How to Go Beyond the Black-Box Simulation Barrier

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

November 13, 2001

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 proven 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 $\mathbf{NP}$ 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 1 and 2 has been recently proven to be impossible to achieve using black-box simulators.
3. It is an Arthur-Merlin (public coins) protocol.
   Simultaneously obtaining 1 and 3 was known 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 constant-round, negligible-error zero-knowledge arguments utilized *expected* polynomial-time simulators.

Note: This is a preliminary version of the paper. Updated versions will be available from http://www.wisdom.weizmann.ac.il/~boaz.

Contents

1 Introduction

The *simulation paradigm* is one of the most important paradigms in the definition and design of cryptographic primitives. For example, this paradigm occurs 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 the 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 [GMR89]. 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 (the hamiltonian cycle).

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 show and a taped show. This second technique is called *rewinding* because the simulator who fails to answer a question posed by the verifier, simply rewinds the verifier back and tries again.

Most of the known zero knowledge protocols make 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] is closed under concurrent composition [CKPR01]. It seems also very hard to construct a constant-round zero-knowledge proof with a simulator that runs in *strict* polynomial time (rather than *expected* polynomial running time) or a constant-round proof of knowledge with a strict polynomial time knowledge extractor. Indeed, no such proofs were previously known.

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 *any* arbitrary algorithm V^* that represents the verifier's strategy. Almost all the known protocols simply used a *single* simulator that used the algorithm V^* as an oracle (i.e. as a black-box). 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, several negative results have been proved about the power of black-box simulators, starting with the results of Goldreich and Krawczyk [GK96b] 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.

We show that the belief that one can not construct non-black-box simulators is not true. That is, given the code

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

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 $\mathbf{NP}$ that satisfies the following properties:

1. It has a constant number of rounds and negligible soundness error.
2. It remains zero knowledge 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] (the choice of n is quite arbitrary and could be replaced with any *fixed* polynomial (e.g. n^3) in the security parameter.)
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.
5. It is actually an argument of knowledge.

The above protocol should be contrasted with the following impossibility results regarding *black-box* zero-knowledge arguments: Goldreich and Krawczyk have shown that obtaining Properties 1 and 3 together is impossible to achieve when using black-box simulation.[4] In addition, the proofs of Canetti, Kilian, Petrank and Rosen [CKPR01] show that no constant-round protocol is bounded concurrent zero-knowledge (i.e. satisfies Property 2) with a black-box simulator.

Moreover, this is the first zero-knowledge argument that achieves both Properties 1 and 4. The previous state of affairs, where one needed to allow *expected* polynomial time simulators in order to have constant-round zero-knowledge arguments, was rather unsatisfactory and resolving it was suggested as an open problem [Fei90, Chap. 3], [Gol01, Sec. 4.12.3].[5]

1.1 Related Work

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff in [GMR89]. Goldreich, Micali and Wigderson [GMW91] gave a zero-knowledge proof for any language in $\mathbf{NP}$. Constant-round zero-knowledge arguments and proofs were first presented by Feige and Shamir [FS89], Brassard , Crépeau and Yung [BCY89], and Goldreich and Kahn [GK96a].

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

Concurrent zero-knowledge was defined by Dwork, Naor and Sahai [DNS98]. They also constructed a concurrent zero knowledge protocol in the timing model. Richardson and Kilian constructed a concurrent zero-knowledge argument (in the standard model) with polynomially many rounds [RK99]. This was later improved by Kilian and Petrank to polylogarithmically many rounds [KP00]. As mentioned above, Canetti *et al* [CKPR01] (improving on [KPR98] and [Ros00]) show that this is essentially the best one can hope for a protocol using *black-box* simulation.

Some of our techniques (the use of CS proofs) were first used in a similar context by Canetti, Goldreich and Halevi [CGH98]. CS proofs were defined and constructed by Kilian [Kil92], [Kil95] and Micali [Mic94]. Dwork, Naor, Reingold and Stockmeyer [DNRS99] have explored some connections between the existence of non-black-box 3-round zero-knowledge arguments and other problems in cryptography. Some other "non-black-box results" were shown by Barak *et al* [BGI$^+$01] in the context of code obfuscation.

1.2 Organization of this paper

Section 2 contains an overview of the main part in our construction. This section does not contain proofs, and is independent of the rest of the paper.

[2] Actually in this paper we need to 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 as is shown in [BG01].

[3] This is in contrast to a standard concurrent zero-knowledge protocol [DNS98] that remains zero-knowledge when executed concurrently any polynomial number of times.

[4] This is not the only reason that the existence of constant-round Arthur-Merlin zero-knowledge arguments is interesting. Subsequently to this paper, [BGGL01] showed that the existence of such protocols has several implications in the *resettable* model.

[5] Subsequently to this paper, [BL01] showed that it is impossible to obtain a constant-round protocol with a strict polynomial-time *black-box* simulator.

The full construction (along with proofs) is presented in the appendices. See Appendix A for the organization of the appendices.

2 Overview of Our Construction

To give a flavor of our techniques, we present in this section a zero-knowledge argument system for $\mathbf{NP}$ satisfying all the properties mentioned in the introduction, other than being bounded concurrent zero-knowledge and an argument of knowledge.[6] That is:

1. The protocol has a constant number of rounds and negligible soundness error.
2. The protocol is an Arthur-Merlin (public coins) protocol.
3. The protocol has a simulator that runs in *strict* polynomial time, rather than *expected* polynomial time.

Items 1 and 2 together imply that the simulator for this protocol *must* be a non-black-box simulator, because if $\mathbf{NP} \not\subseteq \mathbf{BPP}$ then no such protocol can be black-box zero-knowledge [GK96b]. Therefore, this protocol is *inherently* a non-black-box zero-knowledge argument.

Structure of this Section. Our construction uses a general paradigm, which first appeared in a paper by Feige, Lapidot and Shamir [FLS99], called the FLS paradigm. This paradigm is described in Section 2.1. The FLS paradigm gives us a framework into which we need to plug two components in order to obtain a zero-knowledge protocol. The first component, called a generation protocol, is described in Section 2.2. The construction described there involves some novel ideas, which account for the difference between this zero-knowledge argument and previously known protocols. In Section 2.3 we describe the second component that we need to plug into the framework: this is a witness indistinguishable proof that has some special properties. Our construction in Section 2.3 uses previous constructions of Kilian [Kil92] and Micali [Mic94]. In Section 2.4 we review the protocol obtained by plugging the two components into the framework, and sketch why it is indeed a zero-knowledge argument with the required properties. In Section 2.5 we discuss some technical aspects of the construction and compare it to the full proof that can be found in Sections C-E.

Witness Indistinguishable Proofs. We shall use the notion of *witness indistinguishable* (WI) proofs as defined by Feige and Shamir ([FS90] , see also Section 4.6 of Goldreich's book [Gol01]). Loosely speaking, a witness indistinguishable proof is a proof system where the view of *any* polynomial-size verifier is "computationally independent" of the witness that is used by the (honest) prover as auxiliary (private) input. Specifically, recall that when proving that a string x is in $L(R)$, where R is a binary relation, the prover gets as auxiliary input a *witness* w such that $(x, w) \in R$. If the proof system is witness indistinguishable, then for any w, w' such that $(x, w), (x, w') \in R$, and for *any* polynomial-size verifier V^*, the view of V^* when interacting with the (honest) prover that gets w as auxiliary input, is computationally indistinguishable from its view when interacting with the (honest) prover that gets w' as auxiliary input. Constant-round Arthur-Merlin WI proofs of knowledge are known to exist under the assumption of existence of one-way functions (for example, the parallel version of Blum's hamiltonicity protocol [Blu87]).

Notation For a binary relation R, we let $L(R) \stackrel{def}{=} \{x | \exists w \text{ s.t. } (x, w) \in R\}$. A string w such that $(x, w) \in R$ is called a *witness* or a *solution* for x. In the context of a proof system for a language L, we may refer to a string x as a *theorem*, where by proving the theorem x we mean proving the statement "$x \in L$".

2.1 The FLS Paradigm

Our construction follows the well-known paradigm, introduced by Feige, Lapidot and Shamir, which we call the FLS paradigm[7]. In the FLS paradigm, we convert a witness indistinguishableproof into a zero-knowledge proof by providing the simulator with a *fake witness*. This fake witness allows the simulator to simulate a proof, where the witness indistinguishable property is used to ensure that the verifier cannot tell the difference between the real

[6] The protocol, described in the full version of this paper, that satisfies all 5 properties is a modified version of the protocol described in this section.

[7] This paradigm has been introduced in the context of non-interactive zero-knowledge, but has been used also in the interactive setting (e.g. [RK99]).

- Common Input: x such that $x \in L$, security parameter.
- Auxiliary input to prover: w - a witness to the fact that $x \in L$.

1. *Phase 1: Generation Phase.* The prover and verifier engage in some protocol (to be specified later) whose output is a string τ.
2. *Phase 2: WI Proof Phase.* The prover proves to the verifier using a witness indistinguishable argument of knowledge that it knows either a string w such that $(x, w) \in R$ or a string σ such that $(\tau, \sigma) \in S$.

   More formally, the prover proves to the verifier using a WI proof of knowledge that it knows a witness to the fact that $\langle x, \tau \rangle \in L(S')$ where S' is defined so that $\langle x, \tau \rangle \in L(S')$ if and only if there exists w' such that either $(x, w') \in R$ or $(\tau, w') \in S$.

Figure 1: A zero-knowledge argument for $L(R)$

interaction and the simulated one. Of course, one must ensure that no polynomially bounded prover will be able to obtain such a fake witness, or else the resulting system will no longer be sound.

To be more concrete, the protocol will consists of two phases. In the first phase, called the *generation phase*, the prover and verifier will generate some string, denoted τ. The prescribed prover and verifier both *ignore* their input in performing this stage (and so it can be performed even before knowing what is the theorem that will be proved). In the second stage, called the *WI proof phase*, the prover proves (using a witness indistinguishable proof system) that *either* the theorem is true or that the generated string τ satisfies some property (in our case the property will be that τ is a member of a particular language $L(S)$, for some fixed relation S)[8]. This framework is depicted in Figure 1. For technical reasons we actually need to use a (witness indistinguishable) *proof of knowledge* for the second phase, rather than just a (WI) proof of membership. Also, for our purposes it is enough to use an argument (i.e., a computationally sound proof) rather than a (statistically sound) proof. It should be noted that if both the generation protocol and the WI proof of knowledge are constant-round Arthur-Merlin protocols, then so will be the resulting zero-knowledge argument.

The generation phase will be designed such that the following holds:

1. On the one hand, if the verifier follows its prescribed strategy, then it is *guaranteed* that the string τ does not satisfy the above property (e.g. $\tau \notin L(S)$), or at least that no polynomial-size prover will be able to prove that τ does satisfy the property.

2. On the other hand for *any* (possibly cheating) verifier V^*, there exists a simulator M^* that is able to simulate V^*'s interaction with the (honest) prover during the generation protocol in such a way that not only will the generated string τ satisfy the property, but the simulator M^* will be able to prove that this is the case. As our property will be membership in some language $L(S)$, this means that the simulator will "know" a witness σ such that $(\tau, \sigma) \in S$. This string σ is the *fake witness* that allows the simulator to simulate the second stage.

Previous protocols that used a similar framework employed relatively simple generation protocols, and the simulators used *rewinding* to obtain the solution σ for the generated string τ (see for example [RK99]). In particular, no previous generation protocol was a constant-round Arthur-Merlin protocol, as such a protocol would necessarily have a non-black-box simulator (because otherwise one could have plugged such a protocol into the FLS framework (with a constant-round Arthur-Merlin WI proof of knowledge) and obtain a constant-round Arthur-Merlin protocol that is black-box zero-knowledge).

2.2 Our Generation Protocol

Our generation protocol is a constant-round Arthur-Merlin protocol, and so has a non-black-box simulator. Before presenting the protocol itself, we describe more precisely the conditions that such a protocol should satisfy. For some fixed relation S we require that the protocol satisfies that:

[8] At this point the reader may think of S as an **NP** relation, however in our actual construction we will use a relation S of slightly higher complexity.

1. (Hardness) The prescribed verifier has the property that for *any* (possibly cheating) polynomial-size prover P^*, for τ denoting the string generated by the interaction of the prescribed verifier with P^*, the probability that P^* outputs a solution for τ (i.e., a string σ such that $(\tau, \sigma) \in S$) is negligible.
2. (Easiness) For *any* (possibly cheating) polynomial-size verifier V^*, we require that there exists a simulator M^* that outputs a pair (e, σ) that satisfies the following:
   (a) The first element e is a simulation of the view of V^* in a real execution with the (honest) prover. That is, the view of V^* when interacting with the prescribed prover is computationally indistinguishable from the first element, e. Recall that the *view* of V^* consists of all the messages V^* receives and its random tape. Without loss of generality we assume that this view contains also all messages that V^* sends and so also contains the string τ generated in this execution.
   (b) For τ denoting the string generated in an execution with view e, the string σ is a solution for τ (i.e., $(\tau, \sigma) \in S$).

2.2.1 The basic approach

The general approach in our generation protocol is to try to set up a situation where finding a solution to the generated string τ will be equivalent to predicting one of the verifier's messages before it is sent. We will then obtain the hardness condition by having the prescribed verifier choose this message at random, and so infer that no prover can predict this message with non-negligible probability. In contrast, for *any* (possibly cheating) verifier V^*, the corresponding simulator, knowing (i.e. getting) V^*'s code and random tape *can* predict the messages that V^* sends.

In this subsection, we will construct a generation protocol for which the easiness property holds only with respect to verifiers that, when run with security parameter 1^n, the total length of their code (where by *code* we mean the description of the verifier's Turing machine along with any auxiliary input that the verifier gets) and random tape is at most n^3. We call such verifiers n^3*-bounded* (the choice of n^3 is quite arbitrary). We will later modify the protocol in order to remove this restriction.

Consider a two-round protocol in which the prover first sends a message z and then the verifier responds with a message v. We know that for any (possibly cheating) verifier V^*, the message v is obtained by applying the *next-message function*[9] of V^* , denoted NM_{V^*}, to the message z; that is, $v = NM_{V^*}(z)$. Given the code and random tape of V^*, one can construct in polynomial-time a program that computes the function NM_{V^*}. For n^3-bounded verifiers, the length of this program will be at most n^3, where 1^n is the security parameter (and so we can assume without loss of generality that the length of this program is exactly n^3).

In a protocol of the above type, we say that the message z *predicts* the message v if z is a commitment to a program Π such that $\Pi(z) = v$. That is, z predicts v if there exists a program Π and a string s (serving as coins for the commitment) such that $z = \mathcal{C}(\Pi; s)$ and $\Pi(z) = v$, where $\mathcal{C}$ is some fixed (perfect-binding and computationally-hiding) commitment scheme. We see that for any z, if the message v is chosen uniformly and independently of z in $\{0,1\}^n$, then the probability that $v = \Pi(z)$, where $\Pi = \mathcal{C}^{-1}(z)$ [10], is at most 2^{-n}.

These observations lead us to the following suggestion for a generation protocol:

Protocol 1:

- Input: 1^n - security parameter

1. Prover's first step (P1): Compute $z \leftarrow \mathcal{C}(0^{n^3})$. Send z.
2. Verifier's first step (V1): Choose v uniformly in $\{0,1\}^n$. Send v.

We let the string τ generated by the protocol be its transcript (i.e., $\tau = \langle z, v \rangle$). We define the following relation S_1: for $\tau = \langle z, v \rangle$ and $\sigma = \langle \Pi, s \rangle$ we say that $(\tau, \sigma) \in S_1$ if and only if:

1. z is a commitment to Π using coins s. That is, $z = \mathcal{C}(\Pi; s)$.
2. $\Pi(z) = v$.

[9] Note that once we fix the random tape of V^*, this is a (deterministic) function.

[10] That is, Π is the unique string committed to by z

Clearly, Protocol 1 and the relation S_1 satisfy the hardness property. In fact they satisfy this property even with respect to *computationally unbounded* provers, because (regardless of the prover's strategy) as long as the verifier follows its prescribed strategy, it will be the case that $\tau \notin L(S_1)$ with probability at least $1 - 2^{-n}$. Protocol 1 for S_1 also satisfies the easiness property with respect to n^3-bounded verifiers: Let V^* be such a verifier. The simulator M^* for V^* will compute z to be a commitment (with coins s) for a program Π that computes next message function NM_{V^*} (the simulator will compute such a program Π of length n^3 using the code and random tape of V^*). By the security of the commitment scheme, we know that the distribution of z is computationally indistinguishable from the distribution of the first message of the prescribed prover. Moreover, if we let v be V^*'s reply to z (i.e. $v = NM_{V^*}(z)$) then we know that it holds that $(\langle z, v\rangle, \langle \Pi, s\rangle) \in S_1$. We see that we have a simulator that satisfies both Parts (a) and (b) of the easiness condition. The following observation will be useful for us in the future:

Observation 2.1. *For any n^3-bounded verifier V^*, the simulator M^* defined above actually satisfies a stronger condition than is needed for Part (b) of the easiness condition: not only that it holds that $\Pi(z) = v$, but that if V^* runs in time $t(|z|)$, then on input z, the program Π outputs v* within $t(|z|)^2$ steps.

Observation 2.1 implies that the fact that the simulator M^*'s second output σ is a solution for τ can be verified in time which is some fixed polynomial in the running time of V^*. From now on we shall make this requirement, that the simulator outputs an *efficiently verifiable solution*, a part of the easiness condition.

2.2.2 Difficulties and Resolving them

We have two problems with the construction above:

1. Our simulator works only for n^3-bounded verifiers (i.e., verifiers for which the total size of the code and randomness is at most n^3). In particular this means that when we plug this generation protocol into the FLS frameworkof Figure 1, we will obtain a protocol that is zero-knowledge in some meaningful sense but *not* with respect to verifiers with (polynomial-size) auxiliary input. This is less desirable, because the condition of zero knowledge with respect to auxiliary input is needed for many applications (e.g., for proving preservation of zero-knowledge with respect to sequential composition).
2. As defined above, the relation S_1 is undecidable (this can be seen via a direct reduction from the halting problem). This is a more serious problem, because it implies that when plugging our generation protocol into the FLS frameworkof Figure 1, we will need to use a witness indistinguishable proof for undecidable relations (which does not exist[11]).

The reason for the first problem is that the simulator's first message is a commitment to a program of about the same size as the code and random tape of the verifier, and a perfectly binding commitment cannot reduce the size of its input. Our solution will be to use a collision-resistent hash function $h(\cdot)$, and so have the simulator send a commitment to a *hash* of the program, instead of the program itself. By combining a commitment with a hash function we obtain a commitment scheme that is only *computationally* binding , and so the hardness condition will now hold only against adversaries of bounded computational power. As we want the hash function to be collision-resistent also for non-uniform adversaries, we will need to use a hash function *ensemble* $\{h_\kappa\}_{\kappa\in\{0,1\}^*}$, and so we will have the verifier send the index κ of the hash function to be used as a preliminary step (we assume that h_κ is a function from $\{0,1\}^*$ to $\{0,1\}^{|\kappa|}$). Also, for technical reasons we require that the hardness condition will hold against $n^{\log n}$-size adversaries (rather than just against polynomial-size adversaries), and so we will assume that the hash function remains collision resistent also against $n^{\log n}$-size adversaries[12].

To deal with the second problem, we will change the definition of the relation S_1 to require that on input z, the program Π outputs v *within $n^{\log\log n}$ steps.* This will reduce the complexity of S_1 to being an $\mathbf{Ntime}(n^{\log\log n})$ relation. As a first observation, note that this modification does not harm the hardness condition, because for any string τ, we only eliminate some of the solutions σ that were allowed by the previous definition. This modification also does

[11] This is not entirely accurate, as one might be able to use a WI proof system with some guarantees regarding *instance based complexity* (as opposed to worst-case complexity). In fact, our final approach will be along these lines.

[12] For the sake of simplicity, in this overview section we fix the hardness of the hash function ensemble to be $n^{\log n}$. Actually, as is shown in the full version of this paper, the same analysis works whenever the hardness is a "nice" super-polynomial function (e.g., $n^{\log\log n}$). In [BG01] it is shown that a (slightly more complicated) protocol can be constructed under the more standard assumption that a hash function ensemble exists with arbitrary super-polynomial hardness.

Generation Protocol

- Input: 1^n - security parameter

1. Verifier's first step (V1): Choose $\kappa \leftarrow_{\mathrm{R}} \{0,1\}^n$. Send κ.
2. Prover's first step (P1): Compute $z \leftarrow \mathcal{C}(0^n)$. Send z.
3. Verifier's first step (V2): Choose $v \leftarrow_{\mathrm{R}} \{0,1\}^n$. Send v.

The string outputted by this protocol is its transcript $\tau = \langle \kappa, z, v \rangle$

Definition of relation S: For $\tau = \langle \kappa, z, v \rangle$ and $\sigma = \langle \Pi, s \rangle$ we say that $(\tau, \sigma) \in S$ if and only if:

1. z is a "hashed commitment" to Π using coins s and hash function h_κ. That is, $z = \mathcal{C}(h_\kappa(\Pi); s)$.
2. On input z, the program Π outputs v within $n^{\log \log n}$ steps.

Figure 2: Definitions for the generation protocol and the relation S

not harm the easiness condition: The reason is that, due to Observation 2.1, the solution $\sigma = \langle \Pi, s \rangle$ outputted by the simulator for a string $\tau = \langle z, v \rangle$ will always satisfy that $\Pi(z)$ outputs v within the time complexity of the simulated verifier V^*, and so within less than $n^{\log \log n}$ steps. We will also need to change the definition of S_1 in order to accommodate the hash function. We denote this modified relation by S.

The definitions of the $\mathbf{Ntime}(n^{\log \log n})$ relation S and the generation protocol are shown in Figure 2. We claim that this relation and generation protocol satisfy both the easiness and the hardness condition. Observe that the hardness condition holds against $n^{\log n}$-size provers, rather than just against polynomial-size provers (we will later make use of this fact). Observe also that the generation protocol is a constant-round Arthur-Merlin protocol.

2.3 A Witness Indistinguishable Argument for $\mathbf{Ntime}(n^{\log \log n})$

Suppose that S was an $\mathbf{NP}$ relation. In this case, if we plug the generation protocol of Figure 2 into the FLS frameworkof Figure 1, then the relation S' defined in Phase 2 of this framework will also be an $\mathbf{NP}$ relation. If this is the case then we can use any constant-round Arthur-Merlin WI proof of knowledge for $\mathbf{NP}$ for the second phase, and obtain a zero-knowledge argument system for $\mathbf{NP}$.

Our problem is that the relation S is *not* an $\mathbf{NP}$ relation, but rather an $\mathbf{Ntime}(n^{\log \log n})$ relation. A natural question is whether a WI argument system (with polynomial communication) for $\mathbf{Ntime}(n^{\log \log n})$ relations exists, and if so, does there exist such a system that is a constant-round Arthur-Merlin protocol. The answer to the first question is yes: Kilian [Kil95] has constructed, under suitable complexity assumptions, a constant-round WI argument system for $\mathbf{Ntime}(n^{\log \log n})$ relations. However, his system is not an Arthur-Merlin protocol[13], and so we will need to construct such a system ourselves. In fact we will need an "enhanced" WI argument for $\mathbf{Ntime}(n^{\log \log n})$, that satisfies some extra properties, such as *prover efficiency* and *verifier efficiency* (as defined by Micali in [Mic94]). Let S' be an $\mathbf{Ntime}(n^{\log \log n})$ relation. The properties that we will require (and our construction will fulfill) from a WI argument system for S' will be the following (where Properties 1-3 are the standard definition of WI arguments):

1. (Witness Indistinguishability) For any τ and σ, σ' such that $(\tau, \sigma), (\tau, \sigma') \in S'$, and for any polynomial-size verifier V^*, the view of V^* when interacting with the honest prover that gets σ as auxiliary input is computationally indistinguishable from its view when interacting with the honest prover that gets σ' as auxiliary input.
2. (Perfect Completeness) The honest prover, when given as input $(\tau, \sigma) \in S'$, convinces the prescribed verifier with probability 1 that indeed $\tau \in L(S')$.
3. (Computational Soundness) For any polynomial-size strategy P^*, and any $\tau \notin L(S')$, the probability that P^* can convince the prescribed verifier is negligible.
4. (Verifier Efficiency) The length of all messages and the running time of the prescribed verifier are some fixed polynomial in the security parameter
5. (Prover Efficiency) There is a fixed Turing machine $M_{S'}$ that decides S', such that the honest prover, on input $(\tau, \sigma) \in S'$, runs in time which is some fixed polynomial in the running time of $M_{S'}$ on (τ, σ). That is, the time to prove in the WI system is some fixed polynomial in the time to deterministically verify the solution for τ.

