CHAPTER 8

Overview of Interactive Proof Systems and Zero-Knowledge

J. FEIGENBAUM
A T & T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract—**In traditional computational complexity theory, the informal notion of *efficiently verifiable* sets of statements is formalized as *nondeterministic polynomial time* sets. Recently, an alternative formalization has emerged: sets with *interactive proof systems*. An interactive proof system is called *zero-knowledge* if it succeeds in proving the desired statements and nothing else. This chapter surveys definitions, examples, known results, and open problems in the area of interactive proof systems and zero-knowledge.**

1 INTRODUCTION

Some basic problems in complexity theory and cryptography can be thought of as two-player games in which one player (the ''prover'') tries to convince the other player (the ''verifier'') of the truth of an assertion. Indeed, the complexity class NP can be formulated in these terms. To convince a verifier that a string x is in the language L, where L is in NP, the prover can provide a ''short certificate'' of membership; if x is not in L, then no ''cheating prover'' could produce a certificate because none exists. In cryptography, we have the example of authentication schemes. To gain access to a computer, a building, or some other secure facility, a user must convince an operating system, a guard, or some other type of player that he is who he claims to be and he is entitled to access. Thus the user can be thought of as a prover and the operating system as a verifier.

The work surveyed in this chapter addresses two questions about such games:

- How much ***interaction*** is needed for the prover to convince the verifier?
- How much ***knowledge*** does the verifier gain during the course of the interaction?

Let us look more closely at NP languages in terms of two-player games. For example, consider the language L of pairs (G, k), where G is a finite, undirected graph that contains a k-clique—that is, a k-vertex subgraph every two vertices of which are connected by an edge. The obvious way to play a game in which the prover's objective is to convince the verifier that a pair (G, k) is in L is to require the prover to produce a subset $\{v_1, \ldots, v_k\}$ of the vertex set of G and to require the verifier to check that every $\{v_i, v_j\}$, $1 \leq i < j \leq k$, is in the edge set of G. This game is very efficient in terms of interaction. The prover has to send only one message to the verifier in order to prove his claim. On the other hand, this game may be inefficient in terms of knowledge. After receiving this one message from the prover, the verifier has full knowledge of how to prove that (G, k) is in L; he could turn around and use this knowledge to convince a third party that (G, k) is in L, which is something he might not have been able to do before he received a message from the prover. Finally, note that this game involves no random choices and no probability of error: If (G, k) is in L, then, given enough computing power, the prover can always produce a k-clique, and the verifier will accept the prover's claim with probability 1; if (G, k) is not in L, then no k-clique exists, and the probability is zero that a dishonest prover can convince the verifier to accept.

It is fairly easy to formalize the statement that games with these three properties (one message from prover to verifier, full knowledge of the proof given to the verifier, and no probability of error) exactly characterize NP—that is, the assertions that can be proven with these games are those of the form "x is in L," where L is a language in NP.

What if we allow games in which the prover and verifier exchange more messages? What if we allow a small probability of error—that is, what if a true assertion is rejected once in while or a false assertion believed? Can we then devise games for languages that are not known to be in NP? Can we devise games that are more knowledge-efficient? For example, is it possible for the prover to convince the verifier that his assertion is true without giving him *any* knowledge of the proof?

These questions motivate the theory of *interactive proof systems* and *zero-knowledge*, which is surveyed in this chapter. In Section 2 we give formal definitions of these concepts; the definitions are taken from the seminal papers of Goldwasser, Micali, and Rackoff [24], Babai and Moran [6], and Ben-Or, Goldwasser, Kilian, and Widgerson [10]. Section 3 reviews two well-known examples of interactive proof systems, one drawn from cryptography and one from complexity theory. Major results of the theory are given in Section 4. Section 5 contains a brief discussion of related notions such as *arguments*, *program checkability*, and *instance-hiding;* these notions are related in two ways: Their study is motivated by intuitively similar concerns, and they are used in some of the proofs of the main results stated in Section 4. Finally, open problems are given in Section 6.

2 DEFINITIONS

The reader is assumed to be familiar with the basic notions of computational complexity theory. Refer to the Appendix for a brief review of these notions.

We are concerned with the following model of *interactive computation*. In a *one-prover interactive protocol* (P, V), the *prover* P and *verifier* V have a shared *input* x, two private sources of unbiased random bits (say r_P and r_V, respectively), and a way to send messages to each other. On input x, the *transcript* of the protocol is a sequence of *moves* $(\alpha_1, \beta_2, \alpha_3, \beta_4, \ldots, \alpha_{2t-1}, \beta_{2t})$, where $t = t(n)$ is a polynomially bounded function of $n = |x|$. There is also a polynomial l such that $|\alpha_{2i-1}|$ and $|\beta_{2i}|$ are bounded by $l(n)$, for $1 \leq i \leq t$. Each α_{2i-1} is a message from the verifier to the prover and is a polynomial-time computable function of x, r_V, and $(\alpha_1, \beta_2, \ldots, \alpha_{2i-3}, \beta_{2i-2})$. Each β_{2i} is a message from the prover to the verifier and is a function of x, r_P, and $(\alpha_1, \beta_2, \ldots, \beta_{2i-2}, \alpha_{2i-1})$; note that no restrictions are placed on this function except that its output be bounded in length by $l(n)$. After receiving β_{2t}, the verifier computes a polynomial-time function of x, r_V and the entire transcript; the output of this computation is either ACCEPT or REJECT.

