Verifying and Decoding in Constant Depth

Shafi Goldwasser *
CSAIL, MIT and
Weizmann Institute
shafi@theory.csail.mit.edu

Dan Gutfreund †
SEAS
Harvard University
danny@eecs.harvard.edu

Alexander Healy‡
SEAS
Harvard University
ahealy@fas.harvard.edu

Tali Kaufman
CSAIL, MIT
kaufmant@mit.edu

Guy N. Rothblum §
CSAIL, MIT
rothblum@csail.mit.edu

ABSTRACT

We develop a general approach for improving the efficiency of a computationally bounded *receiver* interacting with a powerful and possibly malicious *sender*. The key idea we use is that of *delegating* some of the receiver's computation to the (potentially malicious) sender. This idea was recently introduced by Goldwasser et al. [14] in the area of program checking. A classic example of such a sender-receiver setting is interactive proof systems. By taking the sender to be a (potentially malicious) prover and the receiver to be a verifier, we show that (p-prover) interactive proofs with k rounds of interaction are equivalent to (p-prover) interactive proofs with $k + O(1)$ rounds, where the verifier is in $\mathbf{NC^0}$. That is, each round of the verifier's computation can be implemented in constant parallel time. As a corollary, we obtain interactive proof systems, with (optimally) constant soundness, for languages in **AM** and **NEXP**, where the verifier runs in constant parallel-time.

Another, less immediate sender-receiver setting arises in considering error correcting codes. By taking the sender to be a (potentially corrupted) codeword and the receiver to be a decoder, we obtain explicit families of codes that are locally (list-)decodable by *constant-depth* circuits of size polylogarithmic in the length of the codeword. Using the tight connection between locally list-decodable codes and average-case complexity, we obtain a new, more efficient, worst-case to average-case reduction for languages in **EXP**.

*Research supported by NSF grant CNS-0430450, NSF grant CFF-0635297 and a Cymerman-Jakubskind award.
†Research supported by ONR grant N00014-04-1-0478 and NSF grant CNS-0430336.
‡Research supported by NSF grant CCR-0205423 and a Sandia Fellowship.
§Research supported by NSF grant CNS-0430450 and NSF grant CFF-0635297.

STOC'07, June 11–13, 2007, San Diego, California, USA.

Categories and Subject Descriptors

F.1.2 [**Computability by Abstract Devices**]: Modes of Computation

General Terms

Algorithms, Theory

Keywords

Interactive Proofs, Error-Correcting Codes, Constant-Depth Circuits

1. INTRODUCTION

In this work we consider settings in which a computationally powerful but possibly malicious *sender* interacts with a weak and honest *receiver*. We explore the approach of improving the receiver's efficiency by *delegating* most of its computational work to the sender. It is not immediately apparent how to do this, as the sender may be malicious, and thus we introduce tools for *reliably* delegating computation to a potentially malicious party. We apply our approach to the settings of proof verification and error correcting codes.

Proof Verification.

Proof verification is a central concept in theoretical computer science. In this setting, a computationally powerful and possibly malicious party, the *prover*, interacts with a weak and honest party, the *verifier*. The prover makes a claim (e.g., that a given graph is 3-colorable), and tries to convince the verifier that this claim is valid. The goal is to design a (possibly randomized and/or interactive) proof system such that the verifier is only convinced when the prover's claim is in fact correct, even when the prover behaves maliciously. Proof verification is perhaps the most natural example in complexity theory of a weak receiver interacting with a potentially malicious powerful sender. Here, the prover plays the part of the sender and the verifier plays the role of the weak receiver.

Seminal works in complexity theory have uncovered the striking power of interactive and randomized proof systems as introduced by Babai [4] and Goldwasser, Micali and Rackoff [15]. The works of Shamir [22], and Babai, Fortnow and Lund [5], give efficient proof systems for every language that could conceivably have them. The works of Babai and Moran [7], Goldwasser and Sipser [16], Feige and Lovasz [12], and the PCP literature, show that interactive proof

systems remain very powerful even when various limitations are placed on the verifier or on the protocol.

In this work we continue this line of research, exploring the power of proof systems with verifiers that have very limited computational resources. We show that proof systems remain powerful even when the verifier is severely weakened computationally to run in $\mathbf{NC^0}$ (constant parallel time).

Error-Correcting Codes.

Error correcting codes are highly useful combinatorial objects. In this setting, a given message is encoded with some redundancy to obtain a codeword (longer than the original message) with the property that the original message can be recovered from any word that is close (in Hamming distance) to the codeword. The process of recovering the message from a (possibly) corrupted codeword is called *decoding*. Several variations on the standard notion of decoding have been considered in the literature. In some cases the codeword is corrupted in so many locations that recovering the original message becomes impossible; however, it might still be possible to recover a (short) list of candidate messages that is guaranteed to contain the original message – this is known as *list decoding*. One can also consider the task of recovering only a single bit of the message (in some given location) by accessing only a small number of locations in the codeword. Codes with such local decoding properties are called *locally decodable codes* and *locally list-decodable codes*. See Trevisan's survey [24] for a more detailed discussion.

In this work we focus on the efficiency of the decoder in locally decodable and locally list-decodable codes. We show explicit families of codes that are locally and locally list decodable by constant depth decoders (circuits) of size polylogarthmic in the length of the codeword.

Worst-case to average-case reductions.

The question of the efficiency of decoding locally list-decodable codes is natural on its own. However, another strong motivation for its study comes from its relation to worst-case to average-case reductions, which we now briefly disucss.

One of the most fundamental questions in complexity theory is proving lower bounds for complexity classes. The state-of-the-art circuit complexity lower bounds only apply to circuits that are well below $\mathbf{NC^1}$. In fact, there exist complexity classes that are not known to include the majority function, yet with our current state of knowledge may include the whole of **EXP** (e.g. constant-depth circuits with mod6 gates). Even if we had a *worst-case* lower bound for such a complexity class C, it would not necessarily imply an *average-case* lower bound; indeed, it would remain entirely possible that for every language in **EXP** there exist a family of circuits in C that decides the language very well on random inputs (i.e. errs only on a small fraction of inputs).

One way to overcome this obstacle is to show a reduction from solving languages in **EXP** in the worst-case to solving (possibly other) languages in **EXP** on average. A series of works [21, 6, 18, 23, 25] has revealed a strong connection between locally (list-)decodable codes and such *worst-case to average-case reductions* (for languages in **EXP**). These results relate the efficiency of the decoder of a locally-decodable code to the efficiency of the reduction. However, the worst-case to average-case reductions implied by previous works involve computing majorities or field operations, and they are thus too expensive to apply to a weak complexity class such as the above C.