[13] This is not surprising, since his protocol is in fact a black-box zero-knowledge argument system.

6. (Weak Proof of Knowledge) For any polynomial-size prover strategy P^* and any input τ, there exists a $n^{O(\log\log n)}$-time knowledge extractor E such that the probability that P^* can convince the prescribed CS verifier that $\tau \in L(S')$ is polynomially related to the probability that $E(\tau)$ outputs a solution to τ[14].

Sketch of our construction. The main tool that we use to construct our WI argument system is a *universal argument system* [BG01].[15] A universal argument system is a proof system that satisfies Properties 2-6 above (that is, everything apart from the witness indistinguishable property). Universal arguments are a very close variant of CS proofs and efficient arguments as defined and constructed by Kilian [Kil92] and Micali [Mic94] based on the $\mathbf{PCP}$ Theorem([ALM+98, FGL+96, BFLS91, BFL91], the form they use is $\mathbf{NEXP} = \mathbf{PCP}(poly, poly)$). It follows from [Kil92] and [Mic94] that under our complexity assumptions (namely the existence of collision-resistent hash functions strong against $n^{\log n}$ size adversaries) there exists a 4-round Arthur-Merlin universal argument system for any $\mathbf{Ntime}(n^{\log\log n})$ relation(See [BG01] for a full proof). We denote the 4 messages sent in the universal argument by $\alpha, \beta, \gamma, \delta$. We assume without loss of generality that $|\alpha| = |\beta| = |\gamma| = |\delta| = n^2$.

Let S' be an $\mathbf{Ntime}(n^{\log\log n})$ relation. Our construction for a WI argument system for S', described in Figure 3, will consist of two phases. In the first phase, the prover and verifier engage in an "encrypted" universal argument that $\tau \in L(S')$, when τ is the theorem that is to be proved. Specifically, the verifier follows the strategy of the CS verifier (i.e., sends random strings of length n^2), but the prover sends only *commitments* to the actual CS prover's messages. In the second phase, the prover proves (using a standard[16] constant-round WI proof of knowledge for $\mathbf{NP}$) that the transcript consisting of the verifier's messages along with the decommitments of the prover's messages, would have convinced the prescribed CS verifier that $\tau \in L(S')$.

2.4 Plugging Everything In

Let $L = L(R)$ be an $\mathbf{NP}$ language. We claim that if we plug the generation protocol and the relation S of Figure 2 along with the WI argument of Figure 3 (applied to the relation S' defined such that $\langle x, \tau\rangle \in L(S')$ if $x \in L(R)$ or $\tau \in L(S)$) into the FLS frameworkof Figure 1, then we obtain a constant-round Arthur-Merlin zero-knowledge argument for L (with a simulator that runs in strict polynomial time). We will mention some issues that are involved in proving this statement:

- The fact that the honest prover runs in polynomial time is proven using the *prover efficiency* condition of the WI argument system. This follows from the fact that when proving that $x \in L$, the honest prover gets as auxiliary input a witness w such that $(x, w) \in R$. As R is an $\mathbf{NP}$ relation, checking whether $(x, w) \in R$ can be done in time which is some fixed polynomial in $|x|$, say $|x|^c$ for some $c > 0$. Yet the *prover efficiency* condition implies that proving that either $x \in L(R)$ or $\tau \in L(S)$ (i.e. proving that $\langle x, \tau\rangle \in L(S')$) using the WI argument system with auxiliary input w takes time which will be some fixed polynomial in $|x|^c$.
- Computational soundness is proven using the *(weak) proof of knowledge* condition of the WI argument and the *hardness* condition of the generation protocol: By the proof of knowledge condition of the WI argument, any prover that convinces the verifier that $x \in L$ "knows" either a witness w such that $(x, w) \in R$ or a witness σ such that $(\tau, \sigma) \in S$ (where τ is the string generated in the generation phase). Yet the hardness condition of the generation protocol implies that the probability of the latter event is negligible. This means that if the prover convinces the verifier with non-negligible probability that $x \in L$, then the prover must "know" a witness w such that $(x, w) \in R$ and in particular it must be the case that such w exists and so indeed $x \in L$.

  Note that we use the fact that the hardness condition holds against $n^{\log n}$-size adversaries, because the knowledge extractor of our WI argument does not run in polynomial time, but rather in $n^{O(\log\log n)}$ time.
- Zero-knowledge is proved using the *witness indistinguishability* condition of the WI argument and the *easiness* condition of the generation protocol. The simulator for the zero-knowledge protocol uses the simulator of the easiness condition to simulate the first phase and then uses the solution σ provided by this simulator as

[14] Note that we do not require that the probability of E's success in outputting a solution will be the same or close to the probability of P^* convincing the verifier. We only require that these probabilities are polynomially related. Note also that we allow the knowledge extractor to run in time $n^{O(\log\log n)}$ rather than just in polynomial time.

[15] In previous versions of this papers, we used the term CS proofs which is not completely accurate.

[16] Actually we will need to require that the WI proof for $\mathbf{NP}$ satisfies a slightly stronger condition called *strong* witness indistinguishablity (see [Gol01], Section 4.6.1.1). All standard WI proof satisfy this condition, because it is implied by zero-knowledge and it is closed under parallel and concurrent composition.

- Common Input: τ (the theorem to be proven in $L(S')$), 1^n: security parameter
- Auxiliary Input to prover: σ such that $(\tau, \sigma) \in S'$.

1. *Phase 1: "Encrypted" universal argument.*
   (a) Verifier's first step (V1): Choose α uniformly in $\{0,1\}^{n^2}$; Send α.
   (b) Prover's first step (P1):
      i. Initiate CS prover algorithm with (σ, τ).
      ii. Feed α to the CS prover, and obtain its response β. ($\beta \in \{0,1\}^{n^2}$)
      iii. Send a commitment to β: Choose s' uniformly in $\{0,1\}^{n^3}$; Compute $\hat{\beta} \leftarrow \mathcal{C}(\beta; s')$; Send $\hat{\beta}$.
   (c) Verifier's second step (V2): Choose γ uniformly in $\{0,1\}^{n^2}$; Send γ;
   (d) Prover's second step (P2):
      i. Feed γ to the CS prover, and obtain its response δ. ($\delta \in \{0,1\}^{n^2}$)
      ii. Send a commitment to δ: Choose s'' uniformly in $\{0,1\}^{n^3}$; Compute $\hat{\delta} \leftarrow \mathcal{C}(\delta; s'')$; Send $\hat{\delta}$.

2. *Phase 2: An* $\mathbf{NP}$ *WI Proof of Knowledge.*
   (a) Prover and verifier engage in a constant-round Arthur-Merlin witness indistinguishable proof of knowledge that $\langle \tau, \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle \in L(T)$ where T is the following $\mathbf{NP}$ relation[a]:
   $(\langle \tau, \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle, \langle \beta, s', \delta, s'' \rangle) \in T$ if and only if the following holds:
      i. $\hat{\beta}$ is a commitment to β using coins s'. That is, $\hat{\beta} = \mathcal{C}(\beta; s')$.
      ii. $\hat{\delta}$ is a commitment to δ using coins s''. That is, $\hat{\delta} = \mathcal{C}(\delta; s'')$.
      iii. The CS transcript $(\tau, \alpha, \beta, \gamma, \delta)$ convinces the CS verifier that $\tau \in L(S')$.

[a] The fact that T is indeed an $\mathbf{NP}$ relation is implied by the *verifier efficiency* condition of the universal argument. This condition implies that membership in T can be verified in polynomial time.

Figure 3: A Witness Indistinguishable Argument for $L(S')$

an auxiliary input (i.e., a fake witness) to the prover strategy for the WI argument. The simulation will be computationally indistinguishable from a real interaction of the first phase by Part (a) of the easiness condition. The string σ will be a solution to the string τ generated in this simulation by Part (b) of the easiness condition.

The prover efficiency condition for the WI argument, along with the condition stated in Observation 2.1 (i.e., that the simulator for the generation protocol outputs an efficiently verifiable solution), assures us that the simulator for the zero-knowledge protocol will run in (strict) polynomial time. This follows from the same reasons (described above) that the honest prover for the zero-knowledge argument runs in polynomial time[17]

Intuitively, the simulator presented above does not use the *rewinding* technique (although it is probably impossible to give a precise meaning to the phrase "not using the rewinding technique" for a simulator that gets as input the description of the code of the verifier). Rather, it uses the code and random tape of the verifier as a "fake" witness that allows it to simulate the behavior of the honest prover, that gets as input a real witness.

2.5 Discussion

The fact that the relation S defined in Section 2.2 was an $\mathbf{Ntime}(n^{\log\log n})$ relation and not an $\mathbf{NP}$ relation posed us with a problem. We have chosen to solve this problem by constructing a WI proof for $\mathbf{Ntime}(n^{\log\log n})$. Alternatively, we could have used universal arguments to transform the generation protocol for S into a generation protocol for a modified relation S^* that is an $\mathbf{NP}$ relation. This approach has some technical advantages and is the approach that we take in the full proof that can be found in the appendices. The zero-knowledge protocol that is obtained in this way is almost identical to the protocol described above.

In Appendix E, it is shown that a modified version of the protocol described above remains zero-knowledge when executed n times concurrently (where n is the security parameter).

In Section 2.2.1, we considered n^3-bounded verifiers, for which the total size of their code (including auxiliary input) and random tape is at most n^3. In fact, constructing an Arthur-Merlin constant-round argument that is zero-knowledge with respect to such verifiers is already very non-trivial. Indeed, such a protocol would satisfy the original definition of [GMR89] of zero-knowledge with respect to uniform probabilistic polynomial time (with no auxiliary input). The reason is that, assuming the existence of one-way functions, any *uniform* probabilistic polynomial time verifier can be simulated by a modified verifier that uses at most n random coins and expands these coins using a pseudorandom generator. Because the size of the code of a uniform algorithm is finite, the modified verifier will be an n^3-bounded verifier[18]. Therefore if we can simulate n^3-bounded verifiers, we can simulate uniform probabilistic polynomial time verifiers with no auxiliary input. Indeed, in the proof of Appendices C-E we start by presenting a protocol that is zero-knowledge with respect to such verifiers. We do this both because this protocol is simpler, and because the ideas used in that construction are later used for obtaining a protocol that is closed under n concurrent compositions.

3 Conclusions and Future Directions

Reverse-Engineering. Arguably, our simulator does not "reverse-engineer" the verifier's code, although it applies some non-trivial transformations (such as a $\mathbf{PCP}$ 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're not optimistic about such attempts.

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 reexamination of whether these negative results holds also for general (non-black-box) zero-knowledge. Indeed, we have already

[17] Note that, in contrast to the prover that runs in some fixed polynomial time, the simulator will run in time which is some fixed polynomial in the running time of the verifier that is being simulated.

[18] We assume without loss of generality that the security parameter n is larger than the length $|x|$ of the theorem that is proved.

shown in this paper 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.

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.

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 paper, they construct an *argument of knowledge*[19] that is zero-knowledge in the *resettable* model. As noted by [CGGM00], this is trivially impossible in the black-box model.

Black-box reductions. There are also several negative results regarding what can be achieved using black-box *reductions* between two cryptographic primitives (rather than black-box use of an adversary by a simulator). Although this paper does not involve the notion of black-box reductions (and non-black-box reductions such as [GMW91, FS89] are already known to exist), the results here may serve as an additional sign that in general, similarly to relativized results in complexity theory, black-box impossibility results cannot serve as strong evidence toward real world impossibility results.

Fiat-Shamir heuristic. The fact that we've 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 by replacing the verifier's messages with a hash function.[20] It is known that the resulting signature scheme will be totally breakable if the original protocol is zero-knowledge [DNRS99]. Thus, there exist some constant-round Arthur-Merlin protocols on which the Fiat-Shamir heuristic cannot be applied.

Number of rounds. Goldreich and Krawczyck [GK96b] 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 (by optimizing the protocol against adversaries of bounded uniformity one can obtain a 4 round protocol). It is an open question whether there exists a 3-round zero-knowledge argument for a language outside $\mathbf{BPP}$.

Strict polynomial time. We've 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], [Gol01, Sec. 4.7.6.3]). In [BL01] 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

[19] Here and in [BL01] we use a slightly relaxed definition of arguments of knowledge, that allows the knowledge extractor access to the description of the prover, rather than limiting it to using the prover as an oracle.

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

like to thank Alon Rosen, Shafi Goldwasser and Yehuda Lindell for very helpful discussions. In particular, it was Alon's suggestion to use the FLS paradigm to construct the zero-knowledge argument, a change which considerably simplified the construction and its presentation.

References

[AAB+88] Martín Abadi, Eric Allender, Andrei Broder, Joan Feigenbaum, and Lane A. Hemachandra. On generating solved instances of computational problems. In S. Goldwasser, editor, *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 297–310. Springer-Verlag, 1990, 21–25 August 1988.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In Baruch Awerbuch, editor, *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 21–31, New Orleans, LS, May 1991. ACM Press.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991. (Preliminary version in Proc. 31st FOCS.).

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahay, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. To appear in *CRYPTO 2001*, 2001.

[BG01] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In preparation, 2001.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. These proceedings., 2001.

[BL01] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In preparation, 2001.

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

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians, Vol. 1, 2 (Berkeley, Calif., 1986)*, pages 1444–1451, Providence, RI, 1987. Amer. Math. Soc.

[BCY89] Gilles Brassard, Claude Crépeau, and Moti Yung. Everything in NP can be argued in *perfect* zero-knowledge in a *bounded* number of rounds. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 192–195. Springer-Verlag, 1990, 10–13 April 1989.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In ACM, editor, *Proceedings of the 32nd annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, [2000]*, pages 235–244. ACM Press, 2000.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, 23–26 May 1998.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. Record 2001/051, Cryptology ePrint Archive, June 2001. An extended abstract appeared in STOC01.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In IEEE, editor, *40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, pages 523–534. IEEE Computer Society Press, 1999.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 409–418, New York, May 23–26 1998. ACM Press.

[FLS99] Feige, Lapidot, and Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SICOMP: SIAM Journal on Computing*, 29, 1999.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–545. Springer-Verlag, 1990, 20–24 August 1989.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In ACM, editor, *Proceedings of the 22nd annual ACM Symposium on Theory of Computing, Baltimore, Maryland, May 14–16, 1990*, pages 416–426, 1990.

[Fei90] Uri Feige. *Alternative Models for Zero Knowledge Interactive Proofs.* PhD thesis, Weizmann Institute of Science, 1990.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, March 1996.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proc. CRYPTO'86*, pages 186–194. LNCS 263, Springer Verlag, 1986.

[Gol01] Oded Goldreich. *Foundations of Cryptography*, volume 1 – Basic Tools. Cambridge University Press, 2001.

[GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, Summer 1996.

[GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of Zero-Knowledge Proof systems. *SICOMP*, 25(1):169–192, 1996. Preliminary version appeared in ICALP90, pages 268–290.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual Symposium on Theory of Computing (STOC '89)*, pages 25–32, New York, May 1989. ACM Association for Computing Machinery.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the Association for Computing Machinery*, 38(3):691–729, July 1991.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[HT99] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. Cryptology ePrint Archive, Report 1999/009, 1999. `http://eprint.iacr.org/`.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396 (electronic), 1999.

[KPR98] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the Internet. In IEEE, editor, *39th Annual Symposium on Foundations of Computer Science: proceedings: November 8–11, 1998, Palo Alto, California*, pages 484–492, 1998.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In ACM, editor, *Proceedings of the 24th annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 4–6, 1992*, pages 723–732, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer-Verlag, 27–31 August 1995.

[KP00] Joe Kilian and Erez Petrank. Concurrent zero-knowledge in poly-logarithmic rounds. Cryptology ePrint Archive, Report 2000/013, 2000. `http://eprint.iacr.org/`, an extended abstract appeared in STOC01.

[Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, 1(1):1–7, 1965.

[Mer90] Ralph C. Merkle. A certified digital signature. In Giles Brassard, editor, *Advances in Cryptology – CRYPTO ' 89*, Lecture Notes in Computer Science, pages 218–238, Santa Barbara, CA, USA, 1990. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[Mic94] S. Micali. CS proofs. In Shafi Goldwasser, editor, *Proceedings: 35th Annual Symposium on Foundations of Computer Science, November 20–22, 1994, Santa Fe, New Mexico*, pages 436–453. IEEE Computer Society Press, 1994.

[RK99] Richardson and Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1999.

[Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO: Proceedings of Crypto*, 2000.

[STV98] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. ECCC Report TR98-074, 1998. `http://www.eccc.uni-trier.de/eccc/`.

A Guide to the rest of the paper.

In Appendix B, we describe some notations and some cryptographic primitives that we use. One of the primitives described there that we use heavily is universal arguments ([BG01], [Kil92],[Mic94]).

In Appendix C we describe *easy-hard relations* which are the main technical tool we use in the construction of our zero-knowledge arguments. In this Appendix, we describe and define easy-hard relations, show how they can be used to construct zero-knowledge arguments, and present a construction for easy-hard relations against adversaries with bounded non-uniformity. In Appendix D we construct an easy-hard relation against non-uniform adversaries.

Appendix E deals with the issue of concurrent composition. We prove there that variants of the zero-knowledge arguments constructed in Appendices C and D remain zero-knowledge when executed concurrently some fixed polynomial number of times.

Appendix F contains a description of (a variant of) [Kil92] and [Mic94]'s construction of universal arguments, and a sketch of a proof that this construction satisfies *our* definition of universal arguments.

B Preliminaries

B.1 Notation

Relations For a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ we denote $R(x) \stackrel{def}{=} \{w|(x,w) \in R\}$, and $L(R) \stackrel{def}{=} \{x|\exists_w \text{ s.t. } (x,w) \in R\}$. If (x,w) is a pair we will sometimes call x an *instance*. In the context of a proof system for $L(R)$, we may call x a *theorem*. If $(x,w) \in R$ then we say that w is a *solution* or a *witness* for x.

Cryptographic protocols and computational model for adversary. In this paper we construct and use several cryptographic protocols. Each such protocol involves two or more parties (in this paper all the protocols will be two-party protocols). The *prescribed* / *honest* strategy for each party is described by a uniform probabilistic polynomial time algorithm. As these protocols are cryptographic, they will usually provide some guarantee as to what happens if one of the parties is played by an adversary that cheats and deviates from its prescribed strategy. This guarantee will hold no matter what strategy the adversary uses, as long as this strategy can be computed in some model.

In order to capture all computational models that we will use in this paper, we use the model of Turing machines with an advice string. We model all algorithms in this paper, both interactive and non-interactive, as a pair $(M, \{A_n\}_{n\in\mathbb{N}})$ of a Turing machine and a sequence of strings. The Turing machine M has access to two additional tapes (other than all other tapes for work, input , output and communication). The first tape is the *random tape*; Each slot in the random tape contains a value in $\{0,1\}$ that is chosen uniformly and independently from the values of all other slots. The second tape is the *advice tape*; When run on input of size n (or for an interactive machine, when run with security parameter n), the advice tape contains the string A_n.

For three functions $t, r, a : \mathbb{N} \to \mathbb{N}$ we say that an algorithm $(M, \{A_n\}_{n\in\mathbb{N}})$ is (t, r, a) bounded if $|A_n| \leq a(n)$ and on input of size n (or for an interactive machine, when run with security parameter n) the machine M runs for at most $t(n)$ steps, and examines at most $r(n)$ slots in its random tape. We denote the set of such algorithms by $\mathbf{TRA}(t, r, a)$.

We denote by $\mathbf{ALG}$ the set of all such algorithms. That is $\mathbf{ALG} \stackrel{def}{=} \cup_{t,r,a}\mathbf{TRA}(t, r, a)$ where t, r, a ranges over all possible functions. We denote by $\mathbf{PPT}$ the set of *uniform* probabilistic polynomial time algorithms. That is, $\mathbf{PPT} \stackrel{def}{=} \cup_{c,d>0}\mathbf{TRA}(n^c, n^d, 0)$. We'll use the shorthand notation $\mathbf{PPT} = \mathbf{TRA}(poly, poly, 0)$. We denote by $\mathbf{Psize}$ the set of polynomial size algorithms $\mathbf{Psize} \stackrel{def}{=} \mathbf{TRA}(poly, poly, poly)$. It is well known that an equivalent representation to a $\mathbf{Psize}$ algorithm is a sequence of probabilistic polynomial size circuits. It is also well known that in most cases a probabilistic $\mathbf{Psize}$ algorithm can be simulated by a deterministic $\mathbf{Psize}$ algorithm (that is, in most cases, whatever can be done by an algorithm in $\mathbf{Psize}$ can be done by an algorithm in $\mathbf{TRA}(poly, 0, poly)$). For a particular function $T' : \mathbb{N} \to \mathbb{N}$ we define $\mathbf{Size}(T') \stackrel{def}{=} \mathbf{TRA}(T', T', T')$.

We have the following inclusions:

$$\mathbf{PPT} \subseteq \mathbf{Psize} \subseteq_{quasi} \mathbf{Size}(T') \subseteq \mathbf{ALG}$$

where $T' : \mathbb{N} \to \mathbb{N}$ is any super-polynomial function (e.g. $T'(n) = n^{\omega(1)}$), by $\subseteq_{quasi}$ we mean that any $\mathbf{Psize}$ algorithm can be simulated by a $\mathbf{Size}(T')$ algorithm.

Our standard model for adversaries in this paper will be $\mathbf{Psize}$, while the honest / prescribed strategy will always be a $\mathbf{PPT}$ algorithm. Although we cannot assume anything about the adversary's strategy (other than the bound on its computational power) we can assume without loss of generality that each message the adversary sends satisfies some syntactical conditions (such as being of the proper length). The reason is that we can stipulate as part of the protocol's definition that messages that fail to satisfy these conditions will be interpreted as some canonical valid message.

Standard Notation. If A is a probabilistic algorithm then we denote by $A(x; R)$ the output of A when executed on input x and randomness R. We denote by $A(x)$ the random variable obtained by picking R uniformly and outputting $A(x; R)$. We'll sometimes identify an *interactive* algorithm A with its *next message function*: the next message function of A is the function that given A's random tape and all messages A has received up until the current point, outputs A's next message. The *view* of an interactive algorithm in an execution of a protocol, consists of its random tape and all messages received by the algorithm.

Universal Turing Machine. We denote by $\mathcal{U}$ a fixed *universal Turing machine*. That is, for any program Π (i.e., Π is a description of a Turing machine), and for $x_1, \ldots, x_k \in \{0,1\}^*$, $\mathcal{U}(\Pi, x_1, \ldots, x_k)$ is the output of the program Π on input $x_1, \ldots, x_k$. We assume that if $\Pi(x_1, \ldots, x_k)$ halts in n steps, then $\mathcal{U}(\Pi, x_1, \ldots, x_k)$ halts in $n^{1.1}$ steps.

B.2 Variants of Zero Knowledge Proofs

We shall use the standard definitions for zero-knowledge and witness indistinguishable proofs and arguments, and for *honest verifier* zero-knowledge (see [Gol01] for these definitions). We shall use a *non-standard* definition of a proof of knowledge. The standard condition for a proof of knowledge is that there exists an expected polynomial-time

knowledge extractor, whose probability of success in outputting a witness is identical (up to a negligible factor) to the probability of the prover's success in convincing the verifier. We shall make the definition stronger in one aspect, and weaker in another. On the one hand, we will require that the knowledge extractor will run in *strict* polynomial time. On the other hand, we will only require that if the prover's success probability was non-negligible, then so will be the extractor's success probability. The resulting definition will be the following:

Definition B.1. A proof system $\langle \text{PROVE}, \text{VERIFY} \rangle$ for a language $L = L(R)$ is a *proof of knowledge* if it satisfies the following condition: there exists a polynomial time oracle machine EXT, a negligible function $\mu : \mathbb{N} \to [0,1]$ and a constant $c > 0$ such that for any prover P^* , string $x \in \{0,1\}^*$ and security parameter 1^n, if we let

$$\epsilon \stackrel{def}{=} \Pr[\langle P^*, \text{VERIFY} \rangle(x, 1^n) = 1]$$

(that is, ϵ is the probability that P^* convinces the prescribed verifier that $x \in L$)

then it holds that

$$\Pr[\text{EXT}^{P^*}(x, 1^n) \in R(x)] \geq \epsilon' \stackrel{def}{=} (\epsilon - \mu(n))^c$$

In particular this means that if ϵ is non-negligible then so is ϵ'.

We state the following claim:

Claim B.2. *The n-time parallel version of Blum's hamiltonicity protocol ([Blu87]) is a proof of knowledge with constant $c \stackrel{def}{=} 2$ and negligible function $\mu(n) \stackrel{def}{=} 2^{-n}$.*

This means that assuming the existence of one-way functions, there exists a constant-round Arthur-Merlin WI proof of knowledge for $\mathbf{NP}$.

