Delegating Computation: Interactive Proofs for Muggles

Shafi Goldwasser [*]
MIT and Weizmann Institute
shafi@theory.csail.mit.edu

Yael Tauman Kalai [†]
Georgia Tech
yael@cc.gatech.edu.

Guy N. Rothblum [‡]
MIT
rothblum@csail.mit.edu

ABSTRACT

In this work we study interactive proofs for tractable languages. The (honest) prover should be efficient and run in polynomial time, or in other words a "muggle".[1] The verifier should be super-efficient and run in nearly-linear time. These proof systems can be used for delegating computation: a server can run a computation for a client and interactively prove the correctness of the result. The client can verify the result's correctness in nearly-linear time (instead of running the entire computation itself).

Previously, related questions were considered in the Holographic Proof setting by Babai, Fortnow, Levin and Szegedy, in the argument setting under computational assumptions by Kilian, and in the random oracle model by Micali. Our focus, however, is on the original interactive proof model where no assumptions are made on the computational power or adaptiveness of dishonest provers.

Our main technical theorem gives a public coin interactive proof for any language computable by a log-space uniform boolean circuit with depth d and input length n. The verifier runs in time $(n+d)\cdot\text{polylog}(n)$ and space $O(\log(n))$, the communication complexity is $d\cdot\text{polylog}(n)$, and the prover runs in time $\text{poly}(n)$. In particular, for languages computable by log-space uniform $\mathcal{NC}$ (circuits of $\text{polylog}(n)$ depth), the prover is efficient, the verifier runs in time $n\cdot\text{polylog}(n)$ and space $O(\log(n))$, and the communication complexity is $\text{polylog}(n)$.

Using this theorem we make progress on several questions:

- We show how to construct short (polylog size) computationally sound *non-interactive* certificates of correctness for any log-space uniform $\mathcal{NC}$ computation, in the public-key model. The certificates can be verified in quasi-linear time and are for a designated verifier: each certificate is tailored to the verifier's public key. This result uses a recent transformation of Kalai and Raz from public-coin interactive proofs to one-round *arguments*. The soundness of the certificates is based on the existence of a PIR scheme with polylog communication.
- Interactive proofs with *public-coin*, log-space, poly-time verifiers for all of $\mathcal{P}$. This settles an open question regarding the expressive power of proof systems with such verifiers.
- Zero-knowledge interactive proofs with communication complexity that is quasi-linear in the *witness* length for any $\mathcal{NP}$ language verifiable in $\mathcal{NC}$, based on the existence of one-way functions.
- Probabilistically checkable arguments (a model due to Kalai and Raz) of size polynomial in the *witness* length (rather than the instance length) for any $\mathcal{NP}$ language verifiable in $\mathcal{NC}$, under computational assumptions.

Categories and Subject Descriptors

F.2.0 [**Theory of Computation**]: Analysis of Algorithms and problem complexity—*General*

General Terms

Theory

Keywords

Proof Verification, Interactive Proofs, Delegating Computation

1. INTRODUCTION

Efficient proof verification lies at the heart of complexity theory. Classically, this was captured by the idea of a deterministic polynomial time verification procedure which receives the proof, a certificate of polynomial length, and verifies its validity. Extending classical proof systems, interactive proof systems [37, 9] provide a model in which the

[*]Supported by NSF Grants CCF-0514167, CCF-0635297, NSF-0729011, the RSA chair, and by the Weizmann Chais Fellows Program for New Scientists.

[†]This work was mainly done while the author was visiting Microsoft Research, Redmond. Supported in part by NSF grant CCF-0635297

[‡]Supported by NSF Grants CCF-0635297, NSF-0729011, CNS-0430336 and by a Symantec Graduate Fellowship.

[1] *In the fiction of J. K. Rowling: a person who possesses no magical powers*; from the Oxford English Dictionary.

STOC'08, May 17–20, 2008, Victoria, British Columbia, Canada.

polynomial time verification procedure (the verifier) is randomized and can interact with a prover that may employ an adaptive strategy.

Much of the complexity-theoretic research on interactive proofs has focused on studying their expressive power while improving the verifier's complexity in terms of various resource measures (e.g. time, space, depth, rounds or randomness). The complexity of proving has received less attention. Indeed, since research focused on proofs for intractable languages, the honest prover is often[2] assumed to be able to perform intractable computations in the interest of efficient verifiability. In Arthur-Merlin games, the honest prover is accordingly named after Merlin, a computationally unbounded magician.

Even in cryptographic contexts, where parties in interactive proof systems must run in polynomial time, the main focus of research remains on intractable languages such as deciding quadratic-residuosity modulo a composite number. To allow the honest prover to perform computations otherwise impossible in polynomial time, he can use auxiliary secrets, e.g. the factorization of the input modulos in the quadratic residuosity example. This model is reasonable in protocol settings where the input is generated by the prover himself. The prover can generate the input along with an auxiliary secret which enables him prove non-$\mathcal{BPP}$ properties. However, in settings where the input is generated by an external source, an efficient prover does not have access to auxiliary information about the input.

Our Setting and Goal. In this paper, we embark on the study of interactive proofs in the real world, where both the verifier *and the prover* are efficient. Thus, we replace the unbounded magical Merlin with a "Muggle" prover who is limited to running probabilistic polynomial-time computations. We think of the input to the interactive proof as dictated by an outside source, possibly even by the verifier. Thus, the prover has no auxiliary information to help him in the proving task.

Clearly, if both the prover and the verifier are efficient, then the language is tractable (in $\mathcal{BPP}$). This may seem puzzling at first glance, since usually one allows the verifier to run *arbitrary* polynomial-time computations, and thus it could compute on its own whether or not the input is in the language! This obviously is not very interesting. Indeed, we want verification to be considerably faster than computing.

The question we ask in this work is which *polynomial-time computable* languages have interactive proofs with a *super-efficient verifier* and an *efficient prover*. We emphasize that although we aim for the *honest* prover to be efficient, we still require the soundness of the proof system to hold unconditionally. Namely, we make no assumptions on the computational power of a dishonest prover. Specifically, let L be an efficiently computable language. We seek an interactive proof system that simultaneously achieves:

- *Verifier* time complexity that is *linear* in the size of the input x and poly-logarithmic in the size of the computation of L. More generally, we ask that the verifier run in time and space that are considerably smaller than those required to compute the language.
- *Prover* time complexity that is polynomial in the size of the input.
- *Communication complexity* that is polylogarithmic in the size of the computation.

Delegating Computation. Beyond its complexity theoretic interest, the question of interactive proofs for efficient players is motivated by real-world applications. The main application we consider is delegating polynomial time computations to an untrusted party. The general setting is of several computational devices of differing computational abilities interacting with each other over a network. Some of these devices are computationally weak due to various resource constraints. As a consequence there are tasks, which potentially could enlarge a device's range of application, that are beyond its reach. A natural solution is to *delegate* computations that are too expensive for one device, to other devices which are more powerful or numerous and connected to the same network. This approach comes up naturally in today's and tomorrow's computing reality as illustrated in the following two examples.

