SIAM J. COMPUT.
Vol. 18, No. 1, pp. 186-208, February 1989

012

THE KNOWLEDGE COMPLEXITY OF INTERACTIVE PROOF SYSTEMS*

SHAFI GOLDWASSER†, SILVIO MICALI†, AND CHARLES RACKOFF‡

Abstract. Usually, a proof of a theorem contains more knowledge than the mere fact that the theorem is true. For instance, to prove that a graph is Hamiltonian it suffices to exhibit a Hamiltonian tour in it; however, this seems to contain more knowledge than the single bit Hamiltonian/non-Hamiltonian.

In this paper a computational complexity theory of the "knowledge" contained in a proof is developed. Zero-knowledge proofs are defined as those proofs that convey no additional knowledge other than the correctness of the proposition in question. Examples of zero-knowledge proof systems are given for the languages of quadratic residuosity and quadratic nonresiduosity. These are the first examples of zero-knowledge proofs for languages not known to be efficiently recognizable.

Key words. cryptography, zero knowledge, interactive proofs, quadratic residues

AMS(MOS) subject classifications. 68Q15, 94A60

1. Introduction. It is often regarded that saying a language L is in NP (that is, acceptable in nondeterministic polynomial time) is equivalent to saying that there is a polynomial time "proof system" for L. The proof system we have in mind is one where on input x, a "prover" creates a string α, and the "verifier" then computes on x and α in time polynomial in the length of the binary representation of x to check that x is indeed in L. It is reasonable to ask if there is a more general, and perhaps more natural, notion of a polynomial time proof system. This paper proposes one such notion.

We will still allow the verifier only polynomial time and the prover arbitrary computing power, but will now allow both parties to flip unbiased coins. The result is a probabilistic version of NP, where a small probability of error is allowed. However, to obtain what appears to be the full generality of this idea, we must also allow the prover and verifier to *interact* (i.e., to talk back and forth) and to *keep secret* their coin tosses. We call these proof systems "interactive proof systems." This notion is formally defined in § 2, where we also define what it means for a language to have an interactive proof system.

It is far from clear how to use this power of interaction. Languages with nondeterministic polynomial time algorithms or with probabilistic polynomial time algorithms have proof systems with little or no interaction. We would therefore like examples of languages that appear to have neither nondeterministic nor probabilistic polynomial time algorithms, and yet have interactive proof systems. Although we do not present any such examples here, there are now examples in the literature. Using ideas from an initial version of this paper [GMR] Goldreich, Micali, and Wigderson [GMW] have shown that the "graph nonisomorphism" language has an interactive

* Received by the editors August 26, 1985; accepted for publication (in revised form) April 18, 1988. A preliminary version of this paper appeared in the Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 174–187.

Editor's Note. This paper was originally scheduled to appear in the February 1988 Special Issue on Cryptography (SIAM J. Comput., 17 (1988)).

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The work of these authors was supported by National Science Foundation grants DCR-84-13577 and DCR-85-09905.

‡ Computer Science Department, University of Toronto, Toronto, ONT M5S 1A4 Canada. The work of this author was supported by the Natural Sciences and Engineering Research Council of Canada under grant A3611.

proof system. Independently of this paper, Babai and Szemeredi [BS] show that certain matrix group problems have what they call "Arthur-Merlin" proof systems, which immediately implies that they have interactive proof systems. In fact, the notion of an Arthur-Merlin proof system consists of a restricted interactive proof system in which the prover sees the coin flips of the verifier. Nevertheless, Goldwasser and Sipser [GS] have shown that a language has an interactive proof system if and only if it has an Arthur-Merlin proof system.

It appears, however, that our notion of interactive proof systems generalizes in the right way to attack a novel problem. The main purpose of this current paper, in fact, is to use interactive proof systems to investigate a natural question; how much knowledge is transmitted to the verifier in an interactive proof system for L? For example, consider SAT, the NP-complete language of satisfiable sentences of the propositional calculus. In the obvious proof system, to prove $F \in \text{SAT}$, the prover gives a satisfying assignment I for the formula F, which the verifier then checks in polynomial time. This assignment gives the verifier much more knowledge than merely the fact that $F \in \text{SAT}$; it also gives a satisfying assignment. At the other extreme, every language that can be accepted in probabilistic polynomial time has a proof system in which the prover does nothing, and hence gives no knowledge to the verifier.

We say an interactive proof system for L is *zero-knowledge* if for each $x \in L$, the prover tells the verifier essentially nothing, other than that $x \in L$; this should be the case even if the verifier chooses not to follow the proof system but instead tries (in polynomial time) to trick the prover into revealing something. The notion of zero-knowledge is formally defined in § 3. This definition is an important contribution of this paper.

The main technical contributions of this paper are the proofs in §§ 5 and 6 that the languages QR and QNR (defined below) both have zero-knowledge interactive proof systems. These are the first zero-knowledge protocols demonstrated for languages not known to be recognizable in probabilistic polynomial time. To understand the languages QR and QNR, it helps to read the (brief) number theory background given in § 4. However, for now, let x, y be integers, $0 < y < x$, such that $\gcd(x, y) = 1$; we say that y is a *quadratic residue* mod x if $y = z^2 \bmod x$ for some z; if not, we say that y is a *quadratic nonresidue* mod x. We define

$$\text{QR} = \{(x, y) | y \text{ is a quadratic residue mod } x\}, \quad \text{and}$$

$$\text{QNR} = \{(x, y) | y \text{ is a quadratic nonresidue mod } x\}.$$

(Actually, QNR will be defined slightly differently in § 4.) Both QR and QNR are in NP, and thus possess an elementary proof system. For instance, to prove membership in QNR, the prover just sends x's factorization. But looking ahead to zero-knowledge proof systems, let us discuss a more interesting example of a proof system for QNR.

Example 1. Let us call the prover A and the verifier B. Say that the input is (x, y). Let $n = |x|$, where $|x|$ is the length of the binary representation of x. We will now describe (omitting some details) an interactive proof system for QNR. B begins by flipping coins to obtain random bits $b_1, b_2, \cdots, b_n$. B then flips additional coins to obtain a string α, from which B computes $z_1, z_2, \cdots, z_n$ such that each z_i is a random z, $0 < z < x$, $\gcd(x, z) = 1$. B then computes $w_1, w_2, \cdots, w_n$ as follows: for each i, if $b_i = 0$ then $w_i = z_i^2 \bmod x$; if $b_i = 1$ then $w_i = (z_i^2 y) \bmod x$. B then sends $w_1, w_2, \cdots, w_n$ to A. For each i, A computes (somehow) whether or not w_i is a quadratic residue mod x, and sends B a sequence of bits $c_1, c_2, \cdots, c_n$, where $c_i = 0$ if and only if w_i is a quadratic residue mod x. B checks if $b_i = c_i$ for every i, and if so is "convinced" that $(x, y) \in \text{QNR}$.

It is not hard to see that if $(x, y) \in \text{QNR}$ and both parties follow the protocol, then B will become "convinced." On the other hand, if y is a quadratic residue mod x, then so is each w_i, and every value for w_i is equally likely; since A does not see the sequence $\{b_i\}$, the probability that every $c_i = b_i$ (and hence that B will be convinced) is 2^{-n}. So this is an interactive proof system for QNR. Let us now address the question of how much knowledge A may release.

It is an interesting question how zero-knowledge should be defined. If the prover is trying to prove to the verifier that y is a quadratic residue mod x, then certainly the verifier should not be able to trick the prover into revealing a square root of y mod x, or the factorization of x, or any information which would help the verifier to compute these things much faster than before. In fact, the prover should not reveal anything which would help the verifier compute *anything* much faster than before. The way to state this formally seems to be that what the verifier sees in the protocol (even if he cheats) should be something which the verifier could have computed for himself, merely from the fact that $(x, y) \in \text{QNR}$. Of course, what the verifier sees in the protocol is really a probability distribution. Thus, zero-knowledge means that one can compute in polynomial time, from $(x, y) \in \text{QNR}$, without a prover, the same (or almost the same) probability distribution that the verifier would see with the prover. This is defined formally in § 3. Here, let us informally discuss whether the above interactive proof system for QNR is zero-knowledge.

Consider a pair $(x, y) \in \text{QNR}$, and say that A follows the protocol. Can B obtain any additional knowledge? For the moment, assume that B follows the protocol. B "sees" $[\{b_i\}, \alpha, \{w_i\}, \{c_i\}]$ distributed according to a particular distribution. Without any help from a prover, we can quickly generate a random string according to this distribution: just choose $\{b_i\}$ and α randomly, and then compute $\{w_i\}$ from them; then compute $c_i = b_i$ for each i.

Now what if B were to cheat? B could begin by setting $w_1 = 42$, and then behave correctly. Consider now the induced distribution on $[\{b_i\}, \alpha, \{w_i\}, \{c_i\}]$; in order to compute a random member of it (without help from a prover), we must compute whether or not 42 is a quadratic residue x, given x and a quadratic nonresidue y. At this time it is not known how to compute this in polynomial time, so this proof system may not be zero-knowledge.

A zero-knowledge proof system for QNR is given in § 6. The ideas of this proof system partially come from the secret exchanging protocol of Luby, Micali, and Rackoff [LMR] and are useful there as well. They have also proved useful in the oblivious transfer protocol of Fischer, Micali, Rackoff, and Witenberg [FMRW] and for the identification scheme of Feige, Fiat, and Shamir [FFS]. These ideas have also helped Goldreich, Micali, and Wigderson [GMW] to show that the languages of "graph isomorphism" and "graph nonisomorphism" have zero-knowledge interactive proof systems.

Although we find the idea of a zero-knowledge interactive proof system fascinating in itself, the main motivation for it and the main applications of it are in the area of cryptographic protocols. For example, in the secret exchanging protocol in [LMR], one person wishes to exchange a secret with another without giving away any additional knowledge. Ideas similar to those here must be used to even define this concept.

