Universal Arguments and their Applications

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

Oded Goldreich*
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded@wisdom.weizmann.ac.il

December 2, 2001

Abstract

We put forward a new type of computationally-sound proof systems, called universal-arguments, which are related but different from both CS-proofs (as defined by Micali) and arguments (as defined by Brassard, Chaum and Crepeau). In particular, we adopt the *instance-based* prover-efficiency paradigm of CS-proofs, but follow the computational-soundness condition of argument systems (i.e., we consider only cheating strategies that are implementable by *polynomial-size circuits*).

We show that universal-arguments can be constructed based on standard intractability assumptions that refer to polynomial-size circuits (rather than assumptions referring to subexponential-size circuits as used in the construction of CS-proofs). As an application of universal-arguments, we weaken the intractability assumptions used in the recent non-black-box zero-knowledge arguments of Barak. Specifically, we only utilize intractability assumptions that refer to polynomial-size circuits (rather than assumptions referring to circuits of some "nice" super-polynomial size).

Keywords: Probabilistic proof systems, computationally-sound proof systems, zero-knowledge proof systems, proofs of knowledge, probabilistic checkable proofs (PCP), collision-free hashing, witness indistinguishable proof systems, error-correcting codes,

*Supported by a MINERVA Foundation, Germany.

1 Introduction

Various types of *probabilistic* proof systems have played a central role in the development of computer science in the last two decades. The best known ones are interactive proofs [20], zero-knowledge proofs [20], and probabilistic checkable proofs [17, 5, 12, 2], but other notions such as various types of computationally-sound proofs (e.g., arguments [10] and CS-proofs [23]) and multi-prover interactive proofs [9] have made a prominent appearance as well. *Do we really need yet another type of probabilistic proof systems?*

We believe that the answer is positive. The number of different related notions we need is exactly the number of different notions that are natural, interesting and/or useful. Confining ourself to usefulness, we note that the new type of computationally-sound proof systems introduced in this paper has emerged in the context of trying to improve the recent constructions of non-black-box zero-knowledge arguments of Barak [6]. Furthermore, these proof system seems to be inherent to certain diagonalization techniques used in [6] and (in a different context) in [11].

1.1 Motivation: Applying diagonalization in cryptography

A naive idea, which was discarded for decades in Cryptography, is to construct a cryptographic scheme by "diagonalization"; for example, enumerating all probabilistic polynomial-time adversaries and making sure that each of them fails. The main reason that this idea was discarded is that the resulting scheme will (necessary) be more complex than the class of adversaries it defeats (while in cryptography a scheme should withstand adversaries that are (at least slightly) more complex than the scheme).

Still, as noted by Canetti, Goldreich and Halevi [11] and Barak [6], a small twist on diagonalization may be useful in Cryptography. The twist is to use diagonalization in order to build a "trapdoor" so that the trapdoor can be used in some "imaginary setting" (e.g., by the simulator [6]), but not in the "real" setting (e.g., an actual execution of the proof system [6]).[1] Thus, the complexity of the simulator is effected by the diagonalization, whereas the complexity of the actual execution of the interactive proof is independent of the diagonalization. Specifically, the trapdoor constructed by Barak [6] is knowledge of the (adversarial) verifier's strategy, where this strategy (which is the locus of diagonalization) may be any polynomial-size circuit (where the polynomial is determined only after the proof system is specified). In the actual execution the (zero-knowledge) prover does not use this trapdoor (but rather uses an NP-witness to the real input), and so its complexity is independent of the complexity of the trapdoor (i.e., the cheating verifier's strategy). However, the simulator uses the trapdoor, and so its complexity depends on the latter (and so every polynomial-size adversary yields a related polynomial-time simulation). The above may make sense because the protocol utilizes a witness indistinguishable (WI) proof for which both the NP-witness to the real input and the trapdoor (i.e., verifier's strategy) are valid witnesses. In particular, it follows that the honest verifier strategy in the WI proof must be independent of the length of the "NP-witness" used by the prover. That is, the witness is of polynomial length, but this polynomial is determined (and fixed) only after the proof system is specified.

We conclude that in order to use diagonalization as above, we should have a (WI) proof system that is capable of handling any "NP-statement" (and not merely statements in any a-priori fixed NP-language). Put in other words, we need a *single* proof system that can be used to provide proofs

[1] In [11], the "imaginary setting" is an implementation of the random-oracle by a function ensemble (shown not to exist), whereas the "real setting" is the ideal (Random Oracle Model) setting in which the scheme uses a random-oracle. (Indeed our perspective here is opposite to the one in [11] where the random-oracle is considered "imaginary" and its implementations by function ensembles are considered "real").

for *any* language L in $\mathcal{NP}$ such that the running time and communication needed for verifying that $x \in L$ is bounded by a fixed (i.e., *single*) polynomial in $|x|$, that does *not* depend on the language L.[2] We stress that in contrast, usually when people talk about "proof systems for $\mathcal{NP}$" what they mean is that every langauge $L \in \mathcal{NP}$ has a different proof system and the complexity of verifying that $x \in L$ is bounded by an L-*dependent polynomial* in $|x|$.

1.2 The notion of universal arguments

For sake of simplicity, we define and present proof systems only for the following *universal language*[3] $L_{\mathcal{U}}$: the tuple (M, x, t) is in $L_{\mathcal{U}}$ if M is a non-deterministic machine that accepts x within t steps. Clearly, every $\mathcal{NP}$-language L is linear-time reducible to $L_{\mathcal{U}}$ (i.e., $x \mapsto (M_L, x, 2^{|x|})$, where M_L is any fixed non-deterministic polynomial-time deciding L). Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all "NP-statements" (in a uniform manner); that is, there exists a single polynomial p such that for every $L \in \mathcal{NP}$, the complexity of verifying that $x \in L$ is bounded by $p(|x|)$. In fact, $L_{\mathcal{U}}$ is $\mathcal{NE}$-complete (by an analogous linear-time reduction).[4] We consider also the natural witness-relation for $L_{\mathcal{U}}$, denoted $R_{\mathcal{U}}$: the pair $((M, x, t), w)$ is in $R_{\mathcal{U}}$ if M (viewed here as a two-input deterministic machine) accepts (x, w) within t steps.

Loosely speaking, a universal argument system (or a universal argument system for $L_{\mathcal{U}}$) is a two-party protocol (P, V), for common inputs of the form (M, x, t), that satisfies the following:

Efficient verification: The total time spent by the (probabilistic) verifier V is polynomial in length of the common input (i.e., polynomial in $|(M, x, t)| = O(|M| + |x| + \log t)$). In particular, all messages exchanged in the protocol have length so bounded.

Completeness by a relatively-efficient prover: For every $((M, x, t), w)$ in $R_{\mathcal{U}}$, on common input (M, x, t), given auxiliary input w the prover P always convinces V. Furthermore, the total time spent by P in this case is bounded by a fixed polynomial in $T_M(x, w) \leq t$, where $T_M(x, w)$ is the number of steps taken by M on input (x, w).

Computational Soundness: For every polynomial-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ and every $(M, x, t) \in \{0,1\}^n \setminus L_{\mathcal{U}}$, the probability that, on common input (M, x, t), the cheater C_n fools V (to accept (M, x, t)) is negligible (as a function of the length of (M, x, t)).

(The actual definition appears in Section 2.)

Universal-arguments are related but different from both CS-proofs (as defined by Micali [23]) and arguments (as defined by Brassard, Chaum and Crepeau [10]). The efficient-verification condition is identical in all definitions, the "completeness by a relatively-efficient prover" condition follows the instance-based paradigm of CS-proofs (but provides the prover with an auxiliary input), and the above computational-soundness condition is exactly as in argument systems (and different from the one in CS-proofs). Thus, in a sense, universal-arguments are a hybrid of arguments and CS-proofs. We comment that computational-soundness seems unavoidable in any proof system for $L_{\mathcal{U}}$ satisfying the efficient-verification condition (even just "uniformly for all $\mathcal{NP}$"): statistical soundness would have implied that $L_{\mathcal{U}}$ (or "just" $\mathcal{NP}$) is in $\mathrm{DSPACE}(p)$, for some fixed polynomial p.

[2]In particular it may be the case that $L \in \mathrm{Ntime}(p)$ where $p(\cdot)$ is a polynomial *larger* than the fixed polynomial bounding the verifier's complexity.

[3]The nice aspect about $L_{\mathcal{U}}$ is that it comes with a natural measure of complexity of instances: the complexity of (M, x, t) is the actual time it takes M to accept x (when using either the best or a given sequence of non-deterministic choices). Such a complexity measure is pivotal to the refined formulation of the prover complexity condition.

[4]Furthermore, every language in $\mathcal{NEXP}$ is polynomial-time (but not linear-time) reducible to $L_{\mathcal{U}}$.

A strange-looking aspect of universal-arguments is that, on some inputs, the designated prover may run more time than allowed to cheating provers[5] (i.e., a fixed polynomial in $T_M(.,.)$ may be more than an arbitrary polynomial in the length of the common input). However, in typical application (such as ours), the designated prover will never be invoked on such inputs (i.e., requiring it to run for time that is super-polynomial in the length of the common input). In fact, one may think of universal-arguments as relating to three categories of inputs (as in promise problems), where the polynomial in Items 1 and 3 is selected *after the system is specified*:

1. YES-instances for which t_M is polynomial in the length of $|M| + |x|$. For such inputs the designated prover is a uniform machine that given an appropriate auxiliary input convinces the verifier while running in time that is polynomial in the length of $|M| + |x|$.
2. NO-instances. For such inputs (even non-uniform) polynomial (in $|M| + |x|$) sized circuits cannot fool the verifier to accept. Note that if t_M is super-polynomial in $|M| + |x|$ then it may be the case that t_M sized circuits *can* fool the verifier into accepting.
3. Other YES-instances (i.e., for which t_M is super-polynomial in the length of $|M| + |x|$). For such inputs there is no guarantee as to whether or not (non-uniform) polynomial-size circuits can convince the verifier to accept.

Thus, for every language in $\mathcal{NP}$, we may select the polynomial (in Items 1 and 3) such that all YES-instances fall into the first category. This means that a universal-argument yields a family of argument systems, one per each $L \in \mathcal{NP}$, but the complexity of verification in all these argument systems is bounded by the same (universal) polynomial (in the length of the common input). The latter holds also for CS-proofs, but we will be able to construct a universal-argument based on weaker assumption than the ones seem necessary for constructing CS-proofs (cf. Footnote 5).