Our constructions of efficient local-decoders give new and more efficient worst-case to average-case reductions that resolve this problem.

1.1 Main results

Proof verification.

We consider proof systems with extremely weak verifiers: i.e., verifiers in $\mathbf{NC^0}$ (or constant parallel time). By that we mean that the verifier's strategy at each interaction round can be computed in $\mathbf{NC^0}$ when given access to the input, the randomness, and the messages exchanged in the previous rounds. We show that such proof systems are surprisingly powerful: essentially, anything that is provable in k communication rounds with a polynomial-time verifier is also provable in $k + O(1)$ communication rounds with an $\mathbf{NC^0}$ verifier.[1] In particular, we obtain the following characterizations:

1. A language is in **AM** (the class of languages that have a proof system with a constant number of communication rounds) if and only if it has a single-prover, two-round, constant-soundness interactive proof that can be verified in constant parallel time (Corollary 3.10).[2] In particular, every language in **NP** has such a proof system.

2. A language is in **NEXP** if and only if it has a two-prover, five-round interactive proof of constant soundness, that can be verified in constant parallel time (Theorem 3.11).

Previous proof systems for complete languages in **NP**, **IP** and **NEXP** require, at the very least, the verification of an **NP** statement at the end of the protocol (even to achieve constant soundness). By the Cook-Levin reduction, this verification is very efficient (i.e. in $\mathbf{AC^0}$); indeed, a key point in the Cook-Levin theorem is that computation can be verified by making many local consistency checks and ensuring that they all hold. However, while the local checks are of constant size, the verifier still needs to verify that *all of them hold* by computing an AND of large fan-in, and therefore is not in $\mathbf{NC^0}$. We show, somewhat surprisingly, that a verifier can use interaction with the prover together with its (*private*) random coins to avoid performing a global test on its entire input and proof! This is done by replacing the global test with a test that the prover is not cheating.

[1] We observe in Section 3.3 that by adding $O(\log n)$ communication rounds it is not hard to transform any protocol into one with an $\mathbf{NC^0}$ verifier. However, we achieve this while only adding a *constant* number of communication rounds. this is what enables us to obtain constant parallel time verification for languages in **AM** and **NEXP**.

[2] Here we follow the standard convention that measures the complexity of the protocol only in terms of the resources used by the verifier, i.e. it is assumed that the prover's messages are generated instantly. Thus, our statement about constant parallel-time verification follows from the fact that the protocols we construct have a constant number of communication rounds and that each round the verifier's strategy can be implemented in constant parallel time.

The size (fan-in) of this new test is only a function of the soundness, independent of the size of the input.[3]

Negative results.

We complement our positive results with two negative results. First, we show that constant-round proof systems with an $\mathbf{NC^0}$ verifier cannot have sub-constant soundness (unless the language itself is in $\mathbf{NC^0}$). Second, we show that there is no public-coin proof system with an $\mathbf{NC^0}$ verifier (again, unless the language itself is in $\mathbf{NC^0}$). This result sheds light on private vs. public coins proof systems and in particular on our protocols (which, naturally, use private coins). In particular, it shows that both *interaction* and *private randomness* are *provably essential* for non-trivial $\mathbf{NC^0}$ verification.

Error-correcting codes.

The application of our methodology to the setting of error-correcting codes is a novel approach to the well-studied problem of efficient decoding: the sender embeds information in the codeword that helps speed up the decoder's computation. In our new codes, the decoder uses the received word not only for reconstructing information about the original message, but also as a *computational resource*. This approach allows us to transform locally (list-)decodable codes with $\mathbf{NC^1}$ decoders into codes that have $\mathbf{AC^0}$ decoders (with similar or slightly worse parameters). We then construct explicit locally-decodable and locally list-decodable codes with $\mathbf{NC^1}$ decoders, and by applying our general transformations to these explicit codes, we obtain the following:

1. An explicit binary code with polynomial rate and $\mathbf{AC^0}$ (probabilistic) local-decoding from a word that is corrupted in a constant fraction of locations (Theorem 4.5). This code has roughly the same parameters as the canonical example of a locally-decodable binary code with polynomial rate (see [23]).

2. An explicit family of (non-binary) codes that is locally list-decodable from agreement ε with list size $\mathrm{poly}(1/\varepsilon)$ by probabilistic $\mathbf{AC^0}$ circuits that are of size $\mathrm{poly}(\log M/\varepsilon)$, where M is the length of the message (Theorem 4.6). The alphabet size and codeword length of these codes match the recent construction of Impagliazzo et al. [17]. (Indeed, our construction is based on the approximate codes of [17].)

We note that all previously known decoders for locally (list-)decodable codes with similar parameters compute majorities or finite field operations that (provably) cannot be implemented in $\mathbf{AC^0}$.

[3]Although we emphasize the locality of the verifier, one should not confuse our verifiers with PCP verifiers. While the latter do look at a constant number of bits in the proof, they still need to check consistency with the whole input (e.g. when computing the PCP reduction) and therefore are not in $\mathbf{NC^0}$ (as a function of the input, the proof and the randomness). In fact, our lower bounds (see Section 3.3) show that non-trivial languages *cannot* have $\mathbf{NC^0}$ PCP verifiers (roughly speaking, this is because PCPs are not interactive).

Efficient worst-case to average-case reductions.

By using the tight connection (discussed above) between locally decodable codes and worst-case to average-case reductions, we show that if $\mathbf{EXP}$ is worst-case hard for any complexity class C containing uniform $\mathbf{AC^0}$, then $\mathbf{EXP}$ is hard on average for C. This gives the first worst-case to average-case reduction (in $\mathbf{EXP}$) where the worst/average-case hardness is with respect to complexity classes that cannot compute the majority function, nor simple operations over finite fields.

1.2 Our approach

We improve the receiver's efficiency by delegating some of its computation to the (possibly malicious) sender. The idea of delegating computation from the receiver to the untrusted sender seems dubious at first glance, as the receiver's computation is the only reliable part in the whole interaction; indeed, this seems to leave the receiver very vulnerable to malicious behavior of the sender. To give the receiver a better guarantee, we ask more from the sender: we ask the sender to convince the receiver that he has performed the computations correctly (in the case of proof verification), or to send the results of the computations with redundancy that will allow the receiver to easily recover the correct results even from a corrupted word (in the case of codes). This may seem to bring us back to square one; namely, the receiver again needs to verify a proof or to decode a code. So where do we gain in efficiency? The key point is that we are not trying now to verify an arbitrary claim, or to recover arbitrary information, but rather we are trying to make sure that a certain *computation* was conducted correctly. Here one can see a connection to program checking and correcting that we discuss below. Specifically, we show that if the receiver's computations have certain properties, which we discuss shortly, then the tasks of verifying their correctness or decoding the correct results of the computations can be done extremely efficiently – much more efficiently than the receiver's original computation. To develop our approach we look at functions that the receiver needs to compute and require them to have two properties:

1. (Random instance reduction) One can compute the function on any given instance by querying another function (say g) at a completely random location. This property allows us to "mask" the receiver's computation as a random instance and to correct the sender's computations.