Let $(P, V)(x)$ denote the verifier's final output. For every x, $(P, V)(x)$ is a random variable whose distribution is induced by the uniform distributions on the random variables r_P and r_V and the functions computed by P and V. Similarly, we denote by *View* (P, V, x) the random variable consisting of the transcript $(\alpha_1, \beta_2, \ldots, \alpha_{2t-1}, \beta_{2t})$ produced by an execution of the protocol, together with the prefix of r_V that the verifier consumes during the execution. More generally, if M is any probabilistic Turing machine, the output of M on input x is a random variable denoted by $M(x)$.

For any interactive protocol (P, V), we denote by (P^*, V) the interactive protocol in which the verifier behaves exactly as in the original protocol (P, V), but the prover computes the functions specified by P^*; this is a "cheating prover" if $P^* \neq P$. Similarly, we denote by (P, V^*) the interactive protocol in which the prover behaves exactly as in the original protocol, but the (potentially cheating) verifier computes the functions specified by V^*. The potentially cheating prover P^* is limited only in that the messages that it sends must be of length at most $l(n)$; the potentially cheating verifier V^* is limited to probabilistic polynomial-time computation.

Definition 2.1 (see also [24]): *The interactive protocol* (P, V) ***recognizes the language*** *L if, for all $x \in L$,*

$$\text{Prob}((P, V)(x) = \text{ACCEPT}) > 2/3$$

and, for all $x \notin L$, for all provers P^*,

$$\text{Prob}((P^*, V)(x) = \text{REJECT}) > 2/3$$

We denote by IP(k) the class of all languages recognized by interactive protocols with at most k moves. In the above discussion, $k = 2t$. IP(*poly*) is the union, over all polynomially bounded k, of IP(k); the shorthand IP is used for IP(*poly*). The statement that (P, V) is ***a (one-prover) interactive proof system for the language*** L is synonymous with the statement that (P, V) recognizes L.

Note that the definition of IP, like the definition of BPP, allows for error probability $1/3$. In both cases, the error probability can be made exponentially small by performing polynomially many repetitions of the computation and taking the majority answer. Note that these are *sequential* repetitions; so, in the case of IP, this procedure reduces the error probability at the expense of increasing the number of moves.

Definition 2.2 (see also [24]): *The interactive proof system (P, V) for the language L is **computational (resp. perfect) zero-knowledge** if, for every verifier V^*, there exists a probabilistic, expected polynomial-time machine M_{V^*}, called the* **simulator,** such that the ensembles $\{View\ (P, V^*, x)\}_{x \in L}$ *and* $\{M_{V^*}(x)\}_{x \in L}$ *are **computationally indistinguishable (resp. the same).***

Informally speaking, two ensembles are said to be *computationally indistinguishable* if no probabilistic polynomial-sized circuit family can tell them apart. The term is supposed to convey the idea that the ensembles are computationally "close" to being equal. Refer to Goldwasser et al. [24] for a formal definition. There is also an intermediate notion of *statistical zero-knowledge* and a corresponding notion of *statistical indistinguishability* of ensembles. Many of the results on perfect zero-knowledge that are stated in Section 4 below actually hold for statistical zero-knowledge. Refer to [2, 19] for details.

The statement that (P, V) is an interactive proof system for L can be viewed as a limitation on the prover: No matter what P^* does, it cannot force the verifier to accept a string x that is not in L, except with negligible probability. Similarly, the statement that (P, V) is zero-knowledge can be viewed as a limitation on the verifier: No matter what V^* does, the only distributions that it computes by interacting with P are distributions that it could have computed in expected polynomial time *without* interacting with P.

A more restricted form of interactive protocol, the *Arthur–Merlin protocol,* is considered in [6]. In an Arthur–Merlin protocol, two players, Arthur and Merlin, have a common input x of length n. Once again, there is a sequence of moves. In odd-numbered moves, Arthur simply sends to Merlin a string of $l(n)$ unbiased random bits. In even-numbered moves, Merlin sends to Arthur a string of length $l(n)$ that is *optimal* (in a sense that will be made precise shortly). After $k(n)$ moves, a deterministic polynomial-time Turing machine, the *referee,* decides the winner; so each polynomial-time referee determines an Arthur–Merlin protocol. Because Arthur's moves are random, Merlin's winning probability depends only on x—call this probability $W(x)$. It is required that $W(x)$ be greater that $\frac{2}{3}$ or less than $\frac{1}{3}$, for each x. The referee is known in advance to both Arthur and Merlin; because the referee is polynomial-time bounded and Merlin is not, Merlin can make *optimal* moves—that is, he can maximize $W(x)$.

Definition 2.3 (see also [6]): *The **language recognized by an Arthur–Merlin protocol** is the set of all x for which* $W(x) > \frac{2}{3}$.

Think of Merlin's objective as "trying to convince" Arthur to accept the string x. Thus he is analogous to the prover in IP.

The class of languages recognized by Arthur–Merlin protocols with at most k moves is denoted AM(k), and AM(*poly*) is the union of AM(k) over all polynomially bounded k. Unlike the notation IP, the notation AM is used for AM(2). Merlin–Arthur protocols are also considered in [6]; they are the same as Arthur–Merlin protocols, except that Merlin makes the odd-numbered moves. So the sequence of movers for a language in MA(4) is MAMA; the sequence for a language in AM(3) is AMA.

