SIAM J. COMPUT.
Vol. 18, No. 4, pp. 711-739, August 1989

006

MINIMUM-KNOWLEDGE INTERACTIVE PROOFS FOR DECISION PROBLEMS*

ZVI GALIL†¶, STUART HABER‡¶, AND MOTI YUNG§¶[1]

Abstract. Interactive communication of knowledge from the point of view of resource-bounded computational complexity is studied. Extending the work of Goldwasser, Micali, and Rackoff [*Proc.* 17*th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 291-304; *SIAM J. Comput.*, 18 (1989), pp. 186-208], the authors define a protocol transferring the result of any fixed computation to be *minimum-knowledge* if it communicates no additional knowledge to the recipient besides the intended computational result. It is proved that such protocols may be combined in a natural way so as to build more complex protocols.

A protocol is introduced for two parties, a prover and a verifier, with the following properties:

(1) Following the protocol, the prover gives to the verifier a proof of the value, 0 or 1, of a particular Boolean predicate, which is (assumed to be) hard for the verifier to compute. Such a *deciding* "interactive proof-system" extends the interactive proof-systems of [*op. cit.*], which are used only to confirm that a certain predicate has value 1.

(2) The protocol is minimum-knowledge.

(3) The protocol is *result-indistinguishable*: an eavesdropper, overhearing an execution of the protocol, does not learn the value of the predicate that is proved.

The value of the predicate is a cryptographically secure bit, shared by the two parties to the protocol. This security is achieved without the use of encryption functions, all messages being sent in the clear. These properties enable one to define a cryptosystem in which each user receives exactly the knowledge he is supposed to receive, and nothing more.

Key words. zero knowledge, interactive proof systems, minimum knowledge cryptographic protocols, cryptography, security, probabilistic computations, factoring, quadratic residuosity, number theory

AMS(MOS) subject classifications. 11Z50, 68Q99, 94A60

1. Introduction.

Transfer and exchange of knowledge is the basic task of any communication system. Recently, much attention has been given to the process of knowledge exchange in the context of distributed systems and cryptosystems. In particular, several authors have concentrated on problems associated with the interactive communication of proofs [17], [1], [24].

In [17] Goldwasser, Micali, and Rackoff developed a computational-complexity approach to the theory of knowledge: a message is said to convey knowledge if it contains information that is the result of a computation that is intractable for the receiver. They introduce the notion of an *interactive proof-system* for a language L. This is a protocol for two interacting probabilistic Turing machines, whereby one of them, the *prover*, proves to the other, the *verifier*, that an input string x is in fact (with

* Received by the editors October 17, 1986; accepted for publication (in revised form) October 10, 1988. A preliminary version of the paper appeared as *A Private Interactive Test of a Boolean Predicate and Minimum Knowledge Public-Key Cryptosystems*, Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 360-371. Most of this work was done while all three authors were at Columbia University, New York, New York 10027.

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

† Department of Computer Science, Columbia University, New York, New York 10027, and Department of Computer Science, Tel Aviv University, Tel Aviv, Israel.

‡ Bell Communications Research, Morristown, New Jersey 07960.

§ IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York 10598.

¶ The work of these authors was supported in part by National Science Foundation grants MCS-8303139 and DCR-851173.

[1] The work of this author was supported in part by an IBM graduate fellowship.

very high probability) an element of L. The verifier is limited to tractable (i.e., probabilistic polynomial-time) computations. We do not limit the computational power of the prover; in the cryptographic context, the prover may possess some secret information—for example, the factorization of a certain integer N. (This is analogous to the following model of a "proof-system" for a language L in NP: given an instance $x \in L$, an NP *prover* computes a string y and sends it to a deterministic polynomial-time *verifier*, which uses y to check that indeed $x \in L$.)

Goldwasser, Micali, and Rackoff called an interactive proof-system for L *zero-knowledge* if it releases no additional knowledge—that is, nothing more than the one bit of knowledge given by the assertion that $x \in L$ [17]. Extending their definition, we consider all two-party protocols for the purpose of transferring from one party to the other the result of a specified computation—$y = f(x)$, say—depending on the input x, and call any such protocol *minimum-knowledge* if it releases nothing more than the assertion that $y = f(x)$. Naturally, such interactive protocols are of particular interest in a cryptographic setting where distrustful users with unequal computing power communicate with each other.

