Hiding Instances in Zero-Knowledge Proof Systems

(Extended Abstract)

Donald Beaver* Joan Feigenbaum† Victor Shoup‡

Abstract

Informally speaking, an *instance-hiding proof system* for the function f is a protocol in which a polynomial-time verifier is convinced of the value of $f(x)$ but does not reveal the input x to the provers. We show here that a boolean function f has an instance-hiding proof system if and only if it is the characteristic function of a language in NEXP $\cap$ coNEXP. We formalize the notion of zero-knowledge for instance-hiding proof systems with several provers and show that all such systems can be made perfect zero-knowledge.

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 computationally limited 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.

In an *interactive proof system* for a language L (cf. [12, 7, 10]) the input x is on a shared tape, accessible to the verifier and provers. If x is in L, the protocol allows V to obtain convincing evidence of this fact. Because it obtains this evidence, V need not trust the provers to behave correctly. One can also consider interactive proof systems for functions f in which the verifier learns $f(x)$ and obtains convincing evidence of the correctness of this value (cf. [11]).

*AT&T Bell Laboratories, Room 2C324, 600 Mountain Avenue, Murray Hill, NJ 07974 USA, beaver@research.att.com. 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 USA, jf@research.att.com.

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

A.J. Menezes and S.A. Vanstone (Eds.): Advances in Cryptology - CRYPTO '90, LNCS 537, pp. 326-338, 1991.

It is known (cf. [14, 16]) that the class IP of languages recognized by 1-prover interactive proof systems is equal to the complexity class PSPACE. Furthermore, it is shown in [2] that the class MIP of languages recognized by multi-prover interactive proof systems is equal to the complexity class NEXP = NTIME(2^{poly}).

In an *instance-hiding scheme* for a function f (cf. [1, 4, 5]), the input x is on a private tape, accessible only to the querier V. The protocol allows V to obtain the value of $f(x)$ without revealing to any prover any information about x (other than its length); however, V does not necessarily obtain any evidence of the the correctness of this value (because of this, the powerful players are referred to as "oracles" in [1, 4, 5]). In this model, V does not entrust any information about x to the provers, but it does have to trust the provers to behave correctly.

Beaver and Feigenbaum [4] have shown that *all* functions f have multi-prover instance-hiding schemes, thus settling a question of Rivest [15].

In this paper, we introduce the notion of an *instance-hiding proof system* for a function f and characterize the functions that have such systems. An instance-hiding proof system is similar to an instance-hiding scheme, except that along with the value of $f(x)$, the protocol allows V to obtain convincing evidence of the correctness of this value. Thus, the verifier need not entrust any information about x to the provers, nor need it trust the provers to behave correctly.

Let fNEXP denote the class of 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 Boolean 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 instance-hiding proof system, then clearly 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 instance-hiding proof systems and show that any instance-hiding proof system can be made zero-knowledge. In any type of proof system, 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 interactive proof systems for a function f on input x captures the intuitive idea that the verifier—even a misbehaving one—learns the value of $f(x)$ and nothing else. In an instance-hiding proof system 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 they cannot hope to prevent a verifier from learning, say, $f(x')$ instead of $f(x)$, where $|x'| = |x|$. Our definition of a zero-knowledge instance-hiding proof system 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 instance-hiding proof system, 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 of $f(x)$.

For the purpose of constructing zero-knowledge protocols, it is convenient to assume, as in [7], that the provers have access to a shared random tape that is not accessible to V. In [7], it is shown that every language in MIP has a zero-knowledge interactive proof system.

We prove the following.

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

The protocols that we construct in order to prove Theorems 1 and 2 involve multiple provers. Feigenbaum and Ostrovsky have recently shown that a function has a one-prover instance-hiding proof system if and only if it has a one-oracle instance-hiding scheme and it is in fPSPACE. They have also shown that the existence of a one-way function implies that all one-prover instance-hiding proof systems can be made (computational) zero-knowledge.

We remark that the notion of *private/adaptive checker*, which was introduced by Blum, Luby and Rubinfeld [9], can be viewed as a restricted form of instance-hiding proof system in which the provers are only asked questions of the form "what is $f(y)$."

The rest of this paper is organized as follows. In Section 2 we give the formal definitions of "instance-hiding proof system" and "zero-knowledge." In Section 3 we prove Theorem 1. In Section 4 we prove Theorem 2, along the way giving a new and simple perfect zero-knowledge protocol for languages in MIP. In Section 5 we state an open problem.

Most of these results first appeared in our Technical Memorandum [6].

2 Definitions

We now formally define instance-hiding proof systems and the corresponding notion of zero-knowledge. The intuition behind these definitions can be found in Section 1. Let V, P_1, ..., P_m be a set of interactive Turing Machines. 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 instance-hiding proof systems. Unlike ordinary MIP, the input x in an instance-hiding proof system 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 $\{0, 1, \textit{reject}\}$ and is denoted by $(V(x), P_1^*, \ldots, P_m^*)$.

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 of the verifier and provers.

