Improved Efficient Arguments

(Preliminary version)

Joe Kilian[1]

NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.
joe@research.nj.nec.com

Abstract. We consider complexity of perfect zero-knowledge arguments [4]. Let T denote the time needed to (deterministically) check a proof and let L denote an appropriate security parameter. We introduce new techniques for implementing very efficient zero-knowledge arguments. The resulting argument has the following features:

- The arguer can, if provided with the proof that can be deterministically checked in $O(T)$ time, run in time $O(TL^{O(1)})$. The best previous bound was $O(T^{1+\epsilon}L^{O(1)})$.
- The protocol can be simulated in time $O(L^{O(1)})$. The best previous bound was $O(T^{1+\epsilon}L^{O(1)})$.
- A communication complexity of $O(L \lg L)$, where L is the security parameter against the prover. The best previous known bound was $O(L \lg T)$.

This can be based on fairly general algebraic assumptions, such as the hardness of discrete logarithms.

Aside from the quantitative improvements, our results become qualitatively different when considering arguers that can run for some superpolynomial but bounded amount of time. In this scenario, we give the first arguments zero-knowledge arguments and the first "constructive" arguments in which the complexity of arguing a proof is tightly bounded by the complexity of verifying the proof.

We obtain our results by a hybrid construction that combines the best features of different PCPs. This allows us to obtain better bounds than the previous technique, which only used a single PCP. In our proof of soundness we exploit the error correction capabilities as well as the soundness of the known PCPs.

1 Introduction

One of the great achievements in the study of interactive proof systems has been the discovery of transparent/probabilistically checkable proofs [6, 17]. While most of this research has been aimed at proving complexity results, it is interesting to consider the original application, proving theorems. By requiring the verifier to look at a vanishing section of a proof, one might hope to use them to speed the verification of large, unwieldy proofs. For example, [6] discusses an application to checking the executions of long computations, saying, "In this setup, a single reliable PC can monitor the operation of a herd of supercomputers

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 311-324, 1995.

working with possibly extremely powerful but unreliable software and untested hardware."

Beyond the practical difficulties of this scenario are problems of a more fundamental nature. First, the prover must work quite hard to produce the transparent proof. The most easy to verify PCPs ([3] and its descendants) can be checked in $O(1)$ probes, but require the prover to spend time superquadratic in the size of the execution trace. Polishchuk and Spielman [27] construct a constant probe PCP using only $O(T^{1+\epsilon})$ work, where T is the time to deterministically verify the original proof. Furthermore, they show that for any positive constant c_2 there exists a constant c_1 such that proofs of size T are converted into transparent proofs in $O(T \lg^{c_1} T)$ time, which may be verified using $O(T^{c_2})$ work. Thus, at present, there is a continuum of achievable proof complexity/verification complexity tradeoffs, but there does not exist any "optimal" PCP that dominates the others. One can either force the prover to work hard or force the verifier to work hard.

Another difficulty is that in order for the proof to work, the verifier must have possession of the entire proof, or at least a guarantee that the prover cannot change any bits of the proof. Thus, it is not clear how to verify a PCP over a network. The work needed to receive such a proof would be much more than the work required to receive and check a standard proof.

By a standard transformation, results for PCPs carry over into two-prover proofs with essentially optimal (logarithmic in the size of the PCP) communication requirements. Furthermore, these proofs may be made to be zero-knowledge [10]. However, it is open how to surmount this last difficulty within the more realistic framework of single-prover interactive proof systems.

1.1 Efficient zero-knowledge arguments

Brassard, Chaum and Crépeau introduce the notion of arguments [4]. Unlike interactive proofs, which place no assumptions the power of potentially malicious provers, the argument framework puts some bound on the capabilities of the prover, weakening the ordinary soundness condition to one of *computational soundness*. This more realistic assumption leads to dramatically improved properties over interactive proof systems. For example, there exist constant-round perfect zero-knowledge arguments for NP based on reasonable number-theoretic complexity assumptions [4]; [26] shows how to base such proofs on one-way functions at the expense of greater round complexity.

Fiat and Shamir [19] introduce a technique whereby interactive arguments of a fairly general form may be converted into noninteractive arguments. Their basic idea is to replace random questions from the verifier by the results of a random-behaving hash function. They observe that this transformation is rigorously analyzable given a truly random hash function as a black box. Damgård also uses similar ideas for developing practical noninteractive arguments [9]. More recently, Bellare and Rogaway have developed a much more extensive treatment black-box hash functions [8]. Unfortunately, there is no known clean computational

assumption under which the soundness of the resulting noninteractive argument can be established.