Multiprover interactive protocols are defined analogously to one-prover interactive protocols. Instead of one prover P, there are m provers $P_1, \ldots, P_m$, where $m = m(n)$ is a polynomially bounded function of $n = |x|$. If the verifier V makes the t^{th} move, he sends m messages $(\alpha_{1t}, \ldots, \alpha_{mt})$ simultaneously. For each i, $1 \leq i \leq m$, P_i receives

α_{it}, and P_i cannot overhear α_{jt}, for any $j \neq i$. Similarly, if the provers make the t^{th} move, each P_i sends a message β_{it} to V, who receives all of $(\beta_{1t}, \ldots, \beta_{mt})$ simultaneously, and no P_i can overhear β_{jt}, for $j \neq i$. The notation IP(m, *poly*) is used for the class of languages recognizable by m-prover interactive protocols with polynomially many moves. Zero-knowledge for multiprover interactive protocols is defined in terms of simulation of transcripts, as it is in the one-prover case; there is one nonobvious part of the definition—the provers $P_1, \ldots, P_m$ use a shared random string that is unknown to the verifier. This is often interpreted as follows: The provers can get together in advance to "agree on a strategy" by picking a shared random string and deciding how to compute the β_{it}'s; during the execution of the protocol, however, the provers are "kept separate," which gives the verifier a chance to catch them in an inconsistency if they try to cheat.

The definitions given above are for (zero-knowledge) interactive proofs of *language membership*. The following notions also have rigorous definitions: proofs that the *functional value* $f(x)$ is what the prover claims it is (see, for example, [21]), and *proofs of knowledge* (see, e.g., [17]). The latter are particularly relevant to the original cryptographic motivation for zero-knowledge proof systems: The prover wants to convince the verifier that he "knows" a secret without revealing that secret. The example in Section 3.1 is a protocol for proofs of knowledge.

3 EXAMPLES

3.1 A Cryptographic Example: Proofs of Identity

The following example, a system for proofs of knowledge, is taken from Feige, Fiat, and Shamir [17]. It is designed to allow a community of users to *authenticate* themselves to each other. For each user, there is a pair (I, S). The public information I can be interpreted as the user's identity and the private information S as his secret key. The goal of the proof system is for a user with identity I to convince another user that he "knows" the corresponding key S without revealing anything about S beyond the fact that he knows it. The effectiveness of the proof system is based on the assumption that it is computationally infeasible to compute square roots modulo a large composite integer with unknown factorization; this is provably equivalent to the assumption that factoring large integers is difficult.

Modulus generation: A *trusted center* generates two large primes each congruent to 3 mod 4. The product m of these primes is published, but the primes are not.

Note that -1 is a quadratic non-residue modulo m—that is, there is no a such that $a^2 = -1 \mod m$. In what follows, $Z_m^*[+1]$ denotes the set of integers between 1 and m that are relatively prime to m and have Jacobi symbol $+1$ with respect to m.

Key generation: Each user chooses t_1 random numbers $S_1, \ldots, S_{t_1}$ in $Z_m^*[+1]$ and t_1 random bits $b_1, \ldots, b_{t_1}$. He sets I_j equal to $(-1)^{b_j}/S_j^2 \mod m$, for $1 \leq j \leq t_1$. This user's identity, which he publishes, is $I = (I_1, \ldots, I_{t_1})$, and his secret key, which he keeps private, is $S = (S_1, \ldots, S_{t_1})$.

Proofs of identity: User A authenticates himself to user B as follows. Let I be A's published identity and S his secret key. A and B repeat the following four steps t_2 times.

1. A picks a random R in $Z_m^*[+1]$ and a random bit c and sends $X = (-1)^c R^2 \bmod m$ to B.
2. B sends a random vector of bits $(E_1, \ldots, E_{t_1})$ to A.
3. A sends $Y = R \cdot \prod_{E_j=1} S_j \bmod m$ to B.
4. B verifies that $Y^2 \cdot \prod E_j I_j \bmod m$ is equal to $\pm X$.

B believes that A is who he claims he is if the verification in Step 4 succeeds in each of the t_2 trials.

In [17], it is shown that this scheme provides *zero-knowledge proofs of knowledge* for the values $t_1 = O(\log\log m)$ and $t_2 = \Theta(\log m)$. A bit more care is needed when implementing the scheme in practice; for example, B should check in Step 2 that A actually sent an element of $Z_m^*[+1]$, lest a cheating prover just send $X = Y = 0$ in each of the t_2 executions (refer to [16] for more discussion of this issue). The most important feature of the scheme is that there is no need for the prover to have significant computational power. These identity proofs require only a few modular multiplications and can be implemented on smart cards.

3.2 A Complexity-Theoretic Example: Proofs for the PERM Function

The following interactive proof system is due to Lund, Fortnow, Karloff, and Nisan [30]. It provides a way for P to convince V that the permanent of an integer matrix is what P claims it is. Because the permanent function is complete for the complexity class #P (see also [36]), this proof system of Lund et al., combined with the seminal theorem of Toda [35] that PH $\subseteq$ P$^{\#P}$, shows that every language in the PH has a one-prover interactive proof system and goes a long way toward a complete characterization of IP.