1. Large Scale Distributed Computing. The idea of *Volunteer Computing* is for a server to split large computations into small units, send these units to volunteers for processing, and reassemble the result (via a much easier computation). The Berkeley Open Infrastructure for Network Computing (BOINC) [5, 6] is such a platform whose intent is to make it possible for researchers in fields as diverse as physics, biology and mathematics to tap into the enormous processing power of personal computers around the world. A famous project using the BOINC platform is SETI@home [3, 1], where large chunks of radio transmission data are scanned for signs of extraterrestrial intelligence. Anyone can participate by running a free program that downloads and analyzes radio telescope data. Thus, getting many computers to pitch into the larger task of scanning space for the existence of extraterrestrial intelligence, and getting people interested in science at the same time. Another example of a similar flavor is the Great Internet Mersenne Prime Search [2], where volunteers search for Mersenne prime numbers and communicate

2. Weak Peripheral Devices. More and more, small or cheap computational devices with limited computational capabilities, such as cell-phones, printers, cameras, security access-cards, music players, and sensors, are connected via networks to stronger remote computers whose help they can use. Consider, for example, a sensor that is presented with an access-card, sends it a random challenge, and receives a digital signature of the random challenge. The computation required to verify the signature involves public-key operations which are too expensive both in time and space for the sensor to run. Instead, it could interact with a remote mainframe (delegatee), which can do the computation.

The fundamental problem that arises is: *how can a delegator verify that the delegatees performed the computation correctly, without running the computation itself?* For example, in the volunteer computing setting, an adversarial volunteer may introduce errors into the computation, by claiming that a chunk of radio transmissions contains no signs of extraterrestrial intelligence. In the Mersenne Prime search example, an adversary may claim that a given range of numbers *does not* contain a Mersenne prime. Or in the sensor example, the communication channel between the main-frame and the sensor may be corrupted by an adversary.

All would be well if the delegatee could provide the delegator with a proof that the computation was performed

[2]We note that there are important exceptions to the above, e.g. the work of Beigel, Bellare, Feigenbaum and Goldwasser [13] on competitive proof systems.

correctly. The challenge is that for the whole idea to pay off, it is *essential* that the time to verify such a proof of correctness be significantly smaller than the time needed to run the entire computation.[3] At the same time, the delegatee should not invest more than a reasonable amount of time in this endeavor. Interactive proofs with efficient provers (the delegatees) and super-efficient verifiers (the delegators) provide a natural solution to this problem.[4]

Prior Work in Other Models. Variants of the question of checking the correctness of computations have been studied previously in several settings *outside* of the interactive proof model. These studies include the work of Babai, Fortnow, Levin, and Szegedy [10] in the Holographic Proofs model (or alternatively the PCP model), and the works of Micali [50] and Kilian [45] on computationally sound argument systems. These works raise similar goals to ours, but in their respective models, requiring super-efficient verifiability, and efficient provability (polynomial time in the non-deterministic time complexity of accepting the input). We elaborate on these seminal related works in Section 1.1. In contrast, our work is in the standard interactive proof model [37] where soundness is achieved unconditionally, making no assumptions on the power of the dishonest prover (as in [50, 45]), nor making assumptions on the non-adaptivity of the dishonest prover(as in [10]).

Roadmap. Our main result is described in Section 1.1 We further use our techniques to obtain several other results: Constructing computationally-sound succinct certificates of correctness for any ($\mathcal{L}$-uniform) $\mathcal{NC}$ computation, under computational assumptions (Section 1.2); Characterizing public-coin log-space interactive proofs (Section 1.3); Constructing low communication zero-knowledge proofs (Section 1.5); Constructing IPCP and PCA proof-systems, improving on [41, 42] (Section 1.6). A high-level overview of our techniques is given Section 2, and a protocol overview is given in Section 3. All details are deferred to the full version of this paper due to lack of space.

1.1 Our Main Result

Our most general result is a public-coin interactive proof for any language computable by an $\mathcal{L}$-uniform family of Boolean circuits[5], where the communication complexity is polynomial in the *depth* of the computation rather than its *size*; the running time of the verifier is *linear in the input* and polynomial in the depth; and *the prover is efficient.*

THEOREM 1. *Let L be a language that can be computed by a family of $O(\log(S(n)))$-space uniform boolean circuits of size $S(n)$ and depth $d(n)$.*

Then, L has a public-coin interactive proof with perfect completeness and soundness $1/2$ where: the prover's running time is $\text{poly}(S(n))$; the verifier runs in time $(n + d(n)) \cdot \text{polylog}(S(n))$ and space $O(\log(S(n)))$; the communication complexity is $d(n) \cdot \text{polylog}(S(n))$.

An overview of the proof idea is given in Sections 2 and 3. The interactive proofs constructed in Theorem 1 provide a natural solution to the delegating computation problem mentioned above. Namely, the *statement to be proved* is that the delegated computation was executed correctly; the *delegator is the verifier* in the interactive proof ; the *delegatee is the prover* in the interactive proof, who convinces the delegatee that he performed the computation correctly (and runs in polynomial time).

As a primary implication, we get that any computation with low *parallel time* (significantly smaller than the computation's total size) has a very efficient interactive proof. In particular, for languages in $\mathcal{L}$-uniform $\mathcal{NC}$, we have:

COROLLARY 1. *Let L be a language in $\mathcal{L}$ uniform $\mathcal{NC}$, i.e. computable by a family of $O(\log(n))$-space uniform circuits of size $\text{poly}(n)$ and depth $\text{polylog}(n)$.*

L has a public-coin interactive proof with perfect completeness and soundness $1/2$: the prover runs in time $\text{poly}(n)$; the verifier runs in time $n \cdot \text{polylog}(n)$ and space $O(\log(n))$; the communication complexity is $\text{polylog}(n)$.

A natural question is how this can be done when the verifier cannot even take the circuit in question as an additional input (it has no time to read it!). This is where the condition on the log-space uniformity of the circuit family comes in. For such circuit families, the circuit has a "short" implicit representation which the verifier can use without ever constructing the entire circuit. We view log-space-uniformity as a relaxed notion of uniformity (though admittedly less relaxed than poly-time-uniformity). In particular, Corollary 1 applies to any language in $\mathcal{NL}$, and even to any language computable by a $PRAM$ in poly-logarithmic parallel time.

Alternatively, by modifying the model (to include an on-line and an off-line stage of computation) we also obtain results for the non-uniform setting. See Sections 1.4 and 2, as well as the full version for details.

Comparison to Prior Work on Interactive Proofs. We emphasize that Theorem 1 improves previous work on interactive proofs, including the works of Lund, Fortnow, Karloff and Nissan [49], Shamir [53], and Fortnow and Lund [33] in terms of the honest prover's running time. In particular, Corollary 1 gives efficient honest provers, whereas the honest provers in previous results run in super-polynomial time (even for log-space languages). Both of the works [49, 53] address complete languages for $\#\mathcal{P}$ and for $\mathcal{PSPACE}$, and thus naturally the honest prover needs to perform non-polynomial time computation (scale-down of the protocols to $\mathcal{P}$ or even to $\mathcal{L}$ retains the non-polynomial time provers). The work of Fortnow and Lund [33], using algebraic methods extending [49, 53], on the other hand, does explicitly address the question of interactive proofs for polynomial time languages and in particular $\mathcal{NC}$. They show how to improve the space complexity of the verifier, in particular achieving log-space and poly-time verifiers for $\mathcal{NC}$ computations. Their protocol, however, has a non-polynomial time prover as in [49, 53].