1.3 The construction of universal arguments

By adapting the construction of Kilian [21], one can easily show that the existence of *strong* collision-free hashing functions implies the existence of universal arguments (and even CS-proofs for $L_{\mathcal{U}}$; cf. Micali [23]). By strong collision-free hashing we mean families of functions for which collisions are hard to find even by using *subexponential-size* circuits. The goal, achieved in this paper, is to construct universal arguments based only on standard collision-free hashing; that is, families of functions for which collisions are hard to find by *polynomial-size* circuits. That is, we obtain:

Theorem 1.1 *The existence of* (standard) *collision-free hashing functions implies the existence of universal arguments. Furthermore, these proof systems are of the public-coin* (*Arthur–Merlin [3]*) *type and use a constant number of rounds.*

Our construction of universal arguments (also) adapts Kilian's construction [21] in a straightforward manner. Our contribution is in the analysis of this construction. Unlike in previous analysis (as in [21] and [23]), in establishing computational-soundness via contradiction, we cannot afford to derive a collision-forming circuit of size that is (at least) polynomial in the complexity of the

[5] This phenomena does not occur in arguments [10] and in CS-proofs [23]: Arguments were defined only for individual languages in $\mathcal{NP}$, and so the issue never arises. In case of CS-proofs, the definition of computational-soundness relates to cheating provers of size exponential in the security parameter, which is typically set to be linear in the length of the common input [23]. Thus, the cheating provers are always allowed more running time than the designated prover (since its running-time is always at most exponential in the length of the common input). However, it seems that allowing the adversaries time that is exponential in the security parameter requires using intractability assumptions that refer to exponential (or sub-exponential) circuits.

designated prover (which may be exponential in the input length).[6] We need to derive collision-forming circuit of size that is polynomial in the input length. Indeed, doing so allows us to use standard collision-free hashing (rather than strong ones).

The analysis is further complicated by our desire to establish a "proof of knowledge" property, which is needed in for our main application (discussed next).

1.4 Application to zero-knowledge arguments

Barak's construction [6] of non-black-box zero-knowledge arguments (for any language in $\mathcal{NP}$) uses a witness indistinguishable (WI) argument of knowledge for $R_{\mathcal{U}}$.[7] In his protocol, the prover uses this WI argument (of knowledge) to prove that it knows either an NP-witness for the original common input or a program that fits the verifier functionality (as reflected in the challenge-respond exchange that follows). Thus, as a first step, we need to transform our universal argument (of knowledge) into a corresponding WI universal argument (of knowledge). The transformation follows Barak's transformation [6], but then we encounter a second place where Barak uses a super-polynomial hardness assumption: Barak uses a collision-free hashing function to hash "$L_{\mathcal{U}}$-witnesses" (into fix-length strings), where the length of these witnesses is bounded by some super-polynomial function (but not by any polynomial). Consequently, a collision on such long strings only yields violation of a super-polynomial collision-free assumption. To avoid super-polynomial hardness assumptions, we hash these witnesses by combining "tree-hashing" (as in Kilian's construction [21]) with an error-correcting code. Specifically, first the witness string is encoded using an error-correcting code, and then the "tree-hashing" is applied to the result. Thus, if two different strings are so hashed to the same value, then we can form a collision with respect to the basic hashing function (used in the "tree-hashing") by considering a uniformly selected leaf (which is quite likely to be assigned different values under an error-correction coding of different strings). Combining the above, we obtain:

Theorem 1.2 *The existence of* (standard) *collision-free hashing functions implies the existence of* (non-black-box) *zero-knowledge arguments, for any language in $\mathcal{NP}$, with the following properties:*

- *The protocol has a constant number of rounds and uses only public-coins;*
- *The simulator runs in strict* (rather than expected) *probabilistic polynomial-time;*
- *The protocol remains zero-knowledge when, say, n^2 copies are executed concurrently.*

Theorem 1.2 establishes the main result of Barak's work [6], under a weaker assumption: We only assume the existence of hashing functions that are resilient with respect to polynomial-size circuits (rather than with respect to some super-polynomial-size circuits).

[6] Specifically, Kilian's construction [21] uses a PCP system, and the contradiction hypothesis is shown to yield a collision-forming circuit that is always bigger than the relevant PCP-oracle. Instead, we show how to obtain a collision-forming circuit that is smaller than the relevant PCP-oracle.

[7] In fact, Barak uses a CS-proof (of knowledge) for $R_{\mathcal{U}}^f \subset R_{\mathcal{U}}$, where f is any "nice" super-polynomial function (e.g., $f(n) = n^{\log_2 n}$) and $((M,x,t),w)$ is in $R_{\mathcal{U}}^f$ only if $t \leq f(|x|)$. He constructs such CS-proof assuming the existence of hashing functions that are resilient with respect to f-size circuits (rather than subexponential hardness which would have been required for CS-proof for $R_{\mathcal{U}}$).

2 The Definition of Universal Arguments

In continuation to the discussion in Subsection 1.2, we now define universal argument systems (for $L_{\mathcal{U}}$). Recall that $L_{\mathcal{U}} = \{(M,x,t) : \exists w \text{ s.t. } ((M,x,t),w) \in R_{\mathcal{U}}\}$, where $((M,x,t),w) \in R_{\mathcal{U}}$ if M accepts (x,w) within t steps. Let $T_M(x,w)$ denote the number of steps made by M on input (x,w); indeed, if $((M,x,t),w) \in R_{\mathcal{U}}$ then $T_M(x,w) \le t$. Recall that $|(M,x,t)| = O(|M|+|x|+\log t)$; that is, t is given in binary.

We consider a pair of (interactive) strategies, denoted (P,V), and let $(P(w),V)(y)$ denote the output of V when interacting with $P(w)$ on common input y, where $P(w)$ denotes the functionality of P when given auxiliary input w. We denote by $\mu : \mathsf{N} \to [0,1]$ an unspecified negligible function; that is, for every positive polynomial p and all sufficiently large n's, it holds that $\mu(n) < 1/p(n)$.

In the following definition, we incorporate a (weak) "proof of knowledge" property (which was mentioned in Subsections 1.3 and 1.4, but not in Subsection 1.2).

Definition 2.1 (universal argument): *A* universal-argument system *is a pair of strategies, denoted (P,V), that satisfies the following properties:*

Efficient verification: *There exists a polynomial p such that for any $y = (M,x,t)$, the total time spent by the* (probabilistic) *verifier strategy V, on common input y, is at most $p(|y|)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y|)$.*

Completeness by a relatively-efficient prover: *For every $((M,x,t),w)$ in $R_{\mathcal{U}}$,*

$$\Pr[(P(w),V)(M,x,t) = 1] \; = \; 1$$

Furthermore, there exists a polynomial p such that the total time spent by $P(w)$, on common input (M,x,t), is at most $p(T_M(x,w)) \le p(t)$.

Computational Soundness: *For every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$, and every $(M,x,t) \in \{0,1\}^n \setminus L_{\mathcal{U}}$,*

$$\Pr[(\widetilde{P}_n,V)(M,x,t) = 1] \; < \; \mu(n)$$

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

A weak Proof of Knowledge Property: *For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds:*[8] *for every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$, and every sufficiently long $y = (M,x,t) \in \{0,1\}^*$ if $\Pr[(\widetilde{P}_n,V)(y) = 1] > 1/p(|y|)$ then*

$$\Pr_r\left[\begin{array}{c} \exists w = w_1 \cdots w_t \in R_{\mathcal{U}}(y) \\ \quad \forall i \in \{1,...,t\} \quad E_r^{\widetilde{P}_n}(y,i) = w_i \end{array}\right] > \frac{1}{p'(|y|)}$$

where $R_{\mathcal{U}}(y) \stackrel{\text{def}}{=} \{w : (y,w) \in R_{\mathcal{U}}\}$ and $E_r^{\widetilde{P}_n}(.,.)$ denotes the function defined by fixing the random-tape of E to equal r, and providing the resulting E_r with oracle access to $\widetilde{P}_n$. The oracle machine E is called a (knowledge) extractor*.*

A few comments regarding the weak proof-of-knowledge property are in place. First, note that (in the good case) $E_r^{\widetilde{P}_n}(y,.)$ is an *implicit representation* of a witness $w \in R_{\mathcal{U}}(y)$ (i.e., any specific bit of

[8] Indeed, the polynomial p' as well as the (polynomial) running-time of E may depend on the polynomial p (which defines the noticeable threshold probability below).

w is obtained by instantiating the second input to $E_r^{\widetilde{P}_n}(y,.)$ accordingly). If $\Pr[(\widetilde{P}_n, V)(y) = 1] > 1/p(|y|)$ then at least an $1/p'(|y|)$ fraction of the possible r's yield such implicit representations (of strings in $R_{\mathcal{U}}(y)$), but these strings are not necessarily equal (i.e., different r's may yield different strings in $R_{\mathcal{U}}(y)$). Implicit (rather than explicit) representation is required because we want the extractor to run in polynomial-time, whereas the length of the strings in $R_{\mathcal{U}}(y)$ may be not bounded by any polynomial (in $|y|$). Finally, we note that the weak proof-of-knowledge property is indeed weaker than the standard definition of a proof of knowledge (cf. [7], [18, Sec. 4.7] and Footnote 8), but it suffices for the applications we have in mind.

3 The Construction of Universal Arguments

As mentioned in Subsection 1.3, by adapting of the construction of Kilian [21], one can easily show that the existence of *strong* collision-free hashing functions implies the existence of universal arguments (and even CS-proofs for $L_{\mathcal{U}}$; cf. Micali [23]). Here we show how a similar adaptation, when using only *standard* collision-free hashing functions, yields universal arguments. Our focus is on demonstrating the computational soundness of this construction, which should now be established under a weaker assumption than the one used in [21, 23].

3.1 Motivation

In order to explain the difficulty and its resolution, let us recall the basic construction of Kilian [21] (used also by Micali [23]), as adapted to our setting.

Our starting point is a $\mathcal{PCP}$[poly, poly] system for $L_{\mathcal{U}} \in \mathcal{NEXP}$, which is used in the universal-argument system as follows. The verifier starts by sending the prover a hashing function. The prover constructs a PCP-proof/oracle (corresponding to the common input and its own auxiliary input), places the bits of this oracle at the leaves of a polynomial-depth full binary tree, and places in each internal node the hash-value obtained by applying the hashing function to the labels of its children. The prover sends the label of the root to the verifier, which responses by sending a random tape of the type used by the PCP-verifier. Both parties determine the queries corresponding to this tape, and the prover responds with the values of the corresponding leaves along with the labels of the vertices along the paths from these leaves to the root (as well as the labels of the siblings of these vertices). The verifier checks that this sequence of labels matches the corresponding applications of the hashing function, and also emulates the PCP-verifier. Ignoring (for now) the issue of prover's complexity, the problem is in establishing computational-soundness.