It is easy to see (and is shown in [30]) that it suffices to prove the values of permanents modulo a large prime p. Let $A = (a_{ij})$ be an $N \times N$ matrix over Z_p and $A_{1|i}$ denote the $(1, i)$-minor of A. Recall the formula

$$PERM(A) = \sum_{\sigma \in S_N} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{N\sigma(N)}$$

for the permanent of A and the fact that $PERM(A)$ can be computed by cofactor expansion:

$$PERM(A) = \sum_{i=1}^{N} a_{1i} \cdot PERM(A_{1|i})$$

If $C = (c_{ij})$ and $D = (d_{ij})$ are $N \times N$ matrices over Z_p and x is an indeterminant, we denote by $(C + x(D - C))$ the $N \times N$ matrix whose ij^{th} entry is the degree-1 polynomial $c_{ij} + x(d_{ij} - c_{ij})$ in $Z_p[x]$. Notice that $PERM(C + x(D - C))$ is a polynomial $f(x)$ in $Z_p[x]$ of degree at most N. Furthermore, $f(0) = PERM(C)$ and $f(1) = PERM(D)$.

The following is a protocol for P to prove that $PERM(A)$ is equal to $a \bmod p$. Each stage of the protocol is either an *expand* stage or a *shrink* stage. The object that is expanded and shrunk is a list $\mathcal{L} = \langle (B_1, b_1), \ldots, (B_t, b_t) \rangle$ in which b_i is P's claimed value for $PERM(B_i) \bmod p$.

Initially, $\mathcal{L}$ contains only $\langle A, a\rangle$. P and V repeat the following steps until $\mathcal{L}$ consists of a single entry $\langle B, b\rangle$, where B is 1×1.

If $\mathcal{L} = \langle B, b\rangle$, where B is $r \times r$ and $r > 1$, then EXPAND:

V: Let $C_i = B_{1|i}$. Ask P for the permanents of C_i, $1 \leq i \leq r$.

P: Send claimed values $c_i = PERM(C_i)$ to V, $1 \leq i \leq r$.

V: If $b \neq \Sigma_{i=1}^{r} b_{1i}c_i \mod p$, then REJECT and terminate the protocol. Otherwise, set $\mathcal{L}$ to $\langle (C_1, c_1), \ldots, (C_r, c_r)\rangle$.

If $\mathcal{L}$ contains two or more pairs, then SHRINK:

V: Take the first two pairs (C, c) and (D, d) from $\mathcal{L}$ and ask P for the permanent of $C + x(D - C)$.

P: Send a claimed value $g(x)$ for $PERM(C + x(D - C))$.

V: If $g(0) \neq c$ or $g(1) \neq d$, then REJECT and terminate the protocol. Otherwise, choose a random $s \in Z_p$, send it to P, and replace the first two pairs of $\mathcal{L}$ with $(C + s(D - C), g(s))$.

It is clear that an honest P will always convince V that $PERM(A)$ is what he claims it is. The essential reason that a cheating P^* cannot convince V to accept a wrong value with high enough probability is that $f(x) = PERM(C + x(D - C))$ has low degree. Specifically, it has degree $r \leq N$, where N is the dimension of the original input. If P^* sends a degree-r polynomial $g \neq f$ in some shrink stage, then g and f can agree on at most r points; if p is sufficiently large with respect to r, then f and g are unlikely to agree on a random $s \in Z_p$. Thus, it is likely that $g(s) \neq PERM(C + s(D - C))$ and that P^* will have to lie in subsequent stages. The matrices whose permanents he lies about get smaller and smaller, and eventually V can catch him.

4 KNOWN RESULTS

One of the most exciting recent developments in complexity theory is the complete characterization of the language-recognition power of interactive proof systems.

Theorem 4.1: AM(*poly*) = IP = PSPACE.

The fact that AM(*poly*) is contained in IP is clear from the definitions, because Arthur and Merlin of Definition 2.3 are just special cases of the verifier and prover of Definition 2.1. The equality of IP and AM(*poly*) was proven by Goldwasser and Sipser [25]; actually, they proved the sharper result IP($k(n)$) $\subseteq$ AM($k(n) + 2$), for all polynomially bounded functions k. The upper bound IP $\subseteq$ PSPACE is also clear from the definitions: IP is contained in the class of languages accepted by *games against nature,* which was defined earlier by Papadimitriou [33] and shown to be equal to PSPACE. The fact that the PSPACE-complete language quantified Boolean formulas (QBF) and hence every PSPACE language, has an interactive proof system was obtained by Shamir [34]; his work relies heavily on the work of Lund et al. discussed in Section 3.2; in particular, it uses a similar low-degree polynomial trick.

Given the widespread interest in interactive proof systems and zero-knowledge, it is clear why a definitive theorem like IP = PSPACE would be exciting. But why was it surprising? One reason is that it is a rare example of a *nonrelativizing* result in complexity theory; see [3] for a discussion of relativization and its relationship to the characterization of IP. Another reason for the surprise is the technical simplicity of Shamir's result and the results leading to it; some of the key ingredients are discussed briefly in Section 5.

Now that we know exactly which languages have interactive proof systems, it is natural to ask how much interaction is required for languages of interest. It is clear from the definitions that $\mathrm{AM}(k(n)) \subseteq \mathrm{AM}(k(n) + 1)$ for every polynomially bounded function k. Babai and Moran have shown that these inclusions are not always proper.