More generally, however, it often arises at some point in a protocol that A wishes to convince B of some fact. Say that we know that the protocol would be secure if at this point an angel or someone B trusted were to tell B (truthfully) if A is telling the truth. We want the notion of zero-knowledge to be such that an appropriate zero-knowledge interactive proof system could be inserted at this point instead of the trusted

party, and the whole protocol would remain secure. (Of course, A would have to possess some additional information enabling her to implement the part of the prover efficiently.) In particular instances, we can prove that such substitution works, but a general framework for discussing protocols must exist before the general theorem can even be stated. However, based on intuition and experience, the authors (and many others who have studied these ideas since their initial appearance) believe that the definition of zero-knowledge proposed here has the required properties.

The most important development since these results first appeared is the proof by Goldreich, Micali, and Wigderson [GMW] that, subject to a common complexity theory assumption, every language in NP has a zero-knowledge interactive proof system. These proof systems for NP languages appear to have applications in just about every protocol problem. It is almost certain that these results will vastly simplify distributed cryptographic protocol design in the future, as demonstrated by the powerful results of [GMW2].

2. Interactive proof systems. Intuitively, what should we require from an efficient theorem-proving procedure?

(1) That it should be possible to "prove" a true theorem.
(2) That it should not be possible to "prove" a false theorem.
(3) That communicating the "proof" should be *efficient.* Namely, regardless of how much time it takes to come up with the proof, its correctness should be efficiently verified.

The NP formalization of the concept of an efficient proof system captures one way of communicating a proof. In this section, we will generalize the NP proof system to capture a more general way of communicating a proof. The verifier will be a probabilistic polynomial time (in the length of the common input) machine that is able to exchange messages (strings) with the prover.

At the same time that we introduce probability into the proof system, we relax the classical notion of a "proof." Our verifier may erroneously be convinced of the truth of a proposition with a very small probability of error (less than n^{-k} for each positive constant k and all sufficiently large input-sizes n).

We proceed to formally define the new system.

2.1. Interactive Turing machines and protocols.

DEFINITION. An *interactive Turing machine* (ITM) is a Turing machine equipped with a read-only input tape, a work tape, a random tape, one read-only communication tape, and one write-only communication tape. The random tape contains an infinite sequence of random bits, and can be scanned only from left to right. We say that an interactive machine *flips a coin,* meaning that it reads the next bit in its own random tape.

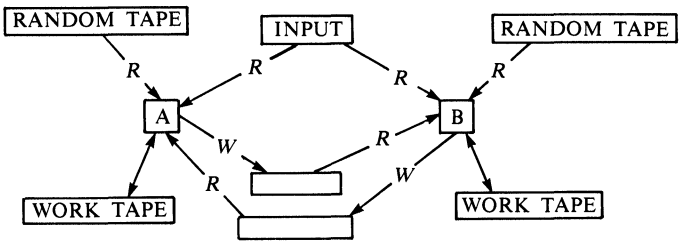

FIG. 1. *An interactive protocol. — denotes a read/write head, R a read-only head, W a write-only head.*

DEFINITION. An *interactive protocol* is an ordered pair of ITM's A and B such that A and B share the same input tape, B's write-only communication tape is A's read-only communication tape and vice versa. Machine A is not computationally bounded, while machine B's computation time is bounded by a polynomial in the length of the common input. The two machines take turns in being active, with B being active first. During an active stage machine $A(B)$ first performs some internal computation using its input tape, work tapes, communication tape and random tape; and, second, it writes a string (for $B(A)$) on its write-only communication tape. The ith message of $A(B)$ is the entire string that $A(B)$ writes on its communication tape during its ith active stage. As soon as machine $A(B)$ writes its message, it is deactivated and machine $B(A)$ becomes active, unless the protocol has been terminated. Either machine can terminate the computation of the protocol by not sending any message in an active stage. Machine B accepts (or rejects) the input by outputting *accept* (or *reject*) and terminating the protocol. The *computation time* of machine B is the sum of the B's computation time during its active stages, and it is this time that is bounded by a polynomial in the length of the input, denoted $|x|$.

2.2. Interactive proof systems.

DEFINITION. Let L be a language over $\{0, 1)^*$. Let (A, B) be an interactive protocol. We say that (A, B) is an *interactive proof system* for L if we have the following:

(1) For each k, for sufficiently large x in L given as input to (A, B), B halts and accepts with probability at least $1-|x|^{-k}$. (The probabilities here are taken over the coin tosses of A and B.)

(2) For each k, for sufficiently large x not in L, for any ITM A', on input x to (A', B), B accepts with probability at most $|x|^{-k}$. (The probabilities here are taken over the coin tosses of A' and B.)

Remark 1. The above probability of error can be decreased, say to smaller than $2^{-|x|}$, by the standard technique of repeating the protocol many times and choosing to accept by majority vote.

We now argue that this definition captures what we intuitively want from an efficient proof system. Condition (1) essentially says that, if $x \in L$, B will accept with overwhelming probability. Condition (2) says that, if x is not in L, there exists no strategy that succeeds with nonnegligible probability for convincing B to accept. In fact, B needs not to trust (or know the program of) the machine with which it is interacting. It is enough for B to trust the randomness of its own coin tosses.

Similar to the NP proof system, note that the definition of an interactive proof system for a language emphasizes the "yes-instances": when a string is in the language, B must be led to acceptance with high probability, but when a string is not in the language A is not required to convince B of the contrary.

A more general version of the above definition is where A is not a Turing machine, but an infinite state machine. However it has been shown by Feldman in [F] that this adds no extra power to the model. In fact, he shows that with respect to language recognition it is sufficient for A to be a deterministic PSPACE machine. The fact that A is probabilistic is of importance to the more subtle definition of zero-knowledge, which is given in the next section.

We define IP, *Interactive Polynomial time*, to be the class of languages for which there exists interactive proof systems.

The first examples of a language in IP but not known to be in NP have been exhibited by Babai and Szemeredi [BS]. Their examples are "matrix group nonmembership" and "matrix group order," where the matrix groups over finite fields are represented by a list of generator matrices. The other, more well-known example

of "graph nonisomorphism" is due to Goldreich, Micali, and Wigderson [GMW]. They have shown that the language of pairs of graphs that are nonisomorphic to each other is in IP.

2.3. Arthur–Merlin games. Babai independently conceived the notion of an "Arthur-Merlin Game," an interactive proof system in which Merlin plays the role of A and Arthur that of B. The interaction, though, is less "liberal" than in our model since Merlin sees all the coin tosses of Arthur. A message from Arthur to Merlin can only consist of a randomly selected string. In an interactive proof system, instead, the verifier is allowed, given a polynomial time computable function f, to secretly select a random string R and transmit only $f(R)$ to the prover (as in the interactive proof system for QNR of Example 1).

This restriction immediately implies that the languages recognized by an Arthur-Merlin game are a subset of those having an interactive proof system. Interestingly, Goldwasser and Sipser [GS] show that they are not a proper subset. However, there is value in having both definitions around. It is easier to design protocols using the interactive proof systems definition, and it is easier to prove complexity results using the Arthur-Merlin definition.

Though the ability to make secret random choices does not help us to recognize more languages, we believe that it is crucial for recognizing languages in zero-knowledge. We make precise this belief in the conjecture of § 3.7.

An Arthur-Merlin type of interactive proof system was already implicitly used in a paper by Blum [B1]. He showed an interactive protocol for recognizing the language of the Blum integers:

$$\mathrm{BL} = \{n \mid p^{\alpha} \text{ divides } n \text{ and } p^{\alpha+1} \text{ does not, where } p \equiv 3 \bmod 4 \text{ is prime and } \alpha \text{ is odd}\}.$$

The prover's goal was to demonstrate membership in BL without having to send n's prime factorization. In this proof system, the verifier talks only once and his message consists of sending the sequence of his coin tosses. Protocols of this type were also found by Goldwasser and Micali [GM1] to prove, without releasing the prime factorization membership in the languages:

$\mathrm{GM1} = \{n \mid n \text{ has exactly two distinct prime divisors}\}$ and

$\mathrm{GM2} = \{(x, n) \mid \gcd(x, \phi(n)) = 1\}$ where $\phi(n)$ is the number of positive integers smaller and relatively prime to n.

3. Zero-knowledge. Rather than giving the definition of zero-knowledge only for interactive proof systems, we will give a more general definition. We will define what it means for any interactive protocol (A, B) to be *zero-knowledge* for a language L, whether or not (A, B) is a proof system for L. Actually the definition will not depend on B at all; as we shall see, it says that for every polynomial time B', the distribution that B' "sees" on all its tapes, when interacting with A on input $x \in L$, is "indistinguishable" from a distribution that can be computed from x in polynomial time. We thus first focus on the notion of indistinguishability for random variables.

3.1. Indistinguishability of random variables. Throughout this paper, we will only consider families of random variables $U = \{U(x)\}$ where the parameter x is from a language L, a particular subset of $\{0, 1\}^*$, and all random variables take values in $\{0, 1\}^*$. Let $U = \{U(x)\}$ and $V = \{V(x)\}$ be two families of random variables. We want to express the fact that, when the length of x increases, $U(x)$ essentially becomes "replaceable" by $V(x)$. To do this, we consider the following framework.

A random sample is selected either from $U(x)$ or from $V(x)$ and it is handed to a "judge." After studying the sample, the judge will proclaim his verdict: 0 or 1. (We

may interpret 0 as the judge's decision that the sample came from $U(x)$; 1 as the decision that the sample came from $V(x)$.) It is then natural to say that $U(x)$ becomes "replaceable" by $V(x)$ for x long enough if, when x increases, the verdict of *any* judge becomes "meaningless," that is, essentially uncorrelated to the distribution from which the sample came.

