction that converts an adversary A that breaks our scheme into an algorithm B that breaks the well-known assumption X (e.g., factoring)."

In almost all cases, the reduction constructs a generic algorithm B that uses the alleged algorithm A as an oracle or a *black-box*. In such cases, we say that the scheme has a *black-box proof of security*.

A natural question to ask is whether using *non-black-box* proofs of security will give us more power and allow us to attain stronger cryptographic goals or use weaker assumptions. Because many schemes that use zero-knowledge as a component (e.g., identification schemes) use the simulator as part of the proof of security, once we use the protocol of the current work the proof of security becomes a non-black-box proof of security. Thus, this work gives a seemingly positive answer to this question. In a recent work [Bar02], we use similar techniques to construct a non-malleable commitment scheme with a non-black-box proof of security. The latter commitment scheme is the first such scheme with a constant number of rounds.

7.3 Black-box impossibility results and concurrent zero-knowledge

There are several negative results regarding the power of *black-box* zero-knowledge arguments. The existence of non-black-box simulators suggests a re-examination of whether these negative results holds also for general (non-black-box) zero-knowledge. Indeed, we have already shown in this work that some of these results *do not* hold in the general setting. The case of concurrent composition is an important example. The results of [CKPR01] imply that (for a constant-round protocol) it is impossible to achieve even *bounded* concurrency using black-box simulation. We have shown that this result does not extend to the non-black-box settings. However, it is still unknown whether one can obtain a constant-round protocol that is (fully) concurrent zero-knowledge. This is an important open question.

7.4 Cryptographic assumptions

In this work we constructed our protocols based on the assumption that collision-resistent hash function exist with some fixed "nice" super-polynomial hardness. It would be nicer to construct the protocol using the more standard assumption that collision-resistent hash functions exist with

arbitrary super-polynomial hardness. Indeed this can be done, and in [BG01], Barak and Goldreich construct universal arguments (by a slightly different definition) based on the more standard assumption. Using this construction, they show that a slightly modified version of the protocol presented here, enjoys the same properties under the more standard assumption.

7.5 The resettable setting

Following this work, Barak, Goldreich, Goldwasser and Lindell [BGGL01] have shown another case where a black-box impossibility result does *not* hold in the general setting. Using results of the current work, they construct a *resettably-sound* zero-knowledge argument for **NP**. As they note, this is trivially impossible to obtain using black-box simulation. They use this resettably-sound zero-knowledge argument to construct a *argument of knowledge* with a *non-black-box knowledge-extractor* that is zero-knowledge in the *resettable* model [CGGM00]. As noted by [CGGM00], this is trivially impossible when using black-box knowledge-extraction.

7.6 Black-box reductions

There are also several negative results regarding what can be achieved using black-box *reductions between two cryptographic primitives*. By such reductions we mean proving that if cryptographic primitive A exists, then cryptographic primitive B exists, by providing:

1. A generic construction of primitive B that uses A as a black-box.
2. A proof that B is secure by showing a generic adversary for primitive A that can be shown to break the security of A, if it is given black-box access to an adversary that breaks B.

Most reductions in cryptography are indeed of this form. The canonical example for such a negative result is the result of Impagliazzo and Rudich [**?**] that it is impossible to reduce a key exchange protocol to one-way functions using a black-box reduction.

Although this paper does not involve the notion of black-box reductions (and non-black-box reductions such as [GMW86, FS89] are already known to exist), the results here may serve as an additional indication that, in general, similarly to relativized results in complexity theory, black-box impossibility results cannot serve as strong evidence toward real world impossibility results.

7.7 Arguments vs. proofs

Our protocol is an *argument system* for **NP** and not a proof system because it is only computationally sound. However, we do not know whether there exists zero-knowledge *proof* for **NP** satisfying Properties 1-5 of Section 1.1. In particular, it is not known whether there exists a constant-round Arthur-Merlin (non-black-box) zero-knowledge *proof* system for **NP**.