Theorem 4.2 (see also [6]): *For any polynomially bounded function* $k(n) \geq 2$, $\mathrm{AM}(2k(n)) = \mathrm{AM}(k(n))$. *In particular,* $\mathrm{AM}(c) = \mathrm{AM}(2)$, *for any constant* $c \geq 2$.

The known proof systems for PERM and QBF involve a polynomial number of moves. It is unknown whether QBF is provable in a constant number of moves; the following results address this question.

Theorem 4.3 (see also [6]): $\mathrm{AM}(2) \subseteq \Pi_2^p$.

Thus, if QBF does have a constant-move interactive proof system, all of PSPACE is contained in the second level of the polynomial-time hierarchy. So such a proof system for QBF may be impossible to obtain and, in any case, would require a revolutionary new idea. A weaker, but still revolutionary, consequence would follow from the weaker assumption that every language in coNP has a constant-move interactive proof system.

Theorem 4.4 (see also [13]): *If* coNP $\subseteq$ AM(2), *then the* PH *collapses at the second level.*

How serious a restriction is the requirement of zero-knowledge? The answer depends on which definition of zero-knowledge is used.

Theorem 4.5: *If one-way functions exist, then every language in* IP *has a computational zero-knowledge proof system.*

This general result follows from Impagliazzo and Yung's theorem [26] that every language in IP has a computational zero-knowledge proof system if *bit-commitment schemes* exist and Naor's theorem [31] that one-way functions yield bit-commitment schemes. Impagliazzo and Yung's work relies heavily on earlier work by Goldreich, Micali, and Wigderson [23] and Brassard, Chaum, and Crépeau [14].

Theorem 4.6: PZK, *the class of languages with perfect zero-knowledge proof systems, is contained in* AM $\cap$ coAM.

The inclusion of PZK in coAM is due to Fortnow [19], and its inclusion in AM is due to Aiello and Håstad [2]. Theorems 4.3 and 4.6 together imply that PZK is contained in $\Sigma_2^p \cap \Sigma_2^p$; so there is no reason to believe that PZK contains all of IP.

What about multiple provers? The first basic result, due to Ben-Or, Goldwasser, Kilian, and Wigderson [10], is that two provers suffice.

Theorem 4.7: *For any polynomially bounded function* m, $\mathrm{IP}(m(n), \mathit{poly}) = \mathrm{IP}(2, \mathit{poly})$.

From now on, we denote by MIP the language class IP(2, *poly*). This class has been fully characterized in terms of traditional complexity classes by Babai, Fortnow, and Lund [5]:

Theorem 4.8: MIP = NEXP.

Unlike the one-prover case, two-prover interactive proof systems can always be made perfect zero-knowledge.

Theorem 4.9 *Every language in* MIP *has a perfect zero-knowledge, two-prover interactive proof system.*

Theorem 4.9 is due to Ben-Or, Goldwasser, Kilian, and Wigderson. For a complete proof, see [27].

So two-prover systems seem to be different from one-prover systems with respect to zero-knowledge. They also seem to be different with respect to move-complexity. Kilian [28] has shown recently that two provers and c moves suffice for any language in MIP, provided that error probability ε is tolerated—here ε is an arbitrarily small constant and c is a constant that depends on ε. The best possible result would be that any language in MIP can be recognized in two moves with exponentially small error probability.*

5 RELATED NOTIONS

In most practical cryptographic situations, it does not make much sense to allow the prover to use unlimited computing power; for this reason, results such as ''all of PSPACE has computational zero-knowledge proofs if one-way functions exist'' are essentially impractical, beautiful as they are. In the model of Brassard, Chaum, and Crépeau [14], all parties are limited to reasonable computing power. The only advantage that the prover has over the verifier is a short piece of advice, such as the factorization of a large integer. Of course, limiting the prover's power reduces the class of languages that can be handled from PSPACE down to MA. On the other hand, under suitable cryptographic assumptions, these limits *increase* the class of problems that can be handled in *perfect* zero-knowledge—that is, the problems in which the prover's secret is protected unconditionally. It is shown in [14] that all languages in MA (and, *a fortiori*, all languages in NP) admit perfect zero-knowledge protocols of this type, assuming that *unconditionally concealing bit-commitment schemes* exist. (This is in sharp contrast with the result of Fortnow [19] mentioned in Section 4.) Brassard, Crépeau, and Yung [15] have shown that all languages in NP admit constant-move, perfect zero-knowledge protocols of this type, under the same assumption.

It is important to understand that the protocols in [14, 15] are *not* IP-protocols—it would be easy for an arbitrarily powerful prover to cheat. In other words, these protocols are *computationally convincing*, as opposed to proof-systems, which are *statistically* convincing. To distinguish computationally convincing protocols from proof

*Note added in proof: This result has been proven by Uriel Feige.

systems, the former are referred to as *arguments*. There is a strong duality between computational zero-knowledge IP-protocols and perfect zero-knowledge arguments for NP-complete problems. However, one important aspect of this duality breaks down. After a computational zero-knowledge IP-protocol is completed, the verifier has complete information about the prover's secret, albeit in enciphered form. Working *offline* for long enough, the verifier may be able to decipher that secret. In sharp contrast, a dishonest prover must break the cryptographic assumption *online*, while the protocol is taking place, if he wishes to cheat in an argument. As a consequence, an algorithm capable of factoring large integers in a month of Cray time would be a severe threat to the security of computational zero-knowledge IP-protocols but of little practical consequences for perfect zero-knowledge arguments.