There are two relevant parameters in this framework: the *size* of the sample and the amount of *time* the judge is given to produce his verdict. By bounding these two parameters in different ways we obtain different notions of indistinguishability for random variables. We focus on the three notions we believe to be the most important: equality, statistical indistinguishability, and computational indistinguishability. Roughly speaking, these notions correspond to the following restrictions on the relevant parameters. If the two families of random variables $\{U(x)\}$ and $\{V(x)\}$ are equal, then the judge's verdict will be meaningless even if he is given samples of arbitrary size and he can study them for an arbitrary amount of time. We will define the two families to be statistically indistinguishable if the judge's verdict becomes meaningless when he is given an infinite amount of time but only random, polynomial (in $|x|$) size samples to work on. We will define the two families to be computationally indistinguishable if the judge's verdict becomes meaningless when he is only given polynomial ($|x|$)-size samples and polynomial ($|x|$) time. Let us now proceed to formalize these notions.

DEFINITION (Statistical indistinguishability). Let $L \subset \{0,1\}^*$ be a language. Two families of random variables $\{U(x)\}$ and $\{V(x)\}$ are *statistically indistinguishable on* L if

$$\sum_{\alpha \in \{0,1\}^*} |\text{prob}(U(x)=\alpha) - \text{prob}(V(x)=\alpha)| < |x|^{-c}$$

for all constants $c>0$ and all sufficiently long $x \in L$.

Notice that, for U and V as above, if a "judge" is handed a polynomial (in $|x|$) size sample, having infinite computing power will not help him to decide whether it came from $U(x)$ or $V(x)$. His answer will be essentially useless, as he will say "1" with essentially the same probability in both cases.

Example 2. Let $U(x)$ assign equal probability to all strings of length $|x|$, and let $V(x)$ assign probability $2^{-|x|}$ to all strings of length $|x|$, except for $0^{|x|}$, which is given probability 0 and for $1^{-|x|}$, which is given probability $2^{-|x|+1}$. Then $\{U(x)\}$ and $\{V(x)\}$ are two families of random variables statistically indistinguishable on $\{0,1\}^*$.

To formalize the notion of computational indistinguishability we make use of nonuniformity (the reasons for this choice can be found in § 3.4). Thus, our "judge," rather than being a polynomial time Turing machine, will be a *poly-size family of circuits.* That is a family $C=\{C_x\}$ of Boolean circuits C_x with one Boolean output such that, for some constant $e>0$, all $C_x \in C$ have at most $|x|^e$ gates. In order to feed samples from our probability distributions to such circuits, we will consider only *poly-bounded* families of random variables, that is, families $U=\{U(x)\}$ such that, for some constant $d>0$, all random variable $U(x) \in U$ assigns positive probability only to strings whose lengths are exactly $|x|^d$. If $U=\{U(x)\}$ is a poly-bounded family of random variables and $C=\{C_x\}$ a poly-size family of circuits, we denote by $P(U, C, x)$ the probability that C_x outputs 1 on input a random string distributed according to $U(x)$. (Here we assume that strings assigned positive probability by $U(x)$ have lengths equal to the number of Boolean inputs of C_x.)

DEFINITION (Computational indistinguishability). Let $L \subset \{0,1\}^*$ be a language. Two poly-bounded families of random variables U and V are *computationally indistinguishable on* L if for all poly-size family of circuits C, for all constants $c>0$ and all

sufficiently long strings $x \in L$,

$$|P(U, C, x) - P(V, C, x)| < |x|^{-c}.$$

This notion of computational indistinguishability was already used by Goldwasser and Micali [GM] in the context of encryption and by Yao [Y] in the context of pseudorandom generation. It is trivial that if U and V are identical, then they are statistically indistinguishable. It is also not hard to see that if U and V are statistically indistinguishable, then they are computationally indistinguishable, as follows. Let C_x be a circuit and let S be the set of inputs on which C_x outputs 1. Since U and V are statistically indistinguishable, the value of $U(x)$ will be in S with almost exactly the same probability that the value of $V(x)$ will be. Hence $P(U, C, x)$ will be very close to $P(V, C, x)$.

Example 3. Consider a probabilistic encryption algorithm that is secure in the sense of Goldwasser and Micali [GM]; for n integer, let $U(1^n)$ and $V(1^n)$ be the random variables taking as values the possible encryptions of 0 and 1, respectively, on security parameter n. Then U and V are computationally indistinguishable on $L = \{1\}^*$.

We believe that the notion of computational indistinguishability for random variables achieves the right level of generality. Thus we will call *indistinguishable* any two families of random variables that are computationally indistinguishable.

Remark 2. Let us point out the robustness of the last definition. In this definition, we are handing our computationally bounded "judge" only samples of size 1. This, however, is not restrictive. We note that two families of random variables $\{U_x\}$ and $\{V_x\}$ are computationally indistinguishable (with respect to samples of size 1) if and only if when C_x is given a polynomial in $|x|$ number of input strings, each independently generated according to the distribution U_x, then the probability of accepting is close to the probability of accepting when V_x is used.

3.2. Approximability of random variables. We now formalize the notion that a random variable U is essentially easy to generate. That is, there exists an efficient algorithm that randomly outputs strings in a way that is indistinguishable from U.

DEFINITION. Let M be a probabilistic Turing machine that on input x halts with probability 1. We denote by $M(x)$ the random variable that, for each string α, takes on α with exactly the same probability that M on input x outputs α.

DEFINITION. Let $L \subset \{0, 1\}^*$ be a language and $U = \{U(x)\}$ a family of random variables. We say that U is *perfectly approximable on* L if there exists a probabilistic Turing machine M, running in expected polynomial time, such that for all $x \in L$, $M(x)$ is equal to $U(x)$. We say that U is *statistically* (*computationally*) *approximable on* L if there exists a probabilistic Turing machine M, running in expected polynomial time, such that the families of random variables $\{M(x)\}$ and $\{U(x)\}$ are statistically (computationally) indistinguishable on L.

In what follows, we will use *approximability* to mean computational approximability. We are now ready to define the notion of zero-knowledge.

3.3. Zero-knowledge protocols and proof systems. We first address the issue of a "cheating verifier," B', who is allowed not to follow the protocol.

DEFINITION. Let (A, B) be an interactive protocol. Let B' be an interactive Turing machine that has as input x and on an extra input tape H, where the length of H is bounded above by a polynomial in the length of x. (Figure 1 must be emended to allow B' this extra tape.) When B' interacts with A, A sees only x on its input tape, whereas B' sees (x, H). A good way to think of H is as some knowledge about x that

the cheating B' already possesses. Alternatively, H can be considered as the history of previous interactions that the cheating B' is trying to use to get knowledge from A; this is discussed in more detail in the next section. We assume that the total computation time of B' when interacting with A will be bounded above by a polynomial in the length of x.

For a run of the protocol on common input x and extra input H, we define the view of B' to be everything that B' sees. Namely, let σ (and ρ) be the strings contained in the random tapes of A (and B'). Say the computation of A and B', with these random choices, consist of n turns with B' going first, where a_i (and b_i) are the ith messages of A (and B'), respectively. Then, we say that $(\rho, b_1, a_1, \cdots, b_n, a_n)$ is the *view* of B' on inputs x and H, and let $\text{View}_{A,B'}(x, H)$ be the random variable whose value is this view. (Note that it would make no difference if we included in the view the material written by B' on its private tape, or excluded the strings that B' sends to A, since these bits can be efficiently computed from the other bits of the view.) For convenience, we consider each view to be a string from $\{0, 1\}^*$ of length exactly $|x|^c$ for some fixed $c > 0$.

DEFINITION. Let $L \subset \{0, 1\}^*$ be a language and (A, B) a protocol. Let B' be as above. We say that (A, B) is *perfectly* (*statistically*) (*computationally*) *zero-knowledge on L for* B' if the family of random variables $\text{View}_{A,B'}$ is perfectly (statistically) (computationally) approximable on

$$L' = \{(x, H) \mid x \in L \text{ and } |H| = |x|^c\}.$$

We say that (A, B) is *perfectly* (*statistically*) (*computationally*) *zero-knowledge on L* if it is perfectly (statistically) (computationally) zero-knowledge on L for all probabilistic polynomial time ITM B'.

Note that the definition of (A, B) being zero-knowledge (in some manner) for B' only depends on A and not at all on B. It might be less misleading to think of A as being zero-knowledge for B'. A similar issue arises in the definition of an interactive proof system in § 2.2. Part (2) of this definition depends only on B, and not at all on A.

Computational zero-knowledge is certainly the most general of the above notions, and we will refer to it simply as *zero-knowledge.* Zero-knowledge really captures any information, which could not have been obtained efficiently in polynomial time, about members of L. That is, if (A, B) is zero-knowledge, it is not possible, in probabilistic polynomial time, to extract any information about members of L by interacting with A, not even by "cheating."

DEFINITION. Let $L \subset \{0, 1\}^*$ be a language. We say that (A, B) is a *perfectly* (*statistically*) (*computationally*) *zero-knowledge proof system for L* if it is an interactive proof system for L and a perfectly (statistically) (computationally) zero-knowledge protocol on L.

We will refer to computationally zero-knowledge proof systems (the most general notion of the three) simply as *zero-knowledge proof systems.* This notion is totally adequate in the real world. That is, if (A, B) is a zero-knowledge proof system for L, it is not possible in polynomial time (not even for a "cheating" B') to interact with A and extract anything else besides proofs of membership in L.

