Instance-Hiding Proof Systems*

D. Beaver† J. Feigenbaum‡ R. Ostrovsky§ V. Shoup¶

July 26, 1993

Abstract

We define the notion of an *instance-hiding proof system* (ihps) for a function f; informally, an ihps is a protocol in which a polynomial-time verifier interacts with one or more all-powerful provers and is convinced of the value of $f(x)$ but does not reveal the input x to the provers. We show here that a function f has a multiprover ihps if and only if it is computable in FNEXP. We formalize the notion of zero-knowledge for ihps's and show that any function that has a multiprover ihps in fact has one that is perfect zero-knowledge. Under the assumption that one-way permutations exist, we show that f has a one-prover, zero-knowledge ihps if and only if it is in FPSPACE and has a one-oracle instance-hiding scheme (ihs).

1 Introduction

In this paper, we show that every function that has a multiprover interactive proof system in fact has one in which the verifier does not learn the proof, and the provers do not learn what they are proving.

Consider interactive protocols involving a probabilistic polynomial-time verifier V and $m \geq 1$ powerful provers $P_1, \ldots, P_m$ in which the provers are allowed to communicate with V but not with each other.

*The results in Sections 3 and 4 of this paper were presented in preliminary form at Crypto '90, Santa Barbara, CA, August 1990. Those in Section 5 were presented in preliminary form at Asiacrypt '91, Fujiyoshida, Japan, November 1991.

†313 Whitmore Laboratory, The Pennsylvania State University, University Park, PA 16802 USA, beaver@cs.psu.edu. Work done at Harvard University, supported in part by NSF grant CCR-870-4513.

‡AT&T Bell Laboratories, Room 2C473, 600 Mountain Avenue, Murray Hill, NJ 07974-0636 USA, jf@research.att.com.

§University of California at Berkeley Computer Science Division and International Computer Science Institute at Berkeley, Berkeley, CA 97420 USA, rafail@melody.berkeley.edu. Supported by an NSF Postdoctoral Fellowship. Part of this work was done at MIT, supported by an IBM Graduate Fellowship, and part at AT&T Bell Laboratories.

¶University of Toronto, Computer Science Department, Toronto, Ontario M5S 1A4, CANADA, shoup@cs.toronto.edu. Work done at AT&T Bell Laboratories as a Postdoctoral Fellow in Theoretical Computer Science.

In an *interactive proof system* (ips) for a language L (cf. [4, 16, 9]), the input x is on a shared tape, accessible to the verifier and provers. If x is in L, the ips allows V to obtain convincing evidence of this fact. Because it obtains this evidence, V need not trust the provers to behave correctly. One may also consider ips's for *functions* f in which the verifier learns $f(x)$ and obtains convincing evidence of the correctness of this value [14].

It is known [19, 25] that the class IP of languages recognized by 1-prover ips's is equal to the complexity class PSPACE. Furthermore, it is shown in [3] that the class MIP of languages recognized by multiprover ips's is equal to the complexity class NEXP = NTIME(2^{poly}).

In an *instance-hiding scheme* (ihs) for a function f (cf. [1, 6, 7]), the input x is on a private tape, accessible only to the querier V. The protocol allows V to obtain the value $f(x)$ without revealing to any P_i any information about x (other than its length); however, V does not necessarily obtain any evidence of the the correctness of this value. In this model, V does not entrust any information about x to the provers, but it does have to trust the provers to answer questions correctly. Because their answers are trusted, the powerful players are referred to as "oracles" in [1, 6, 7], rather than "provers."

Beaver and Feigenbaum [6] have shown that *all* functions f have multioracle ihs's, thus settling a question of Rivest [24].

In this paper, we introduce the notion of an *instance-hiding proof system* (ihps) for a function f and characterize the functions that have such systems. An ihps is similar to an ihs, except that along with the value $f(x)$, the protocol allows V to obtain convincing evidence of the correctness of this value. Thus, V need not entrust any information about x to the P_i's, nor need it trust the P_i's to behave correctly. Adopting the terminology of previous works on ips's and ihs's, we refer to the P_i's as "provers" in an ihps and as "oracles" in an ihs that is not a proof system.

Let FNEXP denote the class of (single-valued) total functions computable by nondeterministic exponential-time Turing-machine transducers. The restriction of FNEXP to Boolean functions consists of the characteristic functions of languages in NEXP $\cap$ coNEXP. We prove the following.

Theorem 1 *Every function $f \in$ FNEXP has an instance-hiding proof system.*

The fact that MIP = NEXP implies that Theorem 1 is the best possible, since if the function f has an ihps, each output bit of f is the characteristic function of a language in MIP $\cap$ coMIP, and hence f is in FNEXP.

We also define in a natural way the notion of zero-knowledge for ihps's and show that any function that has a multiprover ihps in fact has one that is zero-knowledge. In any type of ips, the definition of zero-knowledge should capture the intuitive idea that the provers do not trust the verifier to behave correctly and that the verifier is not to be entrusted with any information other than the fact being proved. The definition of zero-knowledge for ips's for a function f on input x captures the intuitive idea that the verifier—even a misbehaving one—ihps for a function f, the provers do not know the input x, nor can they infer anything about x (except its size) from the messages they receive from the verifier. Thus the provers cannot hope to prevent a verifier from learning, say, $f(x')$ instead of $f(x)$, where $|x'| = |x|$. Our definition of a zero-knowledge ihps captures the intuitive idea that the verifier—even a misbehaving one—learns the value of f at exactly one input of length n and nothing else. Thus, in a zero-knowledge ihps, the verifier and the provers do not trust each other to behave correctly, nor do they entrust each other with any non-essential information: The provers learn nothing about x, and the verifier learns nothing but the value $f(x)$.

For the purpose of constructing zero-knowledge protocols, it is convenient to assume, as in [9], that the provers have access to a shared random tape that is not accessible to V.

We prove the following.

Theorem 2 *Every function $f \in FNEXP$ has a perfect zero-knowledge instance-hiding proof system.*

We remark that the notion of *private/adaptive checker*, which was introduced by Blum, Luby and Rubinfeld [11], may be viewed as a restricted form of ihps in which the provers are only asked questions of the form "what is $f(y)$?"

The ihps's exhibited in the proofs of Theorems 1 and 2 require a polynomial number of provers. We also address the power of *one-prover* ihps's. Let FPSPACE denote the class of functions computable in polynomial space. We prove the following.

Theorem 3 *Assume that one-way permutations exist. Then f has a one-prover, zero-knowledge, instance-hiding proof system if and only if $f \in$ FPSPACE and has a one-oracle instance-hiding scheme.*

The rest of this paper is organized as follows. In Section 2 we give the formal definitions of "instance-hiding scheme," "instance-hiding proof system," and "zero-knowledge" and present a normal form for multiprover ips's. Sections 3 and 4 contains the proofs of Theorems 1 and 2, respectively. Section 5 contains the proof of Theorem 3. In Section 6, we state some open problems.