Our work puts severe restrictions on the sequential runtime (and space) of the verifier. This continues a sequence of works which investigated the power of interactive proofs with weak verifiers (but often with unbounded provers).

[3] With regard to the Mersenne Prime example, we note that current methods for verifying the output of polynomial time deterministic primality tests [4] are not significantly faster than running the test itself.

[4] We note that in practice in BOINC [6], it is suggested to give out each computational task to multiple users, and assume that most of them answer correctly. We will make no such assumptions here.

[5] A circuit family is $s(n)$-space uniform if there exists a Turing Machine that on input 1^n runs in space $O(s(n))$ and outputs the circuit for inputs of length n. A circuit family is $\mathcal{L}$-uniform if it is log-space uniform.

Dwork and Stockmeyer [27, 28] investigated the power of finite state verifiers (with and without zero-knowledge). Condon and Ladner [23], Condon and Lipton [24], and Condon [22] studied space (and time) bounded verifiers. Kilian [43] considered zero-knowledge for space-bounded verifiers. Fortnow and Sipser (see results in [32]) and Fortnow and Lund [33] focused on public-coin restricted space verifiers. A recent work by Goldwasser *et al.* [35] considers the *parallel running time* of a verifier (who can still use a polynomial number of processors and communication complexity), and shows that all of PSPACE can still be recognized by constant-depth ($\mathcal{NC}^0$) verifiers.

Comparison to Prior Work in Other Models. The goal of the work of Babai, Fortnow, Lund and Szegedy [10] on Holographic Proofs for $\mathcal{NP}$ (i.e PCP-proofs where the input is assumed to be given to the verifier in an error-correcting-code format), was to extend Blum and Kannan's program checking model [19] to checking the results of executions (the combination of software and hardware) of long computations. They show how to achieve checking time that is polylogarithmic in the length of the computation (on top of the time taken to convert the input into an error correcting code format), and a proof-string (which can be randomly accessed by the verifier) of length close to the computation time itself. However the soundness of proofs in this PCP like model (as well as its more efficient descendants [52, 16, 17, 25]) *requires* that the verifier/delegator either "posses" the entire PCP proof string (though only a few of its bits are read), or somehow have a guarantee that the prover/delegatee cannot change the PCP proof string after the verifier has started requesting bits of it. Such guarantees seem difficult to achieve over a network as required in the delegation setting.

Kilian [44, 45] gives an argument system for any $\mathcal{NP}$ computation, with communication complexity that is polylogarithmic, and verifier runtime which is linear in the input length (up to polylogarithmic factors). This is achieved by a constant round protocol, in which the prover first constructs a PCP for the correctness of the computation, and then Merkle-hashes it down to a short string and sends it to the verifier. To do this, one must assume the existence of strong collision-intractable hash functions: where collisions cannot be formed in sub-exponential time.[6] We emphasize, that an argument system achieves only computational soundness (soundness with respect to a computationally bounded dishonest prover). In the interactive proof setting soundness is guaranteed against *any* cheating prover.

Finally, Micali raises similar goals to ours in his work on Computationally Sound (CS) proofs [50]. His results are however obtained in the *random oracle model*. This allows him to achieve CS-proofs for the correctness of general time computations with a nearly linear time verifier, a prover whose runtime is polynomial in the time complexity of the computation, and a poly-log length non-interactive ("written down" rather than interactive) proof. Alternatively viewed, Micali's work gets non-interactive CS-proofs under the same assumption as [44], and assuming the existence of Fiat-Shamir-hash-functions [31] to remove interaction. The plausibility of realizing Fiat-Shamir-hash-functions by any explicit function ensemble has been shown to be highly questionable [21, 26, 36].

[6]With standard intractability assumptions, one could get arguments of linear size communication complexity (using Universal arguments [12]).

Finally, we note that all of the above [10, 44, 45, 50] use the full PCP machinery, whereas this is unnecessary to obtain our results.

1.2 Non-Interactive CS Certificates

Throughout the paper until this point we mostly considered interactive settings. We find it very interesting to pursue the question of delegating computation in the **non-interactive** setting as well. One may envision a delegator farming out computations to a computing facility (say by renting computer time at a super-computer facility during the night hours), where the result is later returned via e-mail with a fully written-down "certificate" of correctness.

Thus, we further ask: for which polynomial time computations can a polynomial time prover write down certificates of correctness that are super-efficiently verifiable, and in particular are significantly shorter than the time of computation (otherwise the verifier cannot even receive the certificate!). As discussed above, Micali's CS-proofs result [50] is the only solution known to this problem, and it is in the *random oracle model*.

We address this problem in the *public-key model*. In this model each verifier has a secret and public key pair. A proof (or certificate) is tailored for a specific verifier, in the sense that it depends on the verifier's public key. The verifier, in turn, needs to use his secret key for verification. We note that there is no need for a general public key infrastructure. The keys do not need to be approved or checked. Each verifier simply generates and publishes his public key (say on his web-page) or sends his public-key as a first message to the prover.

We show how to construct computationally sound certificates of correctness for any $\mathcal{L}$-uniform $\mathcal{NC}$ computation in the public key model, assuming the existence of a computational private information retrieval (PIR) scheme with poly(κ)-communication, where κ is the security parameter. We note that such a PIR scheme exists for any $\kappa \geq \log |DB|$ (where $|DB|$ is the database size) under the N-th Residuosity Assumption [48, 40], and under the Φ-Hiding Assumption [20]. For a polynomially small security parameter, such PIR schemes exist under a variety of computational assumptions (see e.g. [46]).

For security parameter κ, the size of the certificates is poly($\kappa, \log n$), the (honest) prover runs in polynomial time, and the verifier runs in time $n \cdot \text{poly}(\kappa, \log n)$ to verify a certificate (as in [50]). Soundness holds only against computationally bounded cheating provers.

Formally, we state the result as a 1-round (2-message) argument system, where the verifier sends his public key to the prover and then receives back the certificate.

THEOREM 2. *Let $\kappa \geq \log n$ be a security parameter. Assume the existence of a PIR scheme with communication complexity* poly(κ) *and receiver work* poly(κ). *Then any language L in $\mathcal{L}$-uniform $\mathcal{NC}$ has a **1-round** (private coin) argument system with the following properties:*
1. *The prover runs in time* poly(n, κ), *the verifier runs in time $n \cdot$ poly(κ).*
2. *The protocol has perfect completeness (assuming the PIR scheme has prefect completeness)*[7] *and computational sound-*

[7]We note that the construction of [20] does not have perfect completeness. Using it would result in a protocol that inherits this imperfect completeness.

ness $\leq 1/2$ *(can be amplified): for any input* $x \notin L$ *and for any cheating prover running in time* $\leq 2^{\kappa^3}$*, the probability that the verifier accepts is* $\leq 1/2$.
3. *The certificate (the prover's message) and the verifier's challenge are of size* $\text{poly}(\kappa, \log n)$*. The verifier's challenge depends only on the parameters* n *and* κ*, and is independent of the language* L *and the input* x.