3.4. Some remarks about the above definitions. First, let us stress that the coin tosses of B are an essential part of the notion of a view in the definition of zero-knowledge for B. Consider the language L of all composite integers and the following

protocol (A, B). On input an integer n, B randomly selects an integer x between 1 and n and relatively prime with n. It then sends A the number $a = x^2 \bmod n$. A responds by sending y, a randomly chosen square root of a mod n. Should the view consist only of the text of the interaction between A and B, then the above-mentioned protocol would be perfectly zero-knowledge on L. However, we define the view so as to contain also the coin tosses of B. Thus, if (x, a, y) is randomly selected in $\text{View}_{(A,B)}(n)$, then $\gcd(x+y, n)$ is not 1 or n with probability at least $\frac{1}{2}$. Thus, if factoring is not in probabilistic polynomial time, the above protocol is not zero-knowledge for B. (It should be noted, however, that in the definition of zero-knowledge (for *all* B'), it is *not* necessary to include the random bits of B' in the view; this is because we have included in the view the messages sent by B', and for every B' there is a B'', which is like B' except that it sends its random bits as part of its last message.)

Second, it should be explained why B' sees an additional string H. (The need for this was independently discovered by the authors of this paper, by Oren [O], and by Tompa and Woll [TW].) H may be thought about in a number of different ways; H may be some extra information that the verifier (cheating or not) happens to know. For example, a zero-knowledge protocol for graph isomorphism should remain zero-knowledge even if the verifier happens to know colorings for the graphs. It is also possible that the protocol will be inserted in the middle of another protocol, where the verifier has seen some history H. There is the fear that this H was generated perhaps by interacting with a machine of unlimited power; we want to rule out the possibility of the verifier then obtaining knowledge by using H when interacting with A. It is for a similar reason that the distinguishing circuits in the definition of computational indistinguishability are allowed to be nonuniform. We want to say that two families of random variables can be computationally distinguished if there are circuits that tell them apart, where the circuits may have wired in some information about x or some information obtained from the history of some protocol in which the protocol of interest is immersed. In other words, the view B' obtains on inputs (x, H) should look like the simulated distribution $M(x, H)$.

One test of a definition is that we should be able to prove those facts that intuition dictates *must* be true. One such fact is that the repetition of a zero-knowledge protocol a polynomial number of times is still zero-knowledge. We can prove this with our current definitions, but we cannot prove it if, for example, we do not give B' the extra string H. We can also prove that if B' wants to decide a special predicate $P(x)$ for $x \in L$, it does not help B' to engage in a zero-knowledge proof system for L. We can prove all the intuitively obvious "facts" we have tried to prove, but only time will tell if these definitions are the right ones, or if they need some further modifications. We feel (and hope) that these definitions capture exactly the intuitive ideas we have tried to capture.

Last, there is the peculiar fact that the machine M simulating the view is allowed to operate in *expected* polynomial time. This appears to be necessary for the zero-knowledge proof system for QNR given in § 6. To see why this is necessary in general, consider a pair (A, B), where B (on input of length n) sends n random bits α to A; if the predicate $P(\alpha)$ holds, then A sends a random β of length n such that $P(\beta)$ holds. Imagine that the predicate P is easily computable, but the number of strings of length n for which P holds is small—maybe a fraction n^{-10} or maybe n^{-20}—but we do not know exactly which; imagine that the only way we know to find a string for which P holds is to select random strings until one satisfying P is found. The only way we know how to simulate the view of B statistically closely (or even computationally indistinguishably) is to choose a random α; if $P(\alpha)$ holds, look through random n-bit

strings until a β is found such that $P(\beta)$ holds. This process is only expected polynomial time.

3.5. Examples of zero-knowledge languages. Trivially, all languages in BPP have perfect zero-knowledge proof systems. (A language is in BPP if there is a probabilistic, polynomial time machine which on each input computes membership in the language with small probability of error.)

The first nontrivial zero-knowledge proof systems (i.e., for recognizing languages not known to be in BPP) are the perfect zero-knowledge proof systems for the quadratic residuosity language QR given in § 5, and the statistically zero-knowledge proof system for QNR given in § 6.

Recently, Goldreich, Micali, and Wigderson have shown in [GMW] that the graph isomorphism language has a perfect zero-knowledge proof system, that the graph nonisomorphism language (though not known to belong to NP) has a statistically zero-knowledge proof system, and that all languages in NP possess computationally zero-knowledge proof systems if secure encryption schemes exist. In [BGGHKMR] it is proved that all languages in IP possess zero-knowledge proof systems.

Results by Boppana, Hastad, and Zachos [BHZ] and Fortnow [Fo] show that if an NP-complete language had a perfect or statistically zero-knowledge proof system, the polynomial time hierarchy would collapse. Thus it may not be surprising that the interactive proof systems in [GMW] for graph coloring were zero-knowledge only in a computational sense. A more immediate reason for their being computationally zero-knowledge is that they make use of probabilistic encryption [GM] (see Example 3). This may tempt us to interpret Fortnow's result as saying that encryption is crucial in any zero-knowledge proof system for NP-complete languages. (Further discussion on Fortnow's result can be found in § 7.)

3.6. A study of earlier proposals. Having reached the notion of a zero-knowledge proof system, let us now have a second look at the earlier, Arthur–Merlin type, proof systems of Blum [Bl] and Goldwasser and Micali [GM1] that we have already mentioned in § 2.3.

The Proof System for BL (defined in § 2.3). In his beautiful paper, assuming that integer factorization is computationally hard, Blum proposes a protocol for flipping a coin over the telephone. For "fairly" flipping a coin, Alice and Bob need an integer n whose prime factorization is known to Alice but not to Bob, and has a special property, namely, $n \in \mathrm{BL}$. Alice is sure that the coin flip is fair because she computes n by multiplying two randomly selected primes both congruent to 3 mod 4, and because she trusts that factoring is hard. Bob, after the coin has come up Head or Tail, checks that the flipping was fair by requesting n's factorization from Alice. Thus a different n should be selected for each coin flip. To make coin flipping more efficient, Blum proposed to test that $n \in \mathrm{BL}$, by means of a protocol that does not give away n's factorization in any obvious way. After slightly modifying it, Blum's protocol can be proved to be perfect zero-knowledge on BL. However, without this modification, it is not clear how much knowledge about n's factorization it releases.

The Proof Systems for GM1 *and* GM2 (defined in § 2.3). The cryptographic protocols of Goldwasser and Micali use the inefficient version of Blum's coin-flipping protocols, and thus assume that factoring integers is computationally difficult. Based on this assumption, they showed that their proof systems do not give away the prime factorization of an input n. That is, no cheating polynomial time verifier can, after participating in the protocol, compute n's factorization much faster than it could before.

More generally, their same protocols can actually be proved to be computationally zero-knowledge on, respectively, the languages GM1 and GM2. Thus, in particular, their protocols do not even give away whether or not, say, 3 is a quadratic residue mod n.

3.7. Interactive proof systems versus Arthur–Merlin games for zero-knowledge. We are now ready to formally express our belief that interactive proof systems are more appropriate than Arthur–Merlin games for recognizing languages in zero-knowledge.

CONJECTURE. There exist languages L that have perfect or statistical zero-knowledge proof systems, but do not have any Arthur–Merlin proof system that is perfect or zero-knowledge on L.

4. The quadratic residuosity problem. In this section we describe the necessary number-theoretic background and notation needed for the proofs in §§ 5 and 6.

Let $\mathbf{N}$ denote the natural numbers, $x \in \mathbf{N}$ and $\mathbf{Z}_x^* = \{y \mid 1 \leqq y < x, \gcd(x, y) = 1\}$. We can determine in time polynomial in $|x|$ and $|y|$ whether or not $y \in \mathbf{Z}_x^*$.

We say that y in $\mathbf{Z}_x^*$ is a *quadratic residue* mod x if there exists a w in $\mathbf{Z}_x^*$ such that $w^2 \equiv y \bmod x$. Otherwise, we call y in $\mathbf{Z}_x^*$ a *quadratic nonresidue* mod x.

FACT 1. Let $x \in \mathbf{N}$ and $y \in \mathbf{Z}_x^*$. Then, y is a quadratic residue mod x if and only if it is a quadratic residue mod all of the prime factors of x.

Define the quadratic residuosity predicate to be

$$Q_x(y) = \begin{cases} 0 & \text{if } y \text{ is a quadratic residue mod } x, \\ 1 & \text{otherwise.} \end{cases}$$

Then we have the following fact.

FACT 2. Let $x \in \mathbf{N}$ and $y \in \mathbf{Z}_x^*$. Given y and the prime factorization of x, $Q_x(y)$ can be computed in time polynomial in $|x|$.

Let $y \in \mathbf{Z}_x^*$ and the prime factorization of x be $\prod_{i=1}^{k} p_i^{\alpha_i}$. Then, the Jacobi symbol of y mod x is defined as

$$(y/x) = \prod_{i=1}^{k} (y/p_i)^{\alpha_i},$$

where $(y/p_i) = 1$ if y is a quadratic residue mod p_i, and -1 otherwise.

FACT 3. Given $x \in \mathbf{N}$ and $y \in \mathbf{Z}_x^*$, (y/x) can be computed in time polynomial in $|x|$.

The Jacobi symbol of y mod x gives some information about whether y is quadratic residue mod x or not. If $(y/x) = -1$, then y is a quadratic nonresidue mod x and $Q_x(y) = 1$. However, when $(y/x) = 1$, no efficient (probabilistic or deterministic polynomial time) solution is known for computing $Q_x(y)$ correctly with probability significantly better than $\frac{1}{2}$. This leads to the formulation of the quadratic residuosity problem.

DEFINITION. We define the *quadratic residuosity problem* as that of computing $Q_x(y)$ on inputs x and y, where y in Z_x^* and $(y/x) = 1$,

The current best algorithm for computing $Q_x(y)$, is to first factor x and then compute $Q_x(y)$. In fact, factoring integers and computing Q_x have been conjectured to be of the same time complexity. The difficulty of the quadratic residuosity problem has been used as a basis for the design of several cryptographic protocols [GM], [LMR], [Bl].