The naive approach is to consider what the prover does on each of the possible random-tapes sent to it. In case it answers consistently (i.e., with leaves labels that depend only on the leave location), we obtain a pcp-oracle and soundness follows by the soundness of the PCP scheme. On the other hand, inconsistent labels for the same leaf yield a (hashing) collision somewhere along the path to the root. However, in order to find such a collision, we must spend time proportional to the size of the tree, which yields contradiction only in case the hashing function is supposed to withstand adversaries using that much time. In case the tree is exponential (or even merely super-polynomial) in the security parameter, we derive contradiction only when using hashing functions of subexponential (respectively, super-polynomial) security.

The approach taken here is to consider each leaf separately rather than all leaves together. That is, the naive analysis distinguishes the case that the prover answers inconsistently on *some* leaf from the case it answer consistently on *all* leaves. Instead, we consider each leave separately, and distinguishes the case that the prover answers inconsistently on *this leaf* from the case it answer consistently on *this leaf*. Loosely speaking, we call a leaf good if the prover answers consistently on it, and observe that if a big fraction of the leaves are good then soundness follows by the soundness of the PCP scheme. In case sufficiently many leaves are not good, we obtain a collision by picking a random leave (hoping that it is not good) and obtaining inconsistent labels for it. This requires being able to uniformly select a random-tape that makes the pcp-verifier make the corresponding query, a property which is fortunately enjoyed by the relevant PCP systems.

We warn that the above is merely a rough description of the main idea in our analysis. Furthermore, in order to establish the proof-of-knowledge property of our construction, we need to rely on an analogous property of the PCP system (which again happens to be satisfied by the relevant PCP systems).

3.2 The PCP system in use

We first recall the basic definition of a PCP system. Loosely speaking, a probabilistically checkable proof (PCP) system consists of a probabilistic polynomial-time verifier having access to an oracle which represents a proof in redundant form. Typically, the verifier accesses only few of the oracle bits, and these bit positions are determined by the outcome of the verifier's coin tosses. It is required that if the assertion holds then the verifier always accepts (i.e., when given access to an adequate oracle); whereas, if the assertion is false then the verifier must reject with high probability (as specified in an adequate bound), no matter which oracle is used. The basic definition of the PCP setting is given in Item (1) below. Typically, the complexity measures introduced in Item (2) are of key importance, but not so in this work.

Definition 3.1 (PCP – basic definition):

1. *A* probabilistic checkable proof system (pcp) with error bound $\epsilon : \mathsf{N} \rightarrow [0, 1]$ for a language L *is a probabilistic polynomial-time oracle machine* (called verifier)*, denoted V, satisfying*
   - Completeness*: For every $x \in L$ there exists an oracle π_x such that V, on input x and access to oracle π_x, always accepts x.*
   - Soundness*: For every $x \notin L$ and every oracle π, machine V, on input x and access to oracle π, rejects x with probability at least $1 - \epsilon(|x|)$.*
2. *Let r and q be integer functions. The complexity class $\mathcal{PCP}_\epsilon[r(\cdot), q(\cdot)]$ consists of languages having a pcp system with error bound ϵ in which the verifier, on any input of length n, makes at most $r(n)$ coin tosses and at most $q(n)$ oracle queries.*

Note that if L has a pcp system with error bound ϵ then $L \in \mathcal{PCP}_\epsilon[p(\cdot), p(\cdot)]$, for some polynomial p. Here we will only care that $L_{\mathcal{U}} \in \mathcal{NE}$ has a pcp system with and exponentially decreasing error bound (i.e., $\epsilon(n) = 2^{-n}$). Instead of caring about the refine complexity measures (of Item 2), we will care about the following *additional properties* satisfied by this specific pcp system, where only some of these properties were explicitly considered before (see discussion below).

Definition 3.2 (PCP – auxiliary properties): *Let V be a pcp verifier with error $\epsilon : \mathsf{N} \rightarrow [0, 1]$ for a language $L \in \mathcal{NEXP}$, and let R be a corresponding witness relation. That is, if $L \in$ Ntime$(t(\cdot))$, then we refer to a* polynomial-time decidable *relation R satisfying $x \in L$ if and only if there exists w of length at most $t(|x|)$ such that $(x, w) \in R$. We consider the following auxiliary properties:*

Relatively-efficient oracle-construction: *This property holds if there exists a polynomial-time algorithm P such that, given any $(x, w) \in R$, algorithm P outputs an oracle π_x that makes V always accept (i.e., as in the completeness condition).*

Non-adaptive verifier: *This property holds if the verifier's queries are determined based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is, V can be decomposed into a pair of algorithms, Q and D, such that on input x and random-tape r, the verifier makes the query sequence $Q(x,r,1), Q(x,r,2), ..., Q(x,r,p(|x|))$, obtains the answers $b_1, ..., b_{p(|x|)}$, and decides by according to $D(x,r,b_1 \cdots b_{p(|x|)})$.*

Efficient reverse-sampling: *This property holds if there exists a probabilistic polynomial-time algorithm S such that, given any string x and integers i and j, algorithm S outputs a uniformly distributed r that satisfies $Q(x,r,i) = j$, where Q is as above.*

A proof-of-knowledge property: *This property holds if there exists a probabilistic polynomial-time oracle machine E such that the following holds:[9] for every x and π, if $\Pr[V^\pi(x) = 1] > \epsilon(|x|)$ then there exists $w = w_1 \cdots w_t$ such that $(x,w) \in R$ and $\Pr[E^\pi(x,i) = w_i] > 2/3$ holds for every i.*

Non-adaptive pcp verifiers were explicitly considered in several works, all popular pcp systems use non-adaptive verifiers, and in fact in some sources PCP is defined in terms of non-adaptive verifiers. The oracle-construction and proof-of-knowledge properties are implicit in some works, and are known to hold for most popular pcp systems (although to the best of our knowledge a proof of this fact has never appeared). To the best of our knowledge, the reverse-sampling property was not considered before. Nevertheless it can be verified that any $L \in \mathcal{NEXP}$ has a pcp system satisfying all the above properties.

Theorem 3.3 *For every $L \in \mathcal{NEXP}$ and for every $\epsilon : \mathsf{N} \to [0,1]$ such that $\epsilon(n) > 2^{-\mathrm{poly}(n)}$, there exist a pcp system with error ϵ for L that satisfies all properties in Definition 3.2.*

Proof sketch: For $L \in \mathrm{Ntime}(t(\cdot))$, we consider a $\mathcal{PCP}_{1/2}[O(\log t(\cdot)), \mathrm{poly}(\cdot)]$ system as in [5] (i.e., the starting point of [2, 1]). (We stress that this pcp system, unlike the one of [12], uses oracles of length polynomial in t.)[10] This pcp system is non-adaptive and is well-known to satisfies the oracle-construction property. It is also known (alas less well-known) that this pcp system satisfies the proof-of-knowledge property. Finally, it is easy to see that this pcp system (as any reasonable pcp system we know of) also satisfies the reverse-sampling property.[11] All these claims will be proven in the full version of this work. Error reduction is obtained without effecting the oracle, and so it is easy to see that the amplified pcp preserves all the auxiliary properties. ■

3.3 The actual construction

The construction is an adaptation of Kilian's construction [21] (used also by Micali [23]). Using Theorem 3.3, we start with a pcp system with error $\epsilon(n) = 2^{-n}$ for $L_{\mathcal{U}}$ that satisfies the auxiliary properties in Definition 3.2. Actually, the corresponding witness relation will not be $R_{\mathcal{U}}$ as defined in Section 1.2, but rather a minor modification of it, denoted $R'_{\mathcal{U}}$: the pair $((M,x,t),(w,1^{t'}))$ is in $R'_{\mathcal{U}}$ if M accepts (x,w) in $t' \leq t$ steps. (The purpose of the modification is to obtain a relation that is decidable in polynomial-time, as required in Definition 3.2.) Let V_{pcp} denote the

[9] For negligible ϵ (as used below) this proof-of-knowledge property is stronger than the standard proof-of-knowledge property (as in [7] and [18, Sec. 4.7]).

[10] Moving to non-binary encoding of objects seems important for achieving this.

[11] This property follows from the structure of the standard pcp systems. In our case, the system consists of a *sum-check* (a la Lund *et. al.* [22]), and a *low-degree test*. In both tests, the queries are selected in a very simple manner, and what is complex (at least in the case of low-degree tests) is the analysis of the test.

above pcp system (or its verifier), and $P_{\rm pcp}, Q_{\rm pcp}, D_{\rm pcp}, S_{\rm pcp}, E_{\rm pcp}$ denote the auxiliary algorithms (or machines) guaranteed by Definition 3.2 (e.g., $P_{\rm pcp}$ is the oracle-constructing procedure, $Q_{\rm pcp}$ determines the verifier's queries, and $S_{\rm pcp}$ provides reverse-sampling).

A second ingredient used in the construction is a family of collision-free hashing functions. That is, a collection of (uniformly polynomial-time computable) functions $\{h_\alpha : \{0,1\}^* \to \{0,1\}^{|\alpha|}\}$ such that for every (non-uniform) family of polynomial-size circuits $\{C_n\}_{n\in\mathbb{N}}$

$$\Pr_{\alpha\in\{0,1\}^n}[C_n(\alpha) = (x,y) \text{ s.t. } x \neq y \text{ and } h_\alpha(x) = h_\alpha(y)] \;=\; \mu(n)$$

where μ is a negligible function.

Construction 3.4 (a universal argument for $L_{\mathcal{U}}$):

Common input: $y = (M,x,t)$ *supposedly in* $L_{\mathcal{U}}$. *Let* $n \stackrel{\rm def}{=} |y|$.

Auxiliary input to the prover: w *such that supposedly* $(y,w) \in R_{\mathcal{U}}$ *holds.*

First verifier step (V1): *Uniformly select* $\alpha \in \{0,1\}^n$*, and send it to the prover.*

First prover step (P1): *When describing the prover's actions, we assume that* $(y,w) \in R_{\mathcal{U}}$.