The idea of the proof is as follows. We apply to the protocol of Corollary 1 a new transformation due to Kalai and Raz in their paper on probabilistically checkable arguments [42]. They use a computational PIR scheme to transform any public-coin interactive proof into a one-round argument system (where the verifier's first and only message can be computed independently and ahead of the input).

More specifically, [42] show how to convert any public-coin interactive proof system $(\mathcal{P}, \mathcal{V})$ (for a language L), with communication complexity ℓ, completeness c, and soundness s, into a one-round (two-message) argument system $(\mathcal{P}', \mathcal{V}')$ (for L), with communication complexity $\text{poly}(\ell, \kappa)$, completeness c, and soundness $s+2^{-\kappa^2}$ against malicious provers of size $\leq 2^{\kappa^3}$, where $\kappa \geq \log n$ is the security parameter. The verifier $\mathcal{V}'$ runs in time $t_{\mathcal{V}} \cdot \text{poly}(\kappa)$, where $t_{\mathcal{V}}$ is $\mathcal{V}$'s running time. The prover $\mathcal{P}'$ runs in time $t_{\mathcal{P}} \cdot \text{poly}(\kappa, 2^{\lambda})$, where $t_{\mathcal{P}}$ is $\mathcal{P}$'s running time, and λ satisfies that each message sent by the prover $\mathcal{P}$ depends only on the λ previous bits sent by $\mathcal{V}$.

Note that if λ is super-logarithmic, then the resulting prover $\mathcal{P}'$ is inefficient. Fortunately, the protocol of Corollary 1 has the property that $\lambda = O(\log n)$, and thus the resulting $\mathcal{P}'$ is efficient.[8]

As discussed above, the resulting one-round argument system $(\mathcal{P}', \mathcal{V}')$ has the property that the first message, sent by $\mathcal{V}'$, depends only on the random coin tosses of $\mathcal{V}'$ (and is independent of the language L and the instance x), and can be computed in time $\text{poly}(\kappa)$. Thus, we can think of this message as a public key, associated with the verifier. For further details, see the full version.

1.3 Public-Coin Log-Space Verifiers

Theorem 1 as presented above, leads to the resolution of an open problem on characterizing public-coin interactive proofs for log-space verifiers.

The power of interactive proof systems with a log-space verifier has received significant attention (see Condon [22] and Fortnow and Sipser [32]). It was shown that any language that has a public-coin interactive proof with a log-space verifier is in $\mathcal{P}$. Fortnow and Sipser [32] showed that such proof systems exist for the class LOGCFL. Fortnow and Lund [33] improved this result, showing such protocols for any language in $\mathcal{NC}$. In fact, for the class $\mathcal{P}$, [33] achieve $\frac{\log^2(n)}{\log\log(n)}$-space verifiers.

We resolve this question. As a corollary of Theorem 1, and using the fact that languages in $\mathcal{P}$ have $\mathcal{L}$-uniform poly-size circuits, we show the following theorem (see the full version for the proof and details):

THEOREM 3. *Let* L *be a language in* $\mathcal{P}$*, i.e. one that can be computed by a deterministic Turing machine in time* $\text{poly}(n)$.

Then, L *has a public-coin interactive proof with perfect completeness and soundness* $1/2$*, where: The prover runs in time* $\text{poly}(n)$*; the verifier runs in time* $\text{poly}(n)$ *and space* $O(\log(n))$*; the communication complexity of the protocol is* $\text{poly}(n)$.

1.4 Non-Uniform Circuit Families

We also obtain results for non-uniform circuits. In the non-uniform setting the verifier must read the entire circuit, which is as expensive as carrying out the computation. Thus, we separate the verification into an off-line (non-interactive) pre-processing phase, and an on-line interactive proof phase. In the *off-line phase*, before the input x is specified, the verifier is allowed to run in $\text{poly}(S)$ time, but retains only $\text{poly}(d, \log(S))$ bits of information about C (where d is the depth and S is the size of the circuit C). These bits are retained for the *on-line* interactive proof phase, where the verifier gets the input x and interacts with the prover who tries to prove $C(x) = 1$. We note that the information computed in the off-line phase can only be used one time, if it is used twice (even for different inputs) then soundness is compromised.

THEOREM 4. *Let* $C : \{0,1\}^n \to \{0,1\}$ *be a boolean circuit of size* S *and depth* d.

There exists an on-line/off-line interactive proof, where both parties take an input x*, and the prover proves that* $C(x) = 1$*. The protocol has completeness 1 and soundness* $\frac{1}{2}$*. The (honest) prover runs in time* $\text{poly}(S)$ *and communication complexity* $\text{poly}(d, log(S))$.
The verifier runs in two phases: (1) an off-line phase before the input x *is specified, where the verifier runs in time* $\text{poly}(S)$*; (2) an on-line phase which is carried out after the input* x *is specified, where the verifier runs in time* $n \cdot poly(d, \log(S))$.

1.5 Short Zero Knowledge Proofs

Aside from the primary interest (to us) of delegating computation, Theorem 1 above, and more importantly the techniques used, enables us to improve previous results on communication efficient zero-knowledge interactive proofs. The literature on zero-knowledge interactive proofs and interactive arguments for $\mathcal{NP}$ is immense. In this setting we have an $\mathcal{NP}$ relation R which takes as input an n-bit instance x and a k-bit witness w. A prover (who knows both x and w) wants to convince a verifier (who knows only x and *does not* know w) in zero-knowledge that $R(x, w) = 1$.

Recently, attention has shifted to constructing zero knowledge interactive proofs with communication complexity that is polynomial or even linear in the length of the witness w, rather than in R's worst case running time, as in traditional zero-knowledge proofs [34, 18].

Working towards this goal, Ishai, Kushilevitz, Ostrovsky, and Sahai [39] showed that if one-way functions exist, then for any $\mathcal{NP}$ relation R that can be verified by an $\mathcal{AC}^0$ circuit (i.e., a constant-depth circuit of unbounded fan-in), there is a zero-knowledge interactive proof with communication complexity $k \cdot \text{poly}(\kappa, \log(n))$, where κ is a security parameter.[9]

[8] Other interactive proofs, such as those in [49, 53, 41], do not yield efficient provers.

[9] A similar result (with slightly higher communication: $\text{poly}(k, \kappa, \log(n))$) was obtained independently by Kalai and Raz [41].