In [21], it is shown (under suitable complexity assumptions) how to

1. Convert PCPs into a type of "perfect zero-knowledge" PCPs, and
2. Convert {"perfect zero-knowledge"} PCPs into {perfect zero-knowledge} arguments.

Asymptotically, this construction requires much less communication than any previous construction. For example, by using the PCPs constructed in [3], the total communication is only $O(L \lg T)$, where T is the number of steps needed to check the original proof and L is the security parameter for the prover (informally, L specifies the size of the problems the prover is assumed unable to solve).

More recently, Micali has put forth the notion of "CS Proofs" [25]. A stronger result may be obtained by a straightforward application of the Fiat-Shamir transformation and the method of [21]. We strongly recommend a careful reading of [4, 19, 9, 21, 8] prior to reading [25].

1.2 Limitations of previous techniques

The arguments of [21] inherit their work/verification time tradeoffs from the work/verification time tradeoffs in the original PCPs. So to obtain the lowest communication costs advertised, one must use proofs that are very difficult to construct.

The time required to simulate the argument is polynomial in the size of the original proof. While in line with previous proof systems and arguments, one can hope to do much better. The verifier only communicates $O(L \lg T)$ bits and performs $O(L^{O(1)} \lg T)$ computations. The intuition behind our notions of zero-knowledge is that what one obtains by participating in a proof should not be more than what one could have obtained using the same resources but not participating in the proof. In the program-checking application of [6], one can by oneself reconstruct the original "proof" in $O(T \lg^{O(1)} T)$ time. Hence, to say that the simulation can be performed in $O(T^{O(1)})$ time doesn't preclude being able to obtain information about entire execution, with a computational investment of only $O(L^{O(1)} \lg T)$. In such a scenario, the standard notion of zero-knowledge is too weak to be meaningful.

Finally, even when optimized for communication, there remains a significant gap between the $O(L \lg T)$ communication required by this protocol what one could reasonable hope for. Intuitively, one might achieve communication of $O(L)$ bits (it would be amazing if one could achieve $o(L)$ communication, without assuming the existence of problems of size $o(L)$ that the prover cannot solve), and since T might conceivably be nearly exponential in L, $O(L \lg T)$ is nearly a quadratic factor off from what one can hope for.

1.3 Our main result

We make progress on the above mentioned difficulties. We show that one can use substantially less communication, even while using the computationally cheap proofs from [27]. As before, this protocol requires the existence of secure perfect zero-knowledge bit-commitment and collision-intractable hash functions, with security parameter L. We will also assume in this paper that the size of ones proofs is much larger than the statement of what is to be proven.

Theorem 1. Under the above assumptions, a proof $\mathcal{P}$ deterministically verifiable in T steps, can be implemented as a perfect zero-knowledge argument for the correctness of $\mathcal{P}$ with the following properties:

(communication efficiency) Only $O(L \lg L)$ bits of communication are required.
(computational efficiency) The prover only has to perform only $O(L^{O(1)}T)$ computational steps
(completeness) If $\mathcal{P}$ is correct and P follows the protocol, then V will always accept.
(soundness) If $\mathcal{P}$ is false, then either V will reject with probability at least $\frac{1}{2}$ or there exists a program which will break the bit-commitment or collision-intractability assumption (with security parameter L) with nonnegligible probability, in $T^{O(1)}$ time and using $T^{O(1)}$ calls to an oracle for P.
(strong perfect zero-knowledge) There exists a simulator that given an oracle for a possibly malicious verifier $\hat{V}$ will perfectly simulate $\hat{V}$'s view of the proof using expected $L^{O(1)}$ computation and $L^{O(1)}$ calls to the oracle for $\hat{V}$.

Note that we are implicitly using a black-box notion of soundness and zero-knowledge in the statement of our theorem, which has ample precedent in the literature. We prefer this approach because is allows one to make meaningful statements about arguments of specific theorems. The older formalisms strictly make sense only in the context of infinite languages L.

1.4 How our protocol scales for large T

The improved efficiency of our protocol is particularly striking if one considers large T. For example, it is not unreasonably to posit a *super-arguer* that can run for superpolynomially many steps (e.g. $T = O(n^{\lg n})$), but cannot perform exponential-time computations. The original arguments required communication at least T, and hence the verifier would also have to run in superpolynomial time. The arguments of [21] don't have this problem (as noted in [25]) but they still are very problematic. Suppose that an arguer works very hard to generate a proof whose verification takes as long as the time to construct the proof. To use the [21] protocol would require him to run for $T' = \Omega(T^{1+\epsilon})$ steps. However, T'/T is also superpolynomial. We contend that this is not in the spirit of superpolynomial time. If one believes that T^2 or $T^{1+\epsilon}$ is "of the same order" as T, then one is really treating T as polynomial time.