Most of these results first appeared in our Technical Memoranda [8, 12].

2 Preliminaries

We now formally define ihs's, ihps's, and the notion of zero-knowledge that is appropriate in our setting, where the prover does not know the input. The intuition behind these definitions can be found in Section 1.

Let V, P_1, ..., P_m be a set of interactive Turing Machines; $m = m(n)$ is a polynomially bounded function of n, the length of the input. As in ordinary MIP, the verifier V is a probabilistic polynomial-time Turing Machine, the provers $P_1, \ldots, P_m$ are computationally unbounded, and the verifier can communicate reliably and privately during the protocol with each of the provers, but the provers cannot communicate with each other. Also as in ordinary MIP, the provers have a shared random tape to which the verifier does not have access; however, this random tape is only required in the construction of zero-knowledge ihps's. Unlike ordinary MIP, the input x in our setting is known only to the verifier. The output produced by V after interacting with a set $\{P_i^*\}$ of (possibly misbehaving) provers is an element of the set $Range(f) \cup \{reject\}$ and is denoted by $(V(x), P_1^*, \ldots, P_m^*)$. We assume without loss of generality that $|f(x)|$ is polynomially bounded in $|x|$ and that $reject \notin Range(f)$.

For each prover P_i, the transcript $T(V, P_i, x)$ of messages sent between V and P_i on input x is a random variable, and its distribution is induced by the random coin-tosses and algorithms of the verifier and provers. The verifier's *view* of the interaction, denoted $View(V, x)$, is $\langle T(V, P_1, x), \ldots, T(V, P_m, x), R \rangle$, where R is the distribution of V's random coins.

Definition 2.1 [1, 6]: *The protocol* $(V, P_1, \ldots, P_m)$ *is an* $\boldsymbol{m}$**-oracle instance-hiding scheme (ihs) for the function** f *if it satisfies the following properties.*

(i) For all x, $\text{Prob}((V(x), P_1, \ldots, P_m) = f(x)) > 3/4$.

(ii) For all inputs x *and* x' *with* $|x| = |x'|$ *and all* i, $1 \le i \le m$, *the random variables* $T(V, P_i, x)$ *and* $T(V, P_i, x')$ *are identically distributed.*

Definition 2.2 *The protocol* $(V, P_1, \ldots, P_m)$ *is an* $\boldsymbol{m}$**-prover instance-hiding proof system (ihps) for the function** f *if it satisfies the following properties.*

(i) For all x,

$$\text{Prob}((V(x), P_1, \ldots, P_m) = f(x)) > 3/4.$$

(ii) For all x *and all* $P_1^*, \ldots, P_m^*$,

$$\text{Prob}((V(x), P_1^*, \ldots, P_m^*) \notin \{f(x), reject\}) < 1/4.$$

(iii) For all $P_1^*, \ldots, P_m^*$, *for all inputs* x *and* x' *with* $|x| = |x'|$, *for* $1 \le i \le m$, *the random variables* $T(V, P_i^*, x)$ *and* $T(V, P_i^*, x')$ *are identically distributed.*

In Definition 2.2, conditions (i) and (ii) capture the notion of a proof system for a function. Condition (iii) captures the notion of instance-hiding—the protocol leaks no more than the length of x to any individual, isolated prover. However, pairs of transcripts, say $T(V, P_i, x)$ and $T(V, P_j, x)$ may be dependent. Thus pairs of provers must be kept physically separated for two reasons: As in ordinary multiprover ips's, colluding provers could cause the verifier to accept a wrong value for $f(x)$; as in ordinary ihs's, colluding provers could compute more information about x than its size. A more general definition of instance-hiding is given in [1]; if we restrict attention to the case in which at most the length of the instance is leaked to the provers, then condition (ii) of Definition 2.1 and condition (iii) of Definition 2.2 are equivalent to the definition in [1]. As usual, the probabilities 3/4 and 1/4 may be replaced by $1 - 1/poly$ and $1/poly$ without changing the set of functions that satisfy the definitions.

Definition 2.3 *An instance-hiding proof system $(V, P_1, \ldots, P_m)$ for the function f is* **computational** *(resp.* **statistical, perfect***)* **zero-knowledge** *if, for any probabilistic polynomial-time verifier V^*, there is a probabilistic, expected-polynomial-time oracle machine M_{V^*} (called the* **simulator***) with the following property. During its execution on input x, M_{V^*} may make exactly one query to an f-oracle, and the query must have length $|x|$. The distribution of the simulator's output $M_{V^*}(x)$ is* **computationally indistinguishable from** *(resp.* **statistically indistinguishable from, the same as***) $View(V^*, x)$.*

In the following normal form for MIP protocols, the verifier's role is extremely limited. This is technically convenient for the proofs of Theorems 1 and 2.

Proposition 2.1 *Any language in $L \in MIP$ has a 2-prover ips with the following structure.*

Protocol N.

N1. *V sends a random string r to P_1, who sends a response a_1.*

N2. *V sends a random string r' to P_1, who sends a response a_2.*

N3. *V sends a_2 to P_2, who sends a response a_3.*

N4. *V computes an NC^1 acceptance predicate accept$(x, r, r', a_1, a_2, a_3)$.*

On inputs $x \in L$, V accepts with probability 1. On inputs $x \notin L$, V rejects with probability at least $1/poly$.

Proof: By the "completeness theorem" of [9] and the "probabilistic oracle machine" characterization of [13], we may assume that there is a polynomial-time deterministic oracle machine ϕ such that

1. for all $x \in L$, there exists an oracle E such that for all r, $\phi^E(x, r) = 1$;
2. for all $x \notin L$, for all oracles E, the probability that $\phi^E(x, r) = 1$ for randomly chosen r is at most 1/3.

The provers choose an oracle E. The verifier V selects a random string r and sends it to P_1. Then P_1 computes a response a_1 that encodes the entire computation of $\phi^E(x, r)$, including all the oracle queries $q_i, i = 1 \ldots m$ and oracle answers $s_i := E(q_i), i = 1 \ldots m$. V sends a random string r' to P_1 that represents a random number $i(r')$ between 1 and m. P_1 sends a response $a_2 := q_{i(r')}$ to the verifier. Now V sends a_2 to P_2. P_2 sends a response $a_3 := E(a_2)$. The acceptance predicate $accept(x, r, r', a_1, a_2, a_3)$ just checks that (1) a_1 encodes a valid accepting computation (imposing no constraints on the oracle responses), (2) $a_2 = q_{i(r')}$, and (3) $a_3 = s_{i(r')}$.