1. Preliminary action by the prover: *The prover invokes* M *on input* (x,w)*, and obtains* $t' = t_M(x,w)$. *Assuming that* $(y,w) \in R_{\mathcal{U}}$ *and letting* $w' = (w, 1^{t'})$*, the prover obtains an* $R'_{\mathcal{U}}$*-witness; that is,* $(y,w') \in R'_{\mathcal{U}}$.
2. Oracle-construction: *Invoking* $P_{\rm pcp}$ *on* (y,w')*, the prover obtains* $\pi_y = P_{\rm pcp}(y,w')$.
3. Construction of a hashing tree: *Letting* $d \stackrel{\rm def}{=} \lceil \log_2 |\pi_y| \rceil$*, the prover constructs a binary tree of depth* d *such that its nodes are associated with binary strings of length at most* d*, and each node is* labeled *as follows. The root is associated with the empty string, and an internal node associated with* γ *has children associated with* $\gamma 0$ *and* $\gamma 1$*. The label of a leaf associated with* $\gamma \in \{0,1\}^d$*, denoted* ℓ_γ*, is the value of* π_y *at position* γ*; that is this oracle answer to the query* γ*. The label of an internal node associated with* $\gamma \in \cup_{i=0}^{d-1}\{0,1\}^i$ *is the value obtained by applying* h_α *to the string* $\ell_{\gamma 0}\ell_{\gamma 1}$.
4. The actual message sent *is the depth of the tree and the label of its root. That is, the prover sends the pair* (d, ℓ_λ) *to the verifier.*

Second verifier step (V2): *The verifier uniformly selects a random-tape* r *for the pcp system, and sends* r *to the prover.*

Second prover step (P2): *The prover provides the corresponding* (pcp) *answers, augmented by* proofs of consistency *of these answers with the label of the root as provided in Step* (P1).

1. Determining the queries: *Invoking* $Q_{\rm pcp}$*, the prover determines the sequence of queries that the pcp system makes on random-tape* r*. That is, for* $i = 1, ..., m$*, it computes* $q_i = Q_{\rm pcp}(y,r,i)$*, where* $m \stackrel{\rm def}{=} {\rm poly}(n)$ *is the number of queries made by the system.*
2. The message sent: *for* $i = 1, ..., m$ *and* $j = 0, ..., d-1$*, the prover sends the pair* $(\ell_{\gamma 0}, \ell_{\gamma 1})$*, where* γ *is the* j*-bit long prefix of* q_i.

Verifier final decision – Step (V3): *The verifier checks that the answers provided by the prover would have been accepted by the pcp-verifier, and that the corresponding proofs of consistency* (*with the label of the root*) *are valid. That is, denoting by* ℓ'_γ *the label provided by the prover for the node associated with* γ*, the verifier accepts if and only if all the following checks pass:*

1. *Invoking $D_{\rm pcp}$, the verifier checks whether, on input y and random-tape r, the pcp-verifier would have accepted the answer sequence $\ell'_{q_1},...,\ell'_{q_m}$. That is, check whether $D_{\rm pcp}(y,r,\ell'_{q_1}\cdots\ell'_{q_m})=1$.*
2. *Check whether the labels provided are consistent with the label of the tree as sent in Step P1. That is, for $i=1,...,m$ and $j=0,...,d-1$, check whether $\ell'_\gamma = h_\alpha(\ell'_{\gamma 0}\ell'_{\gamma 1})$.*

We denote the above verifier and prover strategies by V and P, respectively.

Clearly, Construction 3.4 satisfies the first two requirements of Definition 2.1; that is, the verifier's strategy is implementable in probabilistic polynomial-time, and completeness holds with respect to a prover strategy that (given $y=(M,x,t)$ and w as above) runs in time polynomial in $T_M(x,w)$. We thus focus on establishing the two last requirements of Definition 2.1. In fact, computational soundness follows from the weak proof-of-knowledge property (because whenever some adversary can convince the verifier to accept with non-negligible probability the extractor outputs a valid witness for membership in $L_{\mathcal U}$). Thus, it suffices to establish the latter.

3.4 Establishing the weak proof-of-knowledge property

This subsection contains the main technical contribution of the current section. The novel aspect in the analysis is the "local definition of a conflict" (i.e., a conflicting oracle-bit rather than a conflicting pair of oracles), and the fact that reverse-sampling can be used to derive (in polynomial-time) hashing-collisions (given a conflicting oracle bit-position).

Lemma 3.5 *Construction 3.4 satisfies the weak proof-of-knowledge property of Definition 2.1, provided that the family $\{h_\alpha\}$ is indeed collision-free.*

Combining Lemma 3.5 with the above discussion, we derive Theorem 1.1.

Proof: Fixing any polynomial p, we present a probabilistic polynomial-time knowledge-extractor that extracts witnesses from any feasible prover strategy that makes V accept with probability above the threshold specified by p. Specifically, for any family of (deterministic) polynomial-size circuits representing a possible cheating prover strategy and for all sufficiently long y's, if the prover convinces V to accept y with probability at least $1/p(|y|)$ then, with noticeable probability (i.e., $1/p'(|y|)$), the knowledge-extractor (given oracle access to the strategy) outputs the bits of a corresponding witness.

We fix an arbitrary family, $\{\widetilde{P}_n\}_{n\in\mathbb{N}}$, of (deterministic) polynomial-size circuits representing a possible cheating prover strategy, and a generic n and $y\in\{0,1\}^n$ such that $\Pr[(\widetilde{P}_n,V)(y)=1] > \varepsilon \stackrel{\rm def}{=} 1/p(n)$. We consider a few key notions regarding the interaction of $\widetilde{P}_n$ and the designated verifier V on common input y. First we consider notions that refer to a specific interaction (corresponding to a fixed sequence of verifier coins):

- The $i^{\rm th}$ query in such interaction is $q_i = Q_{\rm pcp}(y,r,i)$, where r is the Step (V2) message.
- The $i^{\rm th}$ answer supplied by (the prover) $\widetilde{P}_n$ is the label (ℓ'_{q_i}) it has provided (in Step (P2)) for the leaf (associated with) the $i^{\rm th}$ query (i.e., $q_i = Q_{\rm pcp}(y,r,i)$). The corresponding authentication is the corresponding sequence of pairs $(\ell'_{\gamma 0},\ell'_{\gamma 1})$, where γ is the j-bit long prefix of q_i.
- The $i^{\rm th}$ answer supplied by $\widetilde{P}_n$ is said to be proper if the corresponding authentication passes the verifier's test (in Step (V3)). That is, $\ell'_\gamma = h_\alpha(\ell'_{\gamma 0}\ell'_{\gamma 1})$ holds, for $j=0,...,d-1$.

Next, we consider the probability distribution induced by the verifier's coins. Note that these coins consist of the pair of choices (α, r) that the verifier took in Steps (V1) and (V2), respectively. Fixing any $\alpha \in \{0,1\}^n$, we consider the conditional probability, denoted $p_{y,\alpha}$, that the verifier accepts when choosing α is Step (V1). Clearly, for at least a $\varepsilon/2$ fraction of the possible α's it holds that $p_{y,\alpha} \geq \varepsilon/2$. We fix any such α for the rest of the discussion. We now consider notions that refer to the probability space induced by a uniformly chosen $r \in \{0,1\}^{\mathrm{poly}(n)}$ (selected by the verifier in Step (V2)).

- For any query $q \in \{0,1\}^d$, a query index $i \in \{1,...,m\}$, possible answer $\sigma \in \{0,1\}$, and $\delta \in [0,1]$, we say that σ is δ-strong for (i,q) if, conditioned on the i^{th} query being q, the probability that $\widetilde{P}_n$ *properly answers* the i^{th} with σ is at least δ. That is,

$$\Pr_r[\ell'_{q_i} = \sigma \text{ is proper} \mid q_i = Q(y,r,i)] \;\geq\; \delta$$

  When i and q are understood from the context, we just say that σ is a δ-strong answer.

- We say that a query $q \in \{0,1\}^d$ has δ-conflicting answers if there exist i and j (possibly $i = j$) such that 0 is δ-strong for (i,q) and 1 is δ-strong for (j,q).

We stress that throughout the rest of the analysis we consider a fixed $\alpha \in \{0,1\}^n$ and a uniformly distributed $r \in \{0,1\}^{\mathrm{poly}(n)}$.

Claim 3.5.1 *The probability that the verifier accepts while receiving only δ-strong answers is at least $p_{y,\alpha} - m\delta$.*[12]

Thus, picking $\delta = p_{y,\alpha}/2m$, we may confine ourselves to the case that all the prover's answers are δ-strong.

Proof: The key observation is that whenever the verifier accepts, all answers are proper. Intuitively, answers that are not δ-strong (i.e., are rarely proper) are unlikely to appear in such interactions. Specifically, we just upper bound the probability that, for some $i \in \{1,...,m\}$, the answer ℓ'_{q_i} is proper but not δ-strong for (i, q_i), where $q_i = Q(y,r,i)$. Fixing any i and any possible value of $q_i = Q(y,r,i)$, by definition (of being proper but not δ-strong), the corresponding event occurs with probability less than δ. Averaging over the possible values of $q_i = Q(y,r,i)$ we are done. $\square$

Claim 3.5.2 *There exist a probabilistic polynomial-time oracle machine that, given α and oracle access to $\widetilde{P}_n$, finds $z' \neq z''$ such that $h_\alpha(z') = h_\alpha(z'')$ with probability that is polynomially related to δ/n and to the probability that the verifier makes a query that has δ-conflicting answers. Specifically, letting η denote the latter probability, the probability of finding a collision is at least $\eta\delta^2/m^3$.*

Thus, on a typical α, the probability η must be negligible (because otherwise we derive a contradiction to the collision-free hypothesis of the family $\{h_\alpha\}$). Consequently, for $\delta = p_{y,\alpha}/2m > 1/\mathrm{poly}(n)$, we may confine ourselves to the case that the prover's answers are not $(\delta/2)$-conflicting.

Proof: We uniformly select $r \in \{0,1\}^{\mathrm{poly}(n)}$ and $i \in \{1,...,m\}$, hoping that $q_i = Q(y,r,i)$ is δ-conflicting (which is the case with probability at least η/m). Uniformly selecting $i', i'' \in \{1,...,m\}$, and invoking the reverse-sampling algorithm S_{pcp} on inputs (y, i', q_i) and (y, i'', q_i), respectively, we obtain uniformly distributed r' and r'' that satisfy $q_i = Q(y, r', i')$ and $q_i = Q(y, r'', i'')$. We now invoke $\widetilde{P}_n$ twice, feeding it with α and r' (resp. α and r'') in the first (resp., second) invocation.

[12] Recall that m is the number of queries asked by the PCP verifier V_{pcp}.