7.8 Fiat-Shamir heuristic

The fact that we have shown a constant-round Arthur-Merlin zero-knowledge protocol, can be viewed as some negative evidence on the soundness of the Fiat-Shamir heuristic [FS86]. This heuristic converts a constant-round Arthur-Merlin identification scheme into a non-interactive signature scheme. The prover/sender in the non-interactive scheme uses the same strategy as in the interactive scheme, but the verifier/receiver's messages are computed by applying a public hash function to the message history, and so can be computed by the prover without any need for

interaction.[15] It is known that the resulting signature scheme will be totally *insecure* if the original protocol is zero-knowledge [DNRS99]. Thus, the current work implies that there exist some constant-round Arthur-Merlin protocols on which the Fiat-Shamir heuristic cannot be applied. In a recent work, Goldwasser and Tauman [GT03] used the current work to show a counter example for the Fiat-Shamir heuristic on 3 rounds.

An open question (related to the previous section) is whether or not the Fiat-Shamir heuristic is secure when applied to interactive *proofs* instead of arguments. In particular, an affirmative answer would imply that there does not exist a constant-round Arthur-Merlin zero-knowledge *proof* system for **NP**.

7.9 Number of rounds

Goldreich and Krawczyck [GK90] have shown that no 3 round protocol can be black-box zero-knowledge. We have presented several non-black-box zero-knowledge protocols, but all of them have more than 3 rounds. It is an open question whether there exists a 3-round zero-knowledge argument for a language outside **BPP**.

One bottleneck in reducing the number of rounds in our protocol is our use of a universal argument scheme, for which the best known construction utilizes 4 rounds. Thus, a related open question is whether or not there exist a non-interactive (or even two-round) universal argument scheme. Micali [Mic94] presents a candidate for such a scheme.

7.10 Strict polynomial-time

We have shown the first constant-round zero-knowledge protocol with a *strict* polynomial-time simulator, instead of an *expected* polynomial-time simulator. Another context in which *expected* polynomial-time arises is in constant-round zero-knowledge proofs *of knowledge* (e.g., [Fei90, Chap. 3], [Gol01b, Sec. 4.7.6.3]). In [BL02] Barak and Lindell construct, using the protocol presented here, a constant-round zero-knowledge argument of knowledge with a *strict* polynomial-time extractor. They also show that non-black-box techniques are *essential* to obtain either a strict polynomial-time simulator or a strict polynomial-time knowledge-extractor, in a *constant-round* protocol.

Acknowledgments

First and foremost, I would like to thank Oded Goldreich. Although he refused to co-author this paper, Oded's suggestions, comments, and constructive criticism played an essential part in the creation of this work. I would also like to thank Alon Rosen, Shafi Goldwasser and Yehuda Lindell for very helpful discussions. In particular, it was Alon's suggestion to present the results using the FLS paradigm, a change that considerably simplified the presentation of this work.

References

[BFL91] L. Babai, L. Fortnow, and C. Lund. Nondeterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, 1(1):3–40, 1991. Preliminaryversion in FOCS' 90.

[15] This heuristic is usually applied to 3 round protocols, but it can be applied to any constant-round Arthur-Merlin protocol.

[BM88] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *J. Comput. Syst. Sci.*, 36:254–276, 1988.

[Bar02] B. Barak. Constant-Round Coin-Tossing With a Man in the Middle or Realizing the Shared Random String Model. In *Proc.* 43*rd FOCS*. IEEE, 2002. Preliminary full version available on `http://www.math.ias.edu/~boaz`.

[BG01] B. Barak and O. Goldreich. Universal Arguments and their Applications. Cryptology ePrint Archive, Report 2001/105, 2001. Extended abstract appeared in CCC' 2002.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-Sound Zero-Knowledge and its Applications. Record 2001/063, Cryptology ePrint Archive, Aug. 2001. Preliminary version appeared in FOCS' 01.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahay, S. Vadhan, and K. Yang. On the (Im)possibility of Obfuscating Programs. In *Crypto '01*, 2001. LNCS No. 2139.

[BL02] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. Cryptology ePrint Archive, Report 2002/043, 2002. Extended abstract appeared in STOC' 02.

[BG93] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. *Lecture Notes in Computer Science*, 740:390–420, 1993.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, November 1993.

[Blu82] M. Blum. Coin Flipping by Phone. In *Proc.* 24*th IEEE Computer Conference (CompCon)*, pages 133–137, 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, Oct. 1988.