B.3 Standard Commitment Schemes

To keep things simple we give here a definition for a simple commitment scheme, which is known to exist under the assumption of existence of one-way permutations ([GL89] , [Blu82]).

Definition B.3. A *simple bit commitment scheme* is a function sequence $\{\mathcal{C}_n\}_{n \in \mathbb{N}}$ where $\mathcal{C}_n : \{0,1\} \times \{0,1\}^n \to \{0,1\}^{n+1}$ that satisfies:

1. (Efficient Evaluation) There exists a polynomial time Turing Machine $EVAL$ such that for any $n \in \mathbb{N}$, $\sigma \in \{0,1\}$, $r \in \{0,1\}^n$, it holds that $EVAL(1^n, \sigma, r) = \mathcal{C}_n(\sigma; s)$.
2. (Binding) For any n , there do not exist r, r' such that $\mathcal{C}_n(0; r) = \mathcal{C}_n(1; r')$.
3. (Secrecy) $\mathcal{C}(0; U_n)$ and $\mathcal{C}(1; U_n)$ are computationally indistinguishable by polynomial size ($\mathbf{Psize}$) algorithms.

As usual we sometimes denote the random variable $\mathcal{C}_n(b; U_n)$ by $\mathcal{C}_n(b)$. We may sometime drop the index n and denote $\mathcal{C}$ instead of $\mathcal{C}_n$ if n can be inferred from the context. For $x = x_1 \ldots x_k \in \{0,1\}^k$ (where $x_1, \ldots, x_k \in \{0,1\}$) and $r = r^1 \ldots r^k \in \{0,1\}^{nk}$ (where $r^1, \ldots, r^k \in \{0,1\}^n$) we denote $\mathcal{C}_n(x_1; r^1) \ldots \mathcal{C}_n(x_k; r^k)$ by $\mathcal{C}_n(x; r)$. We'll also denote the random variable $\mathcal{C}_n(x_1) \ldots \mathcal{C}_n(x_k)$ by $\mathcal{C}_n(x)$.

B.4 Collision Resistant Hash Functions

We shall use an ensemble of hash functions $\{h_\kappa\}_{\kappa \in \{0,1\}^*}$ such that if κ is chosen uniformly in $\{0,1\}^n$ then it is infeasible to find $x \neq x'$ such that $h_\kappa(x) = h_\kappa(x')$.

Definition B.4. Let $\{h_\kappa\}_{\kappa \in \{0,1\}^*}$ be a function ensemble: for any κ , $h_\kappa : \{0,1\}^* \to \{0,1\}^{l(|\kappa|)}$ for some function $l(\cdot)$ where $l(n)$ is polynomially related to n. Let $T' : \mathbb{N} \to \mathbb{N}$ be a (polynomial time computable) super-polynomial function (i.e. $T'(n) = n^{\omega(1)}$). We say that $\{h_\kappa\}_{\kappa \in \{0,1\}^*}$ is a $T'(n)$-*collision resistant hash function ensemble* if for any algorithm $A \in \mathbf{Size}(T')$:

$$\Pr_{\kappa \in \{0,1\}^n}[(x, x') \leftarrow A(\kappa); x \neq x' \wedge h_\kappa(x) = h_\kappa(x')] = negl(n)$$

.

B.5 universal argument Systems

universal arguments were defined and constructed by Kilian and Micali ([Kil92],[Mic94]). A universal argument system is a proof system that has low communication complexity and verifier running time (polynomial in the security parameter and polylogarithmic in the deterministic verification time) and so is suitable for proving membership in languages with complexity higher than $\mathbf{NP}$. Note that a universal argument system is not required to give any security guarantees for the prover (i.e., it is not zero-knowledge or witness indistinguishable). Let $T(\cdot)$ be a super-polynomial function and let S' be an $\mathbf{Ntime}(T)$ relation. A universal argument system for S' satisfies the following properties (we let τ be the theorem to be proved and 1^n be the security parameter):

1. (Perfect Completeness) There's a prover strategy that when given as input $(\tau, \sigma) \in S'$ convinces the prescribed verifier with probability 1 that indeed $\tau \in L(S')$.

2. (Computational Soundness) For any polynomial-size strategy P^*, and any $\tau \notin L(S')$, the probability that P^* can convince the prescribed verifier is negligible.

3. (Verifier Efficiency) The length of all messages and the running time of the prescribed verifier have polylogarithmic dependence in $T(|\tau|)$ and a fixed polynomial dependence on n.

4. (Prover Efficiency) For a fixed Turing machine $M_{S'}$ that decides S', there's a prover strategy (satisfying perfect completeness) that on input $(\tau, \sigma) \in S'$ runs in time which is some fixed polynomial in the running time of $M_{S'}$ on (τ, σ). That is, the time to prove in the WI system is some fixed polynomial in the time to deterministically verify the solution for τ.

5. (Weak Proof of Knowledge) For any $T(n)^{O(1)}$ prover strategy P^* and any input τ, there exists a $T(n)^{O(1)}$ knowledge extractor E such that if P^* can convince the prescribed CS verifier that $\tau \in L(S)$ with non-negligible probability, then $E(\tau)$ outputs a solution to τ with non-negligible probability[21]. Specifically, there exists an oracle $T(n)^{O(1)}$ algorithm EXT, a negligible function $\mu : \mathbb{N} \to [0, 1]$ and a constant $c > 0$ such that for any $T(n)^{O(1)}$ algorithm P^*, input τ and security parameter 1^n, if we let ϵ be the probability that P^* convinces the CS verifier that $\tau \in L(S')$, then

$$\Pr[\text{EXT}^{P^*}(\tau, 1^n) \in S'(\tau)] \geq \epsilon' \stackrel{def}{=} (\epsilon - \mu(n))^c$$

We have the following theorem:

Theorem B.5 (Following [Kil92],[Mic94]). *Let $T : \mathbb{N} \to \mathbb{N}$ be a (polynomial time computable) super-polynomial function such that $T(n) \leq 2^n$. Let $T'(n)$ be a function such that $T'(n) = \omega(poly(T(n)))$ for any polynomial $poly(\cdot)$. If there exists a $T'(n)$-collision resistent hash function ensemble then there exists a universal argument system for any $\mathbf{Ntime}(T)$ relation R. Moreover this proof system is a 4 round Arthur-Merlin (public coins) proof system.*

A proof sketch for Theorem B.5 is in Appendix F.

C Strong Easy-Hard Relations

In this section we construct a new zero-knowledge argument with the following properties:

1. It is zero-knowledge with respect to probabilistic polynomial time adversaries with *bounded non-uniformity*, a concept that we will define later. In particular, it is zero-knowledge with respect to uniform probabilistic polynomial-time adversaries.

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.

[21] Note that we do not require that the probability of E's success in outputting a solution will be the same or close to the probability of P^* convincing the verifier. We only require that if the latter is non-negligible then so will be the former.

In Section E we will show that a variant of this argument remains zero-knowledge when executed concurrently n times (where n is the security parameter).

The main technical tool we will use in our construction is *strong easy-hard relations*. To help understand strong easy-hard relations, and the role they play in constructing a zero-knowledge argument, we will start by introducing in Section C.1 a simpler concept: *weak easy-hard relations*. While weak easy-hard relations can be easily constructed from any one-way function, the construction of strong easy-hard relations is the main technical contribution of this paper.

A rough description of (strong) easy-hard relation is the following. An easy-hard relation is a search problem that is *hard on the average* in the sense that there exists a generator that generates instances for which finding solutions in infeasible for any efficient algorithm. However, in the "simulated world" this search problem is *easy on the average* in the following strong sense: *any* efficient generation algorithm can be simulated by an algorithm that outputs a generated instance along with a solution for this instance.

Structure of this section. In Section C.1 we introduce *weak* easy-hard relations and show how they can be used to construct *honest-verifier* zero-knowledge arguments. Although we will not use any result from Section C.1, the definitions and results presented there serve to illustrate some of the ideas behind the results in the sequel. In Section C.2 we define *strong* easy-hard relations and show how a strong easy-hard $\mathbf{NP}$ relation can be used to construct a general (not just honest-verifier) zero-knowledge argument. Such a relation is constructed in Sections C.3 and C.4: In Section C.3 we construct a strong easy-hard relation that is not an $\mathbf{NP}$ relation but only an $\mathbf{Ntime}(n^{\omega(1)})$ relation; In Section C.4 we show that under suitable assumptions, we convert such a relation to a (slightly relaxed notion of) strong easy-hard relation that is an $\mathbf{NP}$ relation. Finally, in Section C.5, everything is pieced together to obtain the zero-knowledge argument with the desired properties.

As mentioned above, the strong easy-hard relations (and so the zero-knowledge argument) of this section work only against adversaries with bounded non-uniformity (a notion that will be defined later). In Section D we will show a strong easy-hard relation (and as a consequence, a zero-knowledge argument) that works against any polynomial-size algorithm.

Figure 4 shows the general structure of our construction of zero-knowledge arguments. The boxes represent primitives and the arrows represent theorems of the form "if primitive X exists then primitive Y exists".

C.1 A Toy Example: Weak Easy-Hard Relations

In this section we define *weak easy-hard relations* and use these relations (as well as (known) witness indistinguishable proofs of knowledge) to construct an *honest-verifier* zero-knowledge argument. Recall that a proof or argument is called *honest-verifier* zero-knowledge if a simulator exists that can simulate the view of the *honest* verifier when interacting with the (honest) prover. This is a "toy example" in the sense that one can obtain a honest-verifier zero-knowledge argument in much simpler ways. The reason that we present this construction is that the ideas behind it will be useful in the construction of strong easy-hard relations, and in the construction of general (i.e. not just honest verifier) zero-knowledge arguments from the former.

Figure 5 shows the general structure of the honest-verifier zero-knowledge argument using weak easy-hard relations. It may be beneficial to compare it to Figure 4.

A binary relation S can be looked at as a search problem: given an instance τ, the problem is to find a solution σ such that $(\tau, \sigma) \in S$. Intuitively we call S *weak easy-hard* if it satisfies two properties: *hardness* and *easiness*. On the one hand, we require that the search problem corresponding to S is *hard* in the sense that there exists a generator (denoted GEN) such that no polynomial-size algorithm is able to solve the problem on instances generated by GEN. On the other hand we require that solving the search problem is *easy* for the party supplying the generator GEN with its randomness. By this we mean that there exists a polynomial time algorithm M that output pairs (r, σ) such that $(\text{GEN}(1^n; r), \sigma) \in S$. That is, since M determines the random tape r of GEN, the algorithm M *can* solve the search problem for the instance that GEN outputs with randomness r.

Note that we do not make the slightly stronger requirement that the simulator is able, given a string r, to compute a solution σ to $\tau = \text{GEN}(1^n; r)$[22]. Rather, we allow M to generate a pair (r, σ) by itself. We also do not require that

[22]This requirement is equivalent to the requirement that the generator GEN itself always outputs a pair (τ, σ) where $(\tau, \sigma) \in S$. The latter concept has been called before by the name an *invulnerable generator* ([AAB+88], [FS89]).

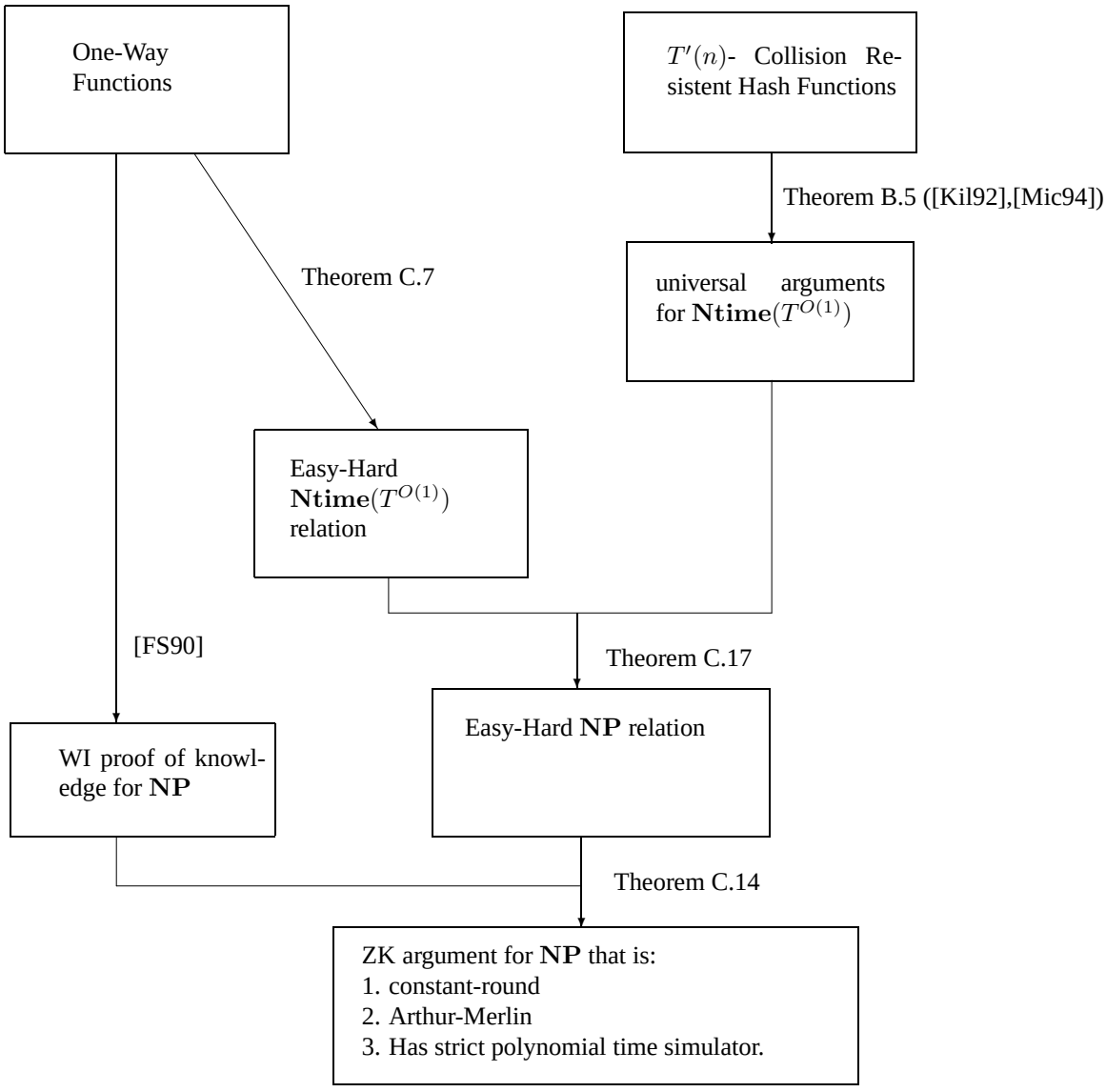

Figure 4: Structure of the construction of a bounded non-uniformity zero-knowledge argument

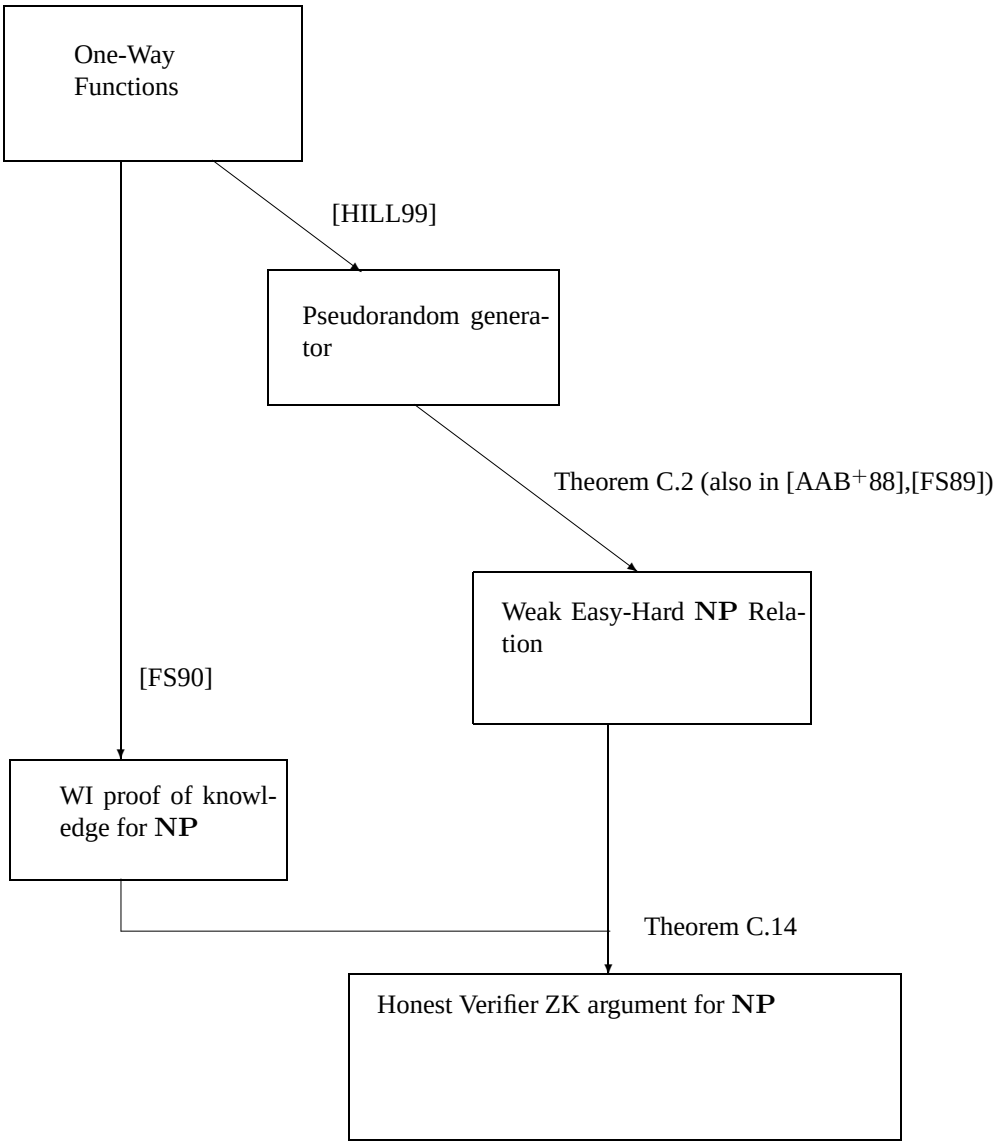

Figure 5: Structure of the construction of an honest-verifier zero-knowledge argument

the first part of M's input (i.e., the string r) is *truly random* (i.e., distributed uniformly among all strings of the proper length). We will require, however, that it is *pseudorandom*.

Definition C.1. Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We say that S is a *weak easy-hard relation* if it satisfies the following properties:

1. (Hardness) There exists a $\mathbf{PPT}$ algorithm GEN such that if $\tau \leftarrow \text{GEN}(1^n)$ then it is infeasible for any $\mathbf{Psize}$ adversary to find a witness σ such that $(\tau, \sigma) \in S$. Formally we say that for any algorithm $A \in \mathbf{Psize}$ and for any $n \in \mathbb{N}$
$$\Pr_{\tau \leftarrow \text{GEN}(1^n)}[A(\tau) \in S(\tau)] = negl(n)$$

2. (Easiness) We require that there exists a $\mathbf{PPT}$ simulator M that is able to simulate an execution of GEN (where GEN is the generator guaranteed to exist by Item 1), but output along with the simulated execution a *witness* for the element that GEN outputs in this execution. That is, we require that on input 1^n, algorithm M outputs a pair (e, σ) of the following form:

   (a) The first e is a pseudorandom string of suitable length (i.e, the length of e will be the maximum number of coins that GEN tosses on input 1^n) .

   (b) If $\tau \stackrel{def}{=} \text{GEN}(1^n; e)$ then $(\tau, \sigma) \in S$.

The existence of weak easy-hard relations is implied by the existence of one-way functions. This is implicit in [AAB$^+$88], [FS89]. We present here a different proof, which lends itself to generalization (for creating strong easy-hard relations).

Theorem C.2. *Suppose that (non-uniformly strong) one-way functions exist, then there exists a weak easy-hard* $\mathbf{NP}$ *relation.*

Proof. Suppose that (non-uniformly strong) one-way functions exist. Then there exists a (non-uniformly strong) pseudorandom generator PRG that triples its input ([HILL99]). That is, for any $s \in \{0,1\}^n$, $|PRG(s)| = 3n$. We define the following relation S_{PRG}: we say that $(\tau, \sigma) \in S_{\text{PRG}}$ if and only if $\tau = \text{PRG}(\sigma)$. We claim that S_{PRG} is a weak easy-hard relation:

1. *Hardness:* Consider the following generator GEN: on input 1^n, the generator GEN outputs τ where τ is selected uniformly in $\{0,1\}^{3n}$. That is, on input 1^n, the generator GEN uses a random string of length $3n$ and simply outputs this random string (i.e., $\text{GEN}(1^n; \tau) = \tau$).

   With very high probability (at least $1 - 2^{-2n}$), the string τ selected uniformly in $\{0,1\}^{3n}$ is *not* a member of $L(S_{\text{PRG}})$. Therefore, no algorithm (and in particular no efficient algorithm) will be able to output a string σ such that $(\tau, \sigma) \in S_{\text{PRG}}$.

2. *Easiness:* Consider the following simulator M: on input 1^n, algorithm M selects $\sigma \leftarrow_{\text{R}} \{0,1\}^n$, lets $\tau = \text{PRG}(\sigma)$ and outputs (τ, σ). On the one hand, the string τ is pseudorandom because PRG is a pseudorandom generator. On the other hand, it is clear that $\tau = \text{GEN}(1^n; \tau)$ and that $(\tau, \sigma) \in S_{\text{PRG}}$.

Note that our construction relied on the fact that the distribution of τ in the simulation is statistically very far (although it is computationally indistinguishable) from its distribution in the actual generation, where with in general τ will not have a solution.

□

As noted above, we can use *weak* easy-hard relations to construct *honest-verifier* zero-knowledge arguments. This is stated in Theorem C.3. The motivation behind the definition of *strong* easy-hard relations is trying to generalize this theorem and its proof to obtain general (not only honest-verifier) zero-knowledge arguments. Therefore, although the statement of Theorem C.3 is not very interesting, understanding its proof is important for motivating the definition of strong easy-hard relations and understanding the construction of our zero knowledge argument.

Theorem C.3. *If weak easy-hard* $\mathbf{NP}$ *relations exists, and there exists a witness indistinguishable proof of knowledge for any language in* $\mathbf{NP}$*, then there exists an* honest verifier *zero-knowledge argument for any language in* $\mathbf{NP}$*. The number of rounds in this argument is at most one more than the number of rounds in the witness indistinguishable proof.*

Proof. This construction uses a technique introduced by Feige, Lapidot and Shamir ([FLS99]) in order to transform a witness indistinguishable proof into a zero-knowledge proof. We suppose that S is a weak easy-hard $\mathbf{NP}$ relation with generator GEN and simulator M.

Let R be an $\mathbf{NP}$ relation and $L = L(R)$. Our protocol for a zero knowledge argument for L will go as follows:

Construction C.4. *A honest-verifier zero-knowledge argument for* $L(R)$

- Common Input: the statement to be proved x, the security parameter 1^n.
- Auxiliary input to prover: w such that $(x, w) \in R$.

1. *Phase 1: Generation*
   - Verifier's first step: Let $\tau \leftarrow \text{GEN}(1^n)$, send τ.
2. *Phase 2:*
   - Prover's and verifier's next steps: The prover proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x, w) \in R$ or σ such that $(\tau, \sigma) \in S$.
     More formally, let R' be the $\mathbf{NP}$ relation defined as follows $(\langle x, \tau\rangle, w') \in R'$ iff $(x, w') \in R$ or $(\tau, w') \in S$. The prover proves, using the witness indistinguishable proof of knowledge with input $(\langle x, \tau\rangle, w)$, that $\langle x, \tau\rangle \in L(R')$ and that it knows a witness to that fact.
     The verifier acts according to the witness indistinguishable protocol and will accept if and only if this proof is valid.