2. (Solved instance generator) We can generate efficiently a random instance of the function g together with g's value on this instance. This property allows us to check the correctness of the sender's computations.

Combining these properties allows us to ensure (w.h.p.) that the computations that the sender conducts for the receiver are indeed correct. Of course, this approach would not give us much if we could not show that the above properties can be implemented more efficiently than the original computations. To that end we show, using techniques that were developed in the field of cryptography [20, 11, 19, 3], that for functions computable in $\mathbf{NC^1}$ these properties can be implemented in probabilistic constant parallel time. Thus, we can take any $\mathbf{NC^1}$ receiver and transform it into one that runs in constant parallel time or constant depth. This

reduces our task to finding a sender-receiver protocol for the required task in which the receiver is in $\mathbf{NC^1}$. For some of our applications, even designing an $\mathbf{NC^1}$ receiver requires technical effort.

Related Work.

Our approach for improving the receiver's efficiency by delegating some of its computation to the (possibly malicious) sender, is based on a delegation methodology and tools developed in a recent work of Goldwasser et al. on improving the efficiency of program checkers [14], and also inspired by the work of Appelbaum, Ishai and Kushilevitz [3] on improving the efficiency of cryptographic primitives. A discussion about the similarities and differences between these works and the results presented here follows.

[14] develops a methodology of delegating computation as a way to increase the efficiency of program checkers (see [9]) and program testers/correctors (see [10]). This idea plays a key role in their general approach of composing program checkers (and testers/correctors).

In fact, one can view program checking as an interactive proof setting where the prover is analogous to the program and the checker is analogous to a verifier. The prover in the program checking setting is fixed in advance and restricted to computing only the language being proved, as opposed to being computationally unbounded and dynamic (according to the messages exchanged in the protocol) in the usual proof verification setting. Moreover, a program checker for a language gives such a proof system both for the language and for its complement. These differences give rise to different challenges in the design of such protocols, and in particular in the implementation of the delegation methodology. The fact that the prover is restricted to answering queries about the language being proved, in the case of program checkers, requires careful design of such protocols that typically use very specific properties of the functions being proved (checked). In fact, it is not at all well understood which languages have such proof systems. [14] gave both a methodology for constructing such systems with very efficient verification (checking) and a family of results for a wide variety of languages. While in this work we consider the (easier) setting of an unbounded prover, we (as opposed to [14]) must deal with the challenge that the prover may change its answers according to the messages exchanged in the interaction. For example, in our setting one cannot design proof systems that first test the prover on random inputs, and then correct it (which is a common methodology for constructing program checkers).

Our results on constructing efficient error-correcting codes are also related to the results in [14] on building efficient program correctors. Again, in this setting we design decoders that delegate work to an encoder that can perform arbitrary (efficient) computations, whereas in the program correction setting one must deal with the difficulty of having a restricted encoder who can only compute membership in some specific language. An extra challenge that we address in this work (as opposed to [14]) is recovering from very large fractions of errors (especially in the list-decoding setting).

The work of [3] can also be viewed as improving the efficiency of players participating in a protocol by pushing computation from one of the participants to another (e.g. improving the efficiency of encryption at the expense of adding to the complexity of decryption). The main difference between this approach and ours is that they consider protocols or objects in which the *goal* of the sender is to reveal the results of its computation to a receiver, so there is no issue of a malicious party. In our case, on the other hand, the sender is *untrusted* but the receiver still wants it to perform computations for him. Given these differences, the tools we use are somewhat different from those used in [3]. Nonetheless, some of the techniques we employ are similar.

1.3 Organization

In Section 2 we give some general preliminaries. Specific background about interactive proofs and error-correcting codes appears in Sections 3.1 and 4.1 respectively. In Section 2.1 we develop the tools that allow us to implement our approach. In Section 3 we present our results in the area of proof verification. Section 3.2 shows how to transform general proof systems to ones with verifiers in $\mathbf{NC^0}$. In Section 3.3 we present our negative results. In Section 4 we present our results in the area of error-correcting codes. In Section 4.2 we show how to transform codes with $\mathbf{NC^1}$ local decoders to codes with $\mathbf{AC^0}$ local decoders. In Section 4.3 we show similar transformations for list-decodable codes. In Section 4.4 we present our explicit constructions, and in Section 4.5 we state the worst-case to average-case reductions that we get using our codes.

Due to space limitations, many proofs are omitted.

2. PRELIMINARIES AND MAIN TOOLS

For a string $x \in \Sigma^*$ (where Σ is some finite alphabet) we denote by $|x|$ the length of the string, and by x_i the i'th symbol in the string. For a finite set S we denote by $y \in_R S$ that y is a uniformly distributed sample from S. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$.

We assume that the reader is familiar with standard complexity classes such as $\mathbf{NP}$, $\mathbf{EXP}$ and $\mathbf{NEXP}$. $\mathbf{AC^0}$ circuits are boolean circuits (with AND, OR and NOT gates) of constant-depth, polynomial size, and unbounded fan-in AND and OR gates. $\mathbf{NC^1}$ circuits are boolean circuits of fan-in 2, polynomial size and logarithmic (in the input size) depth. $\mathbf{NC^0}$ circuits are similar to $\mathbf{NC^1}$, but have constant-depth. Note that in $\mathbf{NC^0}$ circuits, every output bit depends on a constant number of input bits. $\mathbf{AC^0}$, $\mathbf{NC^1}$ and $\mathbf{NC^0}$ are the classes of languages (or functions) computable (respectively) by $\mathbf{AC^0}/\mathbf{NC^1}/\mathbf{NC^0}$ circuits. In this work, circuits may have many output bits (we specify the exact number when it is not clear from the context). Also, all the circuit families that we consider in this paper are log-space uniform (even if we do not explicitly state that), i.e. each circuit in the family can be described by a Turing machine that uses a logarithmic amount of space in the size of the circuit. Thus $\mathbf{NC^0}$ (resp. $\mathbf{NC^1}$) computations in this work are equivalent to constant (resp. logarithmic) parallel time in the CREW PRAM model.

2.1 Randomized images

We start by defining the properties of functions that we need for our approach. The first property says that we can easily generate a random instance together with the evaluation of the function on this input.