The argument that the verifier accepts or rejects with the desired probability is found in [13]. ▮

3 Proof of Theorem 1

3.1 Arithmetization of Boolean Functions

Let $f : \{0,1\}^* \to \{0,1\}$ be any Boolean function.

For any $n \geq 1$, we denote by K a fixed finite field such that $n + 2 \leq |K| = O(n)$. Such a field can be constructed deterministically in polynomial time. In what follows, $\alpha_1, \ldots, \alpha_{n+1}$ will denote fixed nonzero elements in K.

We consider the restriction of f to inputs of length n. We define a polynomial $g \in K[X_1, \ldots, X_n]$ in the following way. For each $A = (a_1, \cdots, a_n) \in \{0,1\}^n$, let

$$\delta_A(X_1, \ldots, X_n) = \prod_{i=1}^{n} (X_i - \overline{a_i})(-1)^{\overline{a_i}} \in K[X_1, \ldots, X_n].$$

So, for each $(x_1, \cdots, x_n) \in \{0,1\}^n$, $\delta_A(x_1, \ldots, x_n)$ is 1 if $x_i = a_i$, for $1 \leq i \leq n$, and it is 0 otherwise. Next, let

$$g(X_1, \ldots, X_n) = \sum_{A \in \{0,1\}^n} f(A)\delta_A(X_1, \ldots, X_n).$$

We may of course view g as a function mapping K^n into K in the usual way. We make the following simple observations.

Proposition 3.1

(i) $g(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ *for all* $(x_1, \ldots, x_n) \in \{0,1\}^n$.

(ii) $\deg g \leq n$.

(iii) If $f \in$ *FNEXP, then* $g \in$ *FNEXP.*

A polynomial such as g that extends f to a larger arithmetic domain is referred to as an "arithmetization" of f.

3.2 An Instance-Hiding Proof System

Now suppose that $f \in$ FNEXP, and let g be its arithmetization. Let $x = (x_1, \ldots, x_n)$ be the input. Define the language L_g as follows. For u_1, ..., u_n, and v in K, $(u_1, \ldots, u_n, v) \in L_g$ if and only if $g(u_1, \ldots, u_n) = v$. Our ihps for f requires $2(n+1)$ provers on inputs of length n; we call the provers $P_1, P'_1, \ldots, P_{n+1}, P'_{n+1}$.

Protocol A.

A1. V picks $r_1, \ldots, r_n \in K$ at random, and computes $y(i,j) := r_j\alpha_i + x_j$ for $i = 1 \ldots n+1$ and $j = 1 \ldots n$. For $i = 1 \ldots n+1$: V sends $(y(i,1), \ldots, y(i,n))$ to provers P_i, P'_i.

A2. For $i = 1 \ldots n+1$: prover P_i computes $z_i := g(y(i,1), \ldots, y(i,n))$; P_i sends z_i to V.

A3. For $i = 1 \ldots n+1$: P_i, P'_i use Protocol N to prove to V that $(y(i,1), \ldots, y(i,n), z_i) \in L_g$. (The basic protocol is repeated polynomially many times in serial to reduce the error probability.)

A4. V interpolates the points (α_i, z_i), $i = 1 \ldots n+1$), to obtain a polynomial $w(X) \in K[X]$. The constant term of $w(X)$ is equal to $f(x)$.

One must verify that (1) Protocol A is a proof system, and (2) Protocol A is instance-hiding. To prove (1), observe that statements (i) and (ii) of Proposition 3.1 guarantee that the constant term of $w(X)$ in step A4 is indeed equal to $f(x)$. Also, observe that statement (iii) of Proposition 3.1 implies that $L_g \in$ NEXP; therefore, the result of [3] that NEXP = MIP shows that Protocol N can indeed be used in step A3. If the provers follow the protocol, the output of the verifier is always $f(x_1, \ldots, x_n)$; otherwise, the verifier will accept a wrong answer with exponentially small probability, provided that a suitable polynomial number of iterations are done in step A3.

Now to prove (2).

The transcript $T(V, P_i^*, x)$ consists of three parts:

(i) the values $y(i,j)$, $j = 1 \ldots n$,

(ii) a sequence of pairs r, r' of bit strings (arising from the many repetitions of Protocol N), and

(iii) P_i^*'s answers in steps A2 and A3.

Part (i) is just a sequence of independently and uniformly chosen elements of K, part (ii) is just a sequence of independently and uniformly chosen bit strings, and parts (i) and (ii) are completely uncorrelated. Furthermore, part (iii) is a function of parts (i) and (ii) and possibly the shared random bits of the provers. So it is clear that $T(V, P_i^*, x)$ is identically distributed for all x of length n.

The transcript $T(V, (P'_i)^*, x)$ also consists of three parts:

(i′) the values $y(i,j)$, $j = 1 \ldots n$,
(ii′) a sequence of messages corresponding to the responses a_2 in Protocol N, and
(iii′) $(P_i')^*$'s sequence of responses a_3.

Part (ii′) is a function of parts (i), (ii), and (iii), and possibly the shared random bits of the provers, and since they are identically distributed for all x of length n, so is part (ii′). Similarly, part (iii′) is a function of parts (i), (ii), (iii), (ii′), and possibly the provers' random bits, and therefore it is identically distributed for all x of length n.

Note that this simple argument relies heavily on the fact that prover P_i never learns any of the responses given by prover P_i'.

4 Proof of Theorem 2

The main new technical ideas required to convert Protocol A in Section 3 to a perfect zero-knowledge ihps are the following.

1. We replace step A3 by a perfect zero-knowledge proof system.

2. In step A2, V learns the value of z_i, which it certainly could not compute on its own. We solve this problem by having P_i send instead $z_i' := z_i + h(\alpha_i)$, where $h(X)$ is a random polynomial over K of degree $\leq n$ and constant term zero. V then interpolates the points (α_i, z_i') in step A4; the constant term of the resulting polynomial has the correct value. As long as the verifier follows the protocol in step A1, z_i' is just the value of a random polynomial of degree $\leq n$ with constant term $f(x_1, \ldots, x_n)$, evaluated at α_i.

3. In step A1, a cheating verifier may not follow the protocol, and may send $y(i,j)$ values that do not correspond in a legitimate way to some point in $\{0,1\}^n$. In particular, this would invalidate our fix to A2, and it could also allow a cheating verifier to learn the value of g at any point in K^n, which we do not want to allow. We prevent this by using a distributed function evaluation protocol that will reveal the true values of $z_1', \ldots, z_n'$ to V only if the $y(i,j)$ values correspond to some input value in $\{0,1\}^n$.

The remainder of this section supplies the details of these steps.