Definition 2.1 *The protocol* $(V, P_1, \ldots, P_m)$ *is an* **instance-hiding proof system for the function** f *if it satisfies the following properties.*

(i) For all constants $c > 0$*, for all sufficiently large* x*,*

$$\text{Prob}((V(x), P_1, \ldots, P_m) = f(x)) > 1 - 1/|x|^c.$$

(ii) For all constants $c > 0$*, for all sufficiently large* x*, for all* $P_1^*, \ldots, P_m^*$*,*

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

(iii) For all $P_1^*, \ldots, P_m^*$*, for all inputs* x *and* x' *with* $|x| = |x'|$*, for* $1 \leq i \leq m$*, the distribution of the transcripts* $T(V, P_i^*, x)$ *and* $T(V, P_i^*, x')$ *are the same.*

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 systems, colluding provers could cause the verifier to accept a wrong value for $f(x)$; as in ordinary instance-hiding schemes, 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 (iii) is equivalent to the definition in [1].

Definition 2.2 *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***) the transcripts* $\langle T(V^*, P_1, x), \ldots, T(V^*, P_m, x)\rangle$*.*

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 can 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 which extends f to a larger arithmetic domain is sometimes referred to as an "arithmetization" of f. Arithmetizations were first used in the context of interactive protocols by Beaver and Feigenbaum [4].

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 instance-hiding proof system 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' prove to V that $(y(i,1), \ldots, y(i,n), z_i) \in L_g$.

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 [2] that NEXP = MIP allows us to implement step A3 with one-sided, exponentially small error probability. 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. Claim (2) can be proved using the line of reasoning found in [4]—the essential point is that each $y(i,j)$ is distributed uniformly over K (although pairs $y(i,j)$ and $y(k,j)$ are correlated); these random elements of K leak only the size of the input.

4 Proof of Theorem 2

The basic steps required to convert Protocol A in Section 3 to a zero-knowledge, instance-hiding proof system are the following.

1. We replace step A3 by a zero-knowledge simulation protocol.
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 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.

4.1.1 Bit Commitment

Using the shared random tape, simple bit commitment can be implemented in a very simple way as in [7]. In the bit commitment scheme, there are two protocols: a *bit commit protocol* and a *bit reveal protocol.* In order to ensure that the scheme works properly, the prover that executes the bit reveal protocol must not have any knowledge of random bits sent by the verifier during the bit commit protocol. One way to guarantee this is simply to dedicate one prover to the task of executing bit reveal protocols.

4.1.2 Committing Shared Random Bits

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.

4.1.3 Multiple-use Notarized Envelopes

In [8] 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 predicate), and if the predicate is not true the verifier can catch a cheating prover with probability at least 1/poly.

The restriction 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 [8]), we can represent b as the sum $c_1 \oplus \cdots \oplus c_m$. Using the same techniques as in [8], which involve Barrington's result on bounded-width branching programs [3], this representation allows b to be involved in $m-1$ proofs, as each proof reveals at most one of the c_i's. This modification has the effect of decreasing slightly the verifier's chances of catching a cheating prover, but the probability is still 1/poly.

4.1.4 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 of $F(u_1, \ldots, u_m)$, rejecting a wrong answer with probability at least 1/poly, and each of the P_i's should learn nothing.

The statement that V learns nothing but the value of F means that (1) during the protocol, V learns the value of F, and (2) there is a simulation procedure that, when given the value of F, will simulate the conversations that take place during the protocol. The statement that each prover learns nothing means that each prover receives messages that consist of uncorrelated random bits.

We briefly sketch an implementation of the protocol using a variant of Kilian's oblivious NC^1 circuit evaluation protocol [13]. Without loss of generality, we can

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

assume that F is a Boolean-valued function. Since $F \in NC^1$, there is a (polynomial-time constructible) branching program M that realizes F [3]. On a given input, M determines a sequence of permutations in S_5, $\sigma_1, \ldots, \sigma_m$, 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 $F = 0$, and it is equal to some fixed nonidentity element in S_5 otherwise.

Let $\rho_1, \ldots, \rho_{m-1}$ be random permutations in S_5, and let ρ_0 and ρ_m be the identity permutation. Let $\tau_j = \rho_{j-1}^{-1}\sigma_j\rho_j$ for $j = 1, \ldots, m$. Then $\prod_j \tau_j = \prod_j \sigma_j$, and the list of τ_j's are uniformly distributed apart from satisfying this equality; therefore, nothing can be inferred from the values of τ_j other than the value of F.

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.

4.2 A Zero-Knowledge MIP Protocol

We now describe a perfect zero-knowledge proof system for any language L in MIP. Our protocol is simpler than that in [7], and it will be easy to see that ours can be embedded as a subprotocol in an instance-hiding proof system in which the input bits to the subprotocol are not on a shared input tape, but are in notarized envelopes, or are known initially only to the verifier. Our protocol does not require oblivious transfer or general oblivious circuit evaluation.

4.2.1 A Normal Form

We begin with a normal form for MIP protocols in which the verifier's role is extremely limited.