Definition 2.1 (Solved instance generator). *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function. We say that a randomized algorithm G is a* solved instance generator *for f if,*

given 1^n, it generates a pair (x, y), where x is a uniformly random element of $\{0,1\}^n$ and $y = f(x)$.

The second property is a reduction from one function to another that says, roughly, that we can evaluate the first function on every instance by querying the second function in a random location.

DEFINITION 2.2 (RANDOM INSTANCE REDUCTION). *Let $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \to \{0,1\}^*$ be two functions. We say that a pair of algorithms (R, E) is a* random instance reduction *from f to g if R is a randomized algorithm that given $x \in \{0,1\}^n$, generates a pair (x', τ), where x' is a uniformly random element of $\{0,1\}^{m(n)}$ and $\tau \in \{0,1\}^*$ and it holds that $E(g(x'), \tau) = f(x)$.*

If $m(n) = n$ we say that the random instance reduction is length-preserving. *If f and g are the same function, we say that it is a random instance self-reduction.*[4] *We call R the* Randomizer *and E the* Evaluator.

The objects that we will be interested in are pairs of functions that have the above two properties.

DEFINITION 2.3 (RANDOMIZED IMAGE). *Let $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \to \{0,1\}^*$ be two functions. We say that g is a* randomized image *of f, if there is a random instance reduction from f to g, and g has a solved instance generator.*

We say that it is length-preserving if the random instance reduction is length-preserving, and that it is a randomized self-image if $f = g$. Finally, we say that the randomized image can be implemented in some complexity class $\mathcal{C}$, if the algorithms G, R and E (from Definitions 2.1 and 2.2) can be implemented in this class.

Another property of functions that we will find useful is the following strong form of downward self-reducibility.

DEFINITION 2.4. *We say that a language L is* strongly downward self-reducible *if, for every constant $\delta > 0$, L can be decided by a family of polynomial-size oracle constant-depth circuits such that the circuit for length n makes queries to an oracle that solves L at input length n^δ.*

We conclude this section with the following lemma, which says that there is an $\mathbf{NC^1}$-complete language that has all the properties that we need. The proof of this lemma (which is omitted) is heavily based on Barrington's theorem [8] and the machinery that was developed in the area of cryptography [20, 11, 19, 3].

LEMMA 2.5. *There is an $\mathbf{NC^1}$-complete language under $\mathbf{NC^0}$ reductions, that is strongly downwards self-reducible, and has a randomized image that can be implemented by $\mathbf{NC^0}$ circuits.*

[4]Random instance self-reductions are a special form of what is called in the literature *random self-reductions*. The word instance in our terminology, should emphasize the fact that the reduction is from one instance to another (random) instance. General random self-reductions can make many *self*-queries to the function in order to compute its value on a given instance.

3. INTERACTIVE PROOFS

3.1 Preliminaries

We give here the standard definition of interactive proof systems, adding some terminology that will be useful for us later.

DEFINITION 3.1. *An* interactive proof system *for a language L with completeness c and soundness s, is a two party game between a probabilistic polynomial-time verifier V and a computationally unbounded prover P. The system has two stages: First, in the* interaction *stage, V and P are given a common input x and they exchange messages to produce a transcript $t = (V(r), P)(x)$ (the entire messages exchange) where r are the internal random coins of V. Then, in the* decision *stage, V decides according to x, t and r, whether to accept or reject. The following should hold:*

1. *(Completeness) There exist a (honest) prover strategy P, such that for every $x \in L$, $\Pr_r[V(x,t,r) = \text{accept}] \geq c$, where $t = (V(r), P)(x)$.*
2. *(Soundness) For every $x \notin L$ and every prover P^*, $\Pr_r[V(x,t,r) = \text{accept}] \leq s$, where $t = (V(r), P^*)(x)$.*

If we do not specify c and s then their respective default values are 2/3 and 1/3.

A multi-prover interactive proof system *with p provers for a language L is the same as an interactive proof system with the only difference that at the interaction stage V exchange messages with p different provers that have no communication between them. Thus the transcript t contains p separate sub-transcripts $t_1, \ldots, t_p$ each with a different prover.*

A round of interaction is an exchange of p messages ($p \geq 1$) that are sent in parallel from the verifier to the p provers (one to each prover) and p messages that are sent in parallel back from the provers to the verifier (one from each prover). Note that the number of rounds may depend on the length of the input to the protocol. We denote by **AM** (i.e., Arthur-Merlin games) the class of languages that have protocols with one prover and a constant number of interaction rounds.

Next we formally define the notion of $\mathbf{NC^0}$ (or constant parallel time) verifiers.

DEFINITION 3.2. *We say that round i of a proof system (with one or more provers) can be computed in $\mathbf{NC^0}$ if the computation of the verifier in this round can be performed by an $\mathbf{NC^0}$ circuit (that may depend on the round i) that is given the input x to the protocol, the randomness r and the partial transcript from the previous $i-1$ rounds.*

We assume that the circuit may depend on the number of the round because $\mathbf{NC^0}$ circuits cannot even increment an integer by 1.[5]

DEFINITION 3.3. *We say that a language can be verified (interactively) in constant parallel time, if it has an interactive proof system (with one or more provers) with a constant*

[5]We could consider a model with the same $\mathbf{NC^0}$ verifier in all rounds. The models are equivalent for protocols with $O(1)$ communication rounds. For other protocols results carry through, except that we can only bound the *expected* number of communication rounds when interacting with a malicious prover.

number of rounds, and the entire verifier's strategy can be implemented in $\mathbf{NC^0}$*, i.e. every round of the interaction stage as well as the decision stage.*

We define a special type of proof systems in which "most" of the verifier's computation is pushed to the decision stage, keeping the computation during the interaction stage extremely efficient.

DEFINITION 3.4. *We say that a proof system (with one or more provers) is* simple *if in every round of the interaction stage the verifier's computation can be performed by an* $\mathbf{NC^0}$ *circuit.*

3.2 Verifiers in constant parallel time

We start by showing how to transform any simple proof system into one in which the entire strategy of the verifier (interaction and decision) is in $\mathbf{NC^0}$.

LEMMA 3.5. *Every language that has a k-round simple interactive proof system with $p \geq 1$ provers, completeness c and soundness s, has an interactive proof system with p provers, $k+2$ rounds, completeness c and soundness $s+\delta$, where $\delta > 0$ is an arbitrarily small constant, and the verifier's entire strategy (both interaction and decision) is in* $\mathbf{NC^0}$.

PROOF. We first consider the case of a single prover, i.e. $p = 1$.