4.1 Building Blocks

We describe here the subprotocols that are used in our zero-knowledge proof system. These are building blocks that appear elsewhere in the literature or slight variations thereof. We present them here in some detail only to make our arguments about zero-knowledge and instance-hiding more transparent.

4.1.1 Bit Commitment

Using the shared random tape, simple bit commitment can be implemented in a very simple way as in [9]. In the bit commitment scheme, there are two protocols: a *bit commit protocol* and a *bit reveal protocol*.

Several provers can easily commit to the same random bit by simply taking that bit from the shared random tape. Only one of the provers actually executes the bit commit protocol. A group of provers can commit to a set of shared random bits in this fashion, and there is no need to "prove" to the verifier that they committed to the same ones—the ordinary reveal protocol will prevent any cheating.

A bit commit protocol allows a prover to choose the value of a bit b without revealing this value to the verifier. At a later time, a bit reveal protocol can be executed that will reveal to the verifier the value of b. The bit commit protocol reveals no information about b to the verifier, and the bit reveal protocol has the property that a cheating prover that attempts to reveal a value of b other than that committed to will be caught by the verifier with non-negligible probability. Following popular usage, we will use the phrase "put b in an envelope" to mean "perform the bit commit protocol for b." If a is a bit string, then the phrase "put a in an envelope" means "perform the bit commit protocol for each bit in a."

The bit commit protocol runs as follows. V sends two random bits r_1 and r_2 to P. P uses two random bits e_1 and e_2 from R and sends $y = \sigma(r_1, r_2, b) \oplus (e_1, e_2)$ to V. Here, $\sigma(r_1, r_2, b) = (r_2, b)$ if $r_1 = 0$, and it equals (b, r_2) if $r_1 = 1$.

The bit reveal protocol will always involve a special prover P' that is never used for any purpose other than to reveal committed bits. To reveal b, P' simply sends the pair (e_1, e_2) to V. Then V computes $y \oplus (e_1, e_2)$, checks that the bit r_2 has not been changed, and extracts the value of b.

When these protocols are used in a zero-knowledge proof system, a simulator that knows the value of b can easily generate the corresponding part of the transcript. Likewise, the protocols are easily employed in an ihps, because all the provers see are two random bits.

4.1.2 Multiple-use Notarized Envelopes

In [10] it is shown how to construct a *notarized envelope scheme* from a protocol for simple bit commitment. There are two protocols: a *notarized bit commit protocol* and a *prove protocol*. The notarized bit commit protocol allows a prover to commit a bit. If a is a bit string, we will use the phrase "put a in a notarized envelope" to mean "perform the notarized bit commit protocol for each bit in a." The prove protocol allows a prover to prove one NC^1 predicate[1] involving bits in notarized envelopes in such a way that no information about these bits is revealed (other than that implied by the truth of the

[1] In this paper, NC^1 means P-uniform NC^1.

predicate), and if the predicate is not true the verifier can catch a cheating prover with probability at least 1/poly.

We now present a notarized envelope scheme that is essentially the same as the one given in [10]. The restriction in [10] that a notarized envelope can be used in only one proof is just an artifact of the implementation that can easily be lifted. Instead of representing a bit b as the sum $c_1 \oplus c_2$, where c_1 and c_2 are committed using an ordinary bit commitment protocol (as done in [10]), we can represent b as the sum $c_1 \oplus \cdots \oplus c_m$, which allows b to be used in $m-1$ proofs.

To put b in a notarized envelope, the prover P chooses random bits c_1, …, c_m subject to the condition $c_1 \oplus \cdots \oplus c_m = b$ and commits the bits c_1, …, c_m using the ordinary bit commit protocol.

Now suppose that $f(x_1, \ldots, x_k)$ is an NC^1 predicate and that b_1, …, b_k are bits in notarized envelopes. A prover P wants to convince V that $f(b_1, \ldots, b_k)$ is true. Suppose that $b_i = c_{i1} \oplus \cdots \oplus c_{im}$, where as above the c_{ij}'s are in ordinary envelopes. Consider the predicate

$$g(c_{11}, \ldots, c_{1m}, \ldots, c_{k1}, \ldots, c_{km}) \equiv f(b_1, \ldots, b_k).$$

Clearly g is itself an NC^1 predicate. By Barrington's theorem [5], there is a branching program N that realizes g. On a given input, N determines a sequence σ_1, …, σ_l of permutations in S_5, where each σ_j is determined by the value of a single input bit. Moreover, the product $\prod_j \sigma_j$ is equal to the identity in S_5 if $g = 0$ and is equal to some fixed nonidentity element in S_5 otherwise.

To prove that $f(b_1, \ldots, b_k)$ is true, prover P selects random permutations ρ_1, …, ρ_{l-1} and computes $\tau_j = \rho_{j-1}^{-1} \sigma_j \rho_j$, $j = 1 \ldots l$. (Here ρ_0 and ρ_l are the identity permutation.) P then puts all of the σ_j's, ρ_j's, and τ_j's in ordinary envelopes.

V chooses a bit r at random. If $r = 0$, V demands that τ_1, …, τ_l be revealed and checks that $\prod_j \tau_j \neq 1$. If $r = 1$, v chooses a random integer $j \in \{1, \ldots l\}$ and demands that τ_j, ρ_{j-1}, σ_j, ρ_j, and the input bit corresponding to σ_j be revealed. He then checks that σ_j was selected according to the input bit and that $\tau_j = \rho_{j-1}^{-1} \sigma_j \rho_j$.

The fact that these protocols satisfy the properties of a notarized envelope scheme is proven in [10].

When the protocols are used in a zero-knowledge proof system, a simulator can generate the corresponding part of the transcript provided it can generate with the correct distribution the values of all bits revealed during the protocols. But this is easy to do: As long as each bit b_i is involved in no more than $m-1$ executions of the prove protocol, the c_{ij}'s that are actually revealed are distributed independently and uniformly. Furthermore, the distribution of the revealed permutations is easily simulatable.

When the protocols are used in an ihps, all any prover sees are the random bits sent during the bit commit protocols and the random bits r and random integers j sent during the prove protocols.

4.1.3 Distributed Function Evaluation

We will need a protocol for the following simple version of distributed function evaluation. Let $F(u_1, \ldots, u_m)$ be an NC^1 function, where the u_i's are bit strings. We have provers $P_1, \ldots, P_m$ and verifier V. Initially, each P_i knows u_i; some of the u_i may be known to V, whereas others may be in notarized envelopes and unknown to V. At the end of the protocol, V should learn nothing but the value $F(u_1, \ldots, u_m)$, rejecting a wrong answer with probability at least 1/poly, and each of the P_i's should learn nothing.