In contrast, our techniques yield a polynomial *multiplicative* blowup in the running time of the super-arguer as opposed to the polynomial *compositional* blowup of previous techniques. Thus, the transformation from a proof to an argument is much more robust than the previous one.

We also note that if one uses the [21] protocol the simulation time is at least T, and thus fails to be zero-knowledge for superpolynomial T. Our protocol continues to be zero-knowledge for as long as a polynomially large security parameter L is appropriate. It is quite possible that the cryptographic primitives we require can be based on problems which have no subexponential time solutions, in which case exponential-sized T may be accommodated.

1.5 Techniques Used

For our new protocol, we use the techniques employed by [21], namely zero-knowledge proofs on committed bits [28, 14], transparent proofs [6, 17] and Merkle's hash-tree commitments [24],[1] and introduce three new techniques.

To improve the communication complexity of our protocols while using computationally inexpensive transparent proofs, we add a further recursive step to our protocol. Interestingly, in these recursive proofs both parties know that the statement is true (with probability extremely close to 1) ahead of time. Rather, the prover convinces the verifier that he knows a particular way of proving this statement. The use of recursive proofs in the transparent proof context is not new; more sophisticated noncryptographic examples of this technique can be found in, for example, [1, 3, 7]. Here, our use of recursion is intermingled with the manner in which individual bits of the transparent proof are revealed, allowing us to obtain much stronger bounds than if we simply restricted ourselves to these techniques.

To achieve the improved zero-knowledge result, we augment the basic hash tree construction with a randomization step. It is difficult to simulate the interior nodes of the hash tree, in particular the root node, in our desired time bound. To get around this problem we use the recursive proof technique to hide the values of all but the root node of the tree. We then show how to randomize this root node so that it may be very efficiently simulated.

Finally, we show how to save random bits by using cryptographically secure pseudorandom generators. This last technique allow us to efficiently exploit transparent proof system in which the verifier flips many more coins than is allowed for in our communication bound. Interestingly, whereas the previous techniques don't say anything about the standard transparent proof/PCP model, this result gives strong evidence for PCPs with a better size/bits of randomness tradeoff than our current techniques can establish without relying on computational assumptions. Note that we do not however obtain fewer random bits than the most efficient systems [1, 3], but merely show how to shrink the number of random bits used in proofs that obtain a very small size in exchange for a large

[1] [21] missed this reference - our apologies to Merkle.

amount of required randomness. It is an interesting open question to achieve these tradeoffs using noncryptographic techniques.

1.6 Outline of the Abstract

In Section 2 we introduce some preliminary definitions and background. In Section 3 we show how to combine PCPs to obtain greater efficiencies than are obtainable using a single PCP. In Section 4 we show how to obtain a protocol with strong simulatability properties. In Section 5 we note that one use complexity assumptions to save random bits. Finally, in Section 6 we give a brief idea of how soundness and zero-knowledge are proven for our interactive arguments.

2 Preliminaries

2.1 Cryptographic Primitives

The original construction of [21] makes use of Merkle's hash-tree technique, which in turn relied on families of *collision-intractable hash functions* and perfect zero-knowledge bit-commitment functions. Let $\phi(L)$ denote ones notion of an intractable amount of time. A family $\{H_{k,r}\}$ of collision-intractable hash functions has the following property: with high probability, if one chooses a function $h : \{0,1\}^{2L} \to \{0,1\}^{L}$ uniformly from $\{H_{L,*}\}$, then one cannot in expected time $\phi(L)$ compute x and y such that $h(x) = h(y)$.

The perfect zero-knowledge commitment scheme has functions $\{E_{L,*}\}$, such that if $E_{L,q}$ is appropriately chosen, then the distribution on $E_{L,q}(0,r)$ will be identical to that of $E_{L,r}(1,r)$, where r is a uniformly distributed infinite sequence of random bits, of which an expected $L^{O(1)}$ bits are actually read. However, one cannot in expected time $\phi(L)$ compute prefixes r_0 and r_1 such that $E_{L,q}(0,r_0) = E_{L,q}(1,r_1)$.

By using some global randomness, these schemes can be decided on before any proofs take place, and the same functions can be used by everyone.