We proceed by showing that any simple proof system can be transformed into a simple proof system (with one more round) in which the decision stage can be implemented in $\mathbf{AC^0}$ (this essentially boils down to evaluating a CNF formula via the Cook-Levin reduction) and hence also in $\mathbf{NC^1}$. Then we add one more round to enable the verification to be performed in $\mathbf{NC^0}$. Details follow.

Let V' be the verifier and P' the honest prover in the original (simple) protocol and let V, P be these entities in the new protocol. By the definition of a simple proof system (Definition 3.4), the computations of V' during the interaction stage are in $\mathbf{NC^0}$. Thus, V and P run the first k rounds as in the original protocol, and we then add two rounds as follows.

Round $k+1$.

In the original protocol, given the input x (of length n), the transcript t and the verifier's random coins r, V' can decide in polynomial time whether to accept or reject. Let $|(x,t,r)| = m(n) = poly(n)$. This round is as follows: V sends all its random coins to P and P sends back the tableau of the computation $V'(x,t,r)$.

If V were an $\mathbf{AC^0}$ circuit, it could at this stage verify that the tableau is correct and deduce the output of V' (accept/reject). This is because checking the validity of the tableau amounts to verifying that x, t, r is written in the first row, and that all the local transitions are legal. However V is not an $\mathbf{AC^0}$ circuit. So we proceed to the next round.

Round $k+2$.

Consider the language L associated with the above $\mathbf{AC^0}$ computation. That is, an instance contains a tableau T and x, t, r as above, and it belongs to L if the tableau T is consistent with the computation $V'(x,t,r)$, and if this computation accepts. In particular $L \in NC^1$ and therefore, by Lemma 2.5, it has a randomized image I.

Let ℓ be an integer that will be determined later. Given an instance $a = (T, x, t, r)$, V does the following: for each $i \in [\ell]$ (in parallel and with independent random coins), choose uniformly $v_i \in_R \{0,1\}$. If $v_i = 0$, then V runs the $\mathbf{NC^0}$ solved instance generator for I on input length $m'(n)$, to obtain a pair (c_i, y_i). If $v_i = 1$, then V runs on a the $\mathbf{NC^0}$ random instance reduction from L to I, to obtain a pair (c_i, τ_i). Here $|c_i| = m'(n) = poly(m(n)) = poly(n)$. V then sends to P the message $(c_1, \ldots, c_\ell)$, and P sends back answers $(b_1, \ldots, b_\ell) \in \{0,1\}^\ell$.

Decision.

V accepts if and only if the following holds:

1. For every $i \in [\ell]$ for which $v_i = 0$, $y_i = b_i$.
2. For every $i \in [\ell]$ for which $v_i = 1$, $E(b_i, \tau_i) = 1$ (recall that E is the evaluator in the random instance reduction from L to I).

Correctness.

Clearly, if ℓ is a constant (independent of n) the entire strategy of V can be implemented in $\mathbf{NC^0}$. We proceed to prove completeness and soundness.

CLAIM 3.6. *The protocol has completeness c.*

PROOF. The honest prover P plays the first k rounds like the honest prover P' of the original protocol. It then sends the correct tableau, and the correct values $b_1, \ldots, b_\ell$, which are the membership values (0/1) of the instances $c_1, \ldots, c_\ell$ in I. By the definition of solved instance generator, this implies that with probability 1, the verifier passes the first test. By the definition of random instance reduction, for every $i \in [\ell]$ for which $v_i = 1$, $E(b_i, \tau_i) = 1$ if and only if $(T, x, t, r) \in L$. This happens exactly when the original verifier V' accepts the original protocol, and the probability for that is at least c. □

CLAIM 3.7. *The protocol has soundness $s + 2^{-\ell}$.*

PROOF. Let x be an instance not in the language. Consider the event:

$$A : E(b_i, \tau_i) = 1 \text{ for every } i \in [\ell] \text{ for which } v_i = 1$$

By the soundness of the original protocol and the definition of random instance reduction, $\Pr[A] \leq s$. If event A does not occur, the only way that a cheating prover, P^*, can convince V to accept is by cheating on c_i for every i for which $v_i = 1$. If the prover cheats on c_j where $v_j = 0$, then by the definition of solved instance generator, V will reject with probability 1. In other words, in order to cheat and not get caught, P^* must cheat on every i for which $v_i = 1$ and give the correct answer on every i for which $v_i = 0$. By the definitions of solved instance generator and random instance reduction, the v_i's are independent of $(c_1, \ldots, c_\ell)$. Thus P^* has to guess exactly the value of ℓ independent unbiased coin tosses which he can do with probability at most $2^{-\ell}$. We conclude that the probability that P^* can convince V to accept is bounded by $s + 2^{-\ell}$. □

Let $\delta > 0$ be an arbitrarily small constant. By setting $\ell = \log(1/\delta)$ we conclude the proof for single-prover systems.

For multi-provers, the same arguments apply where the last two rounds $(k+1, k+2)$ are played only with the first prover P_1. That is, in round $k+1$, V sends to P_1 its random coins as well as the transcripts of messages exchanged with all the other provers, then P_1 and V proceed as above.

REMARK 3.8. *In the proof above, the "hardest" computation that the verifier is performing is an AND of fan-in* $\log(1/\delta)$. *In terms of parallel computing time this amounts to* $\log\log(1/\delta)$. *Generalizing the argument to non-constant* δ *we can obtain proof systems with negligible soundness (e.g.* $n^{-\log n}$*) with a verifier that runs in* $O(\log\log n)$ *parallel time.*

REMARK 3.9. *Vadhan [26] has suggested an alternative implementation of round* $k+2$*: the prover wants to convince* V' *that* $V(x,t,r) = 1$. *Let* $b = V(x,t,r)$, *and for* $c \in \{0,1\}$ *denote* $I_c = \{z : I(z) = c\}$. *The verifier generates an instance* y *that is uniformly distributed in* I_b *restricted to the relevant input length. It also generates an instance* y' *that is uniformly distributed in* I_0 *restricted to the same input length. The ability to sample such* y, y' *follows directly from the techniques used to prove the results in Section 2.1. The verifier then chooses at random one of* y *and* y' *and the prover has to say whether it is from* I_1 *or* I_0. *Note that this is very similar to the protocol for Graph Non-Isomorphism [13].*

General Proof Systems.