We give an implementation of the protocol using a variant of Kilian's oblivious NC^1 circuit evaluation protocol [17]. Without loss of generality, we may assume that F is a Boolean-valued function. Recall the branching program representation of F and the notation of Section 4.1.2.

The function evaluation protocol runs as follows. The provers put shared random permutations $\rho_1, \ldots, \rho_{m-1}$ in notarized envelopes. Each prover P_i computes and sends to V the permutations τ_j corresponding to each input bit of u_i. For each such τ_j, prover P_i proves to V that τ_j was computed correctly. This correctness predicate is an NC^1 predicate involving τ_j, the pair of permutations ρ_{j-1} and ρ_j (which are in notarized envelopes), and the corresponding input bit (which may be in a notarized envelope); therefore, the prove protocol for notarized envelopes described above may be used. Once V has received all such permutations, V can multiply them together to obtain the value of F.

When this protocol is used in a zero-knowledge proof system, a simulator can generate the corresponding part of the transcript provided it known the value of the function F. In this case, the simulator can generate the sequence of permutations τ_1, ..., τ_l so that they are uniformly distributed subject only to the condition that their product has the correct value (determined by the value of F). The simulator can generate the part of the transcript corresponding to the notarized envelope scheme as described in Section 4.1.2.

To see why this protocol can be used in an ihps, note that the verifier does not send anything to the prover except during the prove protocol; as we have seen in Section 4.1.2, this reveals nothing about the input except its length.

4.2 A Zero-Knowledge MIP Protocol

Zero-knowledge multiprover ips's were introduced in [9]. There it is shown that any language in MIP has a *statistical* zero-knowledge multiprover ips. Kilian [18] has shown that, in fact, any language in MIP has a *perfect* zero-knowledge multiprover ips.

In this section, we present a different proof that every language in MIP has a perfect zero-knowledge multiprover ips. Our protocol can easily be embedded as a subprotocol in an ihps in which the input bits to the subprotocol are not on a shared input tape but rather are in notarized envelopes or are known initially only to the verifier.

Our protocol is a perfect zero-knowledge version of Protocol N (see Section 2). The basic idea is the following. The provers will put their responses in notarized envelopes, and the verifier will use the distributed function evaluation protocol (see Section 4.1.3) to evaluate the acceptance predicate. However, difficulties arise in step N3—the response a_2 must somehow be passed to P_2, (1) without letting V know the value of a_2, and (2) without relying on V to follow the protocol. The first problem is solved by having P_1 send $a_2' := a_2 \oplus e$ to V, where e is a shared random string (of length equal to that of a_2) that is put in a notarized envelope at the beginning of the protocol. The second problem is solved by modifying the acceptance predicate so that if V sends anything other than a_2' to P_2, the acceptance predicate becomes trivially true, and hence V can not possibly gain any information by trying to cheat in this way. Since P_2 knows a_2' and e, it can recover a_2 and compute its response a_3.

Here are the details. Let

$$accept'(x, r, r', a_1, a_2', e; c, a_3) = (a_2' \neq c) \vee accept(x, r, r', a_1, a_2' \oplus e, a_3).$$

The distributed function evaluation protocol will be used in the following protocol to evaluate *accept'*, with P_1 supplying the arguments x, r, r', a_1, a_2', e and P_2 supplying the arguments c, a_3.

Protocol Z.

Z1. The provers put a shared random string e in a notarized envelope.

Z2. V sends r to P_1; P_1 puts the response a_1 in a notarized envelope.

Z3. V sends r' to P_1; P_1 sends $a_2' := a_2 \oplus e$ to V.

Z4. V sends $c := a_2'$ to P_2; P_2 puts the response a_3 in a notarized envelope.

Z5. Evaluate the predicate $accept'(x, r, r', a_1, a_2', e; c, a_3)$ using the distributed function evaluation protocol.

If $x \in L$, the verifier will always accept; otherwise, the verifier will reject with probability at least 1/poly. To reduce the error probability, the protocol can be repeated.

This protocol can be embedded in a larger protocol in which some of the input bits are in notarized envelopes. Furthermore, if an input bit b is initially known only to V, V can send $b_1 := b$ to P_1 and $b_2 := b$ to P_2, and the provers can effectively guarantee that $b_1 = b_2$ by replacing the acceptance predicate $\phi(\cdots b \cdots)$ with $(b_1 \neq b_2) \vee \phi(\cdots b_1 \cdots)$. Both of these modifications will be utilized in what follows.

Note that a simulator can generate the transcripts produced by Protocol Z as follows. The transcripts of the notarized commit protocols of steps Z1, Z2, and Z4 can be simulated as described in Section 4.1.2. The string a_2' sent to V in step Z3 is just a random bit string that is easily generated. The distributed function evaluation in step Z5 can be simulated, because the value of the function *accept'* is always true, even if the verifier cheats.

When this protocol is embedded in an ihps, prover P_1 sees the bit strings r and r', which are just random bit strings, and prover P_2 sees the bit string a_2', which is itself a function only of r, r', and the provers' shared random tape. The provers also see the messages sent by V during the bit commit and distributed function evaluation protocols, but it has already been seen that these messages reveal nothing about the verifier's input except its length.

Up to now it has been implicitly assumed that a third prover is dedicated to the bit reveal protocol. This assumption simplifies the protocol, but those researchers whose budget will allow them to purchase only two provers will be happy to know that two provers will suffice. Very briefly, we can't safely use prover P_2 for revealing committed bits after it has received the message c from V in step Z4. However, P_1 can execute its part of the distributed function evaluation protocol before this occurs, allowing P_2 to be used to reveal bits committed by P_1 during this process. We leave the rest of the details to the interested reader.

4.3 A Zero-Knowledge Instance-Hiding Proof System

We now have everything we need to modify Protocol A to obtain a perfect zero-knowledge ihps. We shall the use notation introduced in Section 3.

Let L_g' be the language defined as follows. For $u_1, \ldots, u_n, v, \alpha \in K$, and $h \in K[X]$ a polynomial of degree n with constant term zero (represented as a list of coefficients), $(u_1, \ldots, u_n, v, \alpha, h) \in L_g'$ if and only if $g(u_1, \ldots, u_n) + h(\alpha) = v$.

Let $[y(i,j)]$ $(i = 1 \ldots n+1, j = 1 \ldots n)$ be a collection of elements in K. We shall say that the $y(i,j)$ satisfy the *linearity condition* if there exist (necessarily unique) elements $r_1', \ldots, r_n' \in K$ and $x_1', \ldots, x_n' \in \{0,1\}$ such that $y(i,j) = r_j'\alpha_i + x_j'$ for each $y(i,j)$. It is easy to show that the linearity condition is an NC^1 predicate, and that, if this condition is satisfied, the r_j' and x_j' can be recovered in polynomial time.