2.2 Review of the [21] Protocol

The [21] protocol uses two very separate techniques. The first technique is a method for adding "zero-knowledge" to transparent proof systems. We will leave this basic trick unchanged, though we will alter our method of making our commitments in the next section. The second technique is to use Merkle's hash tree technique to allow a time-bounded prover P to commit to a large number of bits to a verifier V with little communication, and then efficiently reveal these individual bits. This procedure works as follows. Let $b_1, \ldots, b_n$ be the sequence of bits to be committed by the prover, let $h : \{0,1\}^{2L} \to \{0,1\}^{L}$ be uniformly chosen (by the verifier) from a family of collision-intractable hash functions. For ease of exposition, assume that $n = 2^k L$ for some integer k (otherwise, pad the sequence to make it the right size).

To commit to $b_1, \ldots, b_n$, P breaks $b_1, \ldots, b_n$ into $m = 2^k$ blocks,

$$C_{k+1,1}, \ldots, C_{k+1,m},$$

of L bits each. For $1 \leq i \leq k$ and $1 \leq j \leq 2^{i-1}$, P computes

$$C_{i,j} = h(C_{i+1,2j-1}, C_{i+1,2j}).$$

We can think of the array $[C_{i,j}]$ as specifying a hash tree in which each parent is equal to the hashed values of its children. Finally, P sends $C_{1,1}$, the root node, to V.

To reveal block $C_{k+1,l}$, P defines $j_1, \ldots, j_{k+1}$ by $j_{k+1} = l$ and $j_{i-1} = \lceil j_i/2 \rceil$ for $1 \leq i \leq k$. P then sends V

$$(C_{2,2j_1-1}, C_{2,2j_1}), \ldots, (C_{k+1,2j_k-1}, C_{2,2j_k}).$$

V checks that

$$C_{i,j_i} = h(C_{i+1,2j_i-1}, C_{i+1,2j_i})$$

for $1 \leq i \leq k$. The collision intractability of h ensures that P can expand a path in the tree in only one way. We refer to this sequence of pairs as a "witness" for block $C_{k+1,l}$, and denote it as W_l.

In the original protocol, the leaves of the tree were encrypted as perfect zero-knowledge blobs, but in the next section we achieve zero-knowledge in a more efficient way that (in its first step) works on the root of the tree. For the moment, we ignore this issue, and implement transparent proofs as follows:

1. P constructs the transparent proof and commits to it using the commit protocol described above. This step requires $O(L)$ bits of communication and that P compute for time within an $O(L^c)$ multiplicative factor of the time needed to construct the transparent proof.
2. V generates its sequence r of random bits, and sends r to P.
3. P and V compute which bits, and hence which L-bit blocks V would have accessed from the original transparent proof. P reveals these blocks using the above protocol. This requires computing time proportional to that used by in evaluating the original transparent proof, with an $O(L^c \lg n)$ multiplicative overhead. The total communication required by this step is $O(L \lg n)$ times the total number of revealed blocks.
4. V decides whether to accept or reject based on the values of the bits revealed by P. Again, this requires relatively little computation.

3 Greater Efficiencies Using Hybrid Schemes

Assume without loss of generality that V uses relatively few ($O(L)$) random bits. If not, then we can use the straightforward technique in Section 5 to shrink the number of bits required. Most of the communication cost is incurred in Step 3. To reveal a single bit requires $O(\mathrm{L} \lg n)$ communication, which already exceeds our desired bounds.

To get around this problem, we employ the "proof within a proof" technique that was used to achieve zero-knowledge. In the zero-knowledge version of the above protocol, the prover doesn't reveal the actual bits, but rather convinces the verifier that had he revealed these bits the verifier would have accepted. Similarly, the prover does not really need to send the witness for each block the verifier wishes to see. It suffices that he convince the verifier that he could indeed have sent such a message and that the verifier would have accepted. This involves a zero-knowledge proof of knowledge that can be recursively solved using current techniques.

3.1 Using "witnesses" for answers

Given input x, random string r and the root node $C_{1,1}$, we denote a witness $\mathcal{W}$ for $(x, r, C_{1,1})$ as a sequence

$$\mathcal{W} = \left((l_1, W_{l_1}), \ldots, (l_p, W_{l_p})\right),$$

where

- $l_1, \ldots, l_p$ denote those sections of the original transparent proof that V would have looked at on input x and with random string r,
- W_{l_i} is a valid witness for a block C_{k+1,l_i}, and
- On seeing the bits given in blocks

$$C_{k+1,l_1}, \ldots, C_{k+1,l_p},$$

V will accept.