We improve the results of [39, 41] significantly. We enlarge the set of languages that have zero-knowledge proofs with communication complexity quasi-linear in the witness size, from relations R which can be verified by $\mathcal{AC}^0$ circuits (constant depth) to relations R which can be verified by $\mathcal{NC}$ (polylog depth) circuits. More generally, we relate the communication complexity to the *depth* of the relation R:

THEOREM 5. *Assume one-way functions exist, and let $\kappa = \kappa(n) \geq \log(n)$ be a security parameter. Let L be an $\mathcal{NP}$ language whose relation R can be computed on inputs of length n with witnesses of length $k = k(n)$ by Boolean circuits of size* $\text{poly}(n)$ *and depth $d(n)$.*

Then L has a zero-knowledge interactive proof with perfect completeness, soundness $\frac{1}{2}$ (can be amplified), and communication complexity $k \cdot poly(\kappa, d(n))$; the run time of the prover (given a witness) is $\text{poly}(n)$.

In particular, for relations R in $\mathcal{NC}$, the protocol of Theorem 5 matches the communication complexity achieved by [39] for $\mathcal{AC}^0$; i.e., the communication complexity is quasi-linear in the witness length.

From an application point of view, enlarging the set of communication efficient protocols from relations verifiable in $\mathcal{AC}^0$ to relations verifiable in $\mathcal{NC}$, is significant. Many typical statements that one wants to prove in zero knowledge involve proving the correctness of cryptographic operations in zero knowledge, such as "The following is the result of proper decryption" or "The following is a result of a pseudo random function". Many such operations are generally not implementable in $\mathcal{AC}^0$ (see [47]), but can often be done in $\mathcal{NC}$.

The idea behind this theorem is to use our public-coin interactive protocol from Theorem 1, and carefully apply to it the (standard) transformation from public-coin interactive proofs to zero knowledge interactive proofs of [14]. This is done using statistically binding commitments, which can be implemented using one-way functions [51, 38]. Details are in the full version.

For languages in $\mathcal{NC}$ that are $\mathcal{L}$-uniform, the verifier in our zero knowledge proof runs in time that is quasi-linear in the input size. The works of [39, 41] mentioned above on zero knowledge interactive proofs for $\mathcal{AC}^0$ computable relations do not address (nor do they achieve) improvements in the verifier's computation time.

We note that in the setting of arguments and computational soundness, it is known by [44] how to obtain asymptotically very efficient zero-knowledge argument systems with polylogarithmic communication complexity for all of $\mathcal{NP}$. Besides the weaker soundness guarantees, those results require assuming collision-resistant hashing (we assume only one-way functions), and use the full PCP machinery.

1.6 Results on IPCP and PCA

Building on our interactive proofs, we show constructions, with better parameters and novel features, of two new proof systems introduced by Kalai and Raz [41, 42].

Low communication and short Interactive PCP. In [41] Kalai and Raz proposed the notion of an interactive PCP (IPCP): a proof system in which a polynomial time verifier has access to a proof-string (a la PCP) as well as an interactive prover. When an $\mathcal{NP}$ relation R is implementable by a constant-depth circuit (i.e., $R \in AC^0$) they show an IPCP for R with polylog query complexity, where the proof-string is of size polynomial in the length of the witness to R (rather than the size of R) and an interactive phase of communication complexity $\text{polylog}(n)$. We extend this result to $\mathcal{NP}$ relations implementable by poly-size circuits of depth d. Namely, we demonstrate an IPCP with a proof-string of length polynomial in the length of the witness and an interactive phase of communication complexity $\text{poly}(\log n, d)$. In particular, this extends the results of [41] from relations in $\mathcal{AC}^0$ to relations in $\mathcal{NC}$. Moreover, the work of [41] focuses on the communication complexity of the proof system, but not the runtime of the verifier[10] (the complexity of their verifier is proportional to the size of R). For relations in $\mathcal{L}$-uniform $\mathcal{NC}$, our techniques yield IPCPs with verifier time complexity that is quasi-linear in the input and witness sizes. See the full version for details.

PCA with Efficient Provers. Another work of Kalai and Raz [42] proposes a new proof system model in the public-key setting called *probabilistically checkable argument* (PCA). A PCA is a relaxation of a probabilistically checkable proof (PCP) in two ways. First, it is assumed that each verifier is associated with a public key and that the PCA is designated-verifier (i.e., depends on the verifier's public key). Second, the soundness property is required to hold only *computationally*. Other than these differences, the setting is the same as that of PCPs: a probabilistic polynomial time verifier only reads a few bits of the proof string in order to verify. A PCA is said to be *efficient* if the honest prover, given a witness, runs in time $\text{poly}(n)$.

Using the assumption that (computational) PIR schemes with polylog communication exist, [42] show a transformation from any IPCP with certain properties to a short PCA. Applying this transformation to our IPCP (the conditions of the transformation are met) yields an *efficient* PCA with proof-string length $\text{poly}(\text{witness size}, \log n, d)$ and query complexity with $\text{poly}(\log n, d)$ for any language in $\mathcal{NP}$ whose relation can be computed by depth d and poly-size circuits. We note that the efficiency of the prover is derived from a special property of our proof system. In particular, previous PCAs (obtained when one starts with the IPCPs of [41]) require non-polynomial time provers. See the full version for details and theorem statements.

2. MAIN TECHNIQUES OVERVIEW

The Big Picture. In a nutshell, our goal is to reduce the verifier's runtime to be proportional to the depth of the circuit C being computed, rather than its size, without increasing the prover's runtime by too much.

To do this we use many of the ideas developed for the MIP and PCP setting starting with the works of [15, 11, 10, 8, 7, 29], applying them to the problem of proving that the computation of a (uniform) circuit C is progressing properly, without the verifier actually performing it or even looking at the entire circuit. Applying the ideas pioneered in the MIP/PCP setting to our setting, however, runs into immediate difficulties. The MIP/PCP constructions require assuming that the verifier somehow has access to a *committed* string (usually the string should contain a low degree extension—a high-distance encoding—of C's computation on the input x). This assumption is built into the PCP

[10] In both this work and in [41], the prover always runs in polynomial time.

model, and is implicitly achieved in the MIP model by the fact that the provers cannot communicate. Our challenge is that in our setting we cannot assume such a commitment! Instead, we force the prover to recursively prove the values he claims for this low-degree extension, and do this while preserving the prover's time complexity.

Elaborating on the above, we proceed to give the idea of the proof of our main theorem.

Assume without loss of generality that the circuit C is a depth d arithmetic circuit in a layered form where there are as many layers as the depth of the circuit.[11]

In previous work, spanning both the single and multi prover models [49, 53, 11, 41],[12] the entire computation of the underlying machine is arithmetized and turned into an algebraic expression whose value is claimed and proved by the prover.

Departing from previous work, here we instead employ an interactive protocol that closely follows the (parallelized) computation of C, layer by layer, from the output layer to the input layer, numbering the layers in increasing order from the top (output) of the circuit to the bottom (input) of the circuit.[13] The verifier has no time to compute points in the low-degree extension of the computation on x in layer i: this is the low-degree extension (a high distance encoding) of the vector of values that the gates in the circuit's i-th layer take on input x, and to compute it one needs to actually evaluate C, which we want to avoid! Thus, the low-degree extension of the i-th layer, will be instead supplied by the prover. Of course, the prover may cheat. Thus, each phase of the protocol lets the verifier reduce verification of a single point in the low-degree extension of an advanced step (layer) in the parallel computation, to verification of a single point in the low-degree extension of the previous step (layer). This process is repeated iteratively (for as many layers as the circuit has), until at the end the verification has been reduced to verifying a single point in the extension of the first step in the computation. In the first step of the computation (the input layer), the only information "computed" is the input x, the verifier can compute the low degree extension of the input x on its own.