In order to prove that this is a honest-verifier zero-knowledge argument, one needs to prove three properties:

1. *Completeness:* It is clear that whenever the prover is given (x, w) such that $(x, w) \in R$, and follows its prescribed strategy, the honest verifier will accept at the end of the execution.
2. *Computational Soundness:* If any polynomial-size prover is able to convince the honest verifier that $x \in L(R)$ with non-negligible probability then this prover can be modified (using the knowledge extractor associated with the witness indistinguishable proof) to output a $\mathbf{NP}$-witness to the fact that either $x \in L(R)$ or $\tau \in L(S)$. Because S is a weak easy-hard relation (and τ is generated by the honest verifier using its prescribed generator τ) we know that the probability of the latter event (the modified prover outputting a solution to τ) is negligible. It follows that with non-negligible probability the prover can be modified to output a $\mathbf{NP}$-witness to the fact that $x \in L(R)$, and so in particular $x \in L(R)$.
3. *Honest verifier zero-knowledge:* We define a simulator M' that will simulate the view of the honest verifier in this protocol. The simulator M' will use the simulator M guaranteed to exist by the easiness property of the weak easy-hard relation R.

   Algorithm M'
   - Input: the statement x, the security parameter 1^n.

   (a) Let $(e, \sigma) \leftarrow M(1^n)$. We know that for $\tau = \text{GEN}(1^n; e)$, it holds that $(\tau, \sigma) \in S$.
   (b) Run the program for the honest verifier, using e as the random coins used by its first phase. Compute the first message $\tau = \text{GEN}(1^n; e)$ that is outputted by this program.
   (c) Run the prover program for the witness indistinguishable proof to prove that either $x \in L(R)$ or $\tau \in L(S)$ (i.e. that $\langle x, \tau\rangle \in L(R')$). Give the pair $(\langle x, \tau\rangle, \sigma)$ to the prover program as input. To simulate the verifier's messages M' will use the actual verifier program.

The fact that simulated view is computationally indistinguishable from the view in the actual interaction is implied by the witness indistinguishability property of the proof system, and by the fact that e is pseudorandom.

□

C.2 Defining Strong Easy-Hard Relations

The proof of Theorem C.3 suggests how we can strengthen the requirements of weak easy-hard relations to obtain general zero-knowledge arguments (instead of just honest-verifier zero-knowledge arguments). Recall that the *verifier* in the constructed argument used the *generator* of the easy-hard relation. However, the easiness property of the relation only guaranteed a simulator for the prescribed generator. It is this limitation that precluded us from simulating any (possibly cheating) verifier. We conclude that to obtain a general zero-knowledge argument from this construction we may need to require that a simulator (as in the easiness condition) will exist for *any* efficient generator and not just for the prescribed generator GEN. That is,

Definition C.5. Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We say that S is a *strong easy-hard relation* if it satisfies the following properties:

1. (Hardness - same as in weak easy-hard relations): We require that there exists a $\mathbf{PPT}$ algorithm GEN such that if $\tau \leftarrow \text{GEN}(1^n)$ then it is infeasible for any $\mathbf{Psize}$ adversary to find a witness σ such that $(\tau, \sigma) \in S$.

2. (Easiness) We require that for any (possible cheating) $\mathbf{Psize}$ generator G^* there exists a $\mathbf{Psize}$ simulator M^* that is able to simulate an execution of G^*, but output along with the simulated execution a *solution* for the element that G^* outputs in this execution.

   That is, similarly to the easiness condition of weak easy-hard relations, we require that on input 1^n (where n is sufficiently large), algorithm M^* outputs a pair (e, σ) such that e is pseudorandom and for $\tau = G^*(1^n; e)$, it holds that $(\tau, \sigma) \in S$.

   We also require that there exists an *effective transformation* that converts the generator G^* to the simulator M^*. That is, there exists a (uniform) polynomial time algorithm that given the code of G^*, outputs the code of M^*. By the *code* of G^* we mean both the description of its Turing machine and the contents of its advice tape for the current security parameter. In particular this requirement implies that for any $G^* \in \mathbf{PPT}$ there exists a simulator $M^* \in \mathbf{PPT}$.

As noted above, we have the following theorem:

Theorem C.6. *If strong easy-hard* $\mathbf{NP}$ *relations exist, and there exists a witness indistinguishable proof of knowledge for any language in* $\mathbf{NP}$*, then there exists a zero-knowledge argument for any language in* $\mathbf{NP}$*. The number of rounds in this argument is at most one more than the number of rounds in the witness indistinguishable proof.*

Theorem C.6 is proven by using the same construction used in the proof of Theorem C.3 and by treating any (possibly cheating) verifier as a (possibly cheating) generator for the strong easy-hard relation. We omit the details of the proof because it uses the same construction as the one used the proof of Theorem C.3 and we will later show a more general theorem (Theorem C.14).

Comment: The statements of both the easiness and hardness properties involve quantification over the set $\mathbf{Psize}$ of polynomial size algorithms. However, one can obtain an analogous definition by quantifying other some other sets (e.g. $\mathbf{PPT}$, $\mathbf{ALG}$). One can also obtain a meaningful definition by using different sets for the easiness and hardness condition. We will consider such variants in the sequel. When using a strong easy-hard relation to construct a zero-knowledge argument (as per Theorem C.6), the hardness condition of the easy-hard relation corresponds to the soundness of the argument, and the easiness condition of the easy-hard relation corresponds to the zero-knowledge condition of the argument. Therefore, if the hardness condition of the easy-hard relation holds against *all* possible algorithms (i.e. the set $\mathbf{ALG}$) then we would get a zero-knowledge *proof* instead of an argument.

C.3 Constructing a Strong Easy-Hard Relation

In this section we present a strong easy-hard relation $\mathbf{S}$. We will not be able to directly use the relation $\mathbf{S}$ to construct a zero-knowledge argument, because it will not be an $\mathbf{NP}$ relation (we'll only be able to construct an $\mathbf{Ntime}(T)$ relation for any "nice" super-polynomial function $T(\cdot)$). Nevertheless, we will be able to the relation $\mathbf{S}$ to construct a (somewhat relaxed) strong easy-hard $\mathbf{NP}$ relation.

Up until now, our standard computational model for adversaries in this paper has been the set $\mathbf{Psize}$ of polynomial size algorithms. It is also quite standard to consider the set $\mathbf{PPT}$ of *uniform* probabilistic polynomial time adversaries. In this section we consider a "hybrid" model of adversaries with bounded non-uniformity . By this we mean probabilistic polynomial time Turing machines that get an advice string whose length is bounded in some *fixed* polynomial in the security parameter. For example, one can consider adversaries that, on security parameter 1^n, get access to an advice tape of length n^2. We stress that the *running time* of our adversaries is not bounded by a fixed polynomial, but rather they can run in any polynomial time.

To reduce the number of parameters (and without loss of generality), we can consider only adversaries that when the security parameter is n, get access to an advice tape of length n. We call such machines *n-bounded*[23]. We define $\mathbf{Bsize}$ to be the set of n-bounded algorithms: That is $\mathbf{Bsize} \stackrel{def}{=} \cup_{c,d>0}\mathbf{TRA}(n^c, n^d, n)$. Note that $\mathbf{PPT} \subseteq \mathbf{Bsize} \subseteq \mathbf{Psize}$.

We call a strong easy-hard relation a *bounded non-uniformity* strong easy-hard relation if the easiness condition holds when quantifying over all possible n-bounded generators G^* (i.e. all $G^* \in \mathbf{Bsize}$). The hardness condition is unchanged (should hold against all $\mathbf{Psize}$ adversaries). The relation $\mathbf{S}$ that we will construct in this section will be a bounded non-uniformity strong easy-hard relation. As noted above, this relation will have complexity slightly above $\mathbf{NP}$ and thus we will not be able to plug it in Theorem C.6 to obtain a zero-knowledge argument. However, this relation will play an important role in the construction of a (slightly relaxed) bounded non-uniformity strong easy-hard $\mathbf{NP}$ relation, that will be used in a (slightly generalized) variant of Theorem C.6 (Theorem C.14).

The main theorem of this section is the following:

Theorem C.7. *Suppose that one-way functions exist. Let $T : \mathbb{N} \to \mathbb{N}$ be a super-polynomial function (i.e., $T(n) = n^{\omega(1)}$) that is polynomial-time computable. There exists a bounded non-uniformity strong easy-hard $\mathbf{Ntime}(T^2)$ relation $\mathbf{S}$. Furthermore, the hardness condition of the relation $\mathbf{S}$ holds against all adversaries $A \in \mathbf{ALG}$ (and not just $A \in \mathbf{Psize}$).*

The remainder of this section will be devoted to proving Theorem C.7. We will fix a function $T : \mathbb{N} \to \mathbb{N}$ as in the statement of the theorem. We will first prove (in Claim C.8) that there exists a relation $\hat{\mathbf{S}}$ that satisfies all conditions of Theorem C.7, apart from being an $\mathbf{Ntime}(T^2)$ relation (i.e., $\hat{\mathbf{S}}$ is a bounded easy-hard relation with the hardness property holding against any $A \in \mathbf{ALG}$, but membership in $\hat{\mathbf{S}}$ cannot be decided in $T(n)^2$ time). We will then modify the definition of the relation $\hat{\mathbf{S}}$ to obtain the relation $\mathbf{S}$ which is an $\mathbf{Ntime}(T^2)$ relation and show (in Claim C.10) that the relation $\mathbf{S}$ is a bounded easy-hard relation with the hardness property holding against any $A \in \mathbf{ALG}$.

Our basic approach in constructing the strong easy-hard relation will be to generalize the weak easy-hard relation S_{PRG} defined in the proof of Theorem C.2. Recall that the relation S_{PRG} was defined in the following way: $(\tau, \sigma) \in S_{\text{PRG}}$ iff $\tau = \text{PRG}(\sigma)$ where PRG is a pseudorandom generator that triples its input. A string $\tau \in \{0,1\}^{3n}$ is in $L(S_{\text{PRG}})$ if $\tau \in \text{PRG}(\{0,1\}^n)$. Therefore the language $L(S_{\text{PRG}})$ can be looked at as the language of "pseudorandom" strings (we use here "pseudorandom" in the sense of *not random* as opposed to the sense of *random looking*), and a witness for a string τ is a proof that it is indeed "pseudorandom" (as opposed to being truly random). The generator GEN for this relation on input 1^n outputted a uniformly chosen string τ of length $3n$. We then argued that with very high probability (at least $1 - 2^{-2n}$) the truly random string τ is not pseudorandom and so is not in $L(S_{\text{PRG}})$.

In order to obtain a *strong* easy-hard relation, we use a more generalized notion of pseudorandomness, inspired by Kolmogorov complexity ([Kol65]). We fix a *universal Turing machine $\mathcal{U}$* (that is a machine $\mathcal{U}$ that given a description of a Turing machine M and input strings $x_1, \ldots, x_k$ outputs $M(x_1, \ldots, x_k)$). The *Kolmogorov complexity* of a string $\tau \in \{0,1\}^*$, denoted by $KC(\tau)$, is the length of the shortest program Π such that $\mathcal{U}(\Pi) = \tau$ (equivalently, the size of the smallest Turing machine that outputs τ when given the empty string as input). We say that a string τ has *low*

[23] Indeed, suppose that we have a zero-knowledge argument Π for n-bounded adversaries, and want to construct a zero-knowledge argument Π' that works against adversaries that use n^3 bits of non-uniformity. We will simply define the protocol Π' as follows: on security parameter n, run the protocol Π with security parameter $n' \stackrel{def}{=} n^3$.

Kolmogorov complexity if $KC(\tau) < \frac{|\tau|}{2}$. It is easy to see that for any $\tau \in \{0,1\}^{3n}$, if $\tau \in \text{PRG}(\{0,1\}^n)$ then τ has low Kolmogorov complexity; Yet many other strings also have low Kolmogorov complexity (for example the string 0^{3n}). It is well known, and can be easily seen by a counting argument, that if τ is chosen uniformly in $\{0,1\}^{3n}$ then with very high probability (at least $1 - 2^{-1.5n}$) , the string τ does *not* have low Kolmogorov complexity.

We define the following relation $\hat{\mathbf{S}}$: $(\tau, \Pi) \in \hat{\mathbf{S}}$ if and only if $|\Pi| \leq \frac{|\tau|}{2}$ and $\mathcal{U}(\Pi) = \tau$. It follows that $L(\hat{\mathbf{S}})$ is the language of strings of low Kolmogorov complexity. The relation $\hat{\mathbf{S}}$ is undecidable. Other than that, it satisfies all conditions of Theorem C.7. That is, we have the following claim:

Claim C.8. *Suppose that one-way functions exist, then the relation $\hat{\mathbf{S}}$ is a strong easy-hard relation. Furthermore, the hardness property of the relation $\hat{\mathbf{S}}$ holds against any adversary $A \in \mathbf{ALG}$ (not just against any $A \in \mathbf{Psize}$)*

Proof. We need to prove two properties: hardness and easiness.

1. *Hardness:* We will use the same generator GEN we used for proving that S_{PRG} is an easy-hard relation. That is, when run with security parameter 1^n, GEN outputs τ, where τ is chosen uniformly in $\{0,1\}^{3n}$.

   The probability that τ has low Kolmogorov complexity (i.e. the probability that $KC(\tau) \leq 1.5n$) is negligible (at most $2^{-1.5n}$) and so with very high probability $\tau \notin L(\hat{\mathbf{S}})$. If this is the case then no algorithm can output a witness Π such that $(\tau, \Pi) \in \hat{\mathbf{S}}$ (as no such witness exists). Therefore for any algorithm $A \in \mathbf{ALG}$

   $$\Pr_{\tau \leftarrow \text{GEN}(1^n)}[A(\tau) \in \hat{\mathbf{S}}(\tau)] = negl(n)$$

2. *Easiness:* Let $G^* \in \mathbf{Bsize}$ be a (possibly cheating) generator. We assume that when given a security parameter 1^n , algorithm G^* runs for at most $t(n)$ steps, uses at most $r(n)$ random coins and at most n bits of advice (where $t(\cdot)$ and $r(\cdot)$ are two polynomials). We also assume without loss of generality that on input 1^n, G^* outputs a string of length $3n$.

   As we assume that one-way functions exist, it follows that there exists a pseudorandom generator PRG_r that takes strings of length $0.1n$ to strings of length $r(n)$. We assume that on input a strings of size $0.1n$, algorithm PRG_r runs for at most $g(n)$ steps.

   We now describe the simulator M^*, M^* uses as hardwired information the code of G^* (i.e., the description of G^*'s Turing machine and the contents of its advice tape).

   Algorithm M^*:

   - Input: 1^n: security parameter.

   (a) Choose $s \leftarrow_{\text{R}} \{0,1\}^{0.1n}$.

   (b) Let $R \stackrel{def}{=} \text{PRG}_r(s)$.

   (c) Construct the following program Π:

   Program Π:

   - Input: none
   - Hardwired information: the string s, the code of G^*.

   Instructions of program Π:

   i. Compute $\text{PRG}_r(s)$ and store as R.
   ii. Compute $G^*(1^n; R)$ and store as τ.
   iii. Output τ.

   (d) Output (R, Π).

   Algorithm M^* runs in polynomial time $(g(n)+O(n))$[24]. Note also that the description of M^* can be obtained by an effective transformation from the description of G^*. All that remains is to prove the following two properties (a) and (b) regarding the pair (R, Π) that algorithm M^* outputs:

[24]Note that M^* does not actually run G^* but only constructs a program that does so.

(a) The string R is pseudorandom: This follows immediately from the fact that PRG_r is a pseudorandom generator and τ was selected uniformly in $\{0,1\}^{0.1n}$.

(b) For $\tau \stackrel{def}{=} G^*(1^n; R)$, the pair (τ, Π) is in $\hat{\mathbf{S}}$: To prove this we need to prove two things:

i. $\mathcal{U}(\Pi) = \tau$: This is immediate from the definition of the program Π. In fact we can make even a stronger statement that will be useful for us in the sequel:

Observation C.9. *$\mathcal{U}(\Pi)$ outputs τ within $t(n) + g(n) + O(n)$ steps.*

ii. $|\Pi| \leq 1.5n$: To see this consider the size of the description of the program Π: this description consists of the following:

- The code G^*: contains the advice tape of G^* which is of length n and the finite description of its Turing machine. We can assume without loss of generality that the total length of the code of G^* is at most $1.1n$.
- The string s: length $0.1n$.
- The description of the algorithm PRG_r: finite length and so we can assume without loss of generality that it is of length at most $0.1n$.

The total length is $1.3n$, which is smaller than $1.5n$.

And with this we've proven the easiness condition.

□

Comment: An interesting aspect of the simulator M^* is that it is *not* a black-box simulator. It actually uses the code of G^* in the process of the simulation. Although this use of G^* seems to involve no true "reverse-engineering" it still makes a significant difference. In fact, we'll show that using these kind of techniques, one can obtain goals that are provably impossible to achieve using black-box simulators.

Claim C.8 does not prove Theorem C.7, because the relation $\hat{\mathbf{S}}$ is undecidable, and to prove Theorem C.7 we need to construct a strong easy-hard relation that is an $\mathbf{Ntime}(T^2)$ relation. We define the following relation $\mathbf{S}$: $(\tau, \Pi) \in \mathbf{S}$ if and only if $|\Pi| \leq \frac{|\tau|}{2}$ and $\mathcal{U}(\Pi)$ outputs τ *within* $T(|\tau|)$ *steps*. It is clear that the relation $\mathbf{S}$ is an $\mathbf{Ntime}(T^2)$ relation. Note also that $\mathbf{S} \subseteq \hat{\mathbf{S}}$ (i.e., for any pair (τ, Π), if $(\tau, \Pi) \in \mathbf{S}$ then $(\tau, \Pi) \in \hat{\mathbf{S}}$).

To prove Theorem C.7, all that is left is to prove the following claim:

Claim C.10. *Assuming the existence of one-way functions, the relation $\mathbf{S}$ is a bounded strong easy-hard relation. Furthermore, the hardness property of $\mathbf{S}$ holds against any adversary $A \in \mathbf{ALG}$.*

Proof. We need to show that $\mathbf{S}$ satisfies both the hardness and easiness properties:

1. *Hardness:* To prove the hardness property we use the same generator GEN we used to prove the hardness property of the relation $\hat{\mathbf{S}}$. Because $\mathbf{S} \subseteq \hat{\mathbf{S}}$ we get that for any $A \in \mathbf{ALG}$

$$\Pr_{\tau \leftarrow \text{GEN}(1^n)}[A(\tau) \in \mathbf{S}(\tau)] \leq \Pr_{\tau \leftarrow \text{GEN}(1^n)}[A(\tau) \in \hat{\mathbf{S}}(\tau)] = negl(n)$$

   where the last equality is implied by the hardness property of the relation $\hat{\mathbf{S}}$.

2. *Easiness:* To prove easiness we use for any $\mathbf{Bsize}$ algorithm G^*, the same simulator M^* used in the proof of the easiness property of the relation $\hat{\mathbf{S}}$. All that is needed to prove is that for sufficiently large n's if we let $(R, \Pi) \leftarrow M^*(1^n)$ and $\tau \stackrel{def}{=} G^*(1^n; R)$ then (τ, Π) is indeed in $\mathbf{S}$ (and not just in $\hat{\mathbf{S}}$). This means that we need to prove that $\mathcal{U}(\Pi)$ outputs τ *within* $T(|\tau|)$ *steps*. Yet, this a consequence of Observation C.9, as for large enough n's, $t(n) + g(n) + O(n) < T(n)$ (this is because $T(\cdot)$ is a super-polynomial function, and so it is asymptotically larger than any polynomial).

□

Our proof implies that our simulator M^* satisfies a stronger property than merely outputting a pair (R, Π) such that for $\tau \stackrel{def}{=} G^*(1^n; R)$, $(\tau, \Pi) \in \mathbf{S}$. Observation C.9 implies that the pair (τ, Π) has the property that the fact that it is a member of $\mathbf{S}$ can be verified in time which is polynomial in the running time of G^*. (In general this is faster than the $T^2(|\tau|)$ upper bound for verification time that is guaranteed by the fact that $\mathbf{S}$ is an $\mathbf{Ntime}(T^2)$ relation). Since we will use this property in the sequel, we state it here for future reference:

Claim C.11. *For the relation $\mathbf{S}$ defined in the proof of Theorem C.7, there exists a Turing machine $M_{\mathbf{S}}$ with the following properties:*

1. *When run with a pair (τ, Π), the machine $M_{\mathbf{S}}$ halts and returns an answer in at most $T^2(|\tau|)$ steps.*
2. *For any pair (τ, Π), it holds that $(\tau, \Pi) \in \mathbf{S}$ if and only if $M_{\mathbf{S}}(\tau, \Pi) = 1$.*
3. *There exists a polynomial $p(\cdot)$ such that for any algorithm $G^* \in \mathbf{Bsize}$, if M^* is the simulator for G^* [25] and $(R, \Pi) \leftarrow M^*(1^n)$, then $M_{\mathbf{S}}(\tau, \Pi)$ halts in at most $p(t(n) + n)$ steps (where $\tau \stackrel{def}{=} G^*(1^n; R)$ and $t(n)$ is the maximum running time of G^* on input 1^n).*

If an easy-hard relation satisfies this property then we say that it has a *simulator that outputs an efficiently verifiable witness*.

C.4 A Strong Easy-Hard NP Relation

Our main problem with the relation $\mathbf{S}$ defined in the previous section was that it was not an $\mathbf{NP}$ relation, but rather only an $\mathbf{Ntime}(T^2)$ relation for some super-polynomial function $T(\cdot)$. Therefore, we could not use it to construct a zero-knowledge argument using Theorem C.6. In this section we show how to use the relation $\mathbf{S}$ to construct a strong easy-hard $\mathbf{NP}$ relation, that can be used to construct a zero-knowledge argument. The main tool that we shall use is universal arguments. For a definition of universal arguments see Section B.5. Note that our definition is not exactly as defined by [Mic94].

The structure of this section is as follows: In Section C.4.1, we show how we could have constructed an easy-hard $\mathbf{NP}$ relation from an $\mathbf{Ntime}(T^{O(1)})$ relation if we had *non-interactive* universal argument systems for $\mathbf{Ntime}(T^{O(1)})$ languages. Sections C.4.2 and C.4.4 cope with the fact that we do not know if non-interactive universal arguments exist (for $\mathbf{Ntime}(T^{O(1)})$ languages). In Section C.4.2 we introduce a relaxation to the definition of easy-hard relations called *interactive* easy-hard relations, and prove that this relaxed notion is sufficient for the construction of zero-knowledge arguments. In Section C.4.4 we show that (assuming that there exist hash function that are collision resistent against $\mathbf{Size}(T(n)^{\omega(1)})$ adversaries) the existence of an (interactive or non-interactive) easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation, implies the existence of an interactive easy-hard $\mathbf{NP}$ relation.

C.4.1 Constructing an Easy-Hard NP Relation using a *non-interactive* universal argument System.

As a motivation suppose that we had a *non-interactive* universal argument system for $\mathbf{Ntime}(T^{O(1)})$ relations. Let us fix $M_{\mathbf{S}}$ to be a machine that decides $\mathbf{S}$. By our assumptions there exists a proof system for $\mathbf{S}$ (w.r.t $M_{\mathbf{S}}$) that satisfies the following properties:

1. (Non-interactive) The proof that $\tau \in L(\mathbf{S})$ consists of a single string p of length bounded by some fixed polynomial in $|\tau|$.
2. (Perfect Completeness) There's a prover algorithm that on input $(\tau, \Pi) \in \mathbf{S}$ outputs a string p such that the CS verifier algorithm always accepts the pair (τ, p).
3. (Computational Soundness) For any $\mathbf{Psize}$ algorithm P^*, and any $\tau \notin L(\mathbf{S})$, the probability that P^* outputs a string p such that the CS verifier accepts (τ, p) is negligible.
4. (Verifier efficiency) Deciding whether p is a valid proof that $\tau \in L(\mathbf{S})$ takes a number of step which is some fixed polynomial in $|\tau|$.

[25] as constructed in the proof of Theorem C.7.

5. (Prover Efficiency) On input any pair $(\tau, \Pi) \in \mathbf{S}$, the CS prover algorithm outputs a valid proof p for τ in time which is some fixed polynomial in the running time of $M_{\mathbf{S}}$ on input (τ, Π).

(We do not use the proof of knowledge property of the universal argument system in this section).

We have the following claim

Claim C.12. *Suppose that one-way functions exist. Let* $\mathbf{S}$ *be the strong easy-hard* $\mathbf{Ntime}(T^{O(1)})$ *relation guaranteed by Theorem C.7 (and that satisfies the additional property of Claim C.11). If there exists a non-interactive universal argument system for* $\mathbf{S}$ *then there exists a strong easy-hard* $\mathbf{NP}$ *relation* $\mathbf{S}'$.

Proof. Under our assumptions we can obtain a strong easy-hard $\mathbf{NP}$ relation $\mathbf{S}'$ in the following way: we say that $(\tau, p) \in \mathbf{S}'$ iff p is a valid proof that $\tau \in L(\mathbf{S})$ (i.e. if the CS verifier accepts (τ, p)).