We note that the existence of a witness for $(x, r, C_{1,1})$ is not sufficient to guarantee that the original theorem is true. However, under the collision intractability assumption, the fact that P *knows* such a witness is strong evidence for the validity of the theorem.

Consider a machine that in the pointer machine model (used by [6] and [27]) nondeterministically guesses $\mathcal{W}$ and then deterministically verifies that $\mathcal{W}$ is a witness for $(x, r, C_{1,1})$. We denote the transcript of this verification by $\mathcal{T}_{\mathcal{W}}$. We bound the length of $\mathcal{T}_{\mathcal{W}}$ up to polylogarithmic factors. First, suppose that the original verifier for the transparent proof used total time T_V. Note that $p \leq T_V$. Guessing $\mathcal{W}$ involves $O(L \lg nT_V)$ operations. Verifying that each witness is consistent requires $O(L^c \lg n)$ operations for some constant c, for a total cost of $O(L^c \lg nT_V)$. Finally, verifying that seeing the given portion of the tableau will cause the verifier to accept requires at most $O(LT_V)$ operations. Thus, the total transcript will be of size $O(L^c \lg nT_V)$.

For illustrative purposes, we first show how to obtain near optimal communication costs, without regard for the computational costs involved. Suppose that the original proof $\mathcal{P}$ was of size n and that we used the PCPs from [3]. Then the size of the resulting transparent proof $\mathcal{P}'$ will be $O(n^{O(1)})$ and the time used by the verifier to check this proof will be $O(\lg n)$ (not counting the initial cost for putting T in error-correcting code format). Thus, the size of the resulting

proof of knowledge will be (up to polylogarithmic factors) $O(L^{c_1} \lg^{c_2} n)$, which for reasonable sized n is $O(L^{c_3})$ for some constant c_3.

For n large compared to L, $O(L^{c_3})$ is small compared to n. P can therefore recursively prove that it knows a valid $\mathcal{W}$ by using the [21] zero-knowledge construction, optimizing for low communication by again using the transparent proof of [3]. The resulting communication for this step will be

$$O(L \lg(O(L^{c_3}))) = O(L \lg L).$$

Since the [3] type proof uses superquadratic time complexity, the above proof is very inefficient. To obtain simultaneously low-communication and low-computation proofs, we use a three-step recursion. On the top level, we use a Polishchuk-Spielman proof which requires $O(n \lg^{c_1} n)$ time and verified using $O(n^{c_2})$ work, where c_2 is sufficiently small ($< \frac{1}{4}$ suffices). The resulting proof of knowledge will therefore be of size $O(L^{O(1)} n^{c_2})$. Since c_2 is so small, we can then use the computationally intensive protocol described above.

Since malicious prover can try to cheat in the proofs at each level of the recursion, we amplify the error probabilities of these proofs so that in each one he can escape detection with probability at most, for example, 1/100. More precisely, our proof of soundness requires that whenever a PCP causes the verifier to accept with probability $\geq 1/1000$ an accepting computation path can be reconstructed (technically, this condition is not needed for the first proof).

The number of rounds of communication required when use of recursive proofs is of course greater than that of the original protocol. We can offset this somewhat by noting a simple optimization to the protocol of [21]. Instead of using proofs on committed bits, which requires multiple rounds, it suffices to use the zero-knowledge PCPs of [10]. These PCPs have the same qualitative properties as those of [3] (though with worse constants) but with the property that the queries are easy to simulate. This eliminates the round cost incurred by the proofs on committed bits.

We note that the techniques in [10] can be applied to the the protocol of [27] to obtain a zero-knowledge PCP in which the prover only performs $T^{1+\epsilon}$ work. However, it is open whether there exists zero-knowledge PCPs in which the prover only performs $T \log^{O(1)} T$ work. Fortunately, the elimination of the proof on committed bits steps occur in the last proof invoked, in which the work required is small in any case.

4 Achieving Strong Zero-Knowledge

In this section, we further modify the construction of [21] in order to make a more efficiently simulatable zero-knowledge protocol. In terms of a verifier's $\hat{V}$ view, the argument from the previous section can be summarized as follows:

1. P sends $\hat{V}$ a string $C_{1,1}$ (the root of the hash tree).
2. $\hat{V}$ sends P a string r (which may not be random).
3. P and $\hat{V}$ engage in a low-communication zero-knowledge proof of knowledge.

Since the transparent proofs we use have perfect completeness, it is straightforward to simulate the last two steps of the protocol. Regardless of the distribution on r, P will always know a witness for $(x, r, C_{1,1})$, and this proof can be simulated in using the original simulator for this protocol. Furthermore, the simulator for this last proof will work in time polynomial in L, since the entire transparent proof is bounded by a polynomial in L.