Next we want to use Lemma 3.5 to obtain our results about general proof systems. For proof systems with a single prover, we can first apply the transformation of Goldwasser and Sipser [16] to obtain a public-coin protocol (which is clearly also a simple protocol). Then, by applying Lemma 3.5 to the resulting protocol we obtain a general transformation from any interactive single-prover proof system to a one in which the verifier is in $\mathbf{NC^0}$ with an addition of $O(1)$ rounds. In particular, we obtain the following corollary:

COROLLARY 3.10. *A language is in AM if and only if it can be verified in constant parallel time with one prover, two rounds of interaction, and an arbitrarily small constant soundness.*

Next we want to apply similar arguments to obtain $\mathbf{NC^0}$ verifiers for multi-prover proof systems. The problem is that now we do not readily have a general transformation to simple proof systems as we had in the case of a single prover. However, we show that a modification of the techniques from [12] gives us a general transformation to simple 2-provers systems. Then, combining this with Lemma 3.5, and the MIP characterization of **NEXP** [12] , we obtain the following theorem.

THEOREM 3.11. *A language is in* **NEXP** *if and only if it can be verified in constant parallel time with two provers, five rounds, perfect completeness and soundness* δ, *where* $\delta > 0$ *is an arbitrarily small constant.*

3.3 Lower bounds

In this section we prove that the use of private coins in our protocol is inherent. We also show that constant soundness is the best one can hope for in proof systems that have a constant number of rounds and an $\mathbf{NC^0}$ verifier. These statements hold unless the language is already in $\mathbf{NC^0}$.

We start with a more refined definition of $\mathbf{NC^0}$.

DEFINITION 3.12. *For* $k \in \mathbb{N}$, $\mathbf{NC^0_k}$ *is the class of* $\mathbf{NC^0}$ *circuits in which every output bit depends on at most* k *input bits. We say that a language belongs to the class* $\mathbf{NC^0_k}$ *if for every* $n \in \mathbb{N}$, *there is an* $\mathbf{NC^0_k}$ *circuit that decides* $L^n = L \cap \{0,1\}^n$.

Note that if a language is in $\mathbf{NC^0_k}$ then its characteristic function (at every input length) is influenced by at most k variables.

THEOREM 3.13. *Let* $L \subseteq \{0,1\}^*$ *be an arbitrary language, then* L *does not have a public-coin interactive protocol with an* $\mathbf{NC^0}$ *verifier, unless* L *is in* $\mathbf{NC^0}$.

PROOF. (sketch) Suppose L is not in $\mathbf{NC^0_k}$ for any constant k and yet it has a public-coin protocol with an $\mathbf{NC^0}$ verifier. In particular, this means that the verifier decides whether to accept its input using an $\mathbf{NC^0}$ circuit that runs on its input, randomness and the transcript. The number of input bits that influence the verifier's decision is constant. Let k be the overall number of input bits that influence the verifier's decision bit. Let n be an input length for which L^n does not have an $\mathbf{NC^0_k}$ circuit. Let $x_1, x_2 \in \{0,1\}^n$ be such that the k bits that the verifier reads are the same in x_1 and x_2, yet $x_1 \in L$ and $x_2 \notin L$. By the fact that L^n does not have an $\mathbf{NC^0_k}$ circuit (and hence its characteristic function is influenced by more than k variables), such a pair of instances exist. Consider the dishonest prover P^*, that on input x_2, for any verifier randomness, plays the strategy of the honest prover on input x_1. Because the protocol is public-coin, the verifier's view in both cases is exactly the same, i.e. for any verifier randomness, the prover's messages on inputs x_1 and x_2 are identical, and thus the bits that the verifier uses to make its decision are also identical. By the protocol's completeness on x_1, the soundness of the protocol on x_2 is violated and we get a contradiction. □

Next we state our negative result regarding sub-constant soundness (the proof is omitted).

THEOREM 3.14. *Let* $L \subseteq \{0,1\}^*$ *be an arbitrary language, then* L *does not have a constant-round interactive protocol with sub-constant soundness and an* $\mathbf{NC^0}$ *verifier, unless* L *is in* $\mathbf{NC^0}$.

Discussion.

At first glance, it may seem that the proof of Theorem 3.13 should also rule out protocols with private coins (at least for constant-round protocols). We want to explain why this is not the case. We believe that this explanation sheds some interesting light on public versus private coins in the context of $\mathbf{NC^0}$ verifiers, and specifically on our protocol (from Lemma 3.5). The idea in the proof of Theorem 3.13 is that we can let the prover choose its strategy regardless of the input. And then we can argue that since the verifier reads only a constant number of bits from the input before he makes his decision, we can change one input with another without the verifier noticing the change. This cannot be done when the verifier has private coins. Now the prover cannot decide on an arbitrary strategy, because it does not

know the private coins of the verifier (i.e. different inputs with the same randomness will not give the same view anymore). This means that if we design our protocol properly, we can force the prover's bits to depend on the entire input. In this case, the decision bit also depends on the entire input via the prover's messages. I.e. even though the decision bit depends on a constant number of prover's bits, each one of them may depend on the entire input. We therefore cannot replace the input without the verifier's noticing the change.

To see how this works in practice, consider the protocol we give in the proof of Lemma 3.5. The last message of the prover contains only a constant number of bits. Let $i \in [\ell]$ be such that $v_i = 1$, and consider the prover's bit b_i. This bit depends on the entire input via the instance c_i that was generated by applying the random instance reduction on the instance $a = (T, x, t, r)$. The protocol, by using private coins, forces the prover to give the correct answer on c_i. The dependency of b_i on the input is then revealed to the verifier by computing $E(b_i, \tau_i)$.

Moving to our result about sub-constant soundness, we want to point out that if we allow a non-constant number of rounds, we can achieve sub-constant soundness. In fact with an addition of $O(\log n)$ rounds we can achieve the soundness of the original protocol (which can be as small as 2^{-n}). This is because we can spread the $\mathbf{AC^0}$ computation at the decision stage of the simple proof system, over $O(\log n)$ rounds of the protocol. That is, consider the $\mathbf{NC^1}$ circuit that computes this $\mathbf{AC^0}$ computation. The $\mathbf{NC^0}$ verifier computes at each round another level of the $\mathbf{NC^1}$ circuit. it sends the prover the results of the computation. The prover sends a dummy message, and the verifier continues the computation by reading from the transcript the results from the previous layer of the circuit.

A key point in our results is that we only add a constant number of rounds to obtain an $\mathbf{NC^0}$ verifier. This is what allows us to obtain constant parallel time proof systems for **AM** and **NEXP**.