Define the following two languages:

$$\mathrm{QR} = \{(x, y) \mid x \in \mathbf{N}, y \in \mathbf{Z}_x^*, \text{ and } Q_x(y) = 0\},$$

$$\mathrm{QNR} = \{(x, y) \mid x \in \mathbf{N}, y \in \mathbf{Z}_x^*, (y/x) = 1, \text{ and } Q_x(y) = 1\},$$

where x and y are presented in binary.

Clearly, by Facts 1 and 2, both QR and QNR are in the intersection of CO-NP and NP. However, no probabilistic polynomial time algorithm is known that accepts these languages, and thus they are not trivially zero-knowledge. In § 5 we show a perfectly zero-knowledge proof system for QR, and in § 6 we show a statistically zero-knowledge proof system for QNR. The following facts will be useful in the zero-knowledge proofs of §§ 5 and 6.

FACT 4. Let $x \in \mathbf{N}$. Then, for all y such that $Q_x(y) = 0$, the number of solutions $w \in \mathbf{Z}_x^*$ to $w^2 \equiv y \bmod x$ is the same (independent of y).

FACT 5. Let $x \in \mathbf{N}$, $y, z \in \mathbf{Z}_x^*$. Then we have the following:

(a) If $Q_x(y) = Q_x(z) = 0$, then $Q_x(yz) = 0$.

(b) If $Q_x(y) \neq Q_x(z)$, then $Q_x(yz) = 1$.

FACT 6. Given x, y, the Euclidean gcd algorithm allows us to compute in polynomial time whether or not $y \in \mathbf{Z}_x^*$.

5. Zero-knowledge proofs of quadratic residuosity. Recall that $\mathrm{QR} = \{(x, y) \mid y$ is a quadratic residue mod $x\}$, where x and y are presented in binary.

We will first *informally* describe our zero-knowledge interactive proof system for QR, and then describe it with more rigor. Say that A and B are given (x, y), $|x| = m$; then the following is done m times:

- A sends B a random quadratic residue mod x, u.
- B sends A a random bit, *bit*.
- If $bit = 0$ then A sends B a random square root of u mod x, w; if $bit = 1$ then A sends B a random square root of (uy) mod x, w.
- B checks that either [$bit = 0$ and $w^2 \bmod x = u$] or [$bit = 1$ and $w^2 \bmod x = (uy) \bmod x$].

More formally, we assume, for convenience, that A starts the protocol.

A's PROTOCOL ON INPUT $(x, y) \in$ QR.

FOR $i = 1$ to m

Use random bits to generate u_i, a random quadratic residue mod x.

SEND u_i to B

GET a string β_i from B; let bit_i = the first bit of β_i (or 0 if β_i is empty). If $bit_i = 0$, use random bits and generate w_i, a random square root of u_i mod x; if $bit_i = 1$, use random bits and generate w_i, a random square root of $(u_i y)$ mod x.

SEND w_i to B

GET a string from B (this will merely indicate that B wishes to continue the protocol).

END FOR

SEND "terminate" (just a string to finish off the protocol) to B.

B's PROTOCOL ON INPUT (x, y).

See if $x \geqq 1$ and $y \in Z_x^*$; if not, halt.

FOR $i = 1$ to m

GET u_i from A. See if $u_i \in Z_x^*$; if not, halt.

Generate a random bit_i.

SEND bit_i to A.

GET w_i from A. See if $w_i \in Z_x^*$ and either [$bit_i = 0$ and $w_i^2 \bmod x = u_i$] or [$bit_i = 1$ and $w_i^2 \bmod x = (u_i y) \bmod x$]; if not, halt.

SEND "okay" (just a string to continue the protocol) to A.

END FOR

GET any string from A, and halt accepting.

This is clearly an interactive protocol.

CLAIM 1. *The above* (A, B) *protocol is an interactive proof system for* QR.

Proof. Say that B is interacting with an arbitrary A'. Say that $x \geqq 1$, $y \in Z_x^*$, and y is *not* a quadratic residue mod x. For each u_i that B receives from A' it cannot be the case that *both* u_i and $(u_i y)$ have square roots mod x. Since A' does not see any bit_i in advance, there is at most a $\frac{1}{2}$ probability that B will "okay" the ith pass. Hence, the probability that B will say "convinced" is at most $1/2^m$.

We will now show that the protocol is zero-knowledge for QR.

THEOREM 1. *The above* (A, B) *protocol is a perfectly zero-knowledge proof system for* QR.

Proof. Let B' be an arbitrary polynomial time ITM that interacts with A. Let $(x, y) \in \text{QR}$ be the common input to the pair (A, B'), $|x| = m$, and let H be the extra input to B'. For convenience we consider the view $\text{View}_{A,B'}((x, y), H)$ to consist of the random variables

$$R, U_1, \text{BIT}_1, W_1, U_2, \text{BIT}_2, W_2, \cdots, U_m, \text{BIT}_m, W_m,$$

where R is the string of random bits generated by B', U_i takes on the value u_i, BIT_i takes on the ith message of B', etc.

One way to describe the distribution of the view is as follows: R is assigned a random bit string r (of the appropriate length). Say that $R, U_1, \text{BIT}_1, W_1, \cdots, U_i, \text{BIT}_i, W_i$ is the random variable V_i. Assume that for some i, $1 \leqq i < m$, V_i has been given the value v_i; we will describe the experiment for giving values to $U_{i+1}, \text{BIT}_{i+1}, W_{i+1}$.

EXPERIMENT

Choose for U_{i+1} a random quadratic residue mod x, u_{i+1}. If B' were B, we would choose BIT_{i+1} to be the $(i+1)$st bit of r. However, all we can say is that BIT_{i+1} is assigned the value $bit_{i+1} = f(x, y, H, v_i, u_{i+1})$, where f is some $\{0, 1\}$-valued function computable in deterministic, polynomial time. If $bit_{i+1} = 0$, then W_{i+1} gets the value w_{i+1}, a random square root of $u_{i+1} \bmod x$; if $bit_{i+1} = 1$, then W_{i+1} gets the value w_{i+1}, a random square root mod x of $(u_{i+1} y) \bmod x$.

Having characterized the view with the above experiment, we will now describe a probabilistic Turing machine M that, given $(x, y) \in \text{QR}$ and a string H, runs in expected polynomial time, and such that its output distribution $M((x, y), H)$ is *exactly* the same as V_m above; that is, $M((x, y), H)$ is the same as $\text{View}_{A,B'}((x, y), H)$. M begins by choosing r equals a random bit string (of the appropriate length). Assume that v_i has been chosen for some i, $1 \leqq i < m$; M outputs $u_{i+1}, bit_{i+1}, w_{i+1}$ according to the following program:

```
DO FOREVER
    bit_{i+1} := a random member of {0, 1}
    w_{i+1} := a random member of Z_x^*
    IF bit_{i+1} = 0 THEN
        u_{i+1} := w_{i+1}^2 mod x
    ELSE
        u_{i+1} := (w_{i+1}^2 y^{-1}) mod x
    IF bit_{i+1} = f(x, y, H, v_i, u_{i+1}) THEN
        OUTPUT u_{i+1}, bit_{i+1}, w_{i+1} and HALT
END DO
```

Two things about M's program must be clarified. First, the way M chooses a random member of Z_x^* is by choosing random m-bit strings until one is found that is in $\mathbf{Z}_x^*$; this halts in expected polynomial time. Secondly, by "y^{-1}" we mean that unique member of Z_x^* which when multiplied by y mod x yields 1.

We now show that M halts in expected time polynomial in m, with exactly the right output distribution.

Let $R', \{U_i', \mathrm{BIT}_i', W_i'\}$ be the random variables corresponding to the output of M, and let V_i' be defined similarly to V_i. Certainly R' has exactly the same distribution as R. Let $1 \leqq i < m$ and assume that V_i' has exactly the same distribution as V_i, and assume that M gives a value to V_i' in expected time polynomial in m. Say that both V_i and V_i' have been given the value v_i; we wish to show that the above piece of program code halts in expected time polynomial in m, with the same output distribution as the above experiment, given that $V_i = V_i' = v_i$.

Consider the body of the DO loop up to but not including the last test. If $bit_{i+1} = 0$ at this point, then since every quadratic residue in Z_x^* has the same number of square roots (mod x), u_{i+1} is equally likely to be any quadratic residue, and w_{i+1} will be a random square root of u_{i+1}; if $bit_{i+1} = 1$ at this point, then u_{i+1} will also be a random quadratic residue in this case (since y is a quadratic residue), and w_{i+1} will be a random square root of $(u_{1+1}y)$ mod x. Therefore, the body of the DO loop has the following *effect* (even though the following code may not be efficiently executable):

EQUIVALENT DO BODY
$u_{i+1} :=$ a random quadratic residue mod x.
$bit_{i+1} :=$ a random bit
IF $bit_{i+1} = f(x, y, H, v_i, u_{i+1})$ THEN
IF $bit_{i+1} = 0$ THEN $w_{i+1} :=$ a random square root mod x of u_{i+1} FI
IF $bit_{i+1} = 1$ THEN $w_{i+1} :=$ a random square root mod x of $(u_{i+1}y)$ FI
HALT and output $(u_{i+1}, \mathrm{bit}_{i+1}, w_{i+1})$
FI

It is clear that the equivalent body halts (and outputs) with probability $\frac{1}{2}$, and therefore that the actual DO loop halts in expected polynomial time. Since for each value of u_{i+1} the equivalent body is equally likely to halt, U_{i+1}' gets assigned (by the DO loop) a random quadratic residue. BIT_{i+1}' will be assigned $f(x, y, H, v_i, u_{i+1})$. Lastly, we can see from the equivalent body that in the case where the DO loop halts, W_{i+1}' gets assigned a random square root of u_{i+1} or of $(u_{i+1}y)$, depending on bit_{i+1}, as required.