Going from Layer to Layer. Given the outline above, the main remaining challenge is how to reduce verification of a single point in the low degree extension of a layer in the circuit, to verification of a single point in the low degree extension of the previous layer.

We observe that every point in the low degree extension (LDE) of the advanced layer (layer i) is a linear combination, or a weighted sum, of the values of that layer's gates. The circuit has fan-in 2, and thus we can express the value of each gate g in layer i as a sum, over all possible pairs (k, ℓ) of gates in the layer below (layer $i+1$), of some (low degree) function of the values of gates k and ℓ, as well as a predicate that indicates whether those gates are indeed the "children" of gate g. Arithmetizing this entire sum of sums, we run a sum-check protocol to verify the value of one point in the low-degree extension of layer i. To simplify matters, we assume for now that the verifier has access to (a low-degree extension of) the predicate that says whether a pair of gates (k, ℓ) are the children of the gate g. Then (modulo many details) at the end of this sum-check protocol the verifier only needs to verify the values of a pair of points in the LDE of layer $i + 1$. This is still not enough, as we need to reduce the verification of a single point in the LDE of layer i to the verification of a *single* point in layer $i + 1$ and not of a pair of points. We finally use an interactive protocol to reduce verifying two points in the LDE of layer $i + 1$ to verifying just one.

We assumed for simplicity of exposition above that the verifier has access to a low degree extension of the predicate describing arbitrary circuit gates. Thus, the (central) remaining question is how the verifier gains access to such LDE's of predicates that decide whether circuit gates are connected, without looking at the entire circuit (as the circuit itself is much larger than the verifier's sunning time). This is where we use the uniformity of the circuit, described below.

The verifier's running time in each of these phases is *poly-logarithmic* in the circuit size. In the final phase, computing one point in the low-degree extension of the input requires only nearly-linear time, independent of the rest of the circuit. Another important point is that the verifier does not need to remember anything about earlier phases of the verification, at any point in time it only needs to remember what is being verified about a certain point in the computation. This results in very space-efficient verifiers. The savings in the prover's running time comes (intuitively) from the fact that the prover does not need to arithmetize the *entire* computation, but rather proves statements about one (parallel) computation step at a time.

Utilizing Uniformity. It remains then to show how the verifier can compute (a low-degree extensions of) a predicate that decides whether circuit gates are connected, without looking at the entire circuit. To do this, we use the uniformity of the circuit. Namely, the fact that it has a very short implicit representation. A similar problem was faced by [11], there a computation is reduced to an (exponential) 3SAT formula, and the (polynomial-time) verifier needs to access a low-degree extension of a function computing which variables are in a specific clause of the formula. In the [11] setting this can be done because the Cook-Levin reduction transforms even exponential-time uniform computations into formulas where information on specific clauses can be computed efficiently. Unfortunately, we cannot use the Cook-Levin reduction as [11] and other works do, because we need to transform uniform computations into low-depth circuits without blowing up the input size.

To do this, we proceed in two steps. First, we examine low space computations, e.g. uniform log-space Turing Machines (deterministic or non-deterministic). A log-space machine can be transformed into a family of boolean circuits with poly-logarithmic depth and polynomial size. We show that in this family of circuits, it is possible to compute the predicate that decides whether circuit gates are connected in poly-logarithmic time and constant ($\mathcal{AC}^0$) depth. This computation can itself be arithmetized, which allows the verifier to compute a *low-degree extension* of the predicate in poly-logarithmic time. Thus we obtain an interactive proof with an efficient prover and super-efficient verifier for any $\mathcal{L}$ or $\mathcal{NL}$ computation.

[11] Every circuit can be converted into this format by at most squaring its size and not changing the depth.

[12] One exception is the work of Feige and Kilian on refereed games [30], which is in a different model. See the full version for details.

[13] I.e., layer 0 is the output layer, and layer d is the input layer.

Still, the result above took advantage of the (strong) uniformity of very specific circuits that are constructed from log-space Turing Machines. We want to give interactive proofs for general log-space uniform circuits, and not only for the specific ones we can construct for log-space languages. How then can a verifier compute even the predicate that decides whether circuit gates in a log-space uniform circuit are connected (let alone its low degree extension)? In general, computing this predicate might require nearly as much time as evaluating the entire circuit. We overcome this obstacle by observing that the verifier does not have to compute this predicate on its own: it can ask the prover to compute the predicate for it! Of course, the prover may cheat, but the verifier can use the above interactive proof for log-space computations to force the prover to *prove* that it computed the (low degree extensions of) the predicate correctly. This final protocol gives an interactive proof for general log-space uniform circuits with low depth.

Finally, we note that even for non-uniform circuits, the only "heavy" computation that the verifier needs to do is computing low-degree extensions of the predicate that decides whether circuit gates are connected. The locations at which the verifier needs to access this predicate are only a function of its own randomness (and not of the input or the prover's responses). This means that even for a completely non-uniform circuit, the verifier can compute these evaluations of the predicate's low-degree extension *off-line* on its own, without knowing the input or interacting with the prover. This off-line phase requires run-time that is proportional to the circuit size. Once the input is specified, the verifier, who has the (poly-logarithmically many) evaluations of the predicate's low degree extension that it computed off-line, can run the interactive proof on-line with the prover. The verifier will be super efficient in this on-line phase. See the full version for further details.

3. A BARE-BONES PROTOCOL

Our goal is constructing a protocol in which a prover, who is given a circuit $C : \{0,1\}^k \to \{0,1\}$ of size S and of depth d, and a string $x \in \{0,1\}^k$, proves to a verifier that $C(x) = 0$. The verifier's running time should be significantly smaller than S (the time it would take him to evaluate $C(x)$ on his own). At the same time, we want the prover to be efficient, running in time that is polynomial in S.

Since we want the verifier to run in time that is smaller than the circuit size, we must utilize the uniformity of the circuit, as discussed in Section 2. In this section, however, we do not focus on this issue. Rather, we work around the circuit uniformity issue by giving the verifier oracle access to (an extension of) the function that on input three gates outputs 1 if one gate is the addition (or the multiplication) of the other two gates. The verifier will run in quasi-linear time given this oracle. We call this protocol a *bare-bones* interactive proof protocol, it should be taken as an abstraction, meant to highlight and clarify some of the new technical ideas in our work. It is *not* an interactive proof in the standard model. For the details on how we realize the bare-bones protocol as an interactive proof (removing the oracle), see the overview in Section 2 and the full version.