With probability at least $(\delta/m)^2$ both answers to q_i will be proper but with opposite values. From the authentication information corresponding to these two (proper) answers, we obtain a collision under h_α (because each of the different values for the same leave is authenticated with respect to the same value of the root). $\Box$

Suppose for a moment, that (for $\delta = p_{y,\alpha}/2m$) all the prover's answers are δ-strong but not $(\delta/2)$-conflicting. Then, we can use the prover's answers in order to construct (and not merely claim the existence of) an oracle for the pcp system that makes it accept with probability at least $p_{y,\alpha}/2$. Specifically, let the q^{th} bit of the oracle be σ if and only if there exists an i such that σ is δ-strong for (i,q). The setting of the oracle bits can be decided in probabilistic polynomial-time by using the reverse-sampling algorithm S_{pcp} to generate multiple samples of interactions in which these specific oracle bits are queried. That is, to determine the q^{th} bit, we try $i = 1, ..., m$, and for each value of i generate multiple samples of interactions in which the i^{th} oracle query equals q. We will use the gap provided by the hypothesis that for some i there is an answer that is δ-strong for (i,q), whereas (by the non-conflicting hypothesis) for every j the opposite answer is not $(\delta/2)$-strong for (j,q).

In general, in contrary to the simplifying assumption above, some queries may either have no strong answers or be conflicting. The procedure may indeed fail to recover the corresponding entries in the pcp-oracle, but this will not matter because with sufficient high probability the pcp verifier will not query these badly-recovered locations.

The oracle-recovery procedure: We present a probabilistic polynomial-time oracle machine that on input (y,α) and $q \in \{0,1\}^d$ and oracle access to the prover $\widetilde{P}_n$, outputs a candidate for the q^{th} bit of a pcp-oracle. The procedure operates as follows, where $T \stackrel{\text{def}}{=} \text{poly}(n/\delta)$ and $\delta = \varepsilon/4m$:

1. For $i = 1, ..., m$ and $j = 1, ..., T$, invoke S_{pcp} on input (y, i, q) and obtain $r_{i,j}$.
2. For $i = 1, ..., m$ and $j = 1, ..., T$, invoke $\widetilde{P}_n$ feeding it with α and $r_{i,j}$, and if the i^{th} answer is proper then *record* (i,j) *as supporting this answer value.*
3. If for some $i \in \{1, ..., m\}$, there are $(2\delta/3) \cdot T$ records for the form $(i, \cdot)$ for value $\sigma \in \{0,1\}$ then define σ as a *candidate*. That is, σ is a candidate if there exists an i and at least $(2\delta/3) \cdot T$ different j's such that the i^{th} answer of $\widetilde{P}_n(\alpha, r_{i,j})$ is proper and has value σ.
4. If a single value of $\sigma \in \{0,1\}$ is defined as a candidate then set the q^{th} bit accordingly. (Otherwise, do whatever you please.)

We call the query q good if it does not have $(\delta/2)$-conflicting answers and there exists an $i \in \{1, ..., m\}$ and a bit value that is δ-strong for (i,q). For a good query, with overwhelmingly high probability, the above procedure will define the latter value as a unique candidate. (The expected number of $(i,\cdot)$-supports for the strong value is at least $\delta \cdot T$, whereas for the opposite value the expected number of $(i',\cdot)$-supports is less than $(\delta/2) \cdot T$, for every i'.) Let use denote the pcp-oracle induced by the above procedure by π.

Claim 3.5.3 *Let $\delta = \varepsilon/4m$ and recall that $p_{y,\alpha} \geq \varepsilon/2$. Suppose that the probability that V makes a query that has $(\delta/2)$-conflicting answers is at most $p_{y,\alpha}/4$. Then, with probability at least $1 - 2^{-n}$ taken over the reconstruction of π, the probability that ${V_{\text{pcp}}}^{\pi}(y)$ accepts is lower bounded by $p_{y,\alpha}/4$.*

Proof: Combining the hypothesis (regarding $(\delta/2)$-conflicting answers) with Claim 3.5.1, we conclude that with probability at least $(p_{y,\alpha} - m\delta) - (p_{y,\alpha}/4) \geq p_{y,\alpha}/4$ the verifier (of the interactive argument) accepts while making only good queries and receiving only δ-strong answers. However, in this case, with probability at least $1-2^{-n}$, the answers of $\widetilde{P}_n$ equal the corresponding bits in π. (This is because for a good query, with overwhelmingly high probability, the answer that is δ-strong will be the only candidate determined by the procedure.) Since the (interactive argument) verifier is accepting after invoking $V_{\rm pcp}$ on the answers it obtained, it follows that in this case $V_{\rm pcp}$ accepts too. $\square$

The weak proof-of-knowledge property (of the interactive argument) now follows from the corresponding property of the pcp system. Specifically, we combine the pcp extractor with the above oracle-recovery procedure, and obtain the desired extractor.

Extractor for the argument system: On input (y,i) and access to a prover strategy $\widetilde{P}_n$, the extractor operates as follows (using $\delta=\varepsilon/4m$):

1. Uniformly select $\alpha\in\{0,1\}^n$, hoping that α is typical (with respect to collision-freeness) and that $p_{y,\alpha}>\varepsilon/2$ (which hold with probability at least $(\varepsilon/2)-\mu(n)>\varepsilon/3$).

   By saying that α is typical with respect to collision-freeness we mean that in the corresponding conditional probability space (of α being chosen in Step (V1)), the probability that the verifier makes a query that has $(\delta/2)$-conflicting answers is less than $p_{y,\alpha}/4$, which in turn is at least $\varepsilon/8$. By Claim 3.5.2, there exists a probabilistic polynomial-time machine that, given any untypical α (and access to $\widetilde{P}_n$), finds a collision (under h_α) with probability at least $(p_{y,\alpha}/4)\cdot(\delta/2)^2/m^3=\Omega(\varepsilon^3/m^5)=1/{\rm poly}(n)$, because $\varepsilon=1/p(n)$. It follows that the fraction of untypical α's must be negligible.

2. Uniformly select coins ω for the oracle-recovery procedure, and fix ω for the rest of the discussion.

   Note that the oracle-recovery procedure (implicitly) provides oracle access to a pcp-oracle π, where (by Claim 3.5.3) with probability at least $1-2^{-n}$ (over the choice of ω), this π convinces $V_{\rm pcp}$ with probability at least $p_{y,\alpha}/4>\varepsilon/8>2^{-n}$.

3. Invoke $E_{\rm pcp}(y,i)$ providing it with oracle access to π. This means that each time $E_{\rm pcp}(y,i)$ makes a query q, we invoke the oracle-recovery procedure on input (y,q) (and with α and ω as fixed above), and obtain the $q^{\rm th}$ bit of π, which we return as answer to $E_{\rm pcp}(y,i)$. When $E_{\rm pcp}(y,i)$ provides an answer (supposedly the $i^{\rm th}$ bit of a suitable witness w), we just output this answer.

Let us call α useful if it is typical w.r.t collisions and $p_{y,\alpha}>\varepsilon/2$. We say that ω is α-useful if when used as coins for the oracle-recovery procedure (which gets main-input (y,α)) yields an oracle that convinces $V_{\rm pcp}$ with probability at least 2^{-n}. Recall that at least a $\varepsilon/3$ fraction of the α's are useful, and that for each useful α at least a $1-2^{-n}$ fraction of the ω's are α-useful. By the proof-of-knowledge property of the pcp system, if α is useful and ω is α-useful then with probability at least $2/3$ (over the coins of $E_{\rm pcp}$), the output of $E_{\rm pcp}$ (and thus of our extractor) will be correct. By suitable amplification, we can obtain the correct answer with probability at least $1-2^{-2n}$ (over the coins of $E_{\rm pcp}$), pending again on α and ω being useful. Denoting the coins of the amplified $E_{\rm pcp}$ by ρ, we conclude that for at least a fraction $1-t\cdot 2^{-2n}\geq 1-2^{-n}$ of the possible ρ's, the amplified $E_{\rm pcp}$ provides correct answers for all t bit locations. We call such ρ's (α,ω)-useful.

Let us denote the above extractor by E. The running-time of E is dominated by the running-time of the oracle-recovery procedure, whereas the latter is dominated by the ${\rm poly}(n/\varepsilon)$ invocations of $\widetilde{P}_n$ (during the oracle-recovery procedure). Using $\varepsilon=1/p(n)$, it follows that E runs in polynomial-time. The random choices of E correspond to the above three steps; that is, they consist of α, ω and ρ. Whenever they are all useful (i.e., α is useful, ω is α-useful, and ρ is (α,ω)-useful), the extractor E recovers correctly all bits of a suitable witness (for y). The event in the condition occurs with probability at least $(\varepsilon/3)\cdot(1-2^{-n})\cdot(1-2^{-n}) > \varepsilon/4 = 1/4p(n)$. Letting $p'(n) = 4p(n)$, the lemma follows. ■

4 Application to Zero-Knowledge Arguments

Using Theorem 1.1, we prove Theorem 1.2 in two steps:

1. Using any constant-round public-coin universal-argument, we derive one that is (strongly) witness-indistinguishable. Here we just follow Barak's construction [6] (which in turn follows [8]), but the analysis is again more complex than in the original work.
2. Using the latter, we modify Barak's construction of a zero-knowledge argument (for $L \in \mathcal{NP}$) [6]. Specifically, rather than using any collision-free hashing (for the very first message in his protocol), we use tree-hashing (as in Step (P1) of Construction 3.4) composed with an error-correcting code.

We stress that when applied to universal-arguments, the (strong) witness-indistinguishability property refers only to witness ensembles that are verifiable in time that is polynomial in the length of the common input (where the polynomial is fixed after the universal-argument system is specified).

4.1 Constructing witness-indistinguishable universal-arguments

Our starting point is any constant-round, public-coin universal-argument (for $L_{\mathcal{U}}$), denoted $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$. For sake of simplicity, we assume (without loss of generality) that, on any n-bit long common input, each message sent by either parties has length $m = \mathrm{poly}(n)$. Using the public-coin clause this means that the protocol proceeds in rounds, where in each round the verifier selects uniformly an m-bit string, and the prover responds with an m-bit string determined based on its inputs and the messages it has received so far. We denote by c the (constant) number of such rounds; in case of Construction 3.4, $c = 2$.

A second ingredient used in the construction is a (constant-round, public-coin) *strong witness-indistinguishable* (*strong WI*) proof-of-knowledge for an NP-complete language. Loosely speaking, the strong notion of WI means that whenever the common inputs to the proof system are computational indistinguishable so are the corresponding views of the verifier (no matter which NP-witnesses the prover uses). Let us denote such a system by $(P_{\mathrm{wi}}, V_{\mathrm{wi}})$. (Such proof systems exist (cf. [18, Sec. 4.6 & 4.7]), provided that (perfectly-binding) commitment scheme exist.) Finally, we use a (perfectly-binding) commitment scheme, denoted $\mathtt{C}$. The commitment to value v using randomness r is denoted $\mathtt{C}_r(v)$.

Construction 4.1 (a witness-indistinguishable universal-argument):

Common input: $y = (M, x, t)$ *supposedly in* $L_{\mathcal{U}}$. *Let* $n \stackrel{\mathrm{def}}{=} |y|$.