6. Zero-knowledge proofs of quadratic nonresiduousity. We define $\mathrm{QNR} = \{(x, y) \mid y \in \mathbf{Z}_n^*, (y/x) = 1, Q_x(y) = 1\}$, where x and y are presented in binary.

Let (A, B) be an interactive protocol given as input (x, y) such that $|x| = m$.

The basic idea of the protocol is that B generates at random elements w of two types: $w \equiv r^2$ mod x (type 1) and $w \equiv r^2 y$ mod x (type 2), and sends these elements to A. If $(x, y) \in \mathrm{QNR}$ then A can tell of which type w is by computing whether w is a quadratic residue (type 1) or not (type 2). If (x, y) is not a member of QNR, w is always a quadratic residue mod x and A cannot guess its type better than guessing at random. Thus, A will not be able to tell the types of the w's and B will not be convinced that $(x, y) \in \mathrm{QNR}$.

This idea is sufficient as a proof system but not as a zero-knowledge proof system. The danger is that B may not have followed the protocol and generated elements w in a manner differently than specified in the protocol. We get over this difficulty by complicating the protocol to force B to convince A that indeed B knows whether w

is of type 1 or type 2. He does this by convincing A that he knows either a square root of w or a square root of $wy^{-1} \bmod x$, without giving A any information (in the information-theoretic sense) of which one he really knows.

Our original protocol, which appeared in [GMR], was more complex to prove than the one presented here. The simplified protocol presented here was suggested by Cohen [Co].

As was done for the QR language, we will first informally describe an interactive protocol, which we claim is a statistical zero-knowledge interactive proof system for QNR, and then describe it with more rigor.

The following (A, B) protocol on input (x, y) should be repeated m times.

- B picks at random $r \in \mathbf{Z}_x^*$ and $bit \in \{0, 1\}$. If $bit = 0$, B sets $w = r^2 \bmod x$; otherwise B sets $w = r^2 y \bmod x$. B sends w to A.
  For $1 \leqq j \leqq m$, B picks random $r_{j1}, r_{j2} \in \mathbf{Z}_x^*$ and a random $bit_j \in \{0, 1\}$. B sets $a_j = r_{j1}^2 \bmod x$, and $b_j = y r_{j2}^2 \bmod x$. If $bit_j = 1$, B sends A the ordered pair, $\text{pair}_j = (a_j, b_j)$; else if $bit_j = 0$, B sends to A $\text{pair}_j = (b_j, a_j)$.
- A sends B an m-long random bit vector $i = i_1 i_2 \cdots i_m$.
- B sends A the sequence $v = v_1, v_2, \cdots, v_m$; if $i_j = 0$ then $v_j = (r_{j1}, r_{j2})$; if $i_j = 1$ then $v_j = r r_{j1} \bmod x$ (a square root of $wa_j \bmod x$) if $bit = 0$, and $v_j = y r r_{j2} \bmod x$ (a square root of $wb_j \bmod x$) if $bit = 1$.
  (The intuition behind this step is as follows: if $i_j = 0$, then B is convincing A that pair_j was chosen correctly; if $i_j = 1$, then B is convincing that *if* pair_j was chosen correctly, then w was chosen correctly.)
- A verifies that the sequence v was properly constructed. If not, A sends *terminate* to B and halts. Otherwise, A sets $answer = 0$ if w is a quadratic residue mod x and 1 otherwise. A sends *answer* to B.
- B checks whether $answer = bit$. If so B continues the protocol, otherwise B rejects and halts.

After m repetitions of this protocol, if B did not reject thus far, B accepts and halts.

More formally, we proceed to describe first the protocol for B and then the protocol for A. The protocol consists of B going through its first stage, followed by A's first stage, followed by B's second stage, followed by A's second stage, etc., until either A or B chooses to terminate the protocol.

Denote by $v = \{v; v_j\}$ the result of extending sequence v with element v_j.

B's PROTOCOL ON INPUT (x, y).
Check that $x \geqq 1$ and that $y \in \mathbf{Z}_x^*$ and that $(y/x) = 1$.
Set $m = |x|$.
Repeat Stages 1-3 m times.
Stage 1.[1]
  use random bits to pick $r \in \mathbf{Z}_x^*$ and $bit \in \{0, 1\}$.
  IF $bit = 0$ set $w \equiv r^2 \bmod x$, else set $w \equiv r^2 y \bmod x$ FI
  FOR $j = 1, 2, \cdots, m$
    choose random $r_{j1}, r_{j2} \in \mathbf{Z}_x^*$ and random $bit_j \in \{0, 1\}$.
    set $a_j \equiv r_{j1}^2 \bmod x$ and $b_j \equiv r_{j2}^2 y \bmod x$.

[1] The careful reader may observe that picking $r \in Z_x^*$ "exactly at random" can be done in expected polynomial time, while our B must by definition run in a fixed polynomial number of steps. Fortunately, B can pick $r \in Z_x^*$ "almost at random" in a fixed polynomial time. This will have a negligible effect on the result and we omit any further details on this point.

IF $bit_j = 0$ set $\text{pair}_j = (a_j, b_j)$ else set $\text{pair}_j = (b_j, a_j)$ FI
END FOR
SEND $(w, \text{pair}_j \text{ for } j = 1, \cdots, m)$ to A.

Stage 2.
GET from A and m-long bit vector $i = i_1 \cdots i_m$, where $i_j \in \{0, 1\}$.
Initialize the sequence v to the empty sequence.
FOR $j = 1, \cdots, m$
IF $i_j = 0$ then set $v_j = (r_{j1}, r_{j2})$ and set $v = \{v; v_j\}$ FI
IF $i_j = 1$ do one of the following:
IF $bit = 0$ then set $v_j = rr_{j1} \bmod x = \sqrt{wa_j} \bmod x$ and set $v = \{v; v_j\}$
otherwise set $v_j = yrr_{j2} \bmod x = \sqrt{wb_j} \bmod x$ and set $v = \{v; v_j\}$ FI
FI
END FOR
SEND v to A.

Stage 3.
GET $answer \in \{0, 1\}$ from A.
IF $answer \neq bit$ then reject and halt
otherwise go to stage 1 FI.

After m iterations of Stages 1-3, if the protocol has not halted by now, accept and halt.

A's PROTOCOL ON INPUT $(x, y) \in$ QNR.

Stage 1.
GET $(w, \text{pair}_j \text{ for } j = 1, \cdots, m)$ from B.
Pick at random $i = i_1 \cdots i_m$ where $i_j \in \{0, 1\}$.
SEND i to B.

Stage 2.
GET sequence v from B
for every $j = 1, \cdots, m$ check that, if $i_j = 0$, then v_j is a pair (s, t) such that $(s^2 \bmod x, t^2 y \bmod x)$ equals pair_j, possibly with the elements interchanged; and if $i_j = 1$, then $(v_j^2) w^{-1} \bmod x$ is a member of pair_j. If not, SEND *terminate* to B and halt.
(Assume that the above checks have succeeded.)
If w is a quadratic residue mod x, set $answer = 0$ and if w is a quadratic nonresidue mod x, set $answer = 1$.
SEND *answer* to B.
Go to Stage 1.

We first prove that (A, B) is an interactive proof system for QNR.

CLAIM 2. (A, B) *is an interactive proof system for* QNR.

Proof. Clearly (A, B) is an interactive protocol. If $(x, y) \in$ QNR and A and B follow the specification of the protocol, then for every execution of Stages 1-2 by B, w is a quadratic nonresidue mod x if and only if $bit = 1$. Thus, in A's Stage 2, A can always decide whether w is a quadratic residue mod x or not and send *answer* to B such that $answer = bit$ and B will always accept.

Suppose that (x, y) not in QNR (i.e., y is a quadratic residue mod x) and that B is interacting with an arbitrary prover A', in the kth iteration of Stages 1-3. Then we claim that even an A' with infinite computation power cannot distinguish an interaction with B where $bit = 0$ from an interaction with B where $bit = 1$. This is argued as follows. At Stage 1, A' gets the list $(w, \text{pair}_j \text{ for } j = 1, \cdots, m)$, where w is a *random* quadratic residue, and where pair_j simply consists of a pair of *random* quadratic residues. This gives absolutely no information about the value of *bit*.

Now consider Stage 2, where $i_j = 0$. A gets (r_{j1}, r_{j2}). Note that r_{j1} is just a random square root of a_j and r_{j2} is a random square root of $b_j/y \bmod x$. So all that A sees is the result of randomly choosing (or not) to reorder pair_j, and then taking a random square root of the first element and a random square root of the result of dividing the second element by y (all mod x, of course). This gives no information about *bit*.

Now consider the case where $i_j = 1$. If $bit = 0$ then A gets $rr_{j1} \bmod x$, which is a random square root of $wa_j \bmod x$. If $bit = 1$ then A gets $yrr_{j2} \bmod x$, which is a random square root of $wb_j \bmod x$. Since pair_j is a random reordering of (a_j, b_j), v_j is equally likely to be a random square root of w times the first element of pair_j as it is to be a random square root of w times the second element of pair_j, no matter what *bit* is.

Thus, from the information that A' receives in Stages 1 and 2, the value of *bit* is as likely to be 0 as it is to be 1 and the chance that A' predicts *bit* correctly is no greater than $\frac{1}{2}$. In m iterations through Stages 1-3, the probability that A' computed *answer* such that *answer* = *bit* is at most $1/2^m$. □

Proving that the (A, B) proof system is statistically zero-knowledge for QNR is much more complex.