Preliminaries. Fix $\mathbb{H}$ to be an extension field of $\mathbb{GF}[2]$, and fix $\mathbb{F}$ an extension field of $\mathbb{H}$, where $|\mathbb{F}| = \text{poly}(|\mathbb{H}|)$. Fix an integer $m \in \mathbb{N}$, and $\alpha : \mathbb{H}^m \to \{0, 1, \ldots, |\mathbb{H}^m| - 1\}$ to be an (efficiently computable) lexicographic order of $\mathbb{H}^m$. We can view a k-element string $(w_0, w_1, \ldots, w_{k-1})$ as a function $W : \mathbb{H}^m \to \mathbb{F}$, where $W(z) \stackrel{\text{def}}{=} w_{\alpha(z)}$ when $\alpha(z) \leq k-1$ and $W(z) = 0$ otherwise.

A *low degree extension* of a function W (or a k-element string W) is a function $\tilde{W} : \mathbb{F}^m \to \mathbb{F}$ that agrees with W on $\mathbb{H}^m$: $\tilde{W}|_{\mathbb{H}^m} \equiv W$, such that $\tilde{W}$ is an m-variate polynomial of degree significantly smaller than $|\mathbb{F}|$. A basic fact is that there exists a unique extension of W into a function (*the* unique low-degree extension) of degree at most $|\mathbb{H}| - 1$ in each variable. Moreover, the unique low-degree extension can be expressed as $\tilde{W}(z) = \sum_{p \in \mathbb{H}^m} \tilde{\beta}(z, p) \cdot w_i$, where $\tilde{\beta} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ is an easily computable m-variate polynomial of degree at most $|\mathbb{H}| - 1$ in each variable.

Notation and Conventions. Recall that the circuit C has size S and depth $d \leq S$. We choose $\mathbb{H}$ such that $\max\{d, \log(S)\} \leq |\mathbb{H}| \leq \text{poly}(d, \log(S))$. We choose m to be an integer such that $S \leq |\mathbb{H}|^m \leq \text{poly}(S)$, and $\mathbb{F}$ such that $|\mathbb{F}| \leq \text{poly}(|\mathbb{H}|)$. Finally, let $\delta \in \mathbb{N}$ be a (degree) parameter such that $|\mathbb{H}|-1 \leq \delta < |\mathbb{F}|$. We assume w.l.o.g that C is a *layered* arithmetic circuit of *fan-in 2* (over the gates $\times$ and $+$ and the field $\mathbb{F}$), and that all its layers are of size S.[14] When considering the d layers of C, we think of the 0 layer as the output layer, and of the d layer as the input layer.

For each $0 \leq i \leq d$, we label the S gates in the i'th layer of C by values in $\{0, \ldots, S-1\}$. For each $i \in [d]$, we associate with C two functions $\text{add}_i, \text{mult}_i : \mathbb{H}^m \times \mathbb{H}^m \times \mathbb{H}^m \to \{0,1\}$. Both these functions take 3 gate labels (j_1, j_2, j_3) as input (gate labels in $[S]$ are represented as vectors in $\mathbb{H}^m$), and return 1 if and only if the gate j_1 in layer $i-1$ is the addition or multiplication (respectively) of gates (j_2, j_3) in layer i, and 0 otherwise. For each $i \in [d]$, let $\tilde{\text{add}}_i, \tilde{\text{mult}}_i : \mathbb{F}^{3m} \to \mathbb{F}$ be (some) low-degree extensions of $add_i, mult_i$ (respectively) with respect to $\mathbb{H}, \mathbb{F}, m$, of degree δ in each variable (recall $\delta < |\mathbb{F}|$). In this *bare-bones* protocol we simply assume that the verifier has oracle access to the functions $\tilde{\text{add}}_i, \tilde{\text{mult}}_i$ for every $i \in [d-1]$.

Finally, for each $0 \leq i \leq d$ we associate a vector $v_i = (v_{i,0}, \ldots, v_{i,S-1}) \in \mathbb{F}^S$ with the i'th layer of the circuit C. These vectors are functions of the input $x = (x_1, \ldots, x_k) \in \mathbb{F}^k$, and are defined as follows: For each $0 \leq i \leq d$ we let v_i be the vector that consists of the values of all the gates in the i-th layer of the circuit when it is evaluated on input x. So, the vector v_0, that corresponds to the output layer, satisfies $v_0 = (C(x), 0, \ldots, 0) \in \mathbb{F}^S$. Similarly, the vector v_d, that corresponds to the input layer, satisfies $v_d = (x_1, \ldots, x_k)$. Similarly, for each $0 \leq i \leq d$, let $\tilde{V}_i : \mathbb{F}^m \to \mathbb{F}$ be the (unique) low degree extension of v_i with respect to $\mathbb{H}, \mathbb{F}, m$. Recall that the function $\tilde{V}_i$ is of degree $\leq |\mathbb{H}| - 1$ in each of its m variables and is efficiently computable.

Protocol Overview. The prover wants to prove $C(x) = 0$, or equivalently, that $\tilde{V}_0(0, \ldots, 0) = 0$. This is done in d phases (where d is the depth of C). In the i'th phase ($1 \leq i \leq d$) the prover reduces the task of proving that $\tilde{V}_{i-1}(z_{i-1}) = r_{i-1}$ to the task of proving that $\tilde{V}_i(z_i) = r_i$, where z_i is a random value determined by the protocol (and $z_0 = (0, \ldots, 0)$, $r_0 = 0$). Finally, after the d'th phase, the verifier checks on his own that $\tilde{V}_d(z_d) = r_d$. Note that $\tilde{V}_d$ is the low degree

[14] The one exception is the input layer, which is of size n, but we ignore this technicality throughout the exposition. See the full version for details.

extension of the input x with respect to $\mathbb{H}, \mathbb{F}, m$. Thus, this last verification task requires computing a single point in the low degree extension of the input x. This is the "heaviest" computation run by the verifier, and this final computation is independent of the circuit C; it can be done in quasi-linear time in the input length. Moreover, if the verifier is given oracle access to the low-degree extension of x, then this only requires a *single* oracle call.

In what follows we describe these phases. In each phase, the communication complexity is $\text{poly}(d, \log S)$, the running time of the prover is at most $\text{poly}(S)$, and the running time of the verifier is $\text{poly}(d, \log S)$.