Auxiliary input to the prover: w *such that supposedly* $(y, w) \in R_{\mathcal{U}}$ *holds.*

Part 1: encrypted emulation of $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$. *The parties emulate the* $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$ *protocol in a partially encrypted manner. Specifically, the verifier generates random messages exactly as* V_{ua}, *but the prover answers with commitments to the corresponding responses of* P_{ua}. *That is, for* $i = 1, ..., c$:

1. *The verifier uniformly selects* $r_i \in \{0,1\}^m$, *and sends it to the prover.*

2. *The prover determines P_{ua}'s answer, and responds with a commitment to it. That is, the prover selects uniformly coins s for P_{ua}, and fixes them for all iterations. The i^{th} answer of P_{ua} is determined by $a_i \leftarrow P_{\mathrm{ua}}(y, w; s, r_1, ..., r_i)$. Finally, the prover selects commitment coins s_i, and sends $e_i \leftarrow \mathtt{C}_{s_i}(a_i)$ to the verifier.*

Part 2: proving that V_{ua} accepts in the encrypted emulation. *The parties invoke the proof system $(P_{\mathrm{wi}}, V_{\mathrm{wi}})$ to prove that the transcript $(r_1, e_1, ..., r_c, e_c)$ generated above corresponds to an encryption of an accepting $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$ transcript, and that the corresponding cleartext is known to the prover. The prover executes the protocol using the NP-witness $((a_1, s_1), ..., (a_c, s_c))$, and the NP-statement being proven (with respect to the input $(y, r_1, e_1, ..., r_c, e_c)$) is*

1. *For $i = 1, ..., c$, it holds that $e_i = \mathtt{C}_{s_i}(a_i)$.*
2. *$V_{\mathrm{ua}}(y; r_1, a_1, ..., r_c, a_c) = 1$.*

Note that the length of the NP-statement being proven (as well as the length of the corresponding NP-witness) is bounded by a fixed polynomial in $n + m$ (and thus by a fixed polynomial in n).

We denote the above verifier and prover strategies by V and P, respectively.

Clearly, Construction 4.1 is constant-round, public-coin, and satisfies the first two requirements of Definition 2.1; that is, the verifier's strategy is implementable in probabilistic polynomial-time, and completeness holds with respect to a prover strategy that (given $y = (M, x, t)$ and w as above) runs in time polynomial in $T_M(x, w)$. It remains to establish two properties of Construction 4.1:

1. The weak proof-of-knowledge property, which in turn implies also the computational soundness property.
2. The witness-indistinguishability property.

The above properties are established by an adaptation of the proof in [6].

Lemma 4.2 *Construction 4.1 satisfies the weak proof-of-knowledge property of Definition 2.1, provided that so does $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$ and that $\mathtt{C}$ is perfectly-binding.*

Proof: We fix an arbitrary family, $\{\widetilde{P}_n\}_{n \in \mathbb{N}}$, of (deterministic) polynomial-size circuits representing a possible prover strategy in the system (P, V). We fix a generic n and $y \in \{0,1\}^n$ we let $p_y \stackrel{\mathrm{def}}{=} \Pr[(\widetilde{P}_n, V)(y) = 1]$.

Using $\widetilde{P}_n$, we construct a corresponding prover strategy $\widetilde{P_{\mathrm{ua}}}$ that makes V_{ua} accept y with probability at least $\mathrm{poly}(p_y)$. The strategy $\widetilde{P_{\mathrm{ua}}}$ operates in c iterations, where in each iteration it obtains from $\widetilde{P}_n$ an encrypted message and using the knowledge extractor of the system $(P_{\mathrm{wi}}, V_{\mathrm{wi}})$ extracts from it the corresponding cleartext message. That is, for $i = 1, ..., c$:

1. $\widetilde{P_{\mathrm{ua}}}$ obtains from the (real) verifier V_{ua} a (uniformly distributed) string $r_i \in \{0,1\}^m$.
2. $\widetilde{P_{\mathrm{ua}}}$ emulates a continuation of an execution of Part 1 of the (P, V) protocol (i.e., the encrypted $(P_{\mathrm{ua}}, V_{\mathrm{ua}})$ argument). The emulation is internal to $\widetilde{P_{\mathrm{ua}}}$ and proceeds by playing the part of the verifier towards an internal copy of $\widetilde{P}_n$. That is, $\widetilde{P_{\mathrm{ua}}}$ feeds $\widetilde{P}_n$ with verifier messages $r_1, ..., r_i, p_{i+1}, ..., p_c$, where the r_j's are as obtained from the real verifier and the p_j's are selected uniformly by $\widetilde{P_{\mathrm{ua}}}$ itself. In response, $\widetilde{P_{\mathrm{ua}}}$ obtains the corresponding answers $e_1, ..., e_c$.

3. Invoking the knowledge-extractor corresponding to the system $(P_{\rm wi}, V_{\rm wi})$ (while providing it with oracle access to the residual strategy $\widetilde{P}_n(r_1, ..., r_i, p_{i+1}, ..., p_c)$), strategy $\widetilde{P_{\rm ua}}$ tries to obtain an NP-witness $((a_1, s_1), ..., (a_c, s_c))$ corresponding to the NP-statement $(y, r_1, e_1, ..., r_i, e_i, p_{i+1}, e_{i+1}, ..., p_c, e_c)$. In particular, for $i^{\rm th}$ pair in the witness (i.e., (a_i, s_i)) it holds that $e_i = \mathsf{C}_{s_i}(a_i)$.

   The knowledge-extractor used here runs in strict polynomial-time, but recovers the witness (only) with probability that is polynomially-related to the probability that the prover convinces $V_{\rm wi}$. An suitable proof system having such an knowledge-extractor can be obtained by parallel repetitions of the basic zero-knowledge proof for Hamiltonicity (cf. [18, Chap. 4, Exec. 28]).[13]

4. $\widetilde{P_{\rm ua}}$ sends a_i to the (real) verifier $V_{\rm ua}$.

Clearly, $\widetilde{P_{\rm ua}}$ is probabilistic polynomial-time (since it merely invokes $\widetilde{P}_n$ and the knowledge-extractor, which in turn invokes $\widetilde{P}_n$). We need to relate its success probability to p_y.

Claim 4.2.1 *$\widetilde{P_{\rm ua}}$ makes $V_{\rm ua}$ accept y with probability at least* $\text{poly}(p_y)$*, where the polynomial depends on the constant c.*

Proof: The probability p_y that $\widetilde{P}_n$ makes V accept y is taken over the probability space that consists of V's random choices in the two parts of the protocol. Specifically, V's random choices are $((r_1, ..., r_c), R)$, where the r_i's are the c messages sent in the c rounds of Part 1.

For $i = 1, ..., c$, we call $(r_1, ..., r_i) \in \{0,1\}^{i \cdot m}$ good if the probability that $\widetilde{P}_n$ makes V accept y conditioned on V selecting $r_1, ..., r_i$ in the first i rounds is at least $p_y/2^i$. Using induction on i, note that at least a $p_y/2^i$ fraction of the $(r_1, ..., r_i) \in \{0,1\}^{i \cdot m}$ are good. Thus, the probability that verifier $V_{\rm ua}$ has selected $(r_1, ..., r_c)$ such that, for every $i \in \{1, ..., c\}$, the i-prefix $(r_1, ..., r_i)$ is good is at least $\prod_{i=1}^{c}(p_y/2^i) > (p_y/2^c)^c$. Suppose that this lucky event has indeed happened. Then, for every $i \in \{1, ..., c\}$, with probability at least $q' \stackrel{\rm def}{=} p_y/2^{i+1}$ over the choice of $(p_{i+1}, ..., p_c) \in \{0,1\}^{(c-i) \cdot m}$ the residual prover strategy $\widetilde{P}_n(r_1, ..., r_i, p_{i+1}, ..., p_c)$ makes $V_{\rm wi}$ accept with probability at least $q'' \stackrel{\rm def}{=} p_y/2^{i+1}$. Thus (for every i), we successfully extract the (unique) answer a_i with probability at least $q' \cdot \text{poly}(q'') > \text{poly}'(p_y/2^{i+1})$. Here we rely on the proof-of-knowledge property of the system $(P_{\rm wi}, V_{\rm wi})$ (when applied to the residual prover strategy $\widetilde{P}_n(r_1, ..., r_i, p_{i+1}, ..., p_c)$), and on the perfect-binding property of the commitment scheme C.

We conclude that $\widetilde{P_{\rm ua}}$ makes $V_{\rm ua}$ accept with probability at least

$$(p_y/2^c)^c \cdot \prod_{i=1}^{c} \text{poly}(p_y/2^{i+1}) \ > \ 2^{-c^2} \cdot \text{poly}(p_y)^c \qquad (1)$$

The claim follows. □

Using the weak proof-of-knowledge property of $(P_{\rm ua}, V_{\rm ua})$, with respect to $\widetilde{P_{\rm ua}}$ (as constructed and analyzed above), the lemma follows. ■

Lemma 4.3 *Construction 4.1 is strong witness-indistinguishable, provided that so is $(P_{\rm wi}, V_{\rm wi})$ and that C is computationally-hiding.*

[13] The extractor just performs two random interactions with the prover and recovers the Hamiltonian cycle if these interactions are different and both convince the verifier. To deal with the annoying case in which the space of possible interactions contains a single convincing interaction, one may augment the extractor with an attempt to guess a Hamiltonian cycle at random.

Proof: Suppose that $\{y'_n\}_{n\in\mathsf{N}}$ and $\{y''_n\}_{n\in\mathsf{N}}$ are computationally indistinguishable, and that $\{w'_n\}_{n\in\mathsf{N}}$ and $\{w''_n\}_{n\in\mathsf{N}}$ are corresponding sequences of witnesses such that $T_{M'_n}(x'_n, w'_n) = \mathrm{poly}(n)$, where $y'_n = (M'_n, x'_n, t'_n)$ (and similarly for $\{(y''_n, w''_n)\}_{n\in\mathsf{N}}$). Then, by the computationally-hiding property of the commitment scheme $\mathtt{C}$, the corresponding Part 1 transcripts are also computationally indistinguishable. That is, $\{(y'_n, r'_1, \mathtt{C}(a'_1), ..., r'_c, \mathtt{C}(a'_c)\}_{n\in\mathsf{N}}$ and $\{(y''_n, r''_1, \mathtt{C}(a''_1), ..., r''_c, \mathtt{C}(a''_c)\}_{n\in\mathsf{N}}$ are computationally indistinguishable, where $a'_i = P_{\mathrm{ua}}(y'_n, w'_n; s, r'_1, ..., r'_i)$ and $a''_i = P_{\mathrm{ua}}(y''_n, w''_n; s, r''_1, ..., r''_i)$. (We rely on the hypothesis that $T_{M'_n}(x'_n, w'_n) = \mathrm{poly}(n)$ and on the prover-efficiency with perfect completeness property of that P_{ua}.) Using the strong witness-indistinguishability property of $(P_{\mathrm{wi}}, V_{\mathrm{wi}})$, it follows that the corresponding Part 2 transcripts are also computationally indistinguishable. ■

4.2 Modifying Barak's zero-knowledge argument

Here our starting point is any (constant-round, public-coin) strong witness-indistinguishable universal-argument (for $L_{\mathcal{U}}$), denoted $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$.