[BCY89] 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*, pages 192–195, 1989. LNCS No. 434.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. In *Proc.* 32*th STOC*, pages 235–244. ACM, 2000.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc.* 30*th STOC*, pages 209–218. ACM, 1998.

[CKPR01] 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. Extended abstract appeared in STOC' 01.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In *Proc.* 40*th FOCS*, pages 523–534. IEEE, 1999.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero Knowledge. In *Proc.* 30*th STOC*, pages 409–418. ACM, 1998.

[FLS99] Feige, Lapidot, and Shamir. Multiple Noninteractive Zero Knowledge Proofs Under General Assumptions. *SIAM J. Comput.*, 29, 1999.

[Fei90] 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.

[FFS87] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988. Preliminary version in STOC 1987.

[FS89] U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *Crypto '89*, pages 526–545, 1989. LNCS No. 435.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proc. 22nd STOC*, pages 416–426. ACM, 1990.

[FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto '86*, pages 186–194, 1986. LNCS No. 263.

[Gol93] O. Goldreich. A Uniform-Complexity Treatment of Encryption and Zero-Knowledge. *J. Cryptology*, 6(1):21–53, 1993.

[Gol01a] O. Goldreich. Concurrent Zero-Knowledge With Timing, Revisited. Technical Report 2001/104, Cryptology ePrint Archive, Nov. 2001. Extended abstract in STOC' 02.

[Gol01b] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on `http://www.wisdom.weizmann.ac.il/~oded/frag.html` .

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. Preliminary version in FOCS' 84.

[GK96] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *J. Cryptology*, 9(3):167–189, Summer 1996.

[GK90] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, Feb. 1996. Preliminary version appeared in ICALP' 90.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *J. ACM*, 38(3):691–729, July 1991. Preliminary version in FOCS' 86.

[GO87] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *J. Cryptology*, 7(1):1–32, Winter 1994. Preliminary version in FOCS' 87.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. Preliminary version in STOC' 85.

[GT03] S. Goldwasser and Y. Tauman. On the (In)security of the Fiat-Shamir Paradigm. Cryptology ePrint Archive, Report 2003/034, 2003. Extended abstract appeared in FOCS' 03.

[HT99] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. Cryptology ePrint Archive, Report 1999/009, 1999. `http://eprint.iacr.org/`.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc.* 24*th STOC*, pages 723–732. ACM, 1992.

[Kil95] J. Kilian. Improved Efficient Arguments (Preliminary version). In *Crypto '95*, pages 311–324, 1995. LNCS No. 963.

[KP00] J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. Cryptology ePrint Archive, Report 2000/013, 2000. Extended abstract appeared in STOC' 01.

[KPR98] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the Internet. In *Proc.* 39*th FOCS*, pages 484–492. IEEE, 1998.

[Lin03] Y. Lindell. Bounded-concurrent secure Two-party Computation without Setup Assumptions. In *Proc.* 35*th STOC*. ACM, 2003.

[Mic94] S. Micali. CS proofs. In *Proc.* 35*th FOCS*, pages 436–453. IEEE, 1994.

[Nao89] M. Naor. Bit Commitment Using Pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. Preliminary version in CRYPTO' 89.

[PRS92] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. In *Proc.* 33*rd FOCS*. IEEE, 1992.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt '99*, 1999. LNCS No. 1592.

[Ros00] A. Rosen. A Note on the Round-Complexity of Concurrent Zero-Knowledge. In *Crypto '00*, 2000. LNCS No. 1880.

[TW87] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. In *Proc.* 28*th FOCS*, pages 472–482. IEEE, 1987.

[Yao82] A. C. Yao. Theory and Applications of Trapdoor Functions. In *Proc.* 23*rd FOCS*, pages 80–91. IEEE, 1982.

A Universal Arguments

A.1 The CS Proof System

In this section we prove Theorem **??**. We let $T, T' : \mathbb{N} \to \mathbb{N}$ be (polynomial time computable) super-polynomial functions such that $T'(n) = \omega(poly(T'(n))$ for any polynomial $poly(\cdot)$ and such that $T(n) \leq n^{\log n}$. We assume that $T'(n)$-collision resistent hash functions exist and construct a CS Proof system for any $\mathbf{Ntime}(T)$ relation R. Given our tools (PCP and random access hash functions), the idea behind the construction is simple: it uses a random access hash function to force the prover to commit to a single string and answer the questions of the PCP verifier according to this string.