Let $[z_i']$ $(i = 1 \ldots n+1)$ be a collection of elements in K. Let the function

$$F([y(i,j)]; [z_i'])$$

be defined as follows. If the $y(i,j)$ satisfy the linearity condition, then $F = (z_1', \ldots, z_{n+1}')$; otherwise, $F = (0, \ldots, 0)$. It is easy to verify that F can be computed in NC^1.

Protocol B.

B1. V picks $r_1, \ldots, r_n \in K$ at random, and computes $y(i,j) := r_j\alpha_i + x_j$ for $i = 1 \ldots n+1$ and $j = 1 \ldots n$. For $i = 1 \ldots n+1$, V sends $(y(i,1), \ldots, y(i,n))$ to provers P_i, P_i'.

B2. The provers put the coefficients of a shared random polynomial h over K of degree $\leq n$ with constant term zero in notarized envelopes.

B3. For $i = 1 \ldots n+1$, prover P_i computes $z_i' := g(y(i,1), \ldots, y(i,n)) + h(\alpha_i)$ and puts z_i' in a notarized envelope.

B4. For $i = 1 \ldots n+1$, P_i and P_i' use Protocol Z to prove to V in zero-knowledge that

$$(y(i,1), \ldots, y(i,n), z_i', \alpha_i, h) \in L_g'.$$

B5. Using the distributed function evaluation protocol, V evaluates

$$F([y(i,j)]; [z_i']),$$

obtaining $z_1', \ldots, z_{n+1}'$.

B6. V interpolates the points $(\alpha_i, z_i') \quad (i = 1 \ldots n+1)$ to obtain a polynomial $w'(X) \in K[X]$. The constant term of $w'(X)$ is the final result.

We must show that (1) Protocol B is a proof system, (2) Protocol B is instance-hiding, and (3) Protocol B is perfect zero-knowledge.

To prove (1), one can easily show that if the provers follow the protocol, the verifier will always learn the correct value $f(x_1, \ldots, x_n)$; otherwise, the verifier will accept the wrong answer with probability at most $1 - 1/\text{poly}$. The error probability can be decreased by iterating steps B2–B6 of the protocol.

Property (2) follows from the remarks made about the instance-hiding properties of the subprotocols discussed above and the fact that, in step B1, each prover sees a sequence of uniform, independently distributed elements of K.

To prove (3), we describe a simulator M. First M generates the random tape of the verifier V^*; it then runs Protocol B using V^* to generate the messages of the verifier. In steps B2 and B3, the transcripts of the bit commit protocols are simulated as described in Section 4.1.1. In step B4, the transcripts of Protocol Z are simulated as described in Section 4.2.

To simulate the conversation that occurs during step B5, the simulator first determines the value of F, which it does by testing the $y(i,j)$ values sent by V^* in step B1 for the linearity condition. If they do not satisfy this condition, then the value of F is $(0, \ldots, 0)$. Otherwise, the simulator recovers the corresponding values x_1', …, x_n' and r_1', …, r_n'. It then consults the oracle for f to obtain $f(x_1', \ldots, x_n')$.

Notice that the interpolating polynomial $w'(x)$ in step B5 can be written as $g(r_1'x + x_1', \ldots, r_n'x + x_n') + h(x)$, which is just a random polynomial whose constant term is $f(x_1', \ldots, x_n')$. Thus the simulator can generate the values $z_1', \ldots, z_n'$ with the correct distribution by first generating the coefficients of $w'(x)$ (the constant term is $f(x_1', \ldots, x_n')$

and the other coefficients are random) and then evaluating this polynomial at the points $\alpha_1, \ldots, \alpha_{n+1}$ to obtain $z'_i = w'(\alpha_i)$, $i = 1 \ldots n+1$. The value of F is $(z'_1, \ldots, z'_{n+1})$.

Once the simulator has generated the value of F, it can simulate the conversation that occurs in step B5 in the manner described in Section 4.1.3.

5 Proof of Theorem 3

The proof of this theorem uses *Combined Oblivious Transfer* (COT), introduced in [26]. In this two-party protocol, player S (the sender) has a private input x, and player R (the receiver) has a private input y. They are also given a polynomial-size circuit $C(\cdot, \cdot)$. In this paper, we allow S to be infinitely powerful and require R to be polynomially bounded. At the end of the protocol, R learns $C(x, y)$ (but does not learn anything about x that is not already revealed by $C(x, y)$), while S learns nothing. COT with reversed roles of S and R is also possible.

In [26], an implementation of COT based on factoring is given. In [15], an implementation based on trapdoor permutations is presented. Finally, in [22], it is shown how to implement COT based on any one-way function.

We separate the proof Theorem 3 into two lemmas. The assumption that one-way permutations exist is only needed for the second.

Lemma 5.1 *Suppose that $f \in$ FPSPACE and has a one-oracle instance-hiding scheme. Then f has an instance-hiding proof system.*

Proof: Let (P, V) be an ihs for f. We use it to construct (P', V'), an ihps for f. Let $n = |x|$, and let $m = m(n)$ be a polynomial upper bound on the number of moves in $T(V, P, x)$; we may assume without loss of generality that m is even. Recall that x is the private input to V'. Let r be the private random string of V' and $l = l(n)$ be the (polynomial) length of r.

Players P' and V' interact to produce a transcript t of the ihs (P, V) for instances of length n. This transcript must have two properties, described informally as follows: It must be instance-hiding, and, with high probability, it must be "correct for all x." We now describe these properties in more detail and show why P' can prove (interactively) to V' that t has them.

The instance-hiding property is just what we expect it to be: For any x_1 and x_2 of length n, the number of r's that cause P and V to produce t on input x_1 is the same as the number that cause P and V to produce t on input x_2. P' must prove that t has this property round by round, and he does so as follows. For his first move, V' simply computes the question $q_1 = V(x, r)$ and sends it to P'. Suppose that $p = (q_1, a_1, \ldots, q_{i-1}, a_{i-1})$ is the prefix of t that has been computed so far. That is, P' has just sent the answer a_{i-1} to V'. P' then proves (interactively) to V' that, for all x_1 and x_2 of length n, for all questions q_i, the number of r's, consistent with input x_1

and prefix p, such that $V(x_1, r, p) = q_i$ is the same as the number of r's, consistent with input x_2 and prefix p, such that $V(x_2, r, p) = q_i$. If, for any i, this interactive proof fails, V' just terminates the overall protocol and outputs "reject." These "subproofs" can be accomplished, because the statement "for all x_1 and x_2 of length n, for all questions q_i, the number of r's, consistent with input x_1 and prefix p, such that $V(x_1, r, p) = q_i$ is the same as the number of r's, consistent with input x_2 and prefix p, such that $V(x_2, r, p) = q_i$" is a PSPACE statement [19, 25]. V' (resp. P') computes the questions q_i (resp. the answers a_i) exactly as V (resp. P') computes them.