M. Blum defined the related notion of *program checkability*. The goal of the work on program checkability (see, for example, [9, 11, 12]) is to be able to check whether a program is correct on a specific input. One way to do this is to regard the program as a prover in an interactive proof system and design a *checker* that plays the role of the verifier. Note that this changes the nature of both the honest prover and the cheating prover; both are constrained to be non-self-modifying functional programs, and the honest prover must be a program that computes the function being checked. Because of the restriction on the honest prover, languages in IP are not necessarily checkable; in fact, Beigel and Feigenbaum [9] give a "natural" complexity-theoretic hypothesis that implies the existence of a language in IP that is not checkable. Because of the restriction on the cheating prover, checkable languages are not necessarily in IP; in fact, Babai, Fortnow, and Lund [5] show that EXP-complete languages are checkable, whereas they are not in IP unless EXP = PSPACE. The fact that checkable languages are all in NEXP follows from the results in [20].

Abadi, Feigenbaum, and Kilian [1] defined the related notion of *instance-hiding schemes.** In such a scheme, a polynomial-time *querier* interacts with an *oracle* in order to learn the value $f(x)$, for some function f that the querier cannot compute on its own, *without revealing x to the oracle*—specifically, revealing nothing (in the information-theoretic sense) except the length of x. There is an approximate analogy with zero-knowledge proof systems, in which the prover convinces the verifier of the truth of a statement, without actually telling him the proof. In [1], it is shown that no NP-hard function has an instance-hiding scheme, unless the PH collapses at the third level. Beaver and Feigenbaum [7] considered a *multioracle* version of instance-hiding schemes; in this version, the querier asks questions of multiple oracles, and the oracles "cannot talk to each other," by analogy with the multiple provers in MIP. Here there are positive results: If the querier can talk to $n + 1$ oracles, where n is the length of the secret query, then *any* function has an instance-hiding scheme (see [7]; this construction was subsequently improved to work with $n/\log n$ oracles; see [8]).

The instance-hiding schemes in [7] have a special form that is of complexity-theoretic significance; they are *locally random reductions*. Informally, f is k-locally random reducible (k-lrr) to g if there is a probabilistic polynomial-time algorithm that maps an arbitrary instance x in the domain of f to a set of random instances $y_1, \ldots, y_k$ in the domain of g in such a way that $f(x)$ is computable in polynomial time from

*"Instance-hiding" is current terminology; in [1], these are called *encryption schemes for functions*.

$g(y_1), \ldots, g(y_k)$ and the distribution of each of the random instances y_i depends only on $|x|$; a formal definition can be found in [8]. If $g = f$, the reduction is called a *k-random-self-reduction* (*k*-rsr). It is shown in [7] that every Boolean function is $(n + 1)$-lrr to a sequence $\{g_n\}_{n \geq 1}$ of polynomials, where *degree* $(g_n) = n$. This was the first use of the fateful *low-degree polynomial trick* that was to be used by Lund et al. and by Shamir in the proof systems discussed above.

Building on Beaver and Feigenbaum [7], Lipton [29] observed that any f that is itself a low-degree polynomial (including the #*P*-complete function PERM) is random-self-reducible. PERM is also *length-decreasing self-reducible*—that is, computing the permanent of A can be reduced in polynomial time to computing permanents of matrices of smaller size; this is done by co-factor expansion, as discussed in Section 3.2. Blum, Luby, and Rubinfeld [12] showed that any function that is both random-self-reducible and length-decreasing self-reducible has a program checker; by the earlier observation of Fortnow, Rompel, and Sipser [20], such a function also has a multi-prover interactive protocol. Nisan [32] put all these observations together into the statement that PERM (and, from Toda [35], the entire PH) is in MIP. This dramatic sequence of results culminated in the complete characterization of the language-recognition power of IP (see also [30, 34]) and MIP (see also [5]) that was discussed in Section 4.

6 OPEN PROBLEMS

One of the most interesting class of problems remaining concerns the necessary power of the prover. For example, in an interactive proof system for a PSPACE-complete language, the prover can be taken to be a PSPACE machine (see [34] for details). The analogous statements can be made about ⊕P-complete languages (see also [4]) and #P-complete functions (see also [30]). Furthermore, in a two-prover interactive proof system for an EXP-complete language, both provers can be taken to be EXP machines (see also [5]).

Question 1: Is there a one-prover interactive proof system for coSAT in which the prover is given only enough power to answer SAT queries?

Question 2: Is there a two-prover interactive proof system for coSAT in which both provers are given only enough power to answer SAT queries? This is equivalent to the question of whether there is a program checker for SAT.

Question 3: An easier question than 1 or 2: Is there a one- or two-prover interactive proof system for coSAT in which the power of the prover(s) is somewhere in the PH?

Question 4: Does every NEXP-complete language S have a two-prover interactive proof system in which both provers are given only enough power to answer queries about membership in S?

The answer to Question 2 would be "yes" if SAT were random-self-reducible. It is known that, unless the PH collapses, any random-self-reduction for SAT would have to be *adaptive*—that is, if instance x is mapped to a set $y_1, \ldots, y_k$ of random instances, then instance y_{i+1} has to depend on the answer to instance y_i (see [18] for details).