Suppose that $\{H_\alpha\}_{\alpha \in \{0,1\}^*}$ is a $T'(n)$ random access hash function ensemble. Let R be an $\mathbf{Ntime}(T)$ relation with recognizing machine M_R. We construct a CS Proof system for R in the following way:

Construction A.1. A CS proof system for R

- Common input: $x \in \{0,1\}^n$
- Prover's auxiliary input: $y \in \{0,1\}^*$ such that $(x, y) \in R$.
- Verifier's first step (V1): Let $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n$, send α.
- Prover's first step (P1): Using x and y generate a string π_x that always convinces the PCP verifier that $x \in L(R)$ (takes time polynomial in the running time of M_R on (x, y)). Send $\beta \stackrel{def}{=} H_\alpha(\pi_x)$.
- Verifier's second step (V2): Select a random tape γ for the PCP verifier. As γ is of length $n \cdot polylog(T)$ which is $n \cdot polylog(n)$ we can assume that $|\gamma| \leq n^2$. Send γ.
- Prover's second step (P2): Run the PCP verifier for $L(R)$ with γ for random coins. For any query i that the verifier makes send (i, π_i, σ) where σ is the certificate that the i^{th} bit of π is indeed π_i. We denote the message sent that contains all the answers to the queries along with their certificates by δ.
- Verifier accepts if and only if:
  1. The PCP verifier indeed makes those queries when run with random tape γ.
  2. All certificates check out.
  3. The PCP verifier accepts when run with randomness γ and the answers to queries as given by the prover.

The prover's efficiency and completeness conditions are straightforward so what we need to do is to prove the computational soundness and proof of knowledge conditions.

Soundness Condition: Suppose that P^* is a $T(n)$ time algorithm that manages to convince the verifier with non-negligible probability ϵ that $x \in L(R)$. Recall that by the definition of $T'(n)$-collision resistent random access hash ensemble any $poly(T'(n))$ time algorithm has probability at most $\frac{1}{T'(n)}$ of outputting a string β along with two contradicting certificates for what is supposed to be the i^{th} bit of β's preimage under H_α.

As $T'(n) = \omega(poly(T(n))$ for any polynomial *poly* we have that $\frac{T(n)}{T'(n)}$ is a negligible function. Furthermore, $\frac{T(n)^c}{T'(n)}$ is a negligible function for any constant c. Therefore we can assume that $\epsilon \geq \frac{16T(n)^8}{T'(n)}$ as otherwise ϵ is negligible and we're done.

Now suppose we run P^* for the first two steps (V1 and P1), that is, give it a string α and get a response β. We call such a truncated execution a "good start" if P^* has at least an $\epsilon/2$ probability of success if we continue this execution. At least an $\epsilon/2$ fraction of the truncated executions are good.

Let us fix a good start. For any $1 \leq i \leq T(n)$ we define $p_i(0)$ to be the probability that in the rest of the execution if we continue the execution, P^* will output a valid certificate that the i^{th} bit of β's preimage is 0. We define $p_i(1)$ analogously.

We have the following claim:

Claim A.2. *Let us call a good start $\langle \alpha, \beta \rangle$* "problematic" *if there exists an i such that both $p_i(0)$ and $p_i(1)$ (when defined with respect to this start) are bigger than $\frac{2}{\sqrt{\epsilon \cdot T'(n)}}$. At most half of the good starts are problematic.*

Proof: Suppose otherwise. We'll derive a contradiction by constructing a $poly(T(n))$ time algorithm A that will find contradicting certificates for the hash function H_α with probability larger than $\frac{1}{T'(n)}$:

Algorithm A:

- Input: $\alpha \in \{0,1\}^n$
- Step 1: Feed α to P^* and get a response β.
- Step 2:We now try to continue the execution two times. That is perform twice the following experiment: choose $\gamma \leftarrow_{\mathrm{R}} n^2$, feed γ to P^* and get a response δ.
- If there's an $1 \leq i \leq T(n)$ such that A got in the first time a valid certificate that the i^{th} bit of β's preimage is 0, and in the second time a valid certificate that this bit is 1, then we say that A is *successful*.
- In case A is successful it outputs i and the two contradicting certificates.