Intuitively, t has the desired correctness property if it is very likely that, on all inputs x of length n, V gets the correct answer $f(x)$ if the transcript produced is t. We will make this notion more precise shortly, but this intuitive description suffices to finish the high level description of the proof of the lemma. We show that, if P' and V' behave properly, then, with high probability, t has the correctness property. This in turn implies that, with high probability, V' will get the right answer $f(x)$. We also require P' to *prove* (interactively) to V' that t has the correctness property. This protocol takes place at the end, i.e., after P' sends his final answer $a_{m/2}$ and before V' computes his candidate for $f(x)$. Once again, it is possible to obtain such a protocol because the statement that t has the correctness property is a PSPACE statement; the fact that $f \in$ FPSPACE is needed here.

We now make the definition of correctness more precise. The difficulty that we must overcome has to do with the following facts about ihs's. For any transcript t, the number of random strings r that cause P and V to produce t on input x_1 is the same as the number of r's that cause P and V to produce t on input x_2; if these numbers were not the same, then (P, V) would not be instance-hiding. For any x, the probability that a random string r causes V to compute a wrong answer when interacting with P is less than $1/4$. However, it does *not* follow from the definition of an ihs that, for any t and any x, of the r's that cause P and V to produce transcript t on input x, the fraction that cause V to compute a wrong answer is less than $1/4$. If t is such that, for some x, a large fraction of the r's that cause P and V to produce t also cause V to compute a wrong answer, then t will not have the correctness property with respect to (P', V'). We must show that such inappropriate t's are produced only with low probability.

Call a random string r *bad* if there is at least one x for which V computes a wrong answer on input x and random string r. We may assume without loss of generality that an r chosen uniformly at random from $\{0,1\}^l$ is bad with probability less than 2^{-n}. If this is not true of the original ihs (P, V), we can replace it with another ihs that proceeds by running n^d copies of (P, V) and outputting the plurality of the answers. The existence of a suitable polynomial n^d has a standard proof using Chernoff bounds (see, e.g., [2, Appendix A]). For any transcript t, there is a set $R(t)$ of random strings r that are consistent with t; these are r's for which there exists at least one x such that t is produced on input x and random string r. Call a transcript t bad if at least $1/8$ of the elements of $R(t)$ are bad. Formally, t has the correctness property if it is not bad.[2]

We claim that, if P' and V' follow the specified protocol, the probability that they produce a transcript that has the correctness property is at least 7/8. Before proving this claim, we note that it implies that V' outputs a wrong answer with probability less than 1/4: V' may output a wrong answer if t is bad, which occurs with probability at most 1/8, or if t is not bad. If t is not bad, then it has the correctness property, and thus V' may output a wrong answer with probability less than 1/8; to do so, it must choose a bad r and fewer than 1/8 of the r's in $R(t)$ are bad.

Let A be the event that t is bad and B be the event that r is bad. We must show that $\text{Prob}(A) \leq 1/8$. The definition of a bad transcript is that $\text{Prob}(B \mid A) \geq 1/8$. Thus

$$\frac{\text{Prob}(B \cap A)}{\text{Prob}(A)} \geq 1/8.$$

This in turn implies that

$$\text{Prob}(A) \leq 8 \cdot \text{Prob}(B \cap A) \leq 8 \cdot \text{Prob}(B) < 8 \cdot 2^{-n}.$$

Note that the correctness property can be proven after a complete transcript t has been produced, but the instance-hiding property must be proven round by round. ∎

Lemma 5.2 *Assume that one-way permutations exist. Then any function f that has an instance-hiding proof system in fact has one that is computational zero-knowledge.*

Proof: Let (P, V) be an ihps for f. We use (P, V) to construct (P', V'), a zero-knowledge ihps for f. The notation x, m, r, and l is as in the proof of Lemma 5.1.

The proof system (P', V') consists of a set-up phase and an execution phase. In the set-up phase, V' and P' each choose at random a sequences of l bits; call these sequences $s_1 = s_{11}s_{12}\ldots s_{1l}$ and $s_2 = s_{21}s_{22}\ldots s_{2l}$, respectively. V' then commits s_1 to P' using the weak-committer/strong-receiver protocol in [21, 23], and P' commits s_2 to V' using the strong-committer/weak-receiver protocol in [20]. In the execution phase of the protocol, the sequence $r = r_1 r_2 \ldots r_l$, where $r_i = s_{1i} \oplus s_{2i}$, plays the role of the verifier's random input in (P, V). Intuitively, the execution phase is constructed by replacing each move of V by a COT protocol and replacing each answer of P with an encrypted answer. This can be accomplished if we have a COT protocol that can be simulated in a zero-knowledge fashion. Fortunately, Ostrovsky, Venkatesan, and Yung [22] provide such a protocol, assuming that one-way permutations exist. (If zero-knowledge simulators are not required, then a general one-way function suffices for COT [22].)

[2] The values 2^{-n} and 1/8 that we use in these definitions are chosen simply because they fit nicely with the error probability 1/4 used in the definition of ihs. They could clearly be replaced with other values that are "very small" and "reasonably large" respectively.

We now give more details. Let $E(y)$ denote the output of the strong-committer/weak-receiver bit-commitment protocol of [20]. Note that the definition of bit-commitment implies that the probabilistic polynomial-time verifier V' cannot distinguish between $E(y)$ and $E(\rho)$, where ρ is a random string of the same length as y. The polynomial-space prover P' can of course compute y from $E(y)$. Recall that the first step of (P, V) is for V to compute the first question $q_1 = V(x, r)$ and send it to P. In the first "step" of (P', V'), the players execute a COT protocol. P' secret input is s_2, and V' secret input is x and s_1; after the execution of the COT the output given to both players is $E(q_1)$. Given $E(q_1)$, P' first computes q_1 and then computes the answer a_1 that would be given by P; he then sends $E(a_1)$ to V'. More generally, suppose that $(E(q_1), E(a_1), \ldots, E(q_i), E(a_i))$ is the transcript prefix. The next "step" of (P', V') is a COT protocol to which P' supplies $(q_1, a_1, \ldots, q_i, a_i)$ and s_2, and V' supplies x and s_1. The output, which is given to both players, is $E(q_{i+1})$. P' then "decrypts" the question q_{i+1}, computes a_{i+1} using the same algorithm as P, and sends $E(a_{i+1})$ to V'. The last step is a COT protocol in which P' supplies s_2 and the entire transcript $(q_1, a_1, \ldots, q_m, a_m)$, V' supplies x and s_1, and the resulting output is the value $f(x)$. This time, of course, only V' gets the output.