Question 5: Is there an adaptive random-self-reduction for SAT?

Question 6: Find other sufficient conditions for interesting classes of provers, besides random-self-reducibility and length-decreasing self-reducibility.

There are also two obvious questions remaining about zero-knowledge.

Question 7: Theorem 4.5 says that one-way functions are sufficient to ensure computational zero-knowledge proofs for all of PSPACE. Is there a weaker assumption that also suffices? For example, does the assumption that $P \neq NP$ suffice to ensure computational zero-knowledge proofs for every language in NP (perhaps with a PSPACE-complete prover)?

Question 8: Do one-way functions imply the existence of perfect zero-knowledge arguments for NP-complete languages?

So far, all that is known about the question of whether more moves allow proof systems to recognize more languages is Theorems 4.1, 4.2, and 4.4. Further results on the move hierarchy would be interesting.

Finally, one can ask which other notions in theoretical computer science can be fruitfully related to interactive proof systems, program checking, random-self-reducibility, and so on. One good candidate is machine-learning.

ACKNOWLEDGMENT

I am extremely grateful to Gilles Brassard for his prompt and thoughtful comments on an earlier draft.

References

[1] M. Abadi, J. Feigenbaum, and J. Kilian, "On hiding information from an oracle," *J. Comput. Syst. Sci.*, vol. 39, pp. 21–50, 1989.

[2] W. Aiello and J. Håstad, "Perfect zero-knowledge languages can be recognized in two rounds," *Proc. 28th IEEE Symposium on Foundations of Computer Science*, pp. 439–448, 1987. Expanded version to appear in *J. Comput. Syst. Sci.*

[3] E. Allender, "Oracles vs. proof techniques that do not relativize." *Proc. Special Interest Group on Algorithms '90*, (T. Asano, T. Ibaraki, H. Imai, T. Nishizeki, eds.), Berlin: Springer-Verlag, *Lecture Notes In Computer Science* 450, pp. 39–52, 1990.

[4] L. Babai and L. Fortnow, "A characterization of *#P* by arithmetic straight line programs," *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pp. 26–34, 1990.

[5] L. Babai, L. Fortnow, and C. Lund, "Nondeterministic exponential time has two-prover interactive protocols," *Proc. 31st, IEEE Symposium on Foundations of Computer Science*, pp. 16–25, 1990.

[6] L. Babai and S. Moran, "Arthur–Merlin games: A randomized proof system and a hierarchy of complexity classes," *J. Comput. Syst. Sci.*, vol. 36, pp. 254–276, 1988.

[7] D. Beaver and J. Feigenbaum, "Hiding instances in multioracle queries," *Proc. 7th Symposium on Theoretical Aspects of Computer Science,* (C. Choffrut, T. Lengauer, eds.), Berlin: Springer-Verlag *Lecture Notes In Computer Science* 415, pp. 37–48, 1990.

[8] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. "Security with low communication overhead," *Proc. CRYPTO* '90 (S. Vanstone, ed.) Berlin: Springer-Verlag, *Lecture Notes In Computer Science,* (in press).

[9] R. Beigel and J. Feigenbaum, "Improved bounds on coherence and checkability," Yale University Tech. Rept. YALEU/DCS/TR-819, Computer Science Department, Sept. 11, 1990. To appear in *Computational Complexity.*

[10] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multiprover interactive proof systems: How to remove intractability assumptions," *Proc. 20th Symposium on Theory of Computing,* Association of Computing Machinery, pp. 113–131, 1988.

[11] M. Blum and S. Kannan, "Designing programs that check their work," *Proc. 21st Symposium on Theory of Computing,* Association of Computing Machinery, pp. 86–97, 1989.

[12] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *Proc. 22nd Symposium on Theory of Computing,* Association of Computing Machinery, pp. 73–83, 1990.

[13] R. Boppana, J. Håstad, and S. Zachos, "Does coNP have short interactive proofs?," *Inform. Proc. Lett.,* vol. 25, pp. 127–132, 1987.

[14] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *J. Comput. Syst. Sci.,* vol. 37, pp. 156–189, 1988.

[15] G. Brassard, C. Crépeau, and M. Yung, "Everything in NP can be argued in perfect zero knowledge in a bounded number of rounds," in *Proc. 16th International Colloquim on Automata, Languages, and Programming,* Berlin: Springer-Verlag, *Lecture Notes In Computer Science, 372,* Ed. G. Ausiello, M. Dezani-Ciancaglini, S. R. D. Rocca, pp. 123–136, 1989. Expanded version to appear in *Theor. Comput. Sci.*

[16] M. Burmester and Y. Desmedt, "Remarks on the soundness of proofs, *Electron. Lett.,*vol. 25 pp. 1509–1511, 1989.

[17] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," *J. Cryptology,* vol. 1, pp. 77–94, 1988.

[18] J. Feigenbaum and L. Fortnow, "On the random-self-reducibility of complete sets," University of Chicago Tech. Rept. 90-22, Computer Science Department, Aug. 20, 1990. Also in *Proc. 6th IEEE Structures,* pp. 124–132, 1991. Expanded version to appear in *SIAM J. Comput.*

[19] L. Fortnow, "On the complexity of perfect zero-knowledge," in *Advances in Computing Research—Vol. 5: Randomness and Computation* (S. Micali, ed.), Greenwich CT: Johnson Associates Inc. Press, pp. 327–343, 1989.