4. ERROR-CORRECTING CODES

4.1 Preliminaries

We view error-correcting codes as functions mapping M symbols (from some finite alphabet Σ) to $N = N(M)$ symbols. In our definitions and theorems we consider an error-correcting code to be an infinite family of such functions, one for each $M \in \mathbb{N}$. However, to ease the presentation, our definitions and statements consider a single function from the family. Thus when we talk in a definition or a statement about a code $C : \Sigma^M \to \Sigma^N$, we actually mean a family of functions $\{C_M : \Sigma^M \to \Sigma^N\}_{M \in \mathbb{N}}$. Similarly, when we associate circuits with codes (say the decoder circuit), we talk about a single circuit meaning a family of circuits. We use $\Delta(x, y)$ to denote the *fractional* Hamming distance between two strings x and y (over some finite alphabet).

DEFINITION 4.1. *[locally decodable codes] We say that a code $C : \Sigma^M \to \Sigma^N$ is an explicit locally-decodable code from distance $0 \le \delta \le 1$ if the following holds:*

1. *C is explicit in the sense that for every $x \in \Sigma^M$ and $i \in [N]$, $C(x)_i$ is computable in time* $\mathrm{poly}(M)$.
2. *There is a probabilistic oracle algorithm* DEC *that runs in time* $\mathrm{poly}(\log N)$*, such that for every $x \in \Sigma^M$, given oracle access to a string $y \in \Sigma^N$ satistfying $\Delta(y, C(x)) \le \delta$, for every $i \in [M]$,* $\mathrm{DEC}^y(i)$ *computes x_i correctly with probability at least 3/4 (over its coins).*

If DEC *makes all the queries to y in parallel we say that it is non-adaptive.*

We often restrict the complexity of DEC. For example, we may say that C is locally-decodable (with the specified parameters) by $\mathbf{AC^0}$ circuits (or $\mathbf{NC^1}$ circuits etc.), meaning that DEC is computable by probabilistic $\mathbf{AC^0}$ circuits of size $\mathrm{poly}(\log N)$.

The next definition follows the formulation of Sudan, Trevisan and Vadhan [23] (with some small modifications).

DEFINITION 4.2. *[locally list-decodable codes] We say that a code $C : \Sigma^M \to \Sigma^N$ is an explicit locally list-decodable code from agreement ε and with list size ℓ (for $\varepsilon = \varepsilon(N) \in (0,1)$ and $\ell = \ell(N, \varepsilon) \in \mathbb{N}$) if it is explicit in the sense of Definition 4.1 and the following holds: there is a probabilistic oracle algorithm* DEC *that runs in time* $\mathrm{poly}(\log(N)/\varepsilon)$ *such that: for every $x \in \Sigma^M$, given oracle access to a string $y \in \Sigma^N$ that satisfies $\Delta(y, C(x)) \le 1 - \varepsilon$,* DEC^y *outputs a list of probabilistic oracle machines $M_1, \ldots, M_\ell$ such that, with probability at least 3/4, there exists $j \in [\ell]$ such that for every $i \in [M]$, $M_j^y(i)$ computes x_i correctly with probability at least 3/4 (over M_j's coin tosses). Furthermore, each M_j runs in probabilistic time* $\mathrm{poly}(\log(N)/\varepsilon)$.

If DEC *and the M_j's make all their queries to y in parallel we say that the decoding procedure is non-adaptive.*

As before, we often restrict the complexity of DEC and of the M_j's. And so we say, for example, that C is locally list-decodable (with the specified parameters) by $\mathbf{AC^0}$ circuits, meaning that both DEC and the M_j's are computable by probabilistic $\mathbf{AC^0}$ circuits of size $\mathrm{poly}(\log(N)/\varepsilon)$.

4.2 Unique decoding

In this section we show how to transform a locally (and uniquely) decodable code that has an $\mathbf{NC^1}$ decoder to a locally decodable code (with almost the same parameters) that has an $\mathbf{AC^0}$ decoder.

THEOREM 4.3. *Let $C : \{0,1\}^M \to \{0,1\}^N$ be an explicit locally-decodable binary code that can be non-adaptively decoded from some constant distance $\delta < 1/4$ by a probabilistic $\mathbf{NC^1}$ circuit of size* $\mathrm{polylog}(N)$*. Then there is an explicit binary code $C' : \{0,1\}^M \to \{0,1\}^{2N}$ that is locally-decodable from distance $\delta/2$ by a probabilistic $\mathbf{AC^0}$ circuit of size* $\mathrm{polylog}(N)$.

Intuition.

This application of our general approach is the simplest one. The idea is that the new encoding of $x \in \{0,1\}^M$ has two (equally long) parts appended together. The first part is the original encoding of x and the second part is the truth-table of the randomized image I of the $\mathbf{NC^1}$-complete language L (given by Lemma 2.5) on instances of length $\log N$. Given a word that is close enough to a codeword in the new code, we know that both parts are close to what they should be. We then simulate in $\mathbf{AC^0}$ the original $\mathbf{NC^1}$ decoder (on the first half of the received word) as follows: every time we need to compute something (in $\mathbf{NC^1}$) that we cannot

do in $\mathbf{AC^0}$, we reduce this computation to an instance y of the language L. We then use the random instance reduction from L to I (computable in $\mathbf{NC^0}$) to compute this y by querying the truth-table of I (i.e. the second half of the received word) at a random location. Since the second half is a string that is close to the truth-table of I, with relatively high probability we compute $L(y)$ correctly. We can increase the success probability (in $\mathbf{AC^0}$) by repeating many times in parallel and taking the approximate majority of the answers (this can be done in $\mathbf{AC^0}$ by [2, 1]). Thus with high probability, the new $\mathbf{AC^0}$ decoder decodes the first half exactly as the old $\mathbf{NC^1}$ decoder does.

A subtle issue that we need to deal with is the fact that the I-instances we reduce to need to be of length exactly $\log N$ (so that the two parts of the code are equally long). To that end we use the strong downwards self reducibility property of L (see Definition 2.4 and Lemma 2.5) to adjust the lengths of the instances we work with.

4.3 List decoding

In this section we show how to transform a locally list-decodable code that has an $\mathbf{NC^1}$ decoder to a locally list-decodable code that has an $\mathbf{AC^0}$ decoder.

THEOREM 4.4. *Let Σ be a finite alphabet and let $C : \Sigma^M \to \Sigma^N$ be an explicit code that is non-adaptively locally list-decodable by probabilistic $\mathbf{NC^1}$ circuits from agreement ε and with list size ℓ. Then there is an explicit code $C' : \Sigma^M \to \Gamma^{N'}$ that is locally list-decodable by probabilistic $\mathbf{AC^0}$ circuits, from agreement ε and with list size 2ℓ, where $|\Gamma| = |\Sigma| \cdot O(1/\varepsilon)$, and $N' = \max\{N, 2^{(\log(N)/\varepsilon)^\delta}\}$, for arbitrarily small constant $\delta > 0$. In particular, for $\varepsilon \geq 1/\mathrm{poly}(\log N)$, we obtain $N' = N$.*

Intuition.