THEOREM 2. *The above protocol* (A, B) *is a statistically zero-knowledge proof system for* QNR.

Proof. Let B' be an arbitrary probabilistic polynomial time interactive Turing machine interacting with A. Let $(x, y) \in \text{QNR}$ be input to (A, B'), let $m = |x|$, and let H be the extra input to B'.

For convenience, consider the random variable $\text{View}_{A,B'}((x, y), H)$ (B''s view of an iteration of the protocol) to consist of the random variables:

$$\text{RAN}, \quad \text{and}$$

$$\{W^k, \{\text{PAIR}_j^k : 1 \leqq j \leqq m\}, \{I_j^k : 1 \leqq j \leqq m\}, V^k, \text{ANSWER}^k \mid 1 \leqq k \leqq m\}.$$

RAN is the string of random bits generated by B'; W^k takes on the value of w in the kth iteration of the protocol; PAIR_j^k takes on the value of pair_j in the kth iteration of the protocol; I_j^k takes on the value of i_j in the kth iteration of the protocol; $V^k = \{V_j^k\}$ takes on the value of the sequence v in the kth iteration of the protocol; and ANSWER^k takes on the value of *answer* in the kth iteration of the protocol.

For simplicity (notational and otherwise) we concentrate on showing that a single iteration of the protocol is zero-knowledge. Doing the general case implies carrying along the view of the protocol so far as was done in the proof of Theorem 1. Thus, from here on we drop all superscripts and work with the random variables: RAN, W, $\{\text{PAIR}_j\}$, $\{I_j\}$, $V = \{V_j\}$, and ANSWER.

Note that in a good execution of the protocol (namely, if B's protocol is followed), we expect that $W = r^2 \bmod x$ or $W = r^2 y \bmod x$, where r is a substring of RAN; and that $\text{PAIR}_j = (r_{j1}^2 \bmod x, r_{j2}^2 y \bmod x)$ or $(r_{j2}^2 y \bmod x, r_{j1}^2 \bmod x)$, where r_{j1} and r_{j2} are substrings of RAN; and that for all $1 \leqq j \leqq m$, if $I_j = 0$ then V_j will equal (r_{j1}, r_{j2}); and if $I_j = 1$ then $V_j = rr_{j1} \bmod x$ or $yrr_{j2} \bmod x$.

However, since B' may not follow the protocol, all we can say about these random variables is that RAN is a random binary string; W (and PAIR_j) are assigned values w (and pair_j) computed by B' on inputs x, y, H and RAN; I is a random binary string of length m; and V_j is a value computed by B' on inputs x, y, H, RAN, I.

We will now describe a probabilistic Turing machine M that, given $(x, y) \in \text{QNR}$ and H, runs in expected polynomial time, and whose output distribution is statistically indistinguishable from $\text{View}_{A,B'}((x, y), H)$.

M starts by outputting a random string *ran* of the appropriate length and running B' on inputs (x, y, H), and random tape *ran* on it. B' goes through Stage 1 outputting w, pair_j, for $1 \leqq j \leqq m$.
Next, M chooses $i_1, \cdots, i_m$ at random in $\{0, 1\}$, sets $i = i_1 \cdots i_m$, and writes i on B'"s communication tape, activating B'"*s* Stage 2.
B' goes into Stage 2, writing on its communication tapes a sequence $v = \{v_j\}$. M outputs $w\{\text{pair}_j\}$, i, v.
Next M does the checking that A does in Stage 2.
If the check fails M outputs "*terminate*" and halts.

Let us assume that the check succeeds. Think of x, y, H, *ran* as being fixed, so that w and $\{\text{pair}_j\}$ are also fixed. The fact that the check succeeds means that A sending i to B' causes B' to send a v to A, which causes A to send a one-bit *answer* to B' (rather than *terminate*); let us call any such i' *special.* M has just computed that i is special, and now wants to compute the value of *answer* that A would send. This value is 0 if w is a quadratic residue mod x, and 1 otherwise. Since B' may not have computed w the way B would have, it is not obvious how to compute the quadratic residuosity of w, i.e., *answer.*

It turns out that finding one other special string $i' \neq i$ will allow M to determine if w is a quadratic residue, as follows:

> Say that $i_j = 0$ and $i'_j = 1$ and i, i' are special. Let v, v' be the sequences sent by B' after receiving i or i' (respectively); these can be computed in polynomial time by running B'. Since $i_j = 0$, $v_j = (s, t)$, where $(s^2 \bmod x, t^2 y \bmod x)$ equals pair_j, possibly with the elements reversed. Since $i'_j = 1$, $(v_j)^2 w^{-1} \bmod x \in \text{pair}_j$. If $(v_j)^2 w^{-1} \bmod x = s^2 \bmod x$, then w is a quadratic residue mod x; if $(v_j)^2 w^{-1} \bmod x = t^2 y \bmod x$ then w is a quadratic nonresidue mod x.

It therefore remains to find a special $i' \neq i \bmod x$. M uses the following algorithm.

ALGORITHM TO FIND A SPECIAL $i' \neq i$.

> Test 2^m random i' of length m (with replacement), halting when either a special $i' \neq i$ is found, or when 2^m strings have been tried. If no special $i' \neq i$ has been found, then test *all* m-bit strings (in order), looking for a special $i' \neq i$.

If a special $i' \neq i$ is found, then M calculates *answer* as explained above and outputs *answer.* If no such i' exists, then M outputs "?"; note that this will happen when i is the *only* special string.

In order to show that M operates in expected polynomial time, it is sufficient to show that M operates in expected polynomial time for each fixed value of (x, y, H, ran). Say that x, y, H, *ran* are fixed, and so w, pair are also fixed. Let k be the number of strings i that are special. If $k = 0$, then the ALGORITHM TO FIND A SPECIAL i' will not be invoked, and the running time is clearly polynomial in m. If $k = 1$, then with probability $(1/2^m)M$ will choose a special i; in that case the ALGORITHM will run for time $2^m m^c$ (for some c), so the expected running time is $(1/2^m)2^m m^c +$ a polynomial in m.

Assume that $k > 1$. M will choose a special i with probability $k/2^m$. To calculate an upper bound on the expected running time of the ALGORITHM, imagine that it was changed so that it tested random i', including i and with replacement, halting if and when a special $i' \neq i$ is found; the expected running time would be at least half that of the ALGORITHM. In effect, a coin is being tossed until "heads" comes up, where the probability of "heads" is exactly $(k-1)/2^m$. It is well known that the expected number of coin tosses is exactly $2^m/(k-1)$. Hence the expected time for the

ALGORITHM is $\leqq(2^m/(k-1))m^c$ (for some c). The total expected time is $\leqq(k/2^m)$-$(2^m/(k-1))m^c$ + a polynomial in m, which is polynomial in m.

Recall that $M((x, y), H)$ is the random variable denoting the distribution of M's output given x, y, H. It remains to show that M is statistically close to $\text{View}_{A,B'}$. Fix x, y, H. If *ran* is such that the number of special strings is not exactly 1, then any output string α beginning with *ran* is taken on by $\text{View}_{A,B'}((x, y), H)$ with exactly the same probability as by $M((x, y), H)$. Let S be the set of $\alpha = ran, w, \{\text{pair}_j\}, i, v, answer$, where i is the *unique* special string determined by *ran*. The probability that $\text{View}_{A,B'}((x, y), H)$ takes on a value in S is $\leqq 1/2^m$, since for each *ran* there is at most one i, which will be the unique special string. Similarly, the probability that $M((x, y), H)$ takes on a value in S is $\leqq 1/2^m$. Thus,

$$\sum_{\alpha} |\text{prob}\,(M((x, y), H) = \alpha) - \text{prob}\,(\text{View}_{A,B'}((x, y), H) = \alpha)|$$

$$= \sum_{\alpha \notin S} |\text{prob}\,(M((x, y), H) = \alpha) - \text{prob}\,(\text{View}_{A,B'}((x, y), H) = \alpha)|$$

$$+ \sum_{\alpha \in S} |\text{prob}\,(M((x, y), H) = \alpha) - \text{prob}\,(\text{View}_{A,B'}((x, y), H) = \alpha)|$$

$$\leqq 0 + \frac{1}{2^m} + \frac{1}{2^m} = \frac{2}{2^m}.$$

And for m iterations of the protocol, the difference is

$$\sum_{\alpha} |\text{prob}\,(M((x, y), H) = \alpha) - \text{prob}\,(\text{View}_{A,B'}((x, y), H) = \alpha)| \leqq \frac{2m}{2^m}.$$

This completes our proof. □

Remarks. In fact, the above protocol can be shown to be perfect zero-knowledge. We just have to change M so that when it discovers that i is the unique special string it *factors* x in time roughly 2^m, and then determines if w is a quadratic residue mod x in polynomial time. This does not change the expected running time by more than a polynomial factor, since when M decides to do this extra work, it has already spent time 2^m.

7. Related work.

7.1. Work related to interactive proof systems. In studying his Arthur-Merlin games, Babai [Ba] has focused on the number of *rounds*, i.e., the number of times the prover and the verifier alternate in sending messages. Babai denotes the set of all languages accepted by i *rounds* in an Arthur-Merlin proof system by AM[i], and AM[$f(n)$] denotes the set of languages accepted by an Arthur-Merlin proof system with $f(n)$ rounds. Here f is a nondecreasing function from natural numbers to natural numbers, and n the length of the input.

The elegant simplicity of Babai's definition allowed him to show that for every constant k, AM[k] collapses to AM[2]. This in turn is a subset of both Π_2^p and nonuniform NP.

We define IP[$f(n)$] as the class of languages having an interactive proof system with $f(n)$ rounds.

Goldwasser and Sipser [GS] show that, for all f, AM[$f(n)$] = IP[$f(n)$].