We want to bound from below the probability of A's success on input α that is chosen uniformly in $\{0,1\}^n$. by our assumption A has probability at least $\frac{\epsilon}{4}$ to hit in step 1 a good start that is problematic. Now suppose that it did hit a problematic good start. This means that there's an $1 \leq i \leq T(n)$ such that both $p_i(0)$ and $p_i(1)$ are greater than $\frac{2}{\sqrt{\epsilon \cdot T'(n)}}$. Consider step 2 of the algorithm. The probability that in its first experiment A will get a valid certificate that the i^{th} bit of β's preimage is 0 is exactly $p_i(0)$. Likewise, the probability that in the second experiment A will get a valid certificate that this bit is 1 is exactly $p_i(1)$. As these experiments are *independent* this means that A's probability of success (conditioned on the start $\langle \alpha, \beta \rangle$) is $p_i(0)p_i(1) \geq \frac{4}{\epsilon T'(n)}$. As A's probability of hitting a problematic good start is at least $\frac{\epsilon}{4}$ the result follows. □

Note that by our assumption on ϵ we see that $\frac{1}{T(n)^4} \geq \frac{2}{\sqrt{\epsilon \cdot T'(n)}}$. Therefore if we make the following definition with respect to a non-problematic good start $\langle \alpha, \beta \rangle$:

$$\pi_i \stackrel{def}{=} \begin{cases} 1 & p_i(1) \geq \frac{1}{T(n)^4} \\ 0 & \text{otherwise} \end{cases}$$

Then we can say that for any i the probability that if we continue this start P^* will output a valid certificate that says that the i^{th} bit is not π_i is at most $\frac{1}{T(n)^4}$. By the union bound we see that the probability that P^* will output a valid certificate that is incompatible with π is a most $\frac{1}{T(n)^3}$.

We have the following claim:

Claim A.3. *Let $\langle \alpha, \beta \rangle$ be any non-problematic good start, if we look at the string π that we defined then* $\Pr[V^{\pi}(x) = 1] \geq \frac{\epsilon}{2} - \frac{1}{T(n)^3} \geq \frac{\epsilon}{4}$.

Proof: Suppose we make the following experiment:

1. Choose $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^{n^2}$
2. Continue the good start with γ as the response of the verifier.
3. If all P^* answers have valid certificates that are compatible with π and V accepts then we say that we have succeeded.

The probability that $V^{\pi}(x)$ accepts is at least the probability that this experiment succeeds. Yet the probability that this experiments succeeds is the probability that P^* succeeds and P^*'s queries are compatible with π. As the overall success probability is at least $\frac{\epsilon}{2}$ the result follows. □

Therefore we see that if $\frac{\epsilon}{4} \geq 2^{-n}$ then there exists a string π such that $\Pr[V^{\pi}(x) = 1] \geq 2^{-n}$ which means that $x \in L(R)$ which is what we wanted to prove.

Proof of Knowledge Condition:

In the soundness condition we needed to prove that if there exists a prover P^* that manages to convince the verifier that $x \in L(R)$ with non-negligible probability ϵ then $x \in L(R)$. Now we need to prove that if this happens then we can actually find a witness $y \in R(x)$. What we've actually proved above is the existence of a string π in this case that convinces the PCP verifier with probability at least 2^{-n} to accept x. If we can show a way to find this string π in $poly(T(n))$ time with $poly(\epsilon)$ probability, then we'll be done, as the PCP itself is a proof of knowledge.

Yet we *can* find the string π: We start by guessing a start $\langle \alpha, \beta \rangle$. With probability $\frac{\epsilon}{4}$ it will be a non-problematic good one. Now in $T(n)^8$ attempts one can recover the string π that is defined above with only 2^{-n} probability of failure (by estimating $p_i(1)$). Therefore we see that we have managed to present a $poly(T(n))$ algorithm with $poly(\epsilon)$ probability of success.