However, as we have written our protocol, it is not at all clear how to simulate the distribution on $C_{1,1}$, even in time polynomial in n. $C_{1,1}$ is a hashed down version of a transparent proof that S does not know. In the original [21] argument, the root node was a hashed version of a large number (polynomial in n) of L-bit perfect zero-knowledge blobs. Since these blobs were easy to simulate, the root node could be simulated in time polynomial in n. However, it is not clear how to speed up this simulation for arbitrary collision-intractable hash functions.

We get around this problem by using zero-knowledge blobs and a further use of hash trees. We modify the naive, stripped down protocol as follows: Instead of sending $\hat{V}$ the L-bit value of $C_{1,1}$, P generates a sequence of L perfect zero-knowledge L-bit blobs, denoted $C'_1, \ldots, C'_L$. Then, P computes a $\lceil \lg L \rceil$-depth hash-tree for $C'_1, \ldots, C'_L$, generating a second root node, C'. P sends C' to $\hat{V}$.

Naively, P can run the basic (nonzero-knowledge) argument as follows: On receiving r, P first sends the entire hash tree for $C'_1, \ldots, C'_L$, and then opens these blobs to reveal $C_{1,1}$. V will later check this part of the proof by verifying that the hash tree is self consistent and that the blob openings were correct. Now, P can then behave just as in the basic protocol without zero-knowledge. We call this concatenation of a valid hash tree rooted at C', the valid opening of these blobs to reveal $C_{1,1}$ and a valid witness for $(x, r, C_{1,1})$ a witness for (x, r, C').

Naively, this extra step involves sending $O(L^2)$ bits just to reveal the secondary hash-tree, which is prohibitive. However, we can use the same recursion trick to demonstrate knowledge of a witness $\mathcal{W}$ for (x, r, C'). The size of $\mathcal{T}_{\mathcal{W}}$ will be $O(L^c)$ bigger than the transcript for a witness for $(x, r, C_{1,1})$ (for some constant c), but this will at most add an $O(L \lg L)$ factor to the communication complexity of the resulting protocol.

5 Saving random bits

The transparent proofs and PCPs in the literature in general use relatively few bits. However, as one optimize ones PCPs in order to minimize the prover's overhead the number of random bits may conceivably become problematic. However, we can use pseudorandom generators to conserve bits in a straightforward manner. Suppose that V would normally send m random bits to P, and let $g : \{0,1\}^n \to \{0,1\}^m$ be a pseudorandom generator. Then if suffices for V to send a random n-bit string x to P, who then behaves as if sent the m-bit string $y = g(x)$. We claim that if g is sufficiently strong, then the modified PCP will remain a PCP, even against infinitely powerful provers.

Suppose that for some supposed theorem T (not necessarily true), and supposed transparent proof/PCP $\mathcal{P}'$ (not necessarily correctly generated), there is a nonnegligible difference between the probability that V accepts using random coin tosses y and the probability that V accepts using pseudorandom coin tosses $y = g(x)$. Consider the circuit $C_{T,\mathcal{P}'}$ which on input y, outputs 1 iff V would accept $\mathcal{P}'$ after generating its queries according to y. It is straightforward to generate $C_{T,\mathcal{P}'}$ given $\mathcal{P}', T$ and V, and its size will typically be $O(\mathcal{P}')$. Furthermore, $C_{T,\mathcal{P}'}$ can then be used to distinguish a random $g(x)$ from a random y.

Thus, by the contrapositive, if one believes that g is resistant against all sufficiently large circuits, then in particular it may be used for all transparent proofs of a given size.

The above argument relies on the nonuniform complexity of g; we don't know how to prove the analogous result result based on uniform complexity. We also note that thus far there has been no advantage to using cryptography based randomness reduction techniques instead of those currently used for PCPs.

6 Establishing Soundness and Zero-knowledge

In this abstract, we omit the proof of soundness and completeness, and give only a brief sketch of how they are proven. how they work.

Theorem 2. For the protocol described above, there exists a breaker B with the following property. Let T be a false statement, and $\hat{P}$ be a malicious prover who claims to have a proof of length n for this theorem. Suppose that $\hat{P}$ can convince V to accept T with probability greater than $\frac{1}{100}$ using the protocol outlined above. Then given black-box access to $\hat{P}$, B can break either the commitment assumption or the sibling-intractable hash function assumption, running in time $L^{c_1}n^{c_2}$, where c_1 and c_2 are explicitly computable constants. □

Thus, if we believe that breaking the assumption with security parameter L requires more than $L^{c_1}n^{c_2}$ times the number of steps that could plausible taken by $\hat{P}$, then it is reasonable to believe such an argument. Note that it is not so crucial to prove an optimal for c_2, since one can modestly increase the security parameter L.