[20] L. Fortnow, J. Rompel, and M. Sipser, "On the power of multiprover interactive protocols," *Proc. 3rd IEEE Structures,* pp. 156–161, 1988.

[21] Z. Galil, S. Haber, and M. Yung, "Minimum-knowledge interactive proofs for decision problems," *SIAM J. Comput.,* vol. 18, pp. 711–739, 1989.

[22] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* San Francisco: Freeman, 1979.

[23] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a method of cryptographic protocol design," *Proc. 27th IEEE Sympo-*

sium on Foundations of Computer Science, pp. 174–187, 1986. Expanded version to appear in *J. Assoc. Comput. Mach.*

[24] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput,* vol. 18, pp. 186–208, 1989.

[25] S. Goldwasser and M. Sipser, "Public coins vs. private coins in interactive proof systems," in *Advances in Computing Research—Vol. 5: Randomness and Computation* (S. Micali, ed.), Greenwich CT: Johnson Associates Inc. Press, pp. 73–90, 1989.

[26] R. Impagliazzo and M. Yung, "Direct minimum-knowledge computations," *Proc. Crypto '87,* Berlin: Springer-Verlag, *Lecture Notes In Computer Science* 293,C. Pomerance, Ed., pp. 40–51, 1988.

[27] J. Kilian, *Use of Randomness in Algorithms and Protocols,* Ph.D. thesis, Massachusetts Institute of Technology: MIT Press, 1989.

[28] J. Kilian, Interactive Proofs Based on the Speed of Light, unpublished manuscript, Nov. 1990.

[29] R. Lipton, "New directions in testing," *Distributed Computing and Cryptography,* DIMACS Series on Discrete Mathematics and Theoretical Computer Science, AMS/ACM, 2 (1991) pp. 191–202.

[30] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *Proc. 31st IEEE Symposium on Foundations of Computer Science,* pp. 2–10, 1990.

[31] M. Naor, "Bit commitment using pseudo-randomness," *Proc. CRYPTO* 89, (G. Brassard, ed.), Berlin: Springer-Verlag *Lecture Notes In Computer Science* 435, pp. 128–136, 1989. Expanded version in *J. Cryptology,* vol. 4, pp. 151–158, 1991.

[32] N. Nisan, "CoSAT has multiprover interactive proofs," e-mail announcement, Nov. 22, 1989.

[33] C. Papadimitriou, "Games against nature," *J. Comput. Syst. Sci.,* vol. 31, pp. 288–301, 1985.

[34] A. Shamir, "IP = PSPACE," *Proc. 31st IEEE Symposium on Foundations of Computer Science,* pp. 11–15, 1990.

[35] S. Toda, "On the computational power of PP and ⊕P," *Proc. 30th IEEE Symposium on Foundations of Computer Science,* pp. 514–519, 1989.

[36] L. Valiant, "The complexity of computing the permanent," *Theor. Comput. Sci.,* vol. 8, pp. 189–201, 1979.

APPENDIX
COMPLEXITY THEORETIC BACKGROUND

The complexity classes that are most central to the ideas discussed in this chapter are as follows:

P: deterministic polynomial time; those membership problems that can be computed efficiently with no errors,

NP: nondeterministic polynomial time; those membership problems that can be verified efficiently,

#P: counting functions associated with NP sets; for example, the set of all satisfiable propositional formulas is called SAT and is in NP, and the function that maps a formula to the number of truth assignments that satisfy it is in #P,

PSPACE: deterministic polynomial space,

EXP: deterministic exponential time,

NEXP: nondeterministic exponential time, and

BPP: probabilistic polynomial time; those membership problems that can be computed efficiently with error probability at most $1/3$.

Other classes of interest include PH (the polynomial-time hierarchy Σ_i^p, Π_i^p, $i \geq 0$) and $\oplus$P (''parity-P''). The class Σ_i^p, is referred to as the ''i^{th} level'' of the PH.
The following inclusions are known.

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \ldots \subseteq \Sigma_i^p \subseteq \ldots \subseteq \mathrm{PH} \subseteq \mathrm{BPP}^{\oplus\mathrm{P}} \subseteq \mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP}$$

The only inclusion known to be proper is P $\subset$ EXP. One often hears that all these inclusions are ''believed'' to be proper, but ''belief'' (like ''knowledge'') is very hard to define mathematically. Suffice it to say that any resolution of the question of whether these inclusions are proper (e.g., a theorem of the form ''the PH collapses at the second level'' or ''PSPACE is contained in the second level of the PH'') would require a revolutionary new idea. See Garey and Johnson [22] for a thorough introduction to these issues.

We use the notation coC for the class of languages whose complements are in C and the notation coS for the complement of language S. Thus coSAT is the set of all unsatisfiable propositional formulas. The symbol Π_i^p is used for coΣ_i^p.

Finally, we will need the notion of *one-way function*. Intuitively, a function is one-way if it is easy to compute but hard to invert. For cryptography theory, we need that it is *often* hard to invert. One standard way to formalize this is as follows. Let f be computable in deterministic polynomial time. Then f is one-way if, for all polynomials q, for all probabilistic polynomial-time algorithms I (for ''inverter''), for all sufficiently large n, the probability that $f(I(f(x))) = f(x)$ is less than $1/q(n)$. This probability is computed over uniformly chosen x of length n and uniform coin-toss sequences for I.