The verifier efficiency property of the universal argument system implies that $\mathbf{S}'$ is indeed an $\mathbf{NP}$ relation.

Additionally the computational soundness property assures us that for $\tau \notin L(\mathbf{S})$ it is hard to find a proof p such that the CS verifier accepts (τ, p). Therefore, if τ is chosen uniformly in $\{0,1\}^{3n}$, then as with very high probability $\tau \notin L(\mathbf{S})$, no polynomial-size algorithm can find a witness p such that $(\tau, p) \in \mathbf{S}'$. Therefore the relation $\mathbf{S}'$ satisfies the hardness property of the easy-hard relation.

The relation $\mathbf{S}'$ also satisfies the easiness property, as for any generator G^*, the simulator M^* shown above for $\mathbf{S}$ can be modified to a simulator M^{**} for $\mathbf{S}'$ in the following way:

Algorithm M^{**}:

- Input: 1^n - security parameter

1. Let $(R, \Pi) \leftarrow M^*(1^n)$
2. Let $\tau \stackrel{def}{=} G^*(1^n; R)$. (We know that $(\tau, \Pi) \in \mathbf{S}$.)
3. Run the CS prover algorithm to obtain a proof p that $\tau \in L(\mathbf{S})$.
4. Output (R, p).

Since the simulator M^{**} uses the CS prover algorithm, it is the *prover efficiency condition* of the universal argument system ensures us that M^{**} runs in polynomial time. This is because Claim C.11 implies that the fact that the pair (τ, Π) is a member of $\mathbf{S}$ can be verified in time which is some fixed polynomial in the running time of G^*, which in turn is assumed to be polynomial time.

Once we've shown that the simulator M^{**} runs in polynomial time, all that is needed to show is that the pair (R, p) it outputs satisfies the following:

1. R is pseudorandom: This is implied by the fact that R is the first output of M^*.
2. For $\tau \stackrel{def}{=} G^*(1^n; R)$, it holds that $(\tau, p) \in \mathbf{S}'$: This is implied by the fact that $(\tau, \Pi) \in \mathbf{S}$ and so by the perfect completeness of the CS prover algorithm, we get that p is a valid proof that $\tau \in L(\mathbf{S})$.

□

To conclude, we see that if we had a *non-interactive* universal argument system then we would have obtained a strong easy-hard $\mathbf{NP}$ relation, which in turn could have been used (via Theorem C.6) to construct a zero-knowledge argument for $\mathbf{NP}$. However, there is no known construction of non-interactive universal argument systems for languages above $\mathbf{NP}$ under standard cryptographic (or complexity) assumptions, and so we will use an *interactive* universal argument system for $\mathbf{Ntime}(T^{O(1)})$, which is known to exist under the assumption of existence of collision resistent (w.r.t. $\mathbf{Size}(T^{\omega(1)})$ adversaries) hash functions exist (see Theorem B.5). We will not be able to construct an $\mathbf{NP}$ relation that satisfies Definition C.5, but rather an $\mathbf{NP}$ relation that satisfies a slightly relaxed notion of strong easy-hard relations, called *interactive* strong easy-hard relations. This notion will be defined in Definition C.13 and is the subject of the next section.

C.4.2 Interactive (Strong) Easy-Hard Relations

Recall that in the standard definition of an easy-hard relation, the generator outputs an instance τ such that it is hard to find a solution σ for τ. In this section we introduce the following generalization/relaxation of strong easy-hard relations: we allow the generator to be an *interactive* algorithm. Now, instead of letting τ be the output of the generator, we let τ be the *transcript* of the interaction of the generator with its partner, that is the concatenation of all messages transmitted during the interaction. We define a *prescribed* partner for the generator, but the hardness requirement will be that it is hard to find a solution for τ even if the partner uses any polynomial-size strategy. That is, the generator can ensure that the transcript of the interaction is a *hard instance* for the relation, regardless of its partner's strategy.

The easiness condition will be that for any (possibly cheating) $\mathbf{Psize}$ (or $\mathbf{Bsize}$) interactive generator G^* there exists a simulator M^* than can output a pair (e, σ), where e is computationally indistinguishable from G^*'s view of the interaction with prescribed partner, and if τ is the transcript that is compatible with the view e [26], then σ is a solution for τ. That is, in the simulated execution, the simulator can ensure that the instance generated is *easy*, regardless of the generator's strategy.

Recall that the *view* of an interactive algorithm in an execution consists of its random tape and all the messages it received during this execution. If the algorithm is non-interactive then its view consists only of its random tape. Therefore we can look at the Definition C.5 of strong (non-interactive) easy-hard relation as a restricted form of the current definition.

For completeness, we present the resulting definition:

Definition C.13. Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We say that S is an *interactive strong easy-hard relation* if there exist two interactive $\mathbf{PPT}$ algorithms GEN, GENPARTNER that satisfy the following properties:

1. (Hardness) For any $\mathbf{Psize}$ algorithm P^*, if τ is the transcript of the interaction of P^* and GEN on input 1^n then the probability that P^* outputs a string σ such that $(\tau, \sigma) \in S$ is negligible.

   Note that this requirement does not involve the prescribed partner GENPARTNER.

2. (Easiness) We require that for any (possible cheating) $\mathbf{Psize}$ generator G^* there exists a $\mathbf{Psize}$ simulator M^* that is able to simulate the view of G^* when interacting with the prescribed partner GENPARTNER, but output along with the simulated view a *solution* for the transcript of this execution.

   That is, we require that for any G^* the simulator M^* outputs a pair (e, σ) such that:

   (a) The first string e is computationally indistinguishable from the actual view of the G^* when interacting with the prescribed partner GENPARTNER.

   (b) Let τ be the transcript that is compatible with the view e (note that for any G^*, the transcript of the execution is a deterministic function of the view of G^* that G^*'s random tape along with all the messages that G^* received). We require that $(\tau, e) \in S$.

   As before we require that the simulator M^* can be obtained by an efficient transformation of G^*.

We say that an interactive strong easy-hard relation is a *bounded non-uniformity* interactive strong easy-hard relation if the *easiness* condition holds only for generators $G^* \in \mathbf{Bsize}$ (instead of $\mathbf{Psize}$). Note that we still require that the *hardness* condition holds against any $P^* \in \mathbf{Psize}$.

Important convention: Due to the inflation of modifiers, from now on we will drop the modifiers strong and interactive. Therefore we will use the name easy-hard relation to denote interactive strong easy-hard relations and the name bounded easy-hard relation to denote bounded non-uniformity interactive strong easy-hard relations.

C.4.3 Application of Interactive Easy-Hard Relations

The next theorem shows that the relaxation of allowing the generator to be interactive does not harm the application of easy-hard relations to constructing zero-knowledge arguments.

[26] As noted below in the formal definition, the transcript τ is determined by the view e and can be computed from e.

Theorem C.14. *If easy-hard* $\mathbf{NP}$ *relations exist, and there exists a witness indistinguishable proof of knowledge for any language in* $\mathbf{NP}$*, then there exists a zero-knowledge negligible-error argument for any language in* $\mathbf{NP}$ *with the following properties:*

1. *The number of rounds in this argument is the number of rounds used by the generator of the easy-hard relation plus the number of rounds in the witness indistinguishable proof.*
2. *It has a strict polynomial time simulator (as opposed to expected polynomial time simulator).*
3. *If both the generator program in the easy-hard relation and the verifier program in the witness indistinguishable proof are of the public coin type, then so is the verifier program in this zero-knowledge argument.*

If the easy-hard relation is a bounded non-uniformity relation then the zero-knowledge condition of the argument holds only for $\mathbf{Bsize}$ *verifiers.*

Proof. The construction is almost the same as the construction shown in the proof of Theorem C.3. We assume that S is a easy-hard $\mathbf{NP}$ relation with generator GEN and prescribed partner GENPARTNER. Let R be an $\mathbf{NP}$ relation and $L = L(R)$. Our protocol for a zero knowledge argument for L will go as follows:

Construction C.15. *A zero-knowledge argument for* $L(R)$

- Common Input: the statement to be proved x, the security parameter 1^n.
- Auxiliary input to prover: w such that $(x, w) \in R$.

1. *Phase 1: Generation protocol for* S
   - Both verifier and prover run the generator protocol of the relation S with security parameter $n' = n + |x|$. The verifier plays the part of the generator and uses the algorithm GEN. The prover plays the part of the partner and uses the algorithm GENPARTNER. We let τ be the transcript of this interaction.
2. *Phase 2: Witness Indistinguishable Proof*
   - The prover proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x, w) \in R$ or σ such that $(\tau, \sigma) \in S$.
     More formally, let R' be the $\mathbf{NP}$ relation defined as follows $(\langle x, \tau\rangle, w') \in R'$ iff $(x, w') \in R$ or $(\tau, w') \in S$. The prover proves using the witness indistinguishable proof of knowledge that $\langle x, \tau\rangle \in L(R')$.
     The verifier acts according to the witness indistinguishable protocol and will accept if and only if this proof is valid.

It is easy to see that the number of rounds in the protocol of Construction C.15 is indeed the number of rounds in the generation protocol of S plus the number of rounds in the witness indistinguishable proof. It is also easy to see that if both the generator for S and the verifier in the witness indistinguishable proof are of the public-coin type then so is the verifier in the protocol of Construction C.15.

To show that Construction C.15 is indeed a zero-knowledge argument, one needs to prove three properties: completeness, soundness and zero-knowledge (with a strict polynomial time simulator):

Using the completeness property of the witness indistinguishable proof, we see that whenever the prover is given (x, w) such that $(x, w) \in R$ and acts according to the prescribed prover strategy, the honest verifier will accept at the end of the protocol. Therefore the protocol of Construction C.15 satisfies the completeness property.

Claim C.15.1. *The protocol of Construction C.15 satisfies the computational soundness property (with negligible soundness error).*

Proof. Suppose, towards contradiction, that there exists a $\mathbf{Psize}$ prover P^* and a sequence $\{x_n\}_{n\in\mathbb{N}}$ where $x_n \notin L(R)$ for any $n \in \mathbb{N}$ such that P^* is able to convince the honest verifier of the false statement that $x_n \in L(R)$ with non-negligible probability $\epsilon(n)$. Let us fix $n \in \mathbb{N}$ and let $x = x_n$, $\epsilon = \epsilon(n)$.

We'll use P^* and x to construct a cheating partner P^{**} that contradicts the hardness property of the easy-hard relation S:

Interactive Algorithm P^{**}**:**

- Input: $1^{n'}$ security parameter (recall that $n' = n + |x|$).
- Hardwired information: x where $x \notin L(R)$.

1. Interact with the generator for the relation S using the (first phase of the) strategy of P^*.
2. Let τ be the transcript of the interaction of Step 1, and let t be the internal state of P^* after this interaction. Let P^*_t denote the strategy that is the result of "hardwiring" the state t into P^*.
3. Let EXT be the knowledge extractor guaranteed by the proof of knowledge property of the witness indistinguishable proof. Compute $w' \leftarrow \text{EXT}^{P^*_t}(\langle x, \tau \rangle)$. Output w'

We claim that P^{**} outputs a witness w' such that $(\tau, w') \in S$ with non-negligible probability, thus contradicting the hardness property of the easy-hard relation S. Indeed, by the proof of knowledge property of the witness indistinguishable proof, P^{**} outputs a witness w' such that $(\langle x, \tau \rangle, w') \in R'$ with non-negligible probability. This is because with $\Omega(\epsilon)$ probability t is such that P^*_t convinces the verifier with $\Omega(\epsilon)$ probability and so knowledge extraction succeeds with non-negligible probability ($\Omega((\epsilon - \mu)^c)$ for some negligible $\mu = \mu(n)$ and $c > 0$) within polynomial time.

However, since $x \notin L(R)$ it follows by the definition of R' that it must hold that $(\tau, w') \in S$ and so we've reached a contradiction to the hardness property of S.

□

Claim C.16. *The protocol of Construction C.15 satisfies the zero-knowledge property (with a strict polynomial time simulator). If S is only a bounded easy-hard relation then Construction C.15 satisfies the zero-knowledge property only with respect to* $\mathbf{Bsize}$ *verifiers.*

Proof. Let V^* be any $\mathbf{Psize}$ verifier (if S is only a bounded easy-hard relation then we let V^* be any $\mathbf{Bsize}$ verifier). We construct the following simulator M^* for V^*:

Algorithm M^*:

- Input: $x \in L(R)$, security parameter 1^n.

1. Construct the following interactive algorithm G^{**} obtained by considering the interaction of V^* in the first phase, on input x. That is, G^{**} is obtained from V^* and x by hardwiring x into V^*. One can look at G^{**} as a generator for the easy-hard relation S. Obtain through the effective transformation guaranteed by the easiness property of the easy-hard relation S, a simulator M^{**} for G^{**}.

   Recall that G^{**} is invoked with security parameter $n' = n + |x|$ and so if V^* was a $\mathbf{Bsize}$ algorithm with advice tape of length n, then G^{**} will also be a $\mathbf{Bsize}$ algorithm with advice tape of length $n + |x| = n'$. Therefore we can indeed obtain a simulator M^{**} for G^{**}, even in the case where S is only a *bounded* easy-hard relation.
2. Let $(e, \sigma) \leftarrow M^{**}(1^{n+|x|})$. Let τ be the transcript that corresponds to the view e of the interaction of G^{**} with the prescribed generator partner for S. We know by Part (b) of the easiness property that (if n is sufficiently large) then $(\tau, \sigma) \in S$.
3. Consider e as the view of the interaction of V^* during the first phase of the zero-knowledge argument, Let V^*_e be the result of "hardwiring" e into V^* (we assume that e contains the random tape of V^* and so V^*_e is a deterministic interactive algorithm).
4. Run the witness indistinguishable prover strategy on input the theorem $\langle x, \tau \rangle$ and witness σ) (recall that $(\langle x, \tau \rangle, \sigma) \in R'$ because $(\tau, \sigma) \in S$. Use V^*_e for the verifier's strategy in the witness indistinguishable proof. Let e' be the prover's messages in this proof.
5. Output (e, e').

We claim that for any sequence $\{(x_n, w_n)\}_{n \in \mathbb{N}}$ where $(x_n, w_n) \in R$, the probability ensemble $\{M^*(x_n)\}_{n \in \mathbb{N}}$ is computationally indistinguishable from the view of V^* when interacting with the (honest) prover on input (x_n, w_n). Indeed, suppose otherwise and define the following three probability ensembles $\{E^0_n\}_{n \in \mathbb{N}}, \{E^1\}_{n \in \mathbb{N}}, \{E^2_n\}_{n \in \mathbb{N}}$:

- We let E_n^0 be the output of the simulator on input $x_n, 1^n$. That is $E_n^0 = (e, e')$ where $(e, e') \leftarrow M^*(x_n, 1^n)$.
- We let E_n^1 be the output of a modified simulator that gets as additional input the witness w_n such that $(x_n, w_n) \in R$ and uses w_n instead of σ in Step 4 as input for the prover strategy in the witness indistinguishable proof.
- E_n^2 is the view of V^* when interacting with the (honest) prover that gets (x_n, w_n) as input (recall that $(x_n, w_n) \in R$).

We need to prove that $\{E_n^0\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{E_n^2\}_{n\in\mathbb{N}}$. We'll do it by proving the following two claims:

- $\{E_n^0\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{E_n^1\}_{n\in\mathbb{N}}$:

  This is a consequence of the fact that the proof system is witness indistinguishable and so the view of the verifier when the prover is using witness σ is computationally indistinguishable from its view when the prover is using the witness w. (That is, any algorithm that manages to distinguish between E^0 and E^1 can be used to contradict the fact that the proof used is witness indistinguishable.)
- $\{E_n^1\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{E_n^2\}_{n\in\mathbb{N}}$:

  This is a consequence of part (a) of the easiness property (of the easy-hard relation S). Indeed any algorithm that distinguishes between E^1 and E^2 can be converted into a distinguisher that distinguishes between the first output e of the generator G^{**}'s simulator M^{**} and the view of G^{**} when interacting with the honest partner (by hardwiring x_n and w_n into this distinguisher).

□

□

C.4.4 Constructing an Interactive Easy-Hard NP Relation

In this section we show a general transformation of an $\mathbf{Ntime}(T^{O(1)})$ easy-hard relation S into an $\mathbf{NP}$ easy-hard relation S^*. The latter relation S^* will be an interactive easy-hard relation even if the former relation S is non-interactive. The theorem that we prove in this section is:

Theorem C.17. *Let $T : \mathbb{N} \to \mathbb{N}$ be a polynomial time computable function that is super-polynomial (e.g. $T(n) = n^{\omega(1)}$). Suppose that there exist $T'(n)$ collision-resistent hash functions[27] for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$. If there exists an easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation S that satisfies the following additional properties:*

1. *The hardness property of S holds even against $\mathbf{Size}(T^{O(1)})$ adversaries.*
2. *It has a simulator that outputs efficiently verifiable witnesses (in the sense of Claim C.11). (This is a technical requirement that can be ignored in a first reading.)*

Then there exists an easy-hard $\mathbf{NP}$ relation S^*. If S is only a bounded *easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation then S^* will also be a* bounded *easy-hard $\mathbf{NP}$ relation. If the prescribed generator for S is of the public-coin type then so will the prescribed generator for S^*.*

Combining Theorem C.7 and Theorem C.17, along with the observation that both additional properties hold in the case of the bounded easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation $\mathbf{S}$ (the hardness property holds against any adversary in $\mathbf{ALG}$[28], and the second property is implied by Claim C.11), we get the following corollary:

Corollary C.18. *Let $T : \mathbb{N} \to \mathbb{N}$ be a polynomial time computable function that is super-polynomial (e.g. $T(n) = n^{\omega(1)}$). Suppose that there exist $T'(n)$-collision-resistent hash functions for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$. Then there exists a bounded easy-hard $\mathbf{NP}$ relation $\mathbf{S}^*$.*

[27] That is, hash functions that are collision resistent against $\mathbf{Size}(T')$ adversaries. See Definition B.4.

[28] Note that Theorem C.17 is stronger than what is necessary to establish Corollary C.18 as it only requires that the hardness property holds against $\mathbf{Size}(T^{O(1)})$ adversaries rather than $\mathbf{ALG}$. However, we will apply Theorem C.17 in the sequel on a relation whose hardness property holds only against $\mathbf{Size}(T^{O(1)})$ adversaries.

The basic idea in proving Theorem C.7 is the one presented in Section C.4.1. However, we will need to use an additional idea, as we will be using an *interactive* universal argument system in this section (as opposed to the hypothetical non-interactive universal argument system used in the mental experiment of Section C.4.1).

We assume that T and T' are two functions as stated in Theorem C.7: $T(n) = n^{\omega(1)}$, $T'(n) = T(n)^{\omega(1)}$, and there exist a $T'(n)$-collision resistent hash function ensemble $\{h_\kappa\}_{\kappa\in\{0,1\}^*}$. This assumption will be used in order to obtain an adequate universal argument system.

We assume that S is an easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation and a fixed machine M_S that decides it. Under our assumptions we have by Theorem B.5 that there exists a 4 round Arthur-Merlin universal argument system for S. We denote the four messages of this proof system by $\alpha, \beta, \gamma, \delta$, where α and γ are messages sent by the CS verifier (the honest verifier chooses them at random) and β and δ are sent by the prover. We briefly recall the properties of the universal argument system:

1. (Perfect Completeness) There's a CS prover strategy that when given as input $(\tau', \sigma') \in S$ convinces the prescribed CS verifier with probability 1 that indeed $\tau' \in L(S)$.

2. (Computational Soundness) For any $\mathbf{Psize}$ strategy P^*, and any $\tau' \notin L(\mathbf{S})$, the probability that P^* can convince the prescribed CS verifier is negligible.

3. (Verifier Efficiency) The length of all messages and the running time of the prescribed CS verifier are some fixed polynomial in the security parameter and polylogarithmic in $|\tau'|$. As our security parameter n will always satisfy $2^{\sqrt{\log n}} \leq |\tau'| \leq 2^{\log^2 n}$ (and so polylogarithmic in $|\tau'|$ means polylogarithmic in n), we can assume without loss of generality that all messages are of length n^2.

4. (Prover Efficiency) For any pair $(\tau', \sigma') \in S$ the running time of the prescribed CS prover is some fixed polynomial in the running time of M_S on input (τ', σ').

5. (Proof of Knowledge) There exists a $T(n)^{O(1)}$ time oracle machine $\textsc{Ext}$, a negligible function μ and a number $c > 0$ such that for any $\mathbf{Size}(T^{O(1)})$ algorithm P^* and instance τ', if the probability that P^* convinces the CS verifier that $\tau' \in L(S)$ is ϵ then

$$\Pr[\textsc{Ext}^{P^*}(\tau') \in S(\tau')] \geq \epsilon' \stackrel{def}{=} (\epsilon - \mu(n))^c$$

Given an easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation, we will construct an easy-hard $\mathbf{NP}$ relation S^*. The definition of the relation S^* is best understood by describing first its generation protocol (i.e., the prescribed algorithm for the generator and generator's partner) and its simulator. The generation protocol for S^* consists of two phases. The first phase is the generation protocol for the relation S. That is, the generator and generator partner simply follow the prescribed strategy of the generation protocol of the relation S.

We call the second phase an "encrypted" universal argument: In this phase, the partner and generator engage in a universal argument that the transcript τ' of the first phase is a member of $L(S)$. The partner plays the part of the CS prover, and the generator plays the part of the CS verifier. We call the universal argument "encrypted" because, whereas the generator sends the messages of the CS verifier in this proof (i.e. random strings of length n^2), the partner only responds by *commitments* to "junk" messages of the appropriate length (i.e. the partner sends commitments to 0^{n^2}). This may seem strange, but in the simulation the messages sent by the partner will be commitments to actual CS prover messages instead of commitments to junk.

If we denote by τ the transcript of the entire interaction, then a *witness* that τ is in $L(S^*)$ will consist of decommitments for the messages sent by the sender, such that the resulting CS transcript is indeed a valid proof (w.r.t. the universal argument system) that $\tau' \in L(S)$.

We now turn to a formal definition of the relation S^*:

Construction C.19. *Definition of the relation S^*:*
For $\tau = \langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$ and $\sigma = \langle \beta, s', \delta, s'' \rangle$ we say that $(\tau, \sigma) \in S^*$ if the following holds:

1. The string $\hat{\beta}$ is a commitment to β using coins s'. That is, $\hat{\beta} = \mathcal{C}(\beta; s')$.

2. The string $\hat{\delta}$ is a commitment to δ using coins s''. That is, $\hat{\delta} = \mathcal{C}(\delta; s'')$.

3. The CS-transcript $(\tau', \alpha, \beta, \gamma, \delta)$ convinces the CS verifier of the statement "$\tau' \in L(S)$".

First of all, note that S^* is indeed an **NP** relation: Let n be the security parameter used for the universal argument system and the commitment. We see that the length of the witness σ is bounded by a polynomial in the length of the instance τ, because $|\beta| = |\delta| = n^2$ and $|s| = |s'| = n^3$. Verifying that $(\tau, \sigma) \in S^*$ is done by checking the commitments and invoking the CS verifier algorithm: this can be done in time which is a fixed polynomial in n (by the verifier efficiency condition).

We claim that S^* is an easy-hard relation. In order to show this we need to present an interactive generator and a prescribed partner for S^*, and then prove both the hardness and the easiness properties.

Construction C.20. *The generation protocol for S^**

- Common input: 1^n, where n is the security parameter.

1. *Phase 1: Generation protocol for S*

   - Run the generation protocol for the relation S (i.e., both generator and partner follow the prescribed algorithms for the generator and partner of the relation S). Let τ' be the transcript of this protocol.

2. *Phase 2 : "Encrypted" universal argument*

   The generator and partner engage in an "encrypted" universal argument that $\tau' \in L(S)$. This encrypted proof consists of the generator sending the actual CS verifier's messages (i.e., random strings of length n^2), and the partner responding by commitments to "junk" messages of the same length as the CS prover's messages (i.e., commitments to 0^{n^2}). Specifically,

   - Generator's first step (G2.1): Choose $\alpha \leftarrow_R \{0,1\}^{n^2}$; Send α.
   - Partner's first step (P2.1): Send a commitment to the all zero string of length n^2. That is, let $\hat{\beta} \leftarrow \mathcal{C}_n(0^{n^2})$; Send $\hat{\beta}$.
   - Generator's second step (G2.2): Choose $\gamma \leftarrow_R \{0,1\}^{n^2}$. Send γ.
   - Partner's second step (P2.2): Again, send a commitment to the all zero string of length n^2. That is, let $\hat{\delta} \leftarrow \mathcal{C}_n(0^{n^2})$; Send $\hat{\delta}$.