Here is the very basic idea - numerous trivial details are omitted. For simplicity of exposition, we concentrate on a single proof; the analysis of a cascade of $O(1)$ recursive proofs behaves similarly. Thus, we consider the simplified protocol where $\hat{P}$ commits to a root C and then expands its leaves in response to V's challenges. In each challenge, $\hat{P}$ either,

1. Opens up paths in the hash-tree, revealing PCP values that cause V to accept, or,
2. Opens up legitimate paths in the hash-tree, but reveals values that cause V to reject, or,
3. Otherwise causes V to reject. We call this a garbage response.

While in the actual protocol only a single challenge is given, B can roll $\hat{P}$ back and ask him several challenges. He can then combine the branches for non-garbage responses in the natural way to recover entries of a partial PCP. If at any time these non-garbage answers cannot reconciled, i.e., the values of the hash-tree (or the root-node commitments and their hash-tree) are inconsistent, then B trivially has either,

1. Two strings that hash to the same value, or
2. Identical bit commitment strings for a 0 or a 1.

If this ever happens then we are done. Otherwise, we can show that if B runs $\hat{P}$ sufficiently many times and receives a Type 1 response with sufficiently high probability on each random run, then he will reconstruct a PCP that would cause the original verifier to accept with sufficiently high probability.

It remains to low-bound how many times B must run $\hat{P}$ in order to obtain a sufficiently good PCP. Suppose that B ran through all of V's coin tosses, which are identified with those used by the initial PCP verifier. Then, provided that no "breaks" were found, the resulting PCP would cause the verifier to accept with at least the probability that $\hat{P}$ makes a Type 1 response to the next question by V. Here we implicitly assume (at least this assumption trivializes the claim) that V's coin tosses specifies what bits of the PCP he will look at - all known PCPs have this property.

We suspect that the above bound on how often B needs to invoke $\hat{P}$ to construct a PCP is sufficient for known PCP's. That is, the verifiers require randomness that is typically within a constant of optimal. But to make sure, we note that it also suffices for B to run $\hat{P}$ $O(n^c)$ times, where c is a sufficiently large constant (2 may suffice) and n is the size of the PCP. Here, no effort has been made to reduce the $O(n^c)$ - sharper bounds are possible but are not needed. The intuition for why this is true is that any portion of the PCP that is not revealed after so many trials cannot have been needed to make the verifier accept with the given probability. Thus, setting these entries to 0 will not significantly damage the resulting PCPs acceptance probability.

Recall that in each recursive proof $\hat{P}$ that he could have given a good response to the preceding question. Using the PCP construction procedure given above and the the self-correcting properties of [3]-style PCPs (and by extension, those of [10]) the breaker can, whenever the verifier still has a sufficiently large chance of accepting, actually produce a good response. The breaking can then proceed recursively.

Of course, one must be a little more careful than described above. The PCP reconstruction step will only work when it holds that for a random challenge $\hat{P}$ will cause V to accept the current proof. One must perform a simple averaging argument to verify that if the acceptance probability for the entire protocol is sufficiently high than for the most of the time it will hold that most of the time $\hat{P}$ will deal with the next challenge so as to make V accept with high probability. Further details are omitted.

Theorem 3. For the protocol described above, there exists a simulator S with the following property. Let $\mathcal{P}$ be a correct proof for T, and let $\hat{V}$ be a malicious verifier. Then given T, $|\mathcal{P}|$ and black-box access to $\hat{V}$, S can generate the view obtained by $\hat{V}$ in time L^c for some explicitly computable constant c. □

Finally, we observe that a simulator can trivially simulate the distribution on C', simply by generating L random blobs (0-blobs and 1-blobs are identically distributed) and creating the hash-tree for them. We can no longer use the simulator for the proof system given in [21], since it implicitly relied on the multi-phase nature of the standard protocol for zero-knowledge proofs on committed bits. However, if one uses the [3] protocol, then $\hat{V}$ is only allowed to look at a constant number of bits. The simulator exploits this fact by committing to a sequence of random bits. With constant probability, the query made by $\hat{V}$ will be satisfiable by the proof, at which point it the simulation of the zero-knowledge proof will be perfect. Here the proof is simplified by the fact that the bit commitments are information theoretically secure. Note that to argue that the concatenation of the simulation phases is zero-knowledge, we can use the fact that our protocols are auxiliary input zero-knowledge, a weaker property than black-box zero-knowledge.