A second ingredient used in the construction is a tree-hashing scheme, denoted $\mathtt{TH}$, as used in Construction 3.4. Loosely speaking, such a scheme can be applied to arbitrary long strings and allow verification of a particular bit in the string within time polynomial in the hash-value. We stress that verification does not require presenting the entire string to which hashing was applied (but rather only auxiliary authentication information specific to that bit position). We denote by $\mathtt{auth}_i$ the authentication information corresponding to bit position i. Recall that tree-hashing is constructed based on some "basic" hashing function (which maps $2n$-bit strings to n-bit strings), and that conflicting values assigned to any bit position in the tree-hashing (easily) yield a collision in the basic hashing.

In addition we use a perfectly-binding commitment scheme $\mathtt{C}$, and any good error-correcting code (i.e., a code correcting a constant fraction of errors with polynomial-time encoding and decoding algorithms), denoted $\mathtt{ECC}$.

The key idea in our modification of Barak's construction [6], is to replace an arbitrary hashing of strings by the following two-step (hashing) process:

1. Apply the error-correcting code to the input string.
2. Apply the tree-hashing to the resulting codeword.

The advantage of this two-step (hashing) process over standard hashing is that if two different strings are hashed to the same value then we can easily obtain a collision in the basic hashing function (underlying the tree-hashing). We stress that this collision is found in time that is polynomial in the hash-value (independent of the length of the strings being hashed). The reason is that, with (positive) constant probability, a uniformly selected bit-position in the codeword will have different values in the two codewords, and in this case we obtain from the corresponding authentications a collision in the basic hashing. (See analysis below.)

Construction 4.4 (a zero-knowledge argument for $L \in \mathcal{NP}$ (with a corresponding witness relation R_L)):

Common input: *x supposedly in L. Let $n \stackrel{\mathrm{def}}{=} |x|$.*

Auxiliary input to the prover: *w such that supposedly $(x, w) \in R_L$ holds.*

Part 1: introducing a trapdoor for the simulation.

1. *The verifier uniformly selects $\alpha \in \{0,1\}^n$* (i.e., a basic hash function)*, and sends it to the prover.*

2. *The prover sends a dummy commitment; that is, it uniformly selects* $s \in \{0,1\}^{\mathrm{poly}(n)}$*, and sends* $c \stackrel{\mathrm{def}}{=} \mathtt{C}_s(0^{2n})$ *to the verifier.*
3. *The verifier uniformly selects* $r \in \{0,1\}^n$*, and sends it to the prover.*

Part 2: proving that either $x \in L$ or the prover could have guessed r. *The parties invoke the proof system* $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$ *such that the prover proves that it knows* $(w, |\mathtt{ECC}(\Pi)|, \mathtt{auth}(\mathtt{ECC}(\Pi)), s)$ *such that either* $(x,w) \in R_L$ *or* $(|\mathtt{ECC}(\Pi)|, \mathtt{auth}(\mathtt{ECC}(\Pi)), s)$ *encodes authentication and decommitment information for a program* Π *such that* $\Pi(c) = r$.[14] *That is, the parties reduce the above instance* (x, α, c, r) *to triplet* $y = (M'_L, (x, \alpha, c, r), 2^n)$ (where $|y| = \mathrm{poly}(n)$ and y is supposedly in $L_{\mathcal{U}}$) *such that* $M'_L((x,\alpha,c,r),(w,m,\eta,(\mathtt{auth}_1,...,\mathtt{auth}_m),s)) \stackrel{\mathrm{def}}{=} 1$ *if and only if one of the following two conditions hold*

1. $(x,w) \in R_L$.
2. $\mathtt{C}_s(m, \mathtt{TH}_\alpha(\eta)) = c$*, the sequence* $(\mathtt{auth}_1, ..., \mathtt{auth}_m)$ *authenticates the corresponding bits of* $\eta \in \{0,1\}^m$*, and* $\Pi(c) = r$*, where* $\Pi \leftarrow \mathtt{ECC}^{-1}(\eta)$ *is a program for a universal Turing machine,* $\mathtt{TH}_\alpha(z)$ *is computed as in Step (P1) of Construction 3.4, and each* $\mathtt{auth}_i$ *encodes the* i^{th} *bit of* z *as well as the corresponding authentication information.*[15]

When invoking $P_{\text{wi-ua}}$*, the prover provides it with the witness* $(w, m_0, \eta_0, \mathtt{auth}_0, s_0)$*, where* $m_0 = n$ *and* $\eta_0 = \mathtt{auth}_0 = s_0 \stackrel{\mathrm{def}}{=} 0^n$ *are* (short) *dummy values.*

Note that the first condition can be evaluated in (fixed) *polynomial-time* (in $|x|$)*, whereas the complexity of evaluating the second condition is dominated by the running-time of* Π *on input* c*. Furthermore, if* $(x,w) \in R_L$ *then* $(y,(w,m_0,\eta_0,\mathtt{auth}_0,s_0)) \in R_{\mathcal{U}}$ *and the running-time of* $P_{\text{wi-ua}}$ *on* $(y,(w,m_0,\eta_0,\mathtt{auth}_0,s_0))$ *is a fixed polynomial in* $|x|$*. We stress that, in any case, the length of the statement being proven is bounded by a fixed polynomial in* $|x|$*.*

We denote the above verifier and prover strategies by V *and* P*, respectively.*

Clearly, Construction 4.4 is constant-round, public-coin, and employs a probabilistic polynomial-time verifier strategy. Furthermore, the designated prover satisfies the completeness property while running in polynomial-time, given x and w as above. Demonstrating that Construction 4.4 is zero-knowledge is done by following the ideas of [6]. We start with a rough sketch of this proof, and then turn to establish the computational-soundness property of Construction 4.4.

Construction 4.4 is zero-knowledge: We present a non-black-box simulator that, given the code of any feasible cheating verifier (represented by a polynomial-size circuit family $\{\widetilde{V}_n\}_{n\in\mathbb{N}}$), simulates the interaction of P with that verifier. Specifically, given $\widetilde{V}_n$, the simulator emulates Part 1 of the protocol, except that it sets $c \stackrel{\mathrm{def}}{=} \mathtt{C}_s(|\mathtt{ECC}(\widetilde{V}_n)|, \mathtt{TH}_\alpha(\mathtt{ECC}(\widetilde{V}_n)))$ (rather than $c \stackrel{\mathrm{def}}{=} \mathtt{C}_s(0^{2n})$). Next, the simulator emulates Part 2 of the protocol by using the witness $(w_0, |\mathtt{ECC}(\widetilde{V}_n)|, \mathtt{ECC}(\widetilde{V}_n), \mathtt{auth}(\mathtt{ECC}(\widetilde{V}_n)), s)$, where $w_0 = 0^n$ is a (short) dummy value, s was selected by the simulator when emulating Part 1, and $\widetilde{V}_n$ was given to it as (auxiliary) input. (Given $\widetilde{V}_n$, the simulator computes $\mathtt{ECC}(\widetilde{V}_n)$ as well as the corresponding sequence of authenticators $\mathtt{auth}(\mathtt{ECC}(\widetilde{V}_n))$, in polynomial-time.)

[14] The reason that we include (in the witness) the authentication information (i.e., $\mathtt{auth}(\mathtt{ECC}(\Pi))$) rather than Π itself will become clear in the proof of Lemma 4.5. Furthermore, for technical reasons, we explicitly include in the witness also the length of $\mathtt{ECC}(\Pi)$. For sake of clarify, we also explicitly include $\mathtt{ECC}(\Pi)$ in the witness (but we do not use it in the analysis).

[15] That is, $\mathtt{TH}_\alpha(z) = (d, \ell_\lambda)$, where $|z| = 2^d$ (for an integer d), ℓ_i is the i^{th} bit of z, and $\ell_\gamma = h_\alpha(\ell_{\gamma 0}\ell_{\gamma 1})$. Actually, here it is more natural to let $\mathtt{TH}_\alpha(z) = (d, \ell_{0,0})$, where $\ell_{d,i}$ is the i^{th} bit of z, and $\ell_{j,i} = h_\alpha(\ell_{j+1,2i}\ell_{j+1,2i+1})$. Similarly, $\mathtt{auth}_i = (\ell_{d,2\lfloor i/2\rfloor}, \ell_{d,2\lfloor i/2\rfloor+1}, \ell_{d-1,2\lfloor i/4\rfloor}, \ell_{d-1,2\lfloor i/4\rfloor+1}, ..., \ell_{1,2\lfloor i/2^d\rfloor}, \ell_{1,2\lfloor i/2^d\rfloor+1})$.

The computational-hiding property of the commitment scheme $\mathtt{C}$ implies that emulation of Part 1 is indistinguishable from the corresponding part in a real execution. Given this fact, the (strong) witness indistinguishability property of $P_{\text{wi-ua}}$ implies that the emulation of Part 2 is indistinguishable from the corresponding part in a real execution. We stress that when using the witness indistinguishability property of $P_{\text{wi-ua}}$, we refer to witnesses that are verifiable in fixed polynomial-time (because $\{\widetilde{V}_n\}_{n\in\mathsf{N}}$ has been fixed for the current discussion). That is, for every polynomial bounding the size of the verifier's strategy, we prove that the simulator's output (produced when given the verifier's strategy as auxiliary input) is computationally indistinguishable from a real execution.

Construction 4.4 is computational-sound: We show that any feasible cheating strategy for the prover, yields a feasible algorithm that form collisions with respect to the basic hashing family $\{h_\alpha : \{0,1\}^{2|\alpha|} \to \{0,1\}^{|\alpha|}\}_{\alpha\in\{0,1\}^*}$. The main idea is to use the (weak) proof-of-knowledge property of $V_{\text{wi-ua}}$ in order to *implicitly* reconstruct an error-correcting codeword that encodes (different) valid witnesses that corresponding to different executions of Part 1. We stress that since there is no a-priori polynomial bound on the length of such witnesses, we cannot afford to *explicitly* reconstruct them (as done in [6], where only a contradiction to super-polynomial hardness is derived). However, implicit reconstruction of valid codewords will suffice, because different witnesses will be encoded by codewords that differ on a constant fraction of the bit-locations.