On the other hand, Aiello, Goldwasser, and Hastad [AGH] have shown that for any two nonconstant functions $g(n)$ and $f(n)$ such that $g(n) = o(f(n))$, there exists an oracle X such that (if we modify the definitions so that one is computing using the oracle X) IP[$g(n)$] is strictly contained in IP[$f(n)$]. This result is tight as Babai and Moran [BM] have shown that for all constants $c > 0$, IP[$f(n)$] = IP[$cf(n)$]. Namely, IP[$O(f(n))$] is well defined.

An interesting question is the following. Where does IP stand with respect to the polynomial time heirarchy? Boppana, Hastad, and Zachos [BHZ] have shown that if CO-NP has a constant-round interactive proof system, then the polynomial time hierarchy collapses. Thus, from the results of [GMW], [GS], and [BHZ], it follows that graph isomorphism is not NP-complete unless the polynomial time hierarchy collapses.

Other works related to the study of randomized and nondeterministic complexity classes appear in [P] and [ZF]. In Papadimitriou's *Games Against Nature*, the verifier is also a probabilistic polynomial time machine that flips coins and presents them to a prover capable of optimal moves. This is different from our model in that L is said to be accepted by a game against nature if $x \in L$ implies that the probability of the prover to win the game is greater than a $\frac{1}{2}$ rather than bounded away from a $\frac{1}{2}$.

Zachos and Furer [ZF], in a work investigating the robustness of probabilistic complexity classes, introduce a framework of probabilistic existential and universal quantifiers and prove several combinatorial lemmas about them. The AM and thus IP complexity classes can be formulated in terms of these special quantifiers.

7.2. Work related to knowledge complexity. Prior to our work, the theory of knowledge had received much attention in a model-theoretic framework (see [FHV] and [HM] for discussion). There are several essential differences between this framework and ours. In the latter, knowledge is defined with respect to a specific computational model with specific computational resources. In the former framework, there are no limitations on the computational power of the participants, i.e., they "know" all logical consequences of the information they possess. (For discussion of this aspect see, *Belief, Awareness, and Limited Reasoning* [FH].) As for another difference, in our model knowledge is defined with respect to an available public input and is gained by computing on this input. In their model-theoretic framework knowledge is gained by being told (or witnessing) that a certain event is true (e.g., the outcome of a coin flip is heads), rather than by computing.

Galil, Haber, and Yung [GHY] proposed the following extension of the concept of a zero-knowledge interactive proof systems. A language L is said to have a result-indistinguishable zero-knowledge proof system if there exists an interactive protocol (A, B) such that for every string $x \in \{0, 1\}^*$, A can convince B that $x \in L$ or x is not in L (whichever is the case) with high probability, such that no passive observer C can get any information of which is the case. They give a result-indistinguishable proof system for QR.

As previously mentioned, Goldreich, Micali, and Wigderson [GMW] have shown, subject to the existence of secure encryption schemes, that all languages in NP have computationally zero-knowledge proof systems. Subsequently, related notions of proof systems and zero knowledge were given by Brassard and Crepeau [BC] and Chaum [Ch]. They found that for any language L in NP, there is an interactive protocol that

(1) is zero-knowledge, and
(2) proves membership in L correctly (i.e., with probability approaching 1) only if factoring is computationally difficult and the prover is polynomial time.

Let us explicitly contrast their protocols with the ones in [GMW]. The latter ones

(1) correctly prove membership in L, and
(2) are zero-knowledge only if secure encryption schemes exist (which is true if factoring is difficult).

Finally, let us mention the recent result of Fortnow.

THEOREM [Fo]. *Assume a language L has an interactive proof system (A, B) that is statistically zero-knowledge with respect to B. Then L's complement has a constant-round interactive proof system.*

As a corollary, if SAT had statistically zero-knowledge proof systems, the polynomial time hierarchy would collapse.

Note that the hypothesis of Fortnow's theorem is much weaker than saying that (A, B) is a statistically zero-knowledge proof system on L, which would mean that, for all verifiers B', A is zero-knowledge on L for B'. Usually, it is defeating this latter quantifier "for all" that makes it hard to find a perfect or statistically zero-knowledge proof system. Thus Fortnow's result has the potential to be widely applicable.

Acknowledgments. We thank Mike Sipser, Steve Cook, and Mike Fischer who helped us focus on this research from its beginning. Without their enthusiastic encouragement we might not have completed this work.

Oded Goldreich and Ron Rivest, as usual, have been generous with comments and ideas. Thanks also go to Leonid Levin, Zvi Galil, and Dan Simon for having helped us in various ways.

Special thanks to Josh Cohen for simplifying our original zero-knowledge protocol for proving quadratic nonresiduosity. Thanks are also due to Manuel Blum for sharing with us so many beautiful ideas about cryptographic protocols.

Finally, we thank the anonymous referees, whose comments greatly improved this paper.

REFERENCES

[AGH] B. AIELLO, S. GOLDWASSER, AND J. HASTAD, *On the power of interaction*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 368–379.

[AH] B. AIELLO AND J. HASTAD, *Perfect zero-knowledge languages can be recognized in two rounds*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 439–448.

[BGGHKRM] M. BEN-OR, O. GOLDREICH, S. GOLDWASSER, J. HASTAD, J. KILIAN, P. ROGAWAY, AND S. MICALI, *Everything provable is provable in zero-knowledge*, in Proc. Crypto88, to appear.

[Bl] M. BLUM, *Coin flipping by telephone*, IEEE COMPCON 1982, pp. 133–137.

[BHZ] R. BOPPANA, J. HASTAD, AND S. ZACHOS, *Does co-NP have short interactive proofs?*, Inform. Process. Lett., 25 (1987) pp. 127–132.

[Ba] L. BABAI, *Trading group theory for randomness*, in Proc. 17th ACM Annual Symposium on Theory of Computation, 1975, pp. 421–429.

[BM] L. BABAI AND S. MORAN, *Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. Sci. Systems; a previous version was entitled *Trading group theory for randomness*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 421–429.

[BS] L. BABAI AND E. SZEMEREDI, *On the complexity of matrix group problems*, in Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, pp. 229–240.

[BC] G. BRASSARD AND C. CREPAU, *Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for* SAT *and beyond*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, October 1986.

[Ch] D. CHAUM, *Demonstrating the public predicate can be satisfied without revealing any information how*, in Proc. Crypto86.

[Co] J. COHEN (Benaloh), *Cryptographic capsules*, in Proc. Crypto86.

[CKS] A. CHANDRA, D. KOZEN, AND L. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[C] S. COOK, *The complexity of theorem-proving procedures*, in Proc. 3rd Annual ACM Symposium of Theory of Computation, 1971, pp. 151–158.

[CR] S. COOK AND R. RECKHOW, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979).

[F] P. FELDMAN, private communication.

[FMRW] M. FISCHER, S. MICALI, C. RACKOFF, AND D. WITENBERG, *A secure protocol for the oblivious transfer*, unpublished manuscript, 1986.

[FFS] U. FEIGE, A. FIAT, AND A. SHAMIR, *Zero knowledge proofs of identity*, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 210–217.

[Fo] L. FORTNOW, *The complexity of perfect zero-knowledge*, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 204–209.

[FH] R. FAGIN AND J. HALPERN, *Belief, awareness, and limited reasoning*, in Proc. 9th International Joint Conference on Artificial Intelligence, 1985, pp. 491–501.

[FHV] R. FAGIN, J. HALPERN, AND M. VARDI, *A model theoretic analysis of knowledge*, in Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, 1984, pp. 268–278.

[GM] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. Science and Systems, 28 (1984), pp. 270–299.

[GM1] ———, *Proofs with untrusted oracles*, unpublished manuscript (submitted to STOC, 1984). Revised version: *The information content of proof systems*, unpublished manuscript (submitted to STOC, 1984).

[GMS] O. GOLDREICH, Y. MANSOUR, AND M. SIPSER, *Interactive proof systems: Provers that never fail and random selection*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 449–460.

[GMR] S. GOLDWASSER, S. MICALI, AND C. RACKOFF, *The knowledge complexity of interactive proof systems*, in Proc. 27th Annual Symposium on Foundations of Computer Science, 1985, pp. 291–304. Earlier version: *Knowledge complexity*, unpublished manuscript, (submitted to FOCS, 1984).

[GS] S. GOLDWASSER AND M. SIPSER, *Private coins versus public coins in interactive proof-systems*, in Proc. 18th Annual Symposium on Theory of Computing, 1986, pp. 59–68.

[GHY] Z. GALIL, S. HABER, AND M. YUNG, *A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 360–371.

[GMW] O. GOLDREICH, S. MICALI, AND A. WIGDERSON, *Proofs that yield nothing but their validity and a methodology of cryptographic protocol design*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 174–187.

[GMW2] ———, *How to play any mental game*, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 218–229.

[HM] J. HALPERN AND Y. MOSES, *Knowledge and common knowledge in a distributed environment*, in Proc. 3rd Principles of Distributed Computing Conference, 1984, pp. 50–61.

[LMR] M. LUBY, S. MICALI, AND C. RACKOFF, *How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin*, in Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 11–22.

[O] Y. OREN, *On the cunning power of cheating verifiers: some observations of zero-knowledge proofs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 462–471.

[P] C. PAPADIMITRIOU, *Games against nature*, in Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 446–450.

[TW] M. TOMPA AND H. WOLL, *Random self reducibility and zero knowledge interactive proofs of possession of information*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 472–482.

[ZF] S. ZACHOS AND M. FURER, *Probabilistic quantifiers vs. distrustful adversaries*, in Proc. Structure of Complexity Classes Conference, 1986.

[Y] A. YAO, *Theory and application of trapdoor functions*, in Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, November 1982, pp. 80–91.