7 Acknowledgments

We would like to acknowledge Lance Fortnow and Carsten Lund for valuable information about the self correction properties of PCPs. Dan Spielman provided early and invaluable information on his work with Polishchuk, which greatly improved the results of an earlier version of this manuscript.

References

1. S. Arora and S. Safra. Probabilistic Checking of Proofs, *Proceedings of STOC 1992.*
2. S. Arora and T. Leighton and B. Maggs. On-line algorithms for path selection in a nonblocking network. *Proceedings of STOC 1990,* pp. 149–158
3. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Hardness of Approximation Problems, *Proceedings of STOC 1992.*
4. G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge, *J. Comput. System Sci.* **37** (1988), 156–189.
5. L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Proofs, *Proceedings of FOCS 1990*
6. L. Babai, L. Fortnow, L. Levin and M. Szegedy. Checking computation in polylogarithmic time. *Proceedings of STOC 1991.*
7. M. Bellare, S. Goldwasser, C. Lund, A. Russell, "Efficient probabilistic checkable proofs and applications to approximation," *Proc. 25th STOC, 1993, pp. 294-304.*
8. M. Bellare, P. Rogaway. Random Oracles are Practical: A paradigm for Designing Efficient Protocols, *Proc. First ACM Conference on Computer and Communications Security,* ACM, November 1993.

9. I. Damgård, Non-interactive Circuit Based Proofs and Non-Interactive Perfect Zero-Knowledge with Preprocessing, Advances in Cryptology - EUROCRYPT 92, pp. 341–355.
10. C. Dwork, U. Feige, J. Kilian, M. Naor and S. Safra. Low communication 2-Prover Zero-Knowledge Proofs for NP. *Advances in Cryptology - Crypto '92*, pp. 215–227.
11. U. Feige, A. Fiat and A. Shamir. Zero knowledge proofs of identity, *Proceedings of* 19^{nd} *Annual Symposium on the Theory of Computation, 1987, pp. 210-217.*
12. U. Feige, S. Goldwasser, L. Lovasz, M. Safra and M. Szegedy. Approximating Clique is Almost NP-Complete, *Proceedings of* 32^{nd} *Annual Symposium on Foundations of Computer Science, 1991, pp. 2-12.*
13. U. Feige, D. Lapidot and A. Shamir. *Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String,* Proceedings of the 22th Annual Symposium on the Theory of Computation, 1990, pp. 308–317
14. C. Bennett. personal communication via Gilles Brassard.
15. M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability, *Proceedings of STOC 1988.*
16. De Santis, A., S. Micali and G. Persiano, "Bounded-Interaction Zero-Knowledge proofs," *Advances in Cryptology - Crypto '88*
17. U. Feige, S. Goldwasser, L. Lovász, S. Safra and M. Szegedy. Approximating clique is almost NP-Complete. *Proceedings of FOCS 1991,* pp. 2-12.
18. L. Fortnow, J. Rompel, and M. Sipser. On the Power of Multi-Prover Interactive Protocols, *Proceedings of Structure 1988.*
19. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solution to Identification and Signature Problems. *Advances in Cryptology - Crypto '86*, pp. 186–189.
20. S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Comput.* **18** (1989), 186–208.
21. J. Kilian. A note on efficient zero-knowledge proofs and arguments, *Proceedings of STOC 1992.*
22. J. Kilian On the complexity of bounded interaction and noninteractive proofs. *Proceedings of FOCS 1994.*
23. C. Lund, L. Fortnow, H. Karloff, and N. Nisan. The polynomial-time hierarchy has interactive proofs, *Proceedings of STOC 1990,* pp. 2-10.
24. R. Merkle. A Certified Digital Signature. *Proceedings of Crypto '89,* pp. 218–238.
25. S. Micali. Computationally Sound Proofs, *Proceedings of FOCS 1994.*
26. M. Naor, R. Ostrovsky, R. Venkatesan and M. Yung. Perfect Zero-Knowledge Arguments for NP can be Based on General Complexity Assumptions. *Advances in Cryptology - Crypto '92,* pp. 196–214.
27. A. Polishchuk and D. Spielman. Nearly-linear Size Holographic Proofs. *Proceedings of STOC 1994.*
28. S. Rudich, Personal communication via Gilles Brassard.
29. A. Shamir. IP = PSPACE, *Proceedings of FOCS 1990,* IEEE.