Lemma 4.5 *Construction 4.4 is computationally-sound* (w.r.t L)*, provided that the family* $\{h_\alpha\}$ *is indeed collision-free.*

Proof: Suppose towards the contradiction that there exists a feasible prover strategy that fools V with non-negligible probability (to accept inputs not in L). Specifically, let $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$ be such a family, and p be a polynomial such that for infinitely many $x \notin L$ it holds that $p_x \stackrel{\text{def}}{=} \Pr[(\widetilde{P}_n, V)(x) = 1] > 1/p(|x|)$. Let us fix a generic n and $x \in \{0,1\}^n \setminus L$ such that $p_x > 1/p(n)$. For simplicity, we incorporate this x in $\widetilde{P}_n$.

Using $\{\widetilde{P}_n\}_{n\in\mathsf{N}}$, we present a family of circuits $\{C_n\}_{n\in\mathsf{N}}$ that try to form collisions. On input $\alpha \in \{0,1\}^n$, the circuit C_n proceeds as follows:

1. Invoking $\widetilde{P}_n$, on input α, we obtain $c \leftarrow \widetilde{P}_n(\alpha)$. This is supposedly a commitment produced by the cheating prover (in the second step of Part 1 of the protocol).
2. Uniformly select $r \in \{0,1\}^n$, feed it to $\widetilde{P}_n$, and derive a residual prover $\widetilde{P}_n(\alpha, r)$ for Part 2 of the protocol. That is, $\widetilde{P_{\text{wi-ua}}} \stackrel{\text{def}}{=} \widetilde{P}_n(\alpha, r)$ is a prover strategy for the (witness-indistinguishable) universal-argument system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$.
3. Invoking the knowledge-extractor guaranteed for the system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$, while providing it with oracle access to $\widetilde{P_{\text{wi-ua}}}$, we reconstruct the values in bit-locations $n+1, ..., 2n$ in the witness. (The knowledge-extractor used corresponds to acceptance probability threshold $1/4p(n)$, and so it runs in polynomial-time and succeeds with noticeable probability when given access to a residual prover that makes $V_{\text{wi-ua}}$ accept with probability at least $p_x/4$.)

   Recall that since $x \notin L$, the witness must encode a program Π such that $\Pi(c) = r$ and the reconstructed bits encode the length of $\mathtt{ECC}(\Pi)$, denoted m. The values in bit-locations $2n+1, ..., 2n+m$ are an error-correcting encoding of Π, and subsequent m strings contain authentication information for the corresponding bits.
4. We uniformly select $i \in \{1, ..., m\}$. Invoking the knowledge-extractor again with oracle access to $\widetilde{P_{\text{wi-ua}}}$, we reconstruct the values in the bit-locations that correspond to the authenticator of the i^{th} bit in the codeword.

5. We repeat Steps 2 and 4 with a new uniformly selected $r' \in \{0,1\}^n$ but with the same value of i as selected in Step 4. That is, analogously to Step 2, we first obtain a corresponding prover strategy $\widetilde{P'_{\text{wi-ua}}} \stackrel{\text{def}}{=} \widetilde{P}_n(\alpha, r')$ (for the (witness-indistinguishable) universal-argument system $(P_{\text{wi-ua}}, V_{\text{wi-ua}})$). We need not repeat Step 3 because the perfect-binding property of $\mathtt{C}$ guarantees the uniqueness of m (obtained in Step 3). Analogously to Step 4, we invoke the knowledge-extractor with oracle access to $\widetilde{P'_{\text{wi-ua}}}$, and obtain the values in the bit-locations that correspond to the authenticator of the i^{th} bit in the codeword.

   Recall that since $x \notin L$, the witness used by $\widetilde{P'_{\text{wi-ua}}}$ must encode a program Π' such that $\Pi'(c) = r'$. Since (with probability $1 - 2^{-n}$ it holds that) $r \neq r'$ (and $\Pi(c) = r$), it must be the case that $\Pi' \neq \Pi$ (whereas $|\mathtt{ECC}(\Pi)| = m = |\mathtt{ECC}(\Pi')|$). We hope that $\mathtt{ECC}(\Pi)$ and $\mathtt{ECC}(\Pi')$ differ on the i^{th} bit (which happens with constant probability), in which case we obtain authenticators to conflicting values (with respect to the same tree-root (where the uniqueness is due to perfect-binding property of $\mathtt{C}$)).

6. Consider the authenticators obtained in Steps 4 and 5. If they authenticate conflicting values, then we obtain a collision under h_α (as in the proof of Claim 3.5.2).

The above circuit family has polynomial-size (because each C_n is implementable by the same probabilistic polynomial-time oracle machine, which in turn is given access to the polynomial-size $\widetilde{P}_n$). We now turn to analyze the success probability of C_n.

Claim 4.5.1 *Given a uniformly distributed $\alpha \in \{0,1\}^n$, with probability at least $1/\text{poly}(n)$, the circuit C_n outputs a collision with respect to h_α.*

Proof: Recall that $\widetilde{P}_n$ makes V accept x with probability $p_x > 1/p(n)$, where the probability is taken over V's random choices in the two parts of Construction 4.4. Moreover, V's choices in Part 1 are (α, r), where α (resp., r) is uniformly selected in $\{0,1\}^n$.

We call α good if the probability that $\widetilde{P}_n$ makes V accept x, conditioned on V selecting α in the first step of Part 1, is at least $p_x/2$. Clearly, at least a $p_x/2$ fraction of the α's are good, and we focus on any such good α.

Similarly, we call r α-good if the probability that $\widetilde{P}_n$ makes V accept x, conditioned on V selecting α and r in Part 1, is at least $p_x/4$. We denote by $G = G_\alpha$ the set of α-good strings, and note that $|G| > (p_x/4) \cdot 2^n$ (since α is good). That is, for every $r \in G$, the residual prover $\widetilde{P}_n(\alpha, r)$ convinces $V_{\text{wi-ua}}$ with probability at least $p_x/4 > 1/4p(n)$, and therefore the knowledge-extractor (given oracle access to $\widetilde{P}_n(\alpha, r)$) implicitly extracts the corresponding witness with probability at least $q_n \stackrel{\text{def}}{=} 1/\text{poly}(n)$.

Observe that, with probability at least $(p_x/4)^2$, both r and r' selected in Steps 2 and 5 respectively reside in G. Conditioned on this event, we (implicitly) extract the corresponding witnesses with probability at least q_n^2. We stress that the two witnesses must be of the same length by the virtue of their length being committed to in c (sent by the prover in the second step of Part 1). Finally, with constant probability, these witnesses differ in bit position i (selected in Step 4), in which case C_n forms a collision under h_α.

We conclude that, for any good α, the circuit C_n forms a collision under h_α with probability at least

$$(p_x/4)^2 \cdot q_n^2 \cdot \Omega(1) \ = \ \Omega(p_x^2 \cdot (1/\text{poly}(n))^2) \tag{2}$$

Using $p_x > 1/p(n)$, we lower-bound Eq. (2) by $\text{poly}(p_x)$. Recalling that at least $p_x/2$ of the α's are good, the claim follows. □

Using Claim 4.5.1, we derive a contradiction to the hypothesis that $\{h_\alpha\}$ is collision-free. The lemma follows. ■

Achieving bounded concurrent zero-knowledge. We have shown that Construction 4.4 satisfies the first two items of Theorem 1.2. To establish the third item we need to modify the construction a little (analogously to the way this is done in [6]). Specifically, for a suitably chosen polynomial ℓ, in Part 1 we select r uniformly in $\{0,1\}^{\ell(n)+n}$ (rather than in $\{0,1\}^n$), and in Part 2 we relax the second case so that now we require that there exists a string $z \in \{0,1\}^{\ell(n)}$ such that $\Pi(c,z) = r$ (rather than requiring that $\Pi(c) = r$). The extra string z allows to encode information from n^2 concurrent executions of the protocol (see details in [6]), and so enables the simulator strategy to insert a "trapdoor" to the second step of Part 1 of the current execution (and thus proceed in an "execution-by-execution" manner). When demonstrating computational-soundness, we merely observe that for a uniformly distributed $r' \in \{0,1\}^{\ell(n)+n}$ (chosen in Step 5), it is unlikely that Π, which is (implicitly) recovered (in Step 4) obliviously of r', will satisfy $\Pi(c,z') = r'$ for some $z' \in \{0,1\}^{\ell(n)}$.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *JACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[2] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *JACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[3] L. Babai. Trading Group Theory for Randomness. In *17th STOC*, pages 421–429, 1985.

[4] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, Vol. 1, No. 1, pages 3–40, 1991. Preliminary version in *31st FOCS*, 1990.

[5] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd STOC*, pages 21–31, 1991.

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

[7] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *Crypto'92*, Springer-Verlag LNCS 740, pages 390–420.

[8] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali and P. Rogaway. Everything Provable is Provable in Zero-Knowledge. In *Crypto'88*, Springer-Verlag LNCS 403, pages 37–56, 1990.

[9] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability. In *20th STOC*, pages 113–131, 1988.

[10] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988. Preliminary version by Brassard and Crépeau in *27th FOCS*, 1986.

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

[12] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *JACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[13] U. Feige and J. Kilian. Making games short (extended abstract). In *29th STOC*, pages 506–516, 1997.

[14] U. Feige, A. Shamir and M. Tennenholtz. The noisy oracle problem. In *Crypto'88*, Springer-Verlag LNCS 403, pages 284–296.

[15] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *Crypto'89*, Springer-Verlag LNCS Vol. 435, pages 526–544, 1990.

[16] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SICOMP*, Vol. 29 (1), pages 1–28, 1999.

[17] L. Fortnow, J. Rompel and M. Sipser. On the power of multi-prover interactive protocols. In *Proc. 3rd IEEE Symp. on Structure in Complexity Theory*, pages 156–161, 1988.

[18] O. Goldreich. *Foundation of Cryptography – Basic Tools.* Cambridge University Press, 2001.

[19] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38, No. 1, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[20] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SICOMP*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985.

[21] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *24th STOC*, pages 723–732, 1992.

[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems. *JACM*, Vol. 39, No. 4, pages 859–868, 1992. Preliminary version in *31st FOCS*, 1990.

[23] S. Micali. Computationally Sound Proofs. *SICOMP*, Vol. 30 (4), pages 1253–1298, 2000. Preliminary version in *35th FOCS*, 1994.