We shall now prove that the relation S^* and the prescribed generation protocol satisfy both the hardness and the easiness properties. This is proven in Propositions C.21 and C.22 below.

Proposition C.21. *The relation S^* with the generation protocol of Construction C.20 satisfies the hardness property of Definition C.13.*

Proof. The structure of the proof will be as follows:

We start by assuming, for the sake of contradiction, that there exists a cheating $\mathbf{Psize}$ partner P^* that is able, after interacting with the prescribed generator for the relation S^* (denoted GEN_{S^*}), to output a solution σ such that $(\tau, \sigma) \in S^*$ with non-negligible probability (where τ is the transcript of the interaction between P^* and the prescribed generator for the relation S^*).

We then make the following steps:

1. We transform the partner P^* into a cheating CS Prover CSP^{**} that is able to prove a statement of the form "$\tau' \in L(S)$" with non-negligible probability, where τ' is the transcript of the interaction of some polynomial size partner with the prescribed generator for the relation S (denoted GEN_S).

2. Using the proof of knowledge property of the universal argument system, we transform the CS proving algorithm CSP^{**} into a cheating $\mathbf{Size}(T(n)^{O(1)})$ partner P^{***} for the relation S; The partner P^{***}, after interacting with GEN_S with transcript τ', will be able to output a witness σ' for the fact that $\tau' \in L(S)$ with non-negligible probability.

   With this, we've reached a contradiction to the hardness property of the relation S, and so the proof is completed.

As outlined, we start by assuming for the sake of contradiction that there exists a cheating $\mathbf{Psize}$ partner P^* that is able (with non-negligible probability $\epsilon = \epsilon(n)$) , after interacting with the prescribed generator of the relation S^* (the generator of Construction C.20, which we denote by GEN_{S^*}) to output a solution σ such that $(\tau, \sigma) \in S^*$, where τ is the transcript of the interaction. In case P^* manages to output such a solution, we say that P^* is *successful*.

Our first step is to use P^* to construct a cheating CS prover CSP^{**}. The cheating CS prover CSP^{**} will be able to prove theorems of the form "$\tau' \in L(S)$" with some non-negligible probability $\epsilon' = \epsilon'(n)$, where τ' is distributed according to the distribution of the transcript of P^*'s interaction with GEN_{S^*} during the first phase (by the definition of GEN_{S^*}, it follows in this phase the strategy of the prescribed generator of the relation S, denoted GEN_S).

As P^* is a non-uniform algorithm, we can assume without loss of generality that it is a deterministic algorithm (this follows by a standard argument of hardwiring a "best" random tape into P^*). It follows that given the transcript τ' of the interaction of P^* with GEN_{S^*} during the first phase, one can compute the internal state of P^* at the end of the first phase. This is because while for a randomized interactive algorithm, the internal state is a function of the algorithm's random tape and all the messages the algorithm receives, for a deterministic algorithm the internal state is only a function of the messages the algorithm receives, which are part of the transcript τ'.

The cheating CS Prover CSP^{**} will be defined as follows:

Interactive Algorithm CSP^{**}:

- Common input: τ' (the objective of CSP^{**} is to convince the CS verifier that $\tau' \in L(S)$), 1^n : security parameter.

1. Obtaining the CS Verifier's first message (Step CSV1): $\alpha \in \{0,1\}^{n^2}$.

   (Recall that the prescribed CS verifier selects α uniformly in $\{0,1\}^{n^2}$.)

2. Generation of CSP^{**}'s first message (CSP1):

   (a) Recall that we have an interactive algorithm P^* that is a cheating partner for the generation protocol of the relation S^* (Construction C.20). We treat the instance τ' as a transcript of an interaction between P^* and GEN_{S^*} during the first phase (recall that during this phase GEN_{S^*} follows the algorithm of GEN_S). Algorithm CSP^{**} computes the internal state, denoted t', that P^* would be in after interacting in the first phase with GEN_{S^*} with transcript τ'.

   (b) Run P^* from state t' with α as the first message of the second phase from the generator (Step G2.1). Obtain P^*'s first message, denoted $\hat{\beta}_1$ (Step P2.1).

   (c) Algorithm CSP^{**} tries to obtain the string β_1 that is the[29] decommitment of the string $\hat{\beta}_1$, if it succeeds it sends β_1 to the verifier. Details follow:

      i. Record the state of P^* at the current point (after step P2.1). Let us denote this state by t. Algorithm CSP^{**} will continue a simulated execution of P^*, this time choosing by itself the messages to feed to P^* (instead of feeding P^* with messages obtained from the CS verifier).

      ii. Choose $\gamma_1 \leftarrow_R \{0,1\}^{n^2}$ and feed it to P^* as the generator's second message (G2.2). Get P^*'s second message, denoted $\hat{\delta}_1$.

      iii. Assume that P^* is successful (otherwise abort[30]). Recall that this means that P^* outputs a witness, denoted $\sigma_1 = \langle \beta_1, s'_1, \delta_1, s''_1 \rangle$, to the fact that the transcript $\tau_1 = \langle \tau', \alpha, \hat{\beta}_1, \gamma_1, \hat{\delta}_1 \rangle$ is in $L(S^*)$. It follows that β_1 is the decommitment to $\hat{\beta}_1$ (because $\hat{\beta}_1 = \mathcal{C}(\beta_1; s'_1)$.

      iv. Send β_1 to the CS verifier.

3. Obtain the CS verifier's second message (CSV2): $\gamma \in \{0,1\}^{n^2}$. (The prescribed CS verifier chooses $\gamma \leftarrow_R \{0,1\}^{n^2}$.)

4. Generation of CSP^{**}'s second message (CSP2):

[29] As the commitment scheme is perfectly binding there is only one decommitment for $\hat{\beta}_1$.

[30] Note that we are not trying to construct a cheating CS prover that *always* succeeds in convincing the CS verifier: we only want CSP^{**} to succeed with non-negligible probability.

(a) Rewind the state of P^* to the recorded state t (after step P2.1).

(b) Feed the message γ to P^* as the generator's second message (G2.2). Get P^*'s second message (step P2.2), denoted $\hat{\delta}_2$.

(c) We assume that P^* is successful again (otherwise abort). This means that P^* outputs a witness, denoted $\sigma_2 = \langle \beta_2, s'_2, \delta_2, s''_2 \rangle$, to the fact that the transcript $\tau_2 = \langle \tau', \alpha, \hat{\beta}_1, \gamma, \hat{\delta}_2 \rangle$ is in $L(S^*)$. In particular we have that δ_2 is a decommitment to $\hat{\delta}_2$ (because $\hat{\delta}_2 = \mathcal{C}(\delta_2; s''_2)$)

(d) Send δ_2 to the CS verifier.

Analysis of CSP^{**}'s success: We have the following claim:

Claim C.21.1. *If the theorem τ' is distributed as the transcript of P^*'s interaction with GEN_{S^*} during the first phase, then CSP^{**} has probability $\Omega(\epsilon^3)$ of convincing the prescribed CS verifier that $\tau' \in L(S)$.*

Proof. We know that P^* has probability ϵ of success when interacting GEN_{S^*}. Suppose that we stop the interaction of P^* and GEN_{S^*} just after step P2.1 (after P^* sends the message denoted $\hat{\beta}$). We call such a partial interaction *good* if there's a $\frac{\epsilon}{2}$ probability that if it is continued then P^* will be successful. The probability mass of good partial interactions among all partial interactions of P^* and GEN_{S^*} is at least $\frac{\epsilon}{2}$. We can identify a partial interaction with the internal state of P^* after step P2.1. We call such a state that corresponds to a good interaction a *good* state.

Consider an interaction between algorithm CSP^{**} and the prescribed CS verifier on input τ' that is chosen according to the distribution stated in the claim. Consider the state t that algorithm CSP^{**} computes in Step 2(c)i. We claim that the probability (taken over the choice of the input τ', and the random coins of both CSP^{**} and the CS verifier) that t is a good state is at least $\frac{\epsilon}{2}$. Indeed, the input that P^* gets is distributed according to the same distribution that P^* gets when interacting with GEN_{S^*}.

Assume that the state t is good. Conditioned on this, the probability that CSP^{**} obtains a solution from P^* in Step 2(c)iii is at least $\frac{\epsilon}{2}$ (as again the input that CSP^{**} feeds P^* is distributed identically to the input that P^* gets in an actual interaction WITH GEN_{S^*}).

Also, conditioned on t being good, the probability that CSP^{**} obtains a witness from P^* in Step 4c is also at least $\frac{\epsilon}{2}$ (as once again the input distribution is identical to the input P^* gets in an actual interaction). Furthermore, after fixing the state t, these two events (CSP^{**} obtaining a witness in Step 2(c)iii and CSP^{**} obtaining a witness in Step 4c) are *independent*. This means that for a good t, with probability at least $\frac{\epsilon}{2} \cdot \frac{\epsilon}{2} = \Omega(\epsilon^2)$, Algorithm CSP^{**} will obtain witnesses from P^* in *both* Step 2(c)iii and Step 4c.

As the probability of CSP^{**} obtaining a good state t is at least $\frac{\epsilon}{2}$, we get that with overall probability $\Omega(\epsilon^3)$, algorithm $CSP^*{*}$ will get witnesses from P^* in both steps 2(c)iii and 4c.

Suppose that CSP^{**} does obtain a witness in each step. In the notation of algorithm CSP^{**}'s description, this means that $(\tau_2, \sigma_2) \in S^*$ where $\tau_2 = \langle \tau', \alpha, \hat{\beta}_1, \gamma, \hat{\delta}_2 \rangle$ and $\sigma_2 = \langle \beta_2, s'_2, \delta_2, s''_2 \rangle$. This means in particular that the CS transcript $(\tau', \alpha, \beta_2, \gamma, \delta_2)$ is accepted by the prescribed CS verifier (i.e., this a transcript of an interaction where the prescribed CS verifier is convinced that $\tau' \in L(S)$). Yet it also means that β_2 is a decommitment to $\hat{\beta}_1$, and as β_1 is also a decommitment to $\hat{\beta}_1$, it follows (by the perfect binding of the commitment scheme $\mathcal{C}$) that $\beta_2 = \beta_1$. This implies that the transcript $(\tau', \alpha, \beta_1, \gamma, \delta_2)$ is accepted by the CS verifier. Since this is exactly the transcript of the interaction between CSP^{**} and the prescribed CS verifier, and so we see that in this case CSP^{**} succeeds in convincing the CS verifier that $\tau' \in L(S)$.

We conclude that with $\Omega(\epsilon^3)$ probability (over the choice of the theorem τ' and the internal coin tosses of both CSP^{**} and the CS verifier), algorithm CSP^{**} succeeds in convincing the prescribed CS verifier that $\tau' \in L(S)$. □

Note: If we were guaranteed that hardness property of the relation S holds against any adversary $A \in \mathbf{ALG}$ (as is the case in the particular relation $\mathbf{S}$ that is constructed in the proof of Theorem C.7), then we could have obtained at this point a contradiction to the computational soundness property of the universal argument system. This is as in this case we would be guaranteed that for any partner P^* , for the transcript τ' of P^*'s interaction with GEN_S, with very high $(1 - negl(n))$ probability $\tau' \notin L(S)$. Combining this with Claim C.21.1 and an averaging argument we would obtain that there exists a string $\tau'_0 \notin L(S)$ for which CSP^{**} has probability $\Omega(\epsilon^3)$ of convincing the prescribed CS verifier of the false statement "$\tau'_0 \in L(S)$". This would contradict the computational soundness of the universal argument system.

As the only guarantee we have is that the hardness condition of the relation S holds against $\mathbf{Size}(T^{O(1)})$ adversaries we have to make an additional step, which will use the proof of knowledge property of the universal argument system. We will use the algorithm CSP^{**} to construct a cheating $T(n)^{O(1)}$ time partner P^{***} that contradicts the hardness property of the easy-hard relation S. Using the proof of knowledge property of the universal argument system we have the following claim:

Claim C.21.2. *There exists a $T(n)^{O(1)}$ size algorithm E^{**} such that if the theorem τ' is distributed as the transcript of P^*'s interaction with the prescribed generator during the first phase, then $E^{**}(\tau')$ has non-negligible probability of outputting a solution σ' such that $(\tau', \sigma') \in S$. Specifically, let μ and c be the negligible function and constant guaranteed in the proof of knowledge condition of the universal argument system, then the probability that $E^{**}(\tau')$ outputs a solution for τ' is at least $\Omega(\epsilon^3)(\Omega(\epsilon^3) - \mu(n))^c$.*

Proof Sketch: We obtain the algorithm E^{**} by using the knowledge extractor EXT of the universal argument system with an oracle to CSP^{**}. That is, we define $E^{**}(\tau') \stackrel{def}{=} \text{EXT}^{CSP^{**}}(\tau')$. The result follows because with $\Omega(\epsilon^3)$ probability, τ' is such that CSP^{**} has probability $\Omega(\epsilon^3)$ of convincing the CS verifier that $\tau' \in L(S)$. □

Using E^{**} one can define algorithm P^{***} (a cheating $\mathbf{Size}(T(n)^{O(1)})$ partner for the relation S) in the following way:

Interactive Algorithm P^{***}:

- Common input: 1^n - security parameter

1. Interact with the GEN_S using algorithm P^* to compute the messages (of its first phase). Let τ' be the transcript of this interaction.
2. Output $E^{**}(\tau')$.

By Claim C.21.2 we see that P^{***} has probability $\Omega(\epsilon^3)(\Omega(\epsilon^3) - \mu(n))^c$ of outputting a solution σ' such that $(\tau', \sigma') \in S$. This is non-negligible because we assume that ϵ is non-negligible, and so P^{***} contradicts the hardness against $\mathbf{Size}(T(n)^{O(1)})$ algorithms property of the easy-hard relation S.

□

Proposition C.22. *If the original relation S is an easy-hard relation, then the relation S^* with the generation protocol of Construction C.20, satisfies the easiness property of Definition C.13. If the original relation S is only a* bounded *easy-hard relation (i.e., it satisfies the easiness property with respect to $\mathbf{Bsize}$ adversaries) then so is the relation S^*.*

Proof. Let G^* be a $\mathbf{Psize}$ (or possibly $\mathbf{Bsize}$) generator for the relation S^*. We will construct a simulator M^* that outputs a pair (e, σ) such that e is computationally indistinguishable from G^*'s view and σ is a solution for the transcript τ that is compatible with the view e. Recall that the generation protocol for S^* (Construction C.20) consists of two phases. If we let G^{**} be the restriction of G^* to the first phase, then we have a (possibly cheating) generator for the relation S. Therefore, by the easiness property of the relation S, we have a simulator M^{**} for the generator G^{**}. We will use the simulator M^{**} to construct a simulator M^* for the generator G^* in the following way:

Algorithm M^***:**

- Common input: 1^n, where n is the security parameter.

1. *Simulation of phase 1 (Generation protocol for S)*:
   - Let $(e', \sigma') \leftarrow M^{**}(1^n)$.
     Recall that the first element e' represents the view of G^{**} during the generation protocol for S (which is the same as the view of G^* during the first phase of the generation protocol for S^*). Note that e' not only contains a simulation of the messages sent to G^* during the first phase, but also G^*'s random tape.

- Let τ' be the transcript that corresponds to the view e' (we know that $(\tau', \sigma') \in S$).
  Note that if we feed e' to G^* then we obtain its internal state after interacting in the first phase with transcript τ'.

2. *Simulation of phase 2 ("encrypted" universal argument):*

   Algorithm M^* uses the prescribed CS prover's algorithm to continue the simulation. This time instead of sending commitments to "junk" strings (as the prescribed partner does) the simulator sends commitments to valid messages of the CS prover.

   (a) Feed the CS prover's algorithm τ' as the theorem to be proved and σ' as the witness for it.

   (b) Feed the view e' to algorithm G^* and obtain its first message of the second stage α (step G2.1). (We assume without loss of generality that α is of the proper length, that is $\alpha \in \{0,1\}^{n^2}$).

   (c) Feed α to the CS prover's algorithm to obtain the prover's first message β. Compute $\hat{\beta}$: a commitment to β (i.e., choose $s' \leftarrow_R \{0,1\}^{n^3}$, compute $\hat{\beta} \leftarrow \mathcal{C}_n(\beta; s')$).

   (d) Feed G^* the message $\hat{\beta}$ and obtain its second message γ (Step G2.2). Again, we assume without loss of generality that $\gamma \in \{0,1\}^{n^2}$.

   (e) Feed the message γ to the CS prover to obtain the CS prover's second message δ. Compute $\hat{\delta}$: a commitment to δ (i.e., choose $s'' \leftarrow_R \{0,1\}^{n^3}$ and let $\hat{\delta} \leftarrow \mathcal{C}_n(\delta; s'')$).

   (f) Output the pair (e, σ) where $e = \langle e', \hat{\beta}, \hat{\delta} \rangle$ and $\sigma = \langle \beta, s', \delta, s'' \rangle$. Note that the transcript corresponding to the view e is $\tau = \langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$.

As in Section C.4.1, it is not obvious that the simulator M^* runs in polynomial time. This is because M^* uses the CS prover algorithm which runs at the worst case in time $T(n)^{O(1)}$. However, the fact that the relation S has the additional property that its simulator outputs efficiently verifiable witnesses (in the sense of Claim C.11) along with the prover efficiency condition of the universal argument system, assures us that M^* does indeed run in polynomial time.

By the perfect completeness of the universal argument system, it is easily seen that σ is indeed a witness that the corresponding transcript τ is in $L(S^*)$. What remains to be proven is that its first input e is computationally indistinguishable from the view of G^* in an interaction with the prescribed partner.

We define the following three probability ensembles:

- $E^0 = E^0_n$ denotes the distribution of the first element of $M^*(1^n)$.
- $E^1 = E^1_n$ denotes the distribution of the first element of $M^*(1^n)$ with one modification: we replace the commitments to the CS prover's messages with commitments to 0^{n^2}.
- $E^2 = E^2_n$ denotes the distribution E^1 with one additional modification: instead of computing e' by using the simulator M^{**}, we compute e' as the actual view of G^* when interacting in the first phase with the prescribed partner for the relation S. Another way to describe E^2 would be that E^2 is the view of G^* when interacting with the prescribed partner for the relation S^*.

We need to prove that $\{E^0_n\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{E^2\}_{n\in\mathbb{N}}$. We'll do so in two steps:

1. $\{E^0_n\}_{n\in\mathbb{N}}$ is computationally indistinguishable from $\{E^1_n\}_{n\in\mathbb{N}}$:

   Indeed, the existence of a $\mathbf{Psize}$ algorithm that distinguishes E^0 from E^1 contradicts the security of the commitment scheme, as the only difference between the two distributions is that E^1 contains only commitments to 0 and E^0 contains also commitments to 1. (Recall that the commitment scheme $\mathcal{C}$ is a bitwise commitment scheme where commitment to strings is obtained by concatenating commitments to bits).

2. E^1 is computationally indistinguishable from E^2:

   Indeed, suppose by contradiction that there exists a $\mathbf{Psize}$ algorithm D that distinguishes between E^1 and E^2 with non-negligible advantage. In this case, we'll use D to construct a $\mathbf{Psize}$ algorithm D' that distinguishes between $M^{**}(1^n)$ and the view of G^{**} when interacting with the prescribed partner for the relation S. By this, we'll reach a contradiction to the easiness property of the relation S.

   We define D' as follows:

   Algorithm D':

   - Input : e'

   (a) Compute $\hat{\beta} \leftarrow \mathcal{C}(0^{n^2})$, $\hat{\delta} \leftarrow \mathcal{C}(0^{n^2})$.

   (b) Output $D(\langle e', \hat{\beta}, \hat{\delta} \rangle)$.

   It is clear that D' distinguishes between $M^{**}(1^n)$ and the view of G^{**} when interacting with the prescribed partner for the relation S, with the same advantage that D distinguishes between E^1 and E^2.

□

C.5 A Zero-Knowledge Argument for $\mathbf{NP}$

It is known that constant round Arthur-Merlin witness indistinguishable proofs of knowledge for $\mathbf{NP}$ exist if one-way functions exist (e.g. the parallel version of the Blum's hamiltonicity proof, [Blu87], [FS90]). Combining this fact with Theorem C.14 , Corollary C.18, and the observation that our prescribed generator only sends messages that are random strings we obtain the following Theorem:

Theorem C.23. *Let $T : \mathbb{N} \to \mathbb{N}$ be polynomial time computable function that is super polynomial (i.e., $T(n) = n^{\omega(1)}$). Assume that $T'(n)$-collision resistent hash functions exist for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$, then there exists a constant-round Arthur-Merlin zero-knowledge argument, against $\mathbf{Bsize}$ verifiers. Also, the simulator for this protocol runs in* strict *polynomial time.*

We will show later (in Section E) that a version of this argument remains zero-knowledge if executed concurrently a fixed polynomial number of times.

For the sake of clarity let us write down the complete description of this zero-knowledge argument and the description of its simulator. Let R be an $\mathbf{NP}$ relation.

Construction C.24. *A Zero-Knowledge Argument for $L(R)$*

- Common Input: the statement to be proved x, the security parameter 1^n.
- Auxiliary input to prover: w such that $(x, w) \in R$.

1. *Phase 1: Generation protocol for $\mathbf{S}^*$ (an easy-hard $\mathbf{NP}$ relation)*

   (a) *Phase 1.1: Generation protocol for $\mathbf{S}$ (an easy-hard $\mathbf{Ntime}(T^{O(1)})$ relation)*

   - Verifier's step (V1.1.1): Choose $\tau' \leftarrow_{\mathrm{R}} \{0,1\}^{m(n)+|x|}$. Send τ'.

     (In the generation protocol for $\mathbf{S}$ we described in Section C.3 we had $m(n) \stackrel{def}{=} 3n$ but other settings for the function $m(\cdot)$ could be used as well.)

   (b) *Phase 1.2: "Encrypted" universal argument*

   The prover and verifier run the universal argument protocol with the prover sending only the commitments to the messages instead of sending them in the clear. Note that in the actual proof the prover only sends commitments to "junk" messages (i.e. all zero strings). Only in the simulation the messages sent are commitments to valid messages of the universal argument system.

- Verifier's step (V1.2.1): Choose $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^{n^2}$. Send α. Note that this step can be joined with Step V.1.1.
- Prover's step (P1.2.1): Let $\hat{\beta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\beta}$.
- Verifier's step (V1.2.2): Choose $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^{n^2}$. Send γ.
- Prover's step (P1.2.3): Choose $\hat{\delta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\delta}$. Note that if we will use a standard 3-round witness indistinguishable proof for the next phase (such as the parallel version of the graph 3-colorability zero-knowledge proof) then this step can be joined with the first step of the prover in this proof.

2. *Phase 2: Witness Indistinguishable Proof*

   - Let $\tau \stackrel{def}{=} \langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$. The prover proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x, w) \in R$ or σ such that $(\tau, \sigma) \in \mathbf{S}^*$. The verifier acts according to the witness indistinguishable protocol and will accept if and only if this proof is valid.

We now describe the simulator for the protocol of Construction C.24. Let V^* be a $\mathbf{Bsize}$ verifier. We assume that $r(n)$ bounds the number of random coins used by V^* when run with security parameter PRG. We let PRG_r be a pseudorandom generator that takes string of length $0.1n$ to strings of length $r(n)$.

Construction C.25. Algorithm M^* (A simulator for V^*):

- Input: the statement x, security parameter 1^n (where $n \geq |x|$).

1. *Initialization: Choosing randomness for* V^*
   (a) Let $s \leftarrow_{\mathrm{R}} \{0,1\}^{0.1n}$.
   (b) Let $R \stackrel{def}{=} \mathrm{PRG}_r(s)$.
2. *Phase 1: Simulation of generation protocol for* $\mathbf{S}^*$ *(an easy-hard* $\mathbf{NP}$ *relation)*
   (a) *Phase 1.1: Simulation of generation protocol for* $\mathbf{S}$ *(an easy-hard* $\mathbf{Ntime}(T^{O(1)})$ *relation)*
      i. Construct the following program Π:

         Program Π**:**
         - Input: none
         - Hardwired information: the string s, the advice tape of V^* , the theorem x.

         A. Let $R \stackrel{def}{=} \mathrm{PRG}_r(s)$.
         B. Let τ' be the message V^* outputs on input $1 \cdot x$ and randomness R (i.e. $\tau' \stackrel{def}{=} V^*(1^n, x; R)$).
         C. Output τ'.