The fact that (P', V') is an ihps follows directly from the fact that (P, V) is an ihps and from the definition of COT. To prove that (P', V') is zero-knowledge, we use the simulatability properties of the COT protocol of [21, 22] and the bit-commitment protocol of [20]. Let C_i denote the transcript of the i^{th} execution of COT that takes place in the overall execution $(P', V')(x)$. The entire transcript of an execution of $(P', V')(x)$ is thus of the form $(C_1, E(q_1), E(a_1), \ldots, C_m, E(q_m), E(a_m), f(x))$. Assume without loss of generality that all q_i and a_i have the same length $t = t(n)$, and let y_t denote the t-bit binary representation of a number y. By "round j" of an execution of $(P', V')(x)$, we mean the part that produces $(C_j, E(q_j), E(a_j))$. Our simulator $M_{V^*}(x)$ works as follows: For round j, it runs the simulator from [21, 22] of a COT protocol that has output $(E(2j_t), E((2j+1)_t))$. After round m, $M_{V^*}(x)$ makes one query to an f-oracle and outputs $f(x)$. The simulator's output for all the rounds is polynomial-time indistinguishable from the actual transcript; otherwise, there is a particular round j on which either $E(q_j)$ (resp. $E(a_j)$) is distinguishable from $E(2j_t)$ (resp. $E((2j+1)_t)$), contradicting the result of [20], or the simulator's output is distinguishable from the actual COT transcript C_j, contradicting the results of [21, 22]. ∎

Suppose that a function f satisfies Definition 2.2. We may conclude immediately that f satisfies Definition 2.1 and that f has a one-prover ips; it is shown in [19, 25] that the latter is equivalent to the conclusion that $f \in$ FPSPACE. Thus, combining Lemmas 5.1 and 5.2 yields a proof of Theorem 3.

6 Open Problems

The protocols given in Sections 3 and 4 require a polynomial number of provers. One may ask whether some fixed number (perhaps 2) of provers would suffice for all ihps's. Note that it is not even known whether a constant number of provers suffice for the construction of instance-hiding *schemes* for boolean functions in FNEXP—the best known upper bound for the number of provers is $n/\log n$ and is given by the generic construction in [7]. Thus obtaining general ihps's with a constant number of provers may be impossible and, in any case, seems to require a new technique.

Theorem 3 (in particular Lemma 5.2) was proven under the assumption that one-way permutations exist. It is not known how to prove this result assuming only the existence of general one-way functions.

References

[1] M. Abadi, J. Feigenbaum, and J. Kilian. On Hiding Information from an Oracle, *J. Comput. System Sci.* **39** (1989), pp. 21–50.

[2] N. Alon and J. Spencer. *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[3] L. Babai, L. Fortnow, and C. Lund. Nondeterministic Exponential Time has Two-Prover Interactive Protocols, *Computational Complexity* **1** (1991), pp. 3–40.

[4] L. Babai and S. Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes, *J. Comput. System Sci.* **36** (1988), pp. 254–276.

[5] D. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1, *J. Comput. System Sci.* **38** (1989), pp. 150–164.

[6] D. Beaver and J. Feigenbaum. Hiding Instances in Multioracle Queries, *Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, vol. 415, Springer, Berlin, 1990, pp. 37–48.

[7] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with Low Communication Overhead, *Advances in Cryptology – Crypto '90*, Lecture Notes in Computer Science, vol. 537, Springer, Berlin, 1991, pp. 62–76.

[8] D. Beaver, J. Feigenbaum, and V. Shoup. Hiding Instances in Zero-Knowledge Proof Systems, *AT&T Bell Laboratories Technical Memorandum*, April 12, 1990.

[9] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multiprover Interactive Proof Systems: How to Remove Intractability Assumptions, *Proc. 20th Annual Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1988, pp. 113–131.

[10] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything Provable is Provable in Zero-Knowledge, *Advances in Cryptology – Crypto '88*, Lecture Notes in Computer Science, vol. 403, Springer, Berlin, 1990, pp. 37–56.

[11] M. Blum, M. Luby, and R. Rubinfeld. Program Result Checking Against Adaptive Programs and in Cryptographic Settings, *Distributed Computing and Cryptography*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 2, American Mathematical Society, Providence, 1991, pp. 107–118.

[12] J. Feigenbaum and R. Ostrovsky. A Note on One-Prover, Instance-Hiding, Zero-Knowledge Proof Systems, *AT&T Bell Laboratories Technical Memorandum*, June 12, 1991.

[13] L. Fortnow, J. Rompel, and M. Sipser. On the Power of Multiprover Interactive Protocols, *Proc. 3rd Annual Structure in Complexity Theory Conference*, Institute of Electrical and Electronics Engineers Computer Society Press, Los Alamitos, 1988, pp. 156–161.

[14] Z. Galil, S. Haber, and M. Yung. Minimum-Knowledge Interactive Proofs for Decision Problems, *SIAM J. Comput.* **18** (1989), pp. 711–739.

[15] S. Goldreich, S. Micali and A. Wigderson. How to Play ANY Mental Game, *Proc. 19th Annual Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1987, pp. 218–229.

[16] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Comput.* **18** (1989), pp. 186–208.

[17] J. Kilian. Founding Cryptography on Oblivious Transfer, *Proc. 20th Annual Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1988, pp. 20–31.

[18] J. Kilian. *Uses of Randomness in Algorithms and Protocols*, MIT Press, Cambridge, 1990.

[19] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems, *J. Assoc. Comput. Mach.* **39** (1992), pp. 859–868.

[20] M. Naor. Bit Commitment Using Pseudo-Randomness, *J. Cryptology* **4** (1991), pp. 151–158.

[21] 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*, Lecture Notes in Computer Science, Springer, Berlin, to appear.

[22] R. Ostrovsky, R. Venkatesan, and M. Yung. Fair Games Against an All-Powerful Adversary, *SEQUENCES '91*, Positano, June, 1991. Final version to appear in DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 1993.

[23] R. Ostrovsky, R. Venkatesan, and M. Yung. Secure Commitment Against A Powerful Adversary, *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, vol. 577, Springer, Berlin, 1992, pp. 439–448.

[24] R. Rivest. *Workshop on Communication and Computing*, MIT, Cambridge, October, 1986.

[25] A. Shamir. IP = PSPACE, *J. Assoc. Comput. Mach.* **39** (1992), pp. 869–877.

[26] A. C. Yao. How to Generate and Exchange Secrets, *Proc. 27th Annual Symposium on Foundations of Computer Science*, Institute of Electrical and Electronics Engineers Computer Society Press, Los Alamitos, 1986, pp. 162–167.