The approach that we used to prove Theorem 4.3 cannot be used to recover from distance more than 1/4 (even if we start with a code that can be list-decoded from a large distance) because we need that the truth-table part of the word will be more than 1/2 close to the respective half in the codeword. One thing we can do to improve the distance that we can recover from, at the price of doubling the alphabet size, is to append the truth-table *componentwise*. That is, we append to each bit of the original codeword an entry of the truth-table (recall that they are of the same length). This allows us to list-decode from distance $1/2 - 1/\log\log(N)$ (assuming the original code has a locally list-decoder in $\mathbf{NC^1}$), and the alphabet size is 4 (two bits per symbol). However, to recover from distance more than 1/2 requires a different technique. The idea now is to append (again, componentwise) the *direct-product* of I's truth table (I is, as before, the randomized image of the $\mathbf{NC^1}$-complete language L from Lemma 2.5). That is, every symbol in the new encoding contains a symbol from the old encoding concatenated with the binary string $I(i_1), \ldots, I(i_s)$, where $(i_1, \ldots, i_s)$ is a tuple of binary strings of some length that is determined in the proof. For every possible tuple we will have a different entry in the codeword.

As in the proof of Theorem 4.3, the new decoder simulates the old decoder. When it needs to compute some $\mathbf{NC^1}$ computation, it creates a uniformly distributed s-tuple where in each entry it either (with probability 1/2) puts a random instance for which it knows its correct membership status in I (this can be done by using the solved instance generator for I, see Definition 2.1), or (with probability 1/2) puts a random instance from which it can conclude the value of the $\mathbf{NC^1}$ computation given the correct membership status of that instance in I (this can be done using the random instance reduction from L to I and the fact that L in $\mathbf{NC^1}$-complete). The new decoder now reads from the corresponding location in the received word the (possibly corrupted) membership status of every instance in the s-tuple. It then checks whether on the entries for which it knows the correct answer, the received word is correct. If not, it declares this location in the received word to be corrupted. Otherwise it is a good indication that the symbol is not corrupted and it assumes that the other values it reads from it are correct and it infers from them the correct value of the $\mathbf{NC^1}$ computation. The success probability of this procedure can be significantly improved by using a careful amplification argument. Given this procedure we can continue the simulation as in the proof of Theorem 4.3.

A subtle issue is the fact that the length of the original codeword and the length of the extra information we append to it (i.e. the number of s-tuples) are not necessarily the same. To solve this, we write many copies of the original codeword in the new one, so that the repeated original codeword is of the same length as the number of s-tuples.

4.4 Explicit constructions

In this section we apply our general theorems from the previous sections to construct codes with local-decoders in $\mathbf{AC^0}$. We do that by first constructing codes with $\mathbf{NC^1}$ decoders and then applying the general transformations to them. Previous locally-decodable codes with the parameters that we need are not known to be in $\mathbf{NC^1}$ (in particular decoding the code given in [23] involves solving a system of linear equations). We therefore construct new explicit codes with $\mathbf{NC^1}$ decoders, and then apply our transformations to them.

THEOREM 4.5. *There is an explicit code $C : \{0,1\}^M \to \{0,1\}^{\mathrm{poly}(M)}$ that can be locally decoded from distance 1/25 by probabilistic $\mathbf{AC^0}$ circuits of size $\mathrm{poly}(\log M)$.*

THEOREM 4.6. *For every $\varepsilon > 0$, there is an explicit code $C : \{0,1\}^M \to \Sigma^N$ that is locally list-decodable by probabilistic $\mathbf{AC^0}$ circuits of size $\mathrm{poly}(\log M/\varepsilon)$ from agreement ε and with list size $\mathrm{poly}(1/\varepsilon)$. Where $|\Sigma| = 2^{\mathrm{poly}(1/\varepsilon)}$, and $N = M^{\mathrm{poly}(1/\varepsilon)}$.*

4.5 Worst-case to average-case reductions

By using our efficiently locally list-decodable codes, together with ideas from [23, 25], we obtain the following worst-case to average-case reduction for **EXP** languages (which we state in terms of lower-bounds).

THEOREM 4.7. *Let $\mathcal{C}$ be a class of algorithms (or Boolean circuits) containing probabilistic uniform $\mathbf{AC^0}$. Then for every **EXP** function $f : \{0,1\}^* \to \{0,1\}$, there is an **EXP** function $\hat{f} : \{0,1\}^* \to \{0,1\}$ such that: for every large enough m, if there is no algorithm (or family of circuits) in the class $\mathcal{C}$ that computes f at length m correctly in the worst-case, then there is no algorithm (or family of circuits) in the class $\mathcal{C}$ that can compute $\hat{f}$ at length $n = \mathrm{poly}(m)$ correctly on at least a $1/2 + 1/(\log n)^\alpha$ fraction of the inputs, where $\alpha > 0$ is a universal constant.*

5. ACKNOWLEDGEMENTS

We thank Salil Vadhan for many illuminating discussions, and for his skepticism which resulted in proofs of lower bounds. Thanks also to Emanuele Viola, for helpful discussions on locally decodable codes, and to Valentine Kabanets for his kind assistance.

6. ADDITIONAL AUTHORS

7. REFERENCES

[1] M. Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances in computational complexity theory*, pages 1–20. Amererican Mathematical Society, 1993.

[2] M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computation. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 471–474, 1984.

[3] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 2004. To appear. Preliminary version in FOCS 2004.

[4] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[5] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 16–25, 1990.

[6] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential simulation unless Exptime has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[7] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[8] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38, 1989.

[9] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[10] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[11] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 554–563, 1994.

[12] U. Feige and L. Lovasz. Two-prover one-round proof systems, their power and their problems. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 733–744, 1992.

[13] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[14] S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. Rothblum. Designing efficient program checkers by delegating their work. Manuscript, 2006.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[16] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *volume 5 of Advances in Computing Research*, pages 73–90, 1989.

[17] R. Impagliazzo, R. Jaiswal, and V. Kabanets. Approximately list-decoding direct product codes and uniform hardness amplification. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 2006.

[18] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[19] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *In proceedins of 29th ICALP*, pages 244–256, 2002.

[20] J. Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[21] R. Lipton. New directions in testing. *Proceedings of DIMACS workshop on distributed computing and cryptography*, 2:191–202, 1991.

[22] A. Shamir. IP = PSPACE. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 11–15, 1990.

[23] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[24] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[25] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 129–138, 2002.

[26] S. Vadhan. Personal communication, 2006.