In the i-th phase ($i \in [d]$), we reduce the task of proving that $\tilde{V}_{i-1}(z_{i-1}) = r_{i-1}$ to the task of proving that $\tilde{V}_i(z_i) = r_i$ where $z_i \in \mathbb{F}^m$ is a random value determined by the verifier, and r_i is a value determined by the protocol. For $p, \omega_1, \omega_2 \in \mathbb{F}^m$, we define

$$f_i(p, \omega_1, \omega_2) \stackrel{\text{def}}{=} \tilde{\text{add}}_i(p, \omega_1, \omega_2) \cdot \left(\tilde{V}_i(\omega_1) + \tilde{V}_i(\omega_2)\right) + \tilde{\text{mult}}_i(p, \omega_1, \omega_2) \cdot \tilde{V}_i(\omega_1) \cdot \tilde{V}_i(\omega_2).$$

For every $p \in \mathbb{H}^m$, $\tilde{V}_{i-1}(p) = \sum_{\omega_1, \omega_2 \in \mathbb{H}^m} f_i(p, \omega_1, \omega_2)$. From the definition of the low-degree extension, for every $z \in \mathbb{F}^m$, we get that $\tilde{V}_{i-1}(z) = \sum_{p \in \mathbb{H}^m} \tilde{\beta}(z, p) \cdot \tilde{V}_{i-1}(p)$, where $\tilde{\beta} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ is a polynomial of degree at most $|\mathbb{H}| - 1$ in each variable, that can be computed in time $\leq \text{poly}(|\mathbb{H}|, m)$. We conclude that for every $z \in \mathbb{F}^m$,

$$\tilde{V}_{i-1}(z) = \sum_{p, \omega_1, \omega_2 \in \mathbb{H}^m} \tilde{\beta}(z, p) \cdot f_i(p, \omega_1, \omega_2).$$

Thus, proving that $\tilde{V}_{i-1}(z_{i-1}) = r_{i-1}$ reduces to proving that $r_{i-1} = \sum_{p, \omega_1, \omega_2 \in \mathbb{H}^m} \tilde{\beta}(z_{i-1}, p) \cdot f_i(p, \omega_1, \omega_2)$. This is done by running an interactive sum-check protocol (see the full version for the details).

Finally, in order to carry out the last verification task in the sum-check protocol, the verifier needs to compute on his own the value $\tilde{\beta}(z_{i-1}, p) \cdot f_i(p, \omega_1, \omega_2)$, on random inputs $p, \omega_1, \omega_2 \in_R \mathbb{F}^m$ (chosen by the verifier). Recall that we assumed for now that the verifier has oracle access to the functions $\tilde{\text{add}}_i$ and $\tilde{\text{mult}}_i$. Computing the function $\tilde{\beta}$ requires time $\leq \text{poly}(|\mathbb{H}|, m)$. So, the main computational burden in this verification task is computing $\tilde{V}_i(\omega_1)$ and $\tilde{V}_i(\omega_2)$, which requires time $\text{poly}(S)$ (and thus cannot be computed by our computationally bounded verifier). In the protocol, the prover now sends both these values, $\tilde{V}_i(\omega_1)$ and $\tilde{V}_i(\omega_2)$, to the verifier. The verifier $\mathcal{V}_1$ (who knows ω_1 and ω_2) receives two values v_1, v_2 and now wants to verify that $\tilde{V}_i(\omega_1) = v_1$ and $\tilde{V}_i(\omega_2) = v_2$.

Thus, so far, using the sum-check protocol, we reduced task of proving that $\tilde{V}_{i-1}(z_{i-1}) = r_{i-1}$ to the task of proving that both $\tilde{V}_i(\omega_1) = v_1$ and $\tilde{V}_i(\omega_2) = v_2$. However, recall that our goal was to reduce the task of proving that $\tilde{V}_{i-1}(z_{i-1}) = r_{i-1}$ to the task of proving a *single* equality of the form $\tilde{V}_i(z_i) = r_i$. Therefore, what remains (in the i'th phase) is to reduce the task of proving two equalities of the form $\tilde{V}_i(\omega_1) = v_1$ and $\tilde{V}_i(\omega_2) = v_2$ to the task of proving a single equality of the form $\tilde{V}_i(z_i) = r_i$. This is done via a more standard interactive process, see the full version for details.

4. ACKNOWLEDGEMENTS

We thank Salil Vadhan for illuminating conversations and insightful comments. Thanks also to Adam Kalai and Ran Raz for their generous assistance.

5. REFERENCES

[1] ET, phone SETI@home!. Science@NASA headlines, 1999.

[2] The great internet mersenne prime search, project webpage. http://www.mersenne.org/, 2007.

[3] SETI@home project website. http://setiathome.berkeley.edu/, 2007.

[4] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[5] D. P. Anderson. Public computing: Reconnecting people to science. In *Conference on Shared Knowledge and the Web*, 2003.

[6] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID*, pages 4–10, 2004.

[7] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *FOCS*, pages 14–23, 1992.

[8] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[9] L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

[10] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[11] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *FOCS*, pages 16–25, 1990.

[12] B. Barak and O. Goldreich. Universal arguments and their applications. In *CCC*, pages 194–203, 2002.

[13] R. Beigel, M. Bellare, J. Feigenbaum, and S. Goldwasser. Languages that are easier than their proofs. In *FOCS*, pages 19–28, 1991.

[14] 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*, pages 37–56, 1988.

[15] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *STOC*, pages 113–131, 1988.

[16] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. In *STOC*, pages 1–10, 2004.

[17] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan. Short pcps verifiable in polylogarithmic time. In *CCC*, pages 120–134, 2005.

[18] M. Blum. How to prove a theorem so no-one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

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

[20] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic

communication. In *EUROCRYPT*, pages 402–414, 1999.

[21] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[22] A. Condon. Space-bounded probabilistic game automata. *Journal of the ACM*, 38(2):472–494, 1991.

[23] A. Condon and R. E. Ladner. Probabilistic game automata. *Journal of Computer and System Sciences*, 36(3):452–489, 1988.

[24] A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *FOCS*, pages 462–467, 1989.

[25] I. Dinur. The pcp theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[26] C. Dwork, M. Naor, O. Reingold, and L. J. Stockmeyer. Magic functions. *Journal of the ACM*, 50(6):852–921, 2003.

[27] C. Dwork and L. J. Stockmeyer. Finite state verifiers i: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.

[28] C. Dwork and L. J. Stockmeyer. Finite state verifiers ii: Zero knowledge. *Journal of the ACM*, 39(4):829–858, 1992.

[29] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.

[30] U. Feige and J. Kilian. Making games short (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 506–516, 1997.

[31] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[32] L. Fortnow. Complexity-theoretic aspects of interactive proof systems. PhD thesis. Technical Report MIT/LCS/TR-447, Massachusetts Institute of Technology, 1989.

[33] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113(1):55–73, 1993.

[34] 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.

[35] S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

[36] S. Goldwasser and Y. T. Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–, 2003.

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

[38] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[39] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

[40] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

[41] Y. T. Kalai and R. Raz. Interactive pcp. Technical Report TR07-031, ECCC, 2007.

[42] Y. T. Kalai and R. Raz. Probabilistically checkable arguments. Manuscript, 2007.

[43] J. Kilian. Zero-knowledge with log-space verifiers. In *FOCS*, pages 25–35, 1988.

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

[45] J. Kilian. Improved efficient arguments (preliminary version). In *CRYPTO*, pages 311–324, 1995.

[46] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[47] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[48] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *ISC*, pages 314–328, 2005.

[49] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[50] S. Micali. Cs proofs (extended abstract). In *FOCS*, pages 436–453, 1994.

[51] M. Naor. Bit commitment using pseudo randomness. In *CRYPTO*, pages 128–136, 1989.

[52] A. Polishchuk and D. A. Spielman. Nearly-linear size holographic proofs. In *STOC*, pages 194–203, 1994.

[53] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.