      Note that $|\Pi| \leq 3n$.
      ii. Compute $\tau' \stackrel{def}{=} V^*(1^n, x; R) = \mathcal{U}(\Pi)$.
   (b) *Phase 1.2: Simulation of "Encrypted" universal argument*
      Algorithm M^* uses the prescribed CS prover's algorithm to continue the simulation. This time instead of sending commitments to "junk" strings (as the prescribed partner does) the simulator sends commitments to valid messages of the CS prover.
      i. Feed the CS prover's algorithm τ' as the theorem to be proved and Π as the witness for it.
      ii. Compute the first message of the second stage α (step V1.2.1). That is, feed the history until this point ($\langle 1^n, x; R \rangle$) to V^* and obtain the message α (We assume without loss of generality that $\alpha \in \{0,1\}^{n^2}$).
      iii. Feed α to the CS prover's algorithm to obtain the prover's first message β. Compute $\hat{\beta}$: a commitment to β (i.e., choose $s' \leftarrow_{\mathrm{R}} \{0,1\}^{n^3}$, compute $\hat{\beta} \leftarrow \mathcal{C}_n(\beta; s')$).

iv. Feed V^* the message $\hat{\beta}$ and obtain its second message γ (i.e. $\gamma \leftarrow G^*(e', \hat{\beta})$). Again, we assume without loss of generality that $\gamma \in \{0,1\}^{n^2}$.

v. Feed the message β to the CS prover to obtain the CS prover's second message δ. Compute $\hat{\delta}$: a commitment to δ (i.e., choose $s'' \leftarrow_R \{0,1\}^{n^3}$, compute $\hat{\delta} \leftarrow \mathcal{C}_n(\delta; s'')$).

vi. Compute the pair (e, σ) where $e = \langle R, 1^n, x, \hat{\beta}, \hat{\delta} \rangle$ and $\sigma = \langle \beta, s', \delta, s'' \rangle$. Note that the transcript corresponding to the view e is $\tau = \langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$.

3. *Phase 2: Witness Indistinguishable Proof*

(a) The simulator proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x, w) \in R$ or σ such that $(\tau, \sigma) \in \mathbf{S}^*$. The simulator uses V^* to compute the verifier's messages.

(b) Algorithm M^* uses the following as input to the witness indistinguishable prover algorithm: the theorem $\tau = \langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$ and the witness $\sigma = \langle \beta, s', \delta, s'' \rangle$.

D A Strong Easy-Hard Relation Against Non-uniform Adversaries

In this section we show a construction for a strong easy-hard relation that works against any polynomial size adversary, and not just adversaries with the amount of non-uniformity bounded by a specific polynomial. Note that since we are not using a black-box simulator, our uniform result does *not* extend automatically to the non-uniform setting (as is the usual case in cryptographic protocols).

The construction of this relation works in the following way. Firstly, similar to what we did before, we construct a strong easy-hard relation $\mathbf{Q}$ that is not an $\mathbf{NP}$ relation, but rather an $\mathbf{Ntime}(T^{O(1)})$ relation for some super-polynomial function $T : \mathbb{N} \to \mathbb{N}$. One difference between this relation $\mathbf{Q}$ and the relation $\mathbf{S}$ we defined in Section C.3, is that the relation $\mathbf{S}$ was a non-interactive easy-hard relation, while the relation $\mathbf{Q}$ will be an *interactive* easy-hard relation.

As the relation $\mathbf{Q}$ will satisfy the additional properties stated in Theorem C.17, we will be able to use Theorem C.17 to obtain an easy-hard $\mathbf{NP}$ relation, and thus a zero-knowledge argument.

Figure 6 shows how the structure of the construction of this zero-knowledge argument. Again, boxes represent primitives and arrows represent theorems of the form "If primitive X exists then primitive Y exists". The new theorem of this section, Theorem D.1, is mark ed with a bold arrow.

D.1 Constructing an Easy-Hard $\mathbf{Ntime}(T^{O(1)})$ Relation Against Non-uniform Adversaries

Fix $T : \mathbb{N} \to \mathbb{N}$ to be some polynomial time computable function that is super-polynomial (e.g. $T(n) = n^{\omega(1)}$). In this section we construct an $\mathbf{Ntime}(T^{O(1)})$ easy-hard relation $\mathbf{Q}$. We start by describing the main idea behind the construction, and then move on to describe the actual construction.

D.1.1 The Main Idea

Consider the problem we're facing in trying to construct an easy-hard relation. We need the simulator to be able to use its knowledge of the generator's code and random tape in some way. In the relation $\mathbf{S}$ of Section C.4, the simulator used this code to learn a *short* explanation of the generator's output. In the current setting, we can no longer use this idea, since the generator's code may be longer than the length of its first message. We thus take a slightly different approach.

Note that we are trying to construct an *interactive* relation and so we're trying to construct a generating protocol, rather than a simple non-interactive generator algorithm. Consider a protocol in which the first message z is sent by the generator's partner and then the generator responds with a message v. Suppose that the honest generator chooses the message v at random in $\{0,1\}^n$. This means that the partner has at most 2^{-n} probability of guessing this message. Yet, if the partner knew the code and random tape of the (possibly cheating) generator, then it *would* be able to determine the generator's next message. This is because, knowing the code and the random tape of the generator, the partner knows the *next message function* of the generator, denoted $NM(\cdot)$. Therefore the partner knows that the generator's reply to the message z would be $NM(z)$.

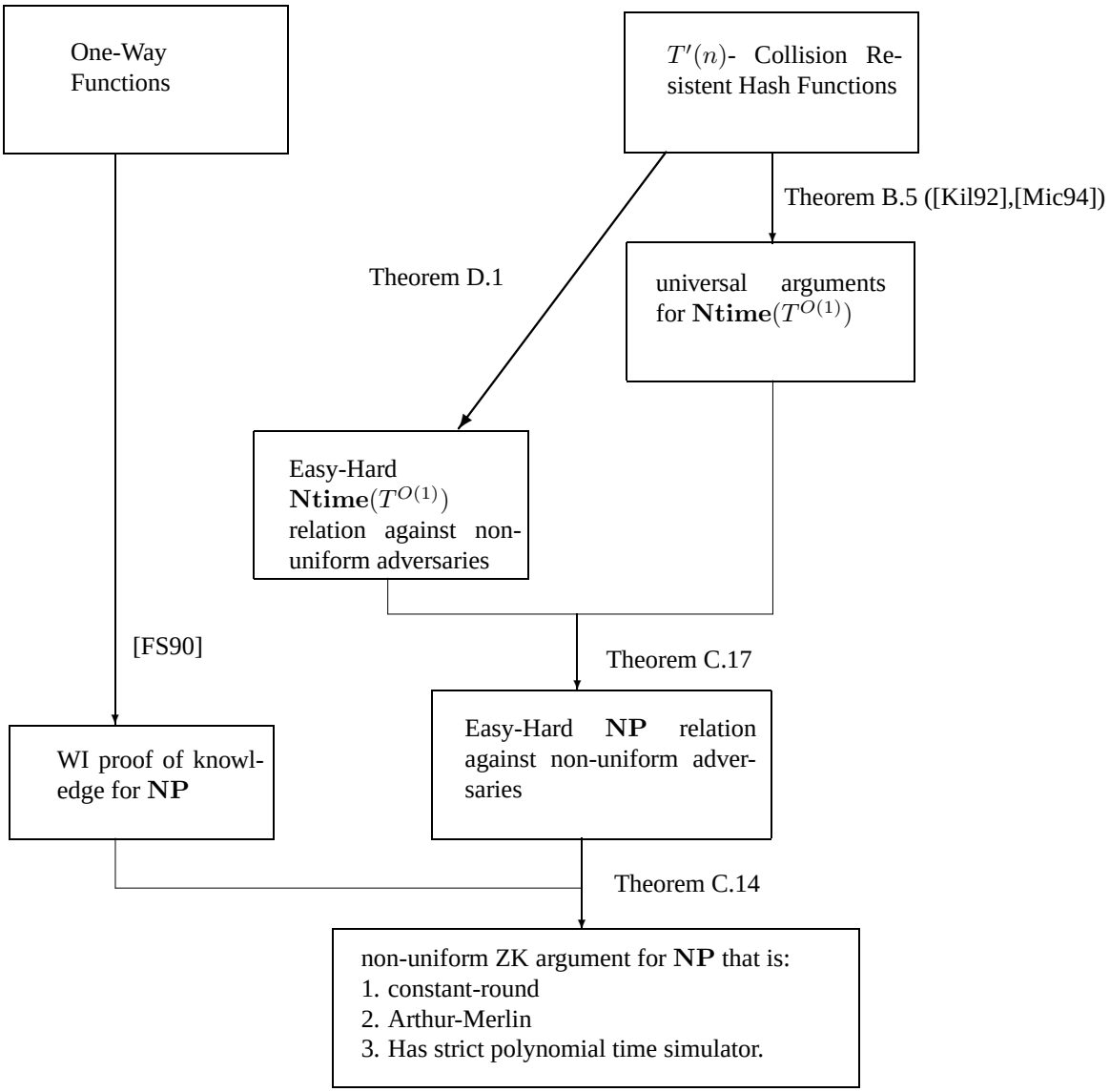

Figure 6: Structure of the construction of a non-uniform zero-knowledge argument (the new theorem of this section is marked by a bold arrow)

The problem is how to convert this knowledge into something useful. Our idea is that the partner's first message z would somehow encode the generator's reply to z (which we denote v). The first idea that comes to mind is that this message z will be a commitment to v (i.e., $z = \mathcal{C}(v; s)$ for some random s) . However there seems to be a problem with this idea, because to implement it the partner would have to find a v and random coins s for the commitment that solve the following equation: $NM(\mathcal{C}(v; s)) = v$.

This does not seem so simple (for instance, consider the case that the next message function is a hash function). Therefore, we make the following modification: the message z would not be a commitment to the reply message v but rather be a commitment to *the code of the next message function* NM. That is the message z is a commitment to a program Π such that $\Pi(z) = v$. In this way the partner's message z does implicitly determine the next message v of the generator, and it is still impossible to produce such a message z with probability higher than 2^{-n} without knowledge of the generator's random tape.

One complication that arises is that the messages in our protocol need to have some fixed polynomial bound on their length, while the code of the generator can be of any polynomial size. We solve this problem by having the message z contain not a commitment to the code of NM itself but rather to a *hash value* of this code. To enable this, we have the generator send as a first message a key κ of the hash function to be used.

Note that checking the condition that $\Pi(z) = v$ can not be done in some *fixed* polynomial time. In this protocol we have a problem not only because Π might run in *any* polynomial time but also because Π might be of *any* polynomial *length*. However, this can be done in time $T(n)^{O(1)}$ which is enough for our purposes.

D.1.2 The Actual Construction

In this section we prove the following theorem:

Theorem D.1. *Let $T : \mathbb{N} \to \mathbb{N}$ be a polynomial time computable function that is super-polynomial (i.e. $T(n) = n^{\omega(1)}$). Let $T' : \mathbb{N} \to \mathbb{N}$ be a function such that $T'(n) = T(n)^{\omega(1)}$. Suppose that $T'(n)$-collision resistent hash functions exist, then there exists an $\mathbf{Ntime}(T^2)$ easy-hard relation $\mathbf{Q}$. Furthermore, this relation satisfies the following additional properties:*

1. *The hardness property holds against $\mathbf{Size}(T^{O(1)})$ adversaries (and not just against $\mathbf{Psize}$ adversaries)*
2. *The simulator for this relation outputs efficiently verifiable witnesses (in the sense of Claim C.11)*

Again, we show a slightly simpler construction under the assumption of existence of simple (1-round) commitment schemes. The construction can be easily modified to use 2-round commitment schemes that are known to exist under the assumption of existence of one-way function (and hence under the assumption of existence of collision resistent hash functions).

Assume that T and T' are functions that have the properties mentioned in the statement of Theorem D.1, and that there exists a $T'(n)$-collision resistent hash function ensemble $\{h_\kappa\}_{\kappa \in \{0,1\}^*}$ (where h_κ maps $\{0,1\}^*$ to $\{0,1\}^{l(|\kappa|}$ for some function $l(\cdot)$ such that $l(n)$ is polynomially related to n). We define the following relation $\mathbf{Q}$: We say that $(\langle \kappa, z, v\rangle, \langle \Pi, s\rangle)$ is in $\mathbf{Q}$ if the following holds:

1. Π is a program with length at most $T(n)$, where $n \stackrel{def}{=} |\kappa|$.
2. $\mathcal{U}(\Pi, z)$ outputs v within $T(n)$ steps. (Recall that $\mathcal{U}$ is a universal Turing machine.)
3. z is a commitment to a hash of Π using s for coins. That is $z = \mathcal{C}_n(h_\kappa(\Pi); s)$. Note that as $|h_\kappa(\Pi)| = l(n)$, the length of z is $nl(n)$‘.

It is easy to see that $\mathbf{Q}$ is an $\mathbf{Ntime}(T^2)$ relation. We claim that it is also an easy-hard relation. We start by defining the generation protocol

Construction D.2. *Generation protocol for* $\mathbf{Q}$

- Common input: 1^n – security parameter.
- Generator's first step (G1): Let $\kappa \leftarrow_{\mathrm{R}} \{0,1\}^n$. Send κ.

- Partner's first step (P1): Let $z \leftarrow \mathcal{C}_n(0^{l(n)})$. Send z.
- Generator's second step (G2): Let $v \leftarrow_R \{0,1\}^n$. Send v.

We claim that the relation $\mathbf{Q}$ satisfies both the hardness and the easiness properties:

Proposition D.3. *The relation $\mathbf{Q}$, with the generation protocol of Construction D.2, satisfies the hardness property with respect to $\mathbf{Size}(T^{O(1)})$ adversaries.*

Proof. Suppose toward contradiction that a $\mathbf{Size}(T^{O(1)})$ adversary P^* exists, that after interacting with the prescribed generator with transcript τ, can output a witness σ such that $(\tau, \sigma) \in \mathbf{Q}$ with non-negligible probability $\epsilon = \epsilon(n)$. In case P^* manages to output such a witness, we say that P^* is *successful*.

We'll use P^* to construct a $T(n)^{O(1)}$ size algorithm A that finds collisions for the hash function, and so derive a contradiction. We define algorithm A as follows:

Algorithm A:

- Input: $\kappa \in \{0,1\}^n$.

1. Feed κ to P^* as the generator's first message (G1). Obtain P^*'s first message, denoted z (step P1). Record the state of P^* at this point (after step P1), we denote this state by t.
2. Perform the following procedure twice. That is for $i = 1, 2$ do:
   (a) Reset P^* to the state t (after step P1).
   (b) Let $v_i \leftarrow_R \{0,1\}^n$.
   (c) Feed P^* with v_i as the generator's second message (step G2). Assume that P^* is successful (otherwise abort). This means that P^* outputs a witness $\langle \Pi_i, s_i \rangle$ such that $(\langle \kappa, z, v_i \rangle, \langle \Pi_i, s_i \rangle) \in \mathbf{Q}$.
3. Output Π_1, Π_2.

Analysis of algorithm A:

We have the following claim, whose proof we omit:

Claim D.3.1. *Suppose that A's input κ is chosen uniformly in $\{0,1\}^n$. With probability $\Omega(\epsilon^3)$, in both times that algorithm A invokes algorithm P^* (Step 2c of algorithm A), algorithm P^* is successful.*

Let us consider this event in which both invocation of P^* are successful. In this case we know that the following holds:

1. $\Pi_1(z) = v_1$
2. $\Pi_2(z) = v_2$
3. $\mathcal{C}_n(h_\kappa(\Pi_1); s_1) = \mathcal{C}_n(h_\kappa(\Pi_2); s_2) = z$

By the perfect binding of the commitment scheme $\mathcal{C}$ it follows that $h_\kappa(\Pi_1) = h_\kappa(\Pi_2)$. Yet as both v_1 and v_2 are chosen uniformly and independently in $\{0,1\}^n$, the probability that $v_1 = v_2$ is negligible (at most 2^{-n}) and so in particular it is $o(\epsilon^3)$. Therefore we see that by Claim D.3.1 with probability $\Omega(\epsilon^3)$ all of the above properties with hold with the additional property that $v_1 \neq v_2$. It follows that $\Pi_2(z) \neq \Pi_1(z)$ and so in particular $\Pi_1 \neq \Pi_2$. Yet this mean that with probability $\Omega(\epsilon^3)$, algorithm A finds a collision for h_κ, contradicting the $T'(n)$-collision resistance of this ensemble. □

Proposition D.4. *The relation $\mathbf{Q}$, with the generation protocol of Construction D.2, satisfies the easiness property.*

Proof. Let G^* be a $\mathbf{Psize}$ generator. Assume that $p(n)$ is a polynomial that bounds G^*'s running time, random tape length, and advice tape length. We construct the following simulator M^* for G^*:

Algorithm M^*:

- Input: 1^n – security parameter.

1. Select random coins for G^*: let $R \leftarrow_R \{0,1\}^{p(n)}$.
2. Compute G^*'s first message: let $\kappa \leftarrow G^*(1^n; R)$. We assume without loss of generality that $\kappa \in \{0,1\}^n$.
3. Using the code of G^* and the string R, compute a program Π of length $O(p(n))$ that computes the *next message function* of G^* at this point. That is for any $z \in \{0,1\}^*$, $\mathcal{U}(\Pi, z) = G^*(1^n, z; R)$.
4. Let $z' \leftarrow h_\kappa(\Pi)$. Note that $|z'| = l(n)$. Choose $s \leftarrow_R \{0,1\}^{nl(n)}$. Compute $z \leftarrow \mathcal{C}_n(z'; s)$.
5. Output $(\langle R, z\rangle, \langle \Pi, s\rangle)$.

It is obvious that M^* runs in polynomial time. We need to show that the first element of M^*'s output is computationally indistinguishable from the view of G^* when interacting with the prescribed partner, and that the second element is a witness to the fact that the transcript that is compatible with this view is in $L(\mathbf{Q})$:

1. The only difference between the distribution of $\langle R, z\rangle$ and the distribution of the view of G^* in the real interaction with the prescribed partner, is that in the latter case z is a commitment to $0^{l(n)}$, instead of being a commitment to z'. Therefore these distributions are computationally indistinguishable by the security of the commitment scheme.
2. If τ is the transcript that corresponds to the execution $\langle R, z\rangle$ (i.e. $\tau = \langle \kappa, z, v\rangle$ where $\kappa = G^*(1^n; R)$ and $v = G^*(1^n, z; R)$) then (if n is sufficiently large) then $(\tau, \langle \Pi, s\rangle) \in \mathbf{Q}$. Indeed, we make the following observation:

   Observation D.5. *$\mathcal{U}(\Pi, z)$ outputs v within $O(p(n))$ steps.*

   As for large enough n, this is smaller than $T(n)$, this proves that $(\tau, \langle \Pi, s\rangle) \in \mathbf{Q}$.

□

Indeed, using Observation D.5, we see that the relation $\mathbf{Q}$ has the property of having a simulator that outputs efficiently verifiable witnesses. This is stated in the following claim:

Claim D.6. *There exists a Turing machine $M_{\mathbf{Q}}$ with the following properties:*

1. *When run with a pair $(\tau, \langle \Pi, s\rangle)$, the machine $M_{\mathbf{Q}}$ halts and returns an answer in at most $T^2(|\tau|)$ steps.*
2. *For any pair $(\tau, \langle \Pi, s\rangle)$, $(\tau, \langle \Pi, s\rangle) \in \mathbf{Q}$ if and only if $M_{\mathbf{Q}}(\tau, \langle \Pi, s\rangle) = 1$.*
3. *There exists a polynomial $q(\cdot)$ such that for any algorithm $G^* \in \mathbf{Psize}$, if M^* is the simulator for G^* (as constructed in the proof of Proposition D.4 that $\mathbf{Q}$ satisfies the easiness property) then for any $(\langle R, z\rangle, \langle \Pi, s\rangle) \leftarrow M^*(1^n)$, $M_{\mathbf{S}}(\tau, \sigma\Pi, z)$ halts in at most $q(p(n))$ steps (where τ is the transcript corresponding to $\langle R, z\rangle$, that is $\tau = \langle G^*(1^n; R), z, G^*(1^n, z; R)\rangle$ and $p(n)$ is a bound on maximum running time, advice tape length, and random tape length of G^* on input 1^n).*

D.2 Obtaining a Non-uniform Zero-Knowledge Argument

Combining Theorems D.1 and C.17, one obtains a easy-hard $\mathbf{NP}$ relation $\mathbf{Q}^*$. From this relation, using Theorem C.14 and the observation that the prescribed generator for both $\mathbf{Q}$ and the relation $\mathbf{Q}^*$ implied by Theorem C.17 only send random messages, one obtains the following Theorem:

Theorem D.7. *Let $T : \mathbb{N} \to \mathbb{N}$ be polynomial-time computable function that is super polynomial (i.e., $T(n) = n^{\omega(1)}$). Assume that $T'(n)$-collision resistent hash functions exist for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$, then there exists a constant-round Arthur-Merlin zero-knowledge argument. Also, the simulator for this protocol runs in* strict *polynomial time.*

Again, for the sake of clarity we write down the description of this zero-knowledge argument. Let R be an $\mathbf{NP}$ relation.

Construction D.8. *A Zero-Knowledge Argument for $L(R)$*

- Common Input: the statement to be proved x, the security parameter 1^n.
- Auxiliary input to prover: w such that $(x, w) \in R$.

1. *Phase 1: Generation protocol for $\mathbf{Q}^*$*

   (a) *Phase 1.1: Generation protocol for $\mathbf{Q}$*

   - Verifier's step (V1.1.1): Choose $\kappa \leftarrow_{\mathrm{R}} \{0,1\}^n$. Send κ.
   - Prover's step (P1.1.1): Let $z \leftarrow \mathcal{C}(0^{l(n)})$. Send z.
   - Verifier's step (V1.1.2): Choose $v \leftarrow_{\mathrm{R}} \{0,1\}^n$. Send v.

   From here on the protocol proceeds exactly as the protocol of Construction C.24. We repeat this description here for the sake of completeness.

   (b) *Phase 1.1: "Encrypted" universal argument*

   The prover and verifier run the universal argument protocol with the prover sending only the commitments to the messages instead of sending them in the clear. Note that in the actual proof the prover only sends commitments to "junk" messages (i.e. all zero strings). Only in the simulation the messages sent are commitments to valid messages of the universal argument system.

   - Verifier's step (V1.2.1): Choose $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^{n^2}$. Send α. Note that this step can be joined with Step V1.1.2.
   - Prover's step (P1.2.1): Let $\hat{\beta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\beta}$.
   - Verifier's step (V1.2.2): Choose $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^{n^2}$. Send γ.
   - Prover's step (P1.2.3): Choose $\hat{\delta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\delta}$. Note that if we will use a standard 3-round witness indistinguishable proof for the next phase (such as the parallel version of the graph 3-colorability zero-knowledge proof) then this step can be joined with the first step of the prover in this proof.

2. *Phase 2: Witness Indistinguishable Proof*

   - The prover proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x, w) \in R$ or σ such that $(\tau, \sigma) \in \mathbf{Q}^*$ (where τ is the transcript of the first phase). The verifier acts according to the witness indistinguishable protocol and will accept if and only if this proof is valid.

E Concurrent Composition

In Sections C and D, we presented two new zero-knowledge arguments: one that is zero-knowledge against adversaries with bounded non-uniformity (Construction C.24), and another that is zero-knowledge against any polynomial size non-uniform adversary (Construction D.8) . We do not know whether these protocols are in fact *concurrent* zero-knowledge protocols.

What we can prove is that both protocols can be modified slightly so that they will remain zero-knowledge when executed concurrently a number of sessions that is bounded by some fixed polynomial in the security parameter. Again, without loss of generality, it is enough to construct a zero-knowledge argument that remains zero-knowledge when executed concurrently n times, where n is the security parameter.

We present first this modification for the first protocol (Construction C.24), and then for the second protocol (Construction D.8).

E.1 A Bounded Concurrency Zero-Knowledge Argument against Bounded Non-uniformity Adversaries

The theorem that we will prove in this section is the following:

Theorem E.1. *Let $T : \mathbb{N} \to \mathbb{N}$ be polynomial time computable function that is super-polynomial (i.e., $T(n) = n^{\omega(1)}$). Assume that $T'(n)$-collision resistent hash functions exist for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$, then there exists a constant-round Arthur-Merlin zero-knowledge argument, against bounded non-uniformity (*$\mathbf{Bsize}$*) verifiers. Also, the simulator for this protocol runs in* strict *polynomial time. Furthermore, this argument remains zero-knowledge when executed n concurrent times, where n is the security parameter.*

We will prove Theorem E.1 by modifying the bounded non-uniformity zero-knowledge argument implied by the proof of Theorem C.23 to an argument that remains zero-knowledge when executed concurrently n times, where n is the security parameter.