Proposition 4.1 *Any language in $L \in$ MIP has a 2-prover protocol 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 (sketch): By the "completeness theorem" of [7] and the "probabilistic oracle machine" characterization of [10], we can assume that there is a poly-time deterministic oracle machine $\phi(x, r)$ 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 which 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, which 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 this protocol has the desired properties is similar to that found in [10]. ∎

4.2.2 Zero-Knowledge Simulation

Now we show how to simulate Protocol N in zero-knowledge. 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 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 $1/\text{poly}$. To reduce the error probability, the protocol can be repeated.

To show that this protocol is perfect zero-knowledge, we describe a simulation program M. In steps Z1–Z4, the messages received by the verifier will just consist of random bits, which M can easily simulate. Since M knows that *accept'* will be true no matter what the verifier does in steps Z1–Z4, M can simulate the conversations that occur during the distributed function evaluation protocol in step Z5.

Clearly, 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.

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 instance-hiding proof system. 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 $[y(i,j)] \quad (i = 1 \ldots n+1, j = 1 \ldots n)$ 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 $[z_i'] \quad (i = 1 \ldots n+1)$ 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 $z'_i := g(y(i,1), \ldots, y(i,n)) + h(\alpha_i)$; P_i puts z'_i in a notarized envelope.

B4. For $i = 1 \ldots n+1$: P_i, P'_i prove in zero-knowledge to V 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 (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 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 zero-knowledge.

To prove (1), one can easily show that if the provers follow the protocol, the verifier will always learn the correct value of $f(x_1, \ldots, x_n)$; 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) is proven as in Theorem 1.

To prove (3), we describe a simulation program M interacting with an arbitrary verifier V^*. We only discuss the simulation of the distributed function evaluation protocol in step B5; the other parts of the protocol can be easily simulated by virtue of the zero-knowledge properties of the various building blocks.

It will suffice to show how M can obtain the value of F, since this will allow it to then simulate the conversations that occur in the distributed function evaluation protocol. If the $y(i,j)$ values given by V^* in step B1 do not satisfy the linearity condition, the value of F is $(0, \ldots, 0)$. Otherwise, M can easily recover the corresponding values $x'_1, \ldots, x'_n$ and $r'_1, \ldots, r'_n$. M can then consult an f-oracle to obtain $f(x'_1, \ldots, x'_n)$. Notice that the polynomial $w'(X)$ can be written as

$$w'(X) = g(r'_1X + x'_1, \ldots, r'_nX + x'_n) + h(X),$$

and so we see that $w'(X)$ is just a random polynomial of degree $\leq n$ over K with constant term equal to $f(x'_1, \ldots, x'_n)$. M can generate such a $w'(X)$ at random, and then generate the z'_i values using the formula $z'_i = w'(\alpha_i)$ $(i = 1 \ldots n+1)$. The value of F is $(z'_1, \ldots, z'_{n+1})$.

5 Open Problem

Our protocols require a polynomial number of provers. One can ask whether some fixed number (perhaps 2) of provers would suffice for all instance-hiding proof systems. Note that it is not even known whether a constant number of provers suffice for the construction of instance-hiding *schemes* (in which no proof is required that the provers' answers are right) 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 [5]. Thus obtaining general instance-hiding proof systems with a constant number of provers may be impossible and, in any case, seems to require a wholely new technique.

References

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

[2] L. Babai, L. Fortnow, and C. Lund. Nondeterministic Exponential Time has Two-Prover Interactive Protocols, *Proc. of the 31st FOCS* (1990), IEEE.

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

[4] D. Beaver and J. Feigenbaum. Hiding Instances in Multioracle Queries, *Proc. of the 7th STACS* (1990), Springer Verlag LNCS 415, 37–48.

[5] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with Low Communication Overhead, *these proceedings.*

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

[7] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multiprover Interactive Proof Systems: How to Remove Intractability Assumptions, *Proc. of the 20th STOC* (1988), ACM, 113–131.

[8] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything Provable is Provable in Zero-Knowledge, *Proc. of the 8th CRYPTO* (1988), Springer Verlag LNCS 403, 37–56.

[9] M. Blum, M. Luby, and R. Rubinfeld. Program Result Checking Against Adaptive Programs and in Cryptographic Settings, *Proc. of the DIMACS Workshop on Distributed Computing and Cryptography* (1989), AMS.

[10] L. Fortnow, J. Rompel, and M. Sipser. On the Power of Multiprover Interactive Protocols, *Proc. of the 3rd Structure in Complexity Theory Conference* (1988), IEEE, 156–161.

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

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

[13] J. Kilian. Founding Cryptography on Oblivious Transfer, *Proc. of 20th STOC* (1988), ACM, 20–31.

[14] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic Methods for Interactive Proof Systems, *Proc. of the 31st FOCS* (1990), IEEE.

[15] R. Rivest. Workshop on Communication and Computing, MIT, October, 1986.

[16] A. Shamir. IP = PSPACE, *Proc. of the 31st FOCS* (1990), IEEE.