Recall that this protocol was obtained from the easy-hard $\mathbf{NP}$ relation $\mathbf{S}^*$, which in turn was obtained from the easy-hard $\mathbf{Ntime}(T(n)^2)$ relation $\mathbf{S}$.

As this protocol is only a bounded non-uniformity zero-knowledge protocol, it is not clear whether it remains zero-knowledge even when executed n times *sequentially*. In fact, we will see that in this protocol it seems that sequential execution is as problematic as concurrent execution, and so the same modification will be needed to make the protocol remain zero-knowledge under n sequential executions and n concurrent executions.

E.1.1 The problem

We do not know if the protocol of Construction C.24 remains zero knowledge under sequential or concurrent execution. We do know that there is a problem when trying to use the simulator of Construction C.25 to simulate several concurrent or consecutive sessions. The problem with both sequential and concurrent execution of this protocol is the following: Recall the definition of the relation $\mathbf{S}$: $(\tau, \Pi) \in \mathbf{S}$ if Π is a *short explanation* for Π. That is $|\Pi| < \frac{|\tau|}{2}$ and $\mathcal{U}(\Pi)$ outputs τ within $T(|\tau|)$ steps. In our protocol, for any verifier V^*, τ was V^*'s first message and so, after using a pseudorandom generator to reduce the amount of randomness V^* used, the simulator could use the code of V^* to provide a short explanation for τ. However, when the protocol is executed more than one time, the message τ could also be a function of the history of previous messages that V^* has seen.

This history consists of all previous messages sent by the simulator to V^* during all other sessions, as in our protocol some of these messages are longer than τ , we can not argue that this history will be shorter than $\frac{|\tau|}{2}$.

E.1.2 The solution

Recall that a basic component in the zero-knowledge argument of Construction C.24 is the generator for the relation $\mathbf{S}$. This generator selects $\tau' \leftarrow_R \{0,1\}^{3n}$ and outputs τ'.

Our approach for the solution is based on the following observation:

Observation E.2. *For any (polynomial time computable) function $m : \mathbb{N} \to \mathbb{N}$ such that $m(n) \geq 3n$ the following generator GEN_m also satisfies the hardness property with respect to the relation $\mathbf{S}$:*
Algorithm GEN_m:

- *Input :* 1^n *- security parameter*

1. *Choose* $\tau' \leftarrow_R \{0,1\}^{m(n)}$
2. *Output* τ'

Observation E.2 implies that the protocol of Construction C.24 remains zero-knowledge also when the function $m(\cdot)$ is set to other values than $m(n) = 3n$. Now let us make the following (unjustified) assumption: suppose that it turned out that all messages sent by the prover in this protocol, will be of length at most $10n^3$, *independently* of the setting of the function $m(\cdot)$. If this was the case, then (as there are less than 10 rounds in the protocol) in n sessions, the length of the entire history of prover messages would be at most $100n^4$. Therefore, if we set $m(n) \stackrel{def}{=} 300n^4$ then we could proceed with the straightforward simulation: the simulator would hardwire the entire history into the code of the verifier.

However, this is not the case. The problem is not with the messages that are sent by the prover in Phase 1 (the generation phase for $\mathbf{S}^*$): we can assume that these messages are of length at most $10n^3$ independently of the setting of $m(\cdot)$. The reason is that these messages are commitments to "junk" strings of the same length as CS prover's

messages. The length of the CS prover's messages has a polynomial dependency on the security parameter but only *polylogarithmic* dependency on the length of the statement that is proved. Therefore we can assume that for any polynomial $m(\cdot)$, the length of the CS prover's messages will be bounded by n^2.

The problem is with the messages sent at phase 2: the witness indistinguishable proof. If we're using a standard witness indistinguishable proof (such as the parallel version of the graph 3-coloring protocol), then the length of the messages in this proof has a *polynomial* dependency on the length of the statement that is proved. Therefore, whatever choice we choose for $m(\cdot)$, the messages sent by the prover in this proof will be of length at least $m(n)$.

However, for the purposes of the simulation it is enough if we have a *short explanation* for the messages of the second phase. That is, if for each message y sent by the prover in the second phase, we have a short (i.e. of length at most $10n^3$) program that outputs y when run on the empty input, then we can carry out the simulation in a straightforward manner. It is this intuition that stands behind the proof of Theorem E.1.

Theorem E.1 is a consequence of the following claim:

Claim E.3. *Under the assumption of existence of $T'(n)$-collision resistent hash functions, the protocol of Construction C.24, when instantiated with $m(n) \stackrel{def}{=} 300n^4$ remains zero-knowledge (w.r.t.* $\mathbf{Bsize}$ *adversaries) when executed n times concurrently.*

Proof Sketch: Let V^* be a $\mathbf{Bsize}$ verifier. We will construct a simulator M^{**} that will simulate an interaction of n concurrent sessions between V^* and the prescribed prover. The simulator M^{**} will be a modified version of the *single session* simulator M^* for V^* which is obtained by Construction C.24.

The simulator M^{**} will do the following. For any $1 \leq i \leq 10n$: Let us denote by y_i the i^{th} prover's message computed by M^{**}. Algorithm M^{**} will compute alongside with y_i a program Π_i which will have the following properties:

1. $|\Pi_i| \leq i10n^3$
2. $\mathcal{U}(\Pi_i)$ outputs $\langle y_1, \ldots, y_i \rangle$ within a fixed polynomial in n number of steps.

Using the programs $\Pi_1, \ldots, \Pi_{10n}$ the rest of the simulation will follow Construction C.24, except for the following: To compute a program Π such that $\mathcal{U}(\Pi) = \tau'$ where τ' is a message of length $m(n) + |x| = 300n^4 + |x|$ sent by V^* after seeing the messages $y_1, \ldots, y_i$, algorithm M^{**} will incorporate the code of Π_i into the program Π and so obtain a program Π such that $|\Pi| \leq \frac{|\tau'|}{2}$.

Computing the programs $\Pi_1, \ldots, \Pi_{10n}$. Algorithm M^{**} will compute the program Π in the following way: If y_i is a message that belongs to the first phase (the generation phase) then it is of length n^3 and one can obtain the program Π_i in the following way:

Program Π_i:

- Input : none
- Hardwired information: The message y_i (of length n^3) , the program Π_{i-1} (of length $(i-1)10n^3$.

1. Let $\langle y_1, \ldots, y_{i-1} \rangle \leftarrow \mathcal{U}(\Pi_{i-1})$.
2. Output $\langle y_1, \ldots, y_{i-1} \rangle$.

Now suppose that the message y_i belongs to the second phase (the witness indistinguishable proof). In this case the message is a function of the following:

1. The prover algorithm. We can assume that the description of this algorithm is finite and so of length at most n.
2. Random coins used by the prover algorithm. By using a pseudorandom generator we can assume that these are also of length at most n.
3. The input to the prover algorithm. This contains the following:

(a) The statement to be proved $\langle \tau', \alpha, \hat{\beta}, \gamma, \hat{\delta} \rangle$. Here we have a problem as the statement is of length at least $m(n) + |x|$. This is because $|\tau'| = m(n) + |x|$. The other elements of the statement are short as $|\alpha| = |\gamma| = n^2$ and $|\hat{\beta}| = |\hat{\delta}| = n^3$.

(b) The witness $\langle \beta, s', \delta, s'' \rangle$. The witness is short as $|\beta| = |\delta| = n^2$ and $|s| = |s''| = n^3$.

(c) (Optionally) a message of the verifier. This message too will be of length at least $m(n) + |x|$.

The key observation is that both the string τ' and the verifier's message can be computed by applying V^* to a prefix of the history up till the i^{th} message. As this prefix can be computed by running $\mathcal{U}(\Pi_{i-1})$ we can compute a program Π_i of length at most $(i-1)10n^3 + 10n^3$ such that $\mathcal{U}(\Pi_i) = \langle y_1, \ldots, y_i \rangle$.

□

E.2 A Bounded Concurrency Non-Uniform Zero Knowledge Argument

In this section we prove that a modification of the zero-knowledge argument of Construction D.8 remains zero-knowledge when executed n times concurrently. That is, we prove the following theorem:

Theorem E.4. *Let $T : \mathbb{N} \to \mathbb{N}$ be polynomial time computable function that is super-polynomial (i.e., $T(n) = n^{\omega(1)}$). Assume that $T'(n)$-collision resistent hash functions exist for some function $T' : \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(n)^{\omega(1)}$, then there exists a constant-round Arthur-Merlin zero-knowledge argument, against polynomial-size (*$\mathbf{Psize}$*) verifiers. Also, the simulator for this protocol runs in* strict *polynomial time. Furthermore, this argument remains zero-knowledge when executed n concurrent times, where n is the security parameter.*

In Section E.2.1 we show how the straightforward approach to proving that the protocol of Construction D.8 is a concurrent zero-knowledge argument fails. In Section E.2.2 we show a modified version of the protocol of Construction D.8, and sketch why this modified version remains zero-knowledge when composed n times concurrently.

E.2.1 The Problem

First of all, note that the zero-knowledge argument of Construction D.8 is an auxiliary input zero-knowledge argument and so is closed under *sequential* composition. This is not hard to show as the simulator can treat the auxiliary information is part of the verifier's advice tape.

To see what happens in the concurrent setting let us recall how our original simulator works. Let V^* be a (possibly cheating) $\mathbf{Psize}$ verifier. One can treat V^* as a function, that given the history of the messages V^* received so far, outputs the next message. Our original simulator sent the message z that was a commitment to a hash to the code of V^*. The next message outputted by the verifier was $v = V^*(z)$. Then the simulator proved that indeed $V^*(z) = v$. The problem in carrying over this analysis to the concurrent setting is that in this model, between the time V^* receives the message z and the time it outputs its response v, other messages may be exchanged in the sessions that are executed concurrently. Let us denote by $y = \langle y_1, \ldots, y_k \rangle$ all the messages that are sent by the prover between the time V^* receives the message z and the times it responds with the message v. In general, the message v that V^* eventually outputs may depend also on y. Although the messages y are also a function of z, it is not clear how the simulator can *prove* in polynomial time that this is the case.

E.2.2 The Solution

Our solution to this problem will be based on a similar idea as the previous section. The idea is that, similarly to what we saw there, if we bound the number of concurrent sessions by n, then there exists a *short explanation* for y; By a short explanation we mean that there exists a program Π such that $\mathcal{U}(\Pi) = y$ and $|\Pi| \leq 100n^4$.

The current definition of the relation $\mathbf{Q}$, on which the protocol of Construction D.8 is based, does not seem to enable us to take advantage of the above observation. Therefore we will base our new zero-knowledge protocol on a relaxed version of the relation $\mathbf{Q}$.

Before defining this relaxation, let us note the following observation regarding the relation $\mathbf{Q}$: For any polynomial time computable function $m(\cdot)$ such that $m(n) \geq n$ the following generation protocol for $\mathbf{Q}$ satisfies both the hardness and the easiness conditions:

Construction E.5. *Generation protocol for* $\mathbf{Q}$

- Common input: 1^n – security parameter.
- Generator's first step (G1): Let $\kappa \leftarrow_R \{0,1\}^n$. Send κ.
- Partner's first step (P1): Let $z \leftarrow \mathcal{C}_n(0^{l(n)})$. Send z.
- Generator's second step (G2): Let $v \leftarrow_R \{0,1\}^{m(n)}$. Send v.

We now define the relation $\mathbf{Q}'$ by introducing the following relaxation to the definition of $\mathbf{Q}$: instead of requiring that z completely determines the next message v, we require that it restricts the possibilities of v to a small subset (of size at most $2^{|v|/2}$). That is we say that $\langle \kappa, z, v \rangle \in L(\mathbf{Q}')$ if z is a commitment to a hash of a program Π that satisfies that $\mathcal{U}(\Pi, z, y') = v$ for some *short* string y' . Formally we say that $(\langle \kappa, z, v \rangle, \langle \Pi, s, y' \rangle) \in \mathbf{Q}'$ if the following holds:

1. Π is a program with length at most $T(n)$ where $n \stackrel{def}{=} |\kappa|$.
2. z is a commitment to a hash of Π using s for coins.
   That is, $z = \mathcal{C}_n(h_\kappa(\Pi); s)$. Note that $|h_\kappa(z)| = l(|n|)$.
3. $\mathcal{U}(\Pi, z, y')$ outputs v within $T(n)$ steps.
4. $|y'| \leq \frac{|v|}{2}$.

The generation protocol of Construction E.5 can be viewed also as a generation protocol for the relation $\mathbf{Q}'$ (not just for the relation $\mathbf{Q}$). In fact, we have the following theorem:

Theorem E.6. *Assuming the existence of $T'(n)$-collision resistent hash functions (where $T'(n) = T(n)^{\omega(1)}$, $T(n) = n^{\omega(1)}$) the relation $\mathbf{Q}'$ is an easy-hard $\mathbf{Ntime}(T^2)$ relation with respect to the generation protocol of Construction E.5. Furthermore, the relation $\mathbf{Q}'$ with this generation protocol satisfies the following additional properties:*

1. *The hardness condition holds against $\mathbf{Size}(T^{O(1)})$ adversaries (rather than just against $\mathbf{Psize}$ adversaries).*
2. *The simulator guaranteed by the easiness condition outputs efficiently verifiable witnesses (in the sense of Claim D.6).*

We omit the proof of Theorem E.6 which is similar to the proof of Theorem D.1. Plugging this generation protocol into Theorems C.17 and C.14, we obtain the following zero-knowledge argument (this is the same protocol as Construction D.8 with the only differences being that the relation $\mathbf{Q}$ is replaced with $\mathbf{Q}'$ and the message v is chosen uniformly in $\{0,1\}^{m(n)}$ instead of $\{0,1\}^n$):

Construction E.7. *A Zero-Knowledge Argument for $L(R)$*

- Common Input: the statement to be proved x, the security parameter 1^n.
- Auxiliary input to prover: w such that $(x, w) \in R$.

1. *Phase 1: Generation protocol for $\mathbf{Q}^*$*

   (a) *Phase 1.1: Generation protocol for $\mathbf{Q}$*
   - Verifier's step (V1.1.1): Choose $\kappa \leftarrow_R \{0,1\}^n$. Send κ.
   - Prover's step (P1.1.1): Let $z \leftarrow \mathcal{C}(0^{l(n)})$. Send z.
   - Verifier's step (V1.1.2): Choose $v \leftarrow_R \{0,1\}^{m(n)}$. Send v.

   (b) *Phase 1.1: "Encrypted" universal argument*
   The prover and verifier run the universal argument protocol with the prover sending only the commitments to the messages instead of sending them in the clear. Note that in the actual proof the prover only sends commitments to "junk" messages (i.e. all zero strings). Only in the simulation the messages sent are commitments to valid messages of the universal argument system.

- Verifier's step (V1.2.1): Choose $\alpha \leftarrow_R \{0,1\}^{n^2}$. Send α. Note that this step can be joined with Step V1.1.2.
- Prover's step (P1.2.1): Let $\hat{\beta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\beta}$.
- Verifier's step (V1.2.2): Choose $\gamma \leftarrow_R \{0,1\}^{n^2}$. Send γ.
- Prover's step (P1.2.3): Choose $\hat{\delta} \leftarrow \mathcal{C}(0^{n^2})$. Send $\hat{\delta}$. Note that if we will use a standard 3-round witness indistinguishable proof for the next phase (such as the parallel version of the graph 3-colorability zero-knowledge proof) then this step can be joined with the first step of the prover in this proof.

2. *Phase 2: Witness Indistinguishable Proof*

   - The prover proves using the witness indistinguishable proof of knowledge that it knows either w such that $(x,w) \in R$ or σ such that $(\tau,\sigma) \in \mathbf{Q}'^*$ (where τ is the transcript of the first phase). The verifier acts according to the witness indistinguishable protocol and will accept if and only if this proof is valid.

Using the observations above, we have the following claim, that proves Theorem E.4:

Claim E.8. *Under the assumption of existence of $T'(n)$-collision resistent hash functions, the protocol of Construction C.24, when instantiated with $m(n) \stackrel{def}{=} 300n^4$ remains zero-knowledge (w.r.t.* $\mathbf{Psize}$ *adversaries) when executed n times concurrently.*

F Universal Arguments

In this section we provide a construction of universal arguments and a proof sketch for Theorem B.5. The proof uses two components: random access hashing (also known as Merkle's hash-tree [Mer90]) and a form of the PCP (Probabilistically Checkable Proofs) Theorem.

> **Note:** Subsequently to this work, together with Goldreich [BG01] we have given a new analysis for the universal arguments constructions. It is shown in [BG01] that for the purposes of this paper, one can construct universal argument system under the standard assumption of hash functions strong against all polynomial-sized circuits. Furthermore, in our opinion, the analysis in [BG01] is presented better than the analysis in this appendix.

F.1 Random Access Hashing

A *random access hash function* is first of all, a hash function that allows a sender to hash down a long string x into a shorter string y. In addition we want that if the receiver asks the sender for a particular bit i of x, then the sender would be able to not only provide the value b of x_i but also to *prove* to the receiver that indeed $b = x_i$. More formally we'll require that the hash function H has the following property: For any $1 \leq i \leq |x|$, given x and H one can compute not only the the value b of x_i, but also a *certificate* σ that indeed $b = x_i$. The certificate σ is validated against the hash value y. We require that for any probabilistic polynomial time[31] time adversary, and for any i, it is infeasible to generate two certificates σ_0, σ_1 such that both pass verification against y and such that σ_b certifies that the value of x_i is b. Of course one way to construct such certificates would be to say that the value x itself is a certificate for the value of all the bits of x, but we require that the certificate for the value of a single bit would be shorter length: only a fixed polynomial in the security parameter and not allowed to depend on the length of x.

In the formal definition we talk about an *ensemble* of hash functions:

Definition F.1. A function ensemble $\{H_\alpha\}_{\alpha \in \{0,1\}^*}$ is called a *$T'(n)$-collision resistent random access hash function ensemble* if the following holds:

1. For any $\alpha \in \{0,1\}^n$, H_α is a function $H_\alpha : \{0,1\}^* \to \{0,1\}^{l(n)}$ where $l(n)$ is a function polynomially related to n. We also require that the ensemble is polynomially computable. That is, there exists a polynomial time Turing machine $EVAL$ such that for any $\alpha, x \in \{0,1\}^*$, $EVAL(\alpha, x) = H_\alpha(x)$.

[31] Actually we'll need to work against any $T'(n)$ time adversary for some super-polynomial function $T'(\cdot)$.

2. There's a polynomial time verification algorithm V that satisfies the following: (V gets as inputs the index α, the hash value y, the bit location i, the alleged value b and the certificate for it σ)

   (a) For any $y \in \{0,1\}^{l(n)}$, it is infeasible to make V accept two contradicting certificates for the same bit of the preimage: For any $poly(T'(n))$ time PPT A the probability that A succeeds in outputting two contradicting certificates for the same location is negligible. Formally we require that

   $$\Pr_{\alpha \leftarrow_R \{0,1\}^n}[(y, i, \sigma_0, \sigma_1) \leftarrow A(\alpha); V(\alpha, y, 0, \sigma_0) = 1 \wedge V(\alpha, y, i, 1, \sigma_1) = 1] \leq \frac{1}{T'(n)}$$

   (b) There is a way to convince V of the correct value of the bit of the preimage: there exists a polynomial time algorithm P such that for any $x \in \{0,1\}^*$, $1 \leq i \leq |x|$:

   $$\Pr_{\alpha \leftarrow_R \{0,1\}^n}[V(\alpha, H_\alpha(x), i, x_i, P(\alpha, x, i)) = 1] = 1$$

   We also require that the certificate length will be bounded by a *fixed* polynomial in n. That is that there exists a polynomial $p(\cdot)$ such that for any $\alpha \in \{0,1\}^n$, $x \in \{0,1\}^*$, $1 \leq i \leq |x|$, $|P(\alpha, x, i)| \leq p(n)$. This means that the length of the certificate is *not* allowed to depend on $|x|$.

We have the following Theorem:

Theorem F.2 ([Mer90]). *Suppose that $T'(n)$-collision resistent hash functions exist for some super-polynomial function $T' : \mathbb{N} \to \mathbb{N}$. Then there exists a $T'(n)$-collision resistent random access hash function ensemble.*

Proof Sketch: FILL IN (Merkle tree) □

F.2 Probabilistically Checkable Proofs (PCP)

In this section we state the result on probabilistically checkable proofs that we need for universal arguments.

Definition F.3. Let $r, q, \epsilon : \mathbb{N} \to \mathbb{N}$ be functions. We say that a language L is in $\mathbf{PCP}_{\epsilon(\cdot)}(r(\cdot), q(\cdot))$ if there exists a probabilistic polynomial time oracle Turing machine V such that:

1. For any $x \in L$ there exists a string π_x such that

   $$\Pr[V^{\pi_x}(x) = 1] = 1$$

2. For any $x \notin L$, $\pi \in \{0,1\}^*$

   $$\Pr[V^{\pi}(x) = 1] < \epsilon(|x|)$$

3. On any input $x \in \{0,1\}^*$, V uses at most $r(|x|)$ random coins, and asks at most $q(|x|)$ *non adaptive* queries.

The following Theorem is is implied in [FGL+96],[ALM+98]:

Theorem F.4. *For any function $T : \mathbb{N} \to \mathbb{N}$ such that $T(n) \leq 2^n$,*

$$\mathbf{Ntime}(T) \subseteq \mathbf{PCP}_{\frac{1}{2}}(polylog(T), polylog(T))$$

We need to use additional properties that are indeed satisfied by the proof of Theorem F.4:

1. For any $\mathbf{Ntime}(T)$ relation R that is accepted by a $T(n)$ time Turing machine M_R, there's a polynomial atime algorithm algorithm P that satisfies the following: If $M_R(x, y)$ outputs 1 within t steps for some $(x, y) \in R$, then $P(x, y, 1^t) = \pi_x$ where $|\pi_x| \leq t^3$ and π_x is a string such that the PCP verifier V for $L(R)$ satisfies

   $$\Pr[V^{\pi_x}(x) = 1] = 1$$

2. (This is implied by the error correcting properties of the codes used in proving in the $\mathbf{PCP}$ theorem, see [STV98]) For any $\mathbf{Ntime}(T)$ relation R, there's a PCP verifier V for $L(R)$ and a $poly(T(n))$ time algorithm E such that for any $x \in \{0,1\}^n$ and any string π that satisfies $\Pr[V^{\pi}(x) = 1] > \frac{1}{2}$, $E(\pi) \in R(x)$

We'll use the following Lemma:

Lemma F.5. *For any function $T : \mathbb{N} \to \mathbb{N}$ such that $T(n) \leq 2^n$,*

$$\mathbf{Ntime}(T) \subseteq \mathbf{PCP}_{2^{-n}}(n \cdot polylog(T), n \cdot polylog(T)) \subseteq \mathbf{PCP}_{2^{-n}}(n^2, n^2)$$

Furthermore for any $\mathbf{Ntime}(T)$ relation R there's a $\mathbf{PCP}_{2^{-n}}(n^2, n^2)$ verifier V for $L(R)$ that satisfies the following properties:

1. *There is a polynomial time algorithm algorithm P that satisfies the following: If $M_R(x, y)$ outputs 1 within t steps for some $(x, y) \in R$, then $P(x, y, 1^t) = \pi_x$ where $|\pi_x| \leq t^3$ and*

$$\Pr[V^{\pi_x}(x) = 1] = 1$$

2. *There's a $poly(T(n))$ time algorithm E such that for any $x \in \{0,1\}^n$ and any string π that satisfies $\Pr[V^{\pi}(x) = 1] > 2^{-n}$, $E(\pi) \in R(x)$*

Proof Sketch: The proof is simple. Let R be an $\mathbf{Ntime}(T)$ relation. Consider the $\mathbf{PCP}_{\frac{1}{2}}(polylog(T), polylog(T))$ verifier V guaranteed to exist by Theorem F.4 and suppose that it satisfies the additional properties.

Now let V_k be a verifier that uses $k \cdot polylog(T)$ randomness and $k \cdot polylog(T)$ queries that does the following with input x and oracle access to π: Run k independent times $V^{\pi}(x)$ and reject if V rejects in any one of these times.

It is easy to see that if $\Pr[V^{\pi}(x) = 1] \leq \frac{1}{2}$ then $\Pr[V_k^{\pi}(x) = 1] \leq 2^{-k}$. Therefore the verifier V_n is the verifier that we need to get the result of the theorem. □

F.3 The universal argument System

In this section we prove Theorem B.5. 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 universal argument 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 universal argument system for R in the following way:

Construction F.6. A universal argument 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 $\alpha \leftarrow_R \{0,1\}^n$, 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 F.7. *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 $\gamma \leftarrow_R n^2$, 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 F.8. *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 $\gamma \leftarrow_R \{0,1\}^{n^2}$
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.