After giving our definition of minimum-knowledge protocols, we prove that the concatenation of two minimum-knowledge protocols is minimum-knowledge. This suggests the importance of the minimum-knowledge property for the modular design of complex protocols. In fact, it is by serially composing several minimum-knowledge subprotocols that we formulate the more complex minimum-knowledge protocol that we introduce in this paper.

In this paper we extend the ability of interactive proof-system protocols from *confirming* that a given string x is in a language L to *deciding* whether $x \in L$ or $x \notin L$. That is, we give the first (nontrivial) example of a language L so that both L and its complement have minimum-knowledge interactive proof-systems for confirming membership, where both the proof of membership in L and the proof of nonmembership in L are by means of the *same* protocol, which releases no more knowledge than the value of the membership bit $(x \in L)$.

Furthermore, by following the protocol, the *prover* demonstrates to the *verifier* either that $x \in L$ or that $x \notin L$ in such a way that the two cases are indistinguishable to an eavesdropping third party that is limited to feasible computations. In fact, the protocol releases *no knowledge at all* to such an eavesdropper. As usual, we assume that the eavesdropper knows both the prover's and the verifier's algorithms, and we allow him access to all messages passed during an execution of the protocol. In spite of the fact that our protocol makes no use of encryption functions, the eavesdropper receives *no knowledge* about whether he has just witnessed an interactive proof of the assertion that $x \in L$ or of the assertion that $x \notin L$. We call this property of our protocol *result-indistinguishability.*

The proof that our protocol is minimum-knowledge with respect to the verifier and result-indistinguishable with respect to the eavesdropper relies on no unproved assumptions about the complexity of a number-theoretic problem.

The work of [17], [1], [24] concentrates on the knowledge transmitted by a prover to an active verifier. Introducing a third party to the scenario, we analyze the knowledge gained both by an active verifier and by a passive eavesdropper.

If membership or nonmembership in L is an intractable computation, then a result-indistinguishable minimum-knowledge proof-system for L can be used as a tool in building a cryptographic system. After an execution of our protocol, the string x can serve as a cryptographically secure encoding—shared only by the prover and the verifier—of the membership-bit $(x \in L)$. The use of x as an encoding of the membership-bit exemplifies what we may call "minimum-knowledge cryptography": it is a probabilistic encryption with the property that neither its specification (i.e., the interactive proof of the value encoded by x) nor its further use in communication can release any compromising knowledge, either to the verifier or to an eavesdropper. The minimum-knowledge property ensures that each party receives exactly the knowledge he is supposed to receive and nothing more. A cryptosystem based on such a minimum-knowledge protocol has the strongest security against passive attack that we could hope to prove; in particular, it is secure against both chosen-message and chosen-ciphertext attack.

The predicate that our protocol tests is that of being a quadratic residue or nonresidue modulo N for a certain number N (whose factorization may be the prover's secret information). We note that the language for which we show membership and nonmembership is in NP $\cap$ co-NP. A conventional membership proof for these languages releases the factorization of N, while in the interactive proof-system presented below no extra knowledge (about the factorization or about anything else) is given either to the verifier or to an eavesdropper.

An important motivation in our work on this protocol comes from our desire to guarantee the security of cryptographic keys, especially in situations where the generation of new keys is very costly or is otherwise limited by the context. If the integer N is the prover's public key in a public-key cryptosystem, then N is not compromised by polynomially many executions of our protocol; a polynomially bounded opponent knows no more after witnessing or participating in these executions than he knew before the key was used at all.

2. Preliminaries.

2.1. Interactive Turing machines.

We specify the model for which we describe our protocol; this is an extension of the model used in [17]. Two probabilistic Turing machines A and B form an *interactive pair of Turing machines* if they share a read-only *input tape* and a pair of *communication tapes*; one of the communication tapes is exclusive-write for A, while the other is exclusive-write for B. (The writing heads are unidirectional; once a symbol has been written on a communication tape, it cannot be erased.) We model each machine's probabilistic nature by providing it with a read-only *random tape* with a unidirectional read-head; the machine "flips a coin" by reading the next bit from its random tape. The two machines take turns being *active.* While it is active, a machine can read the communication tapes, perform computation using its own work tape and consulting its random tape, and send a message to the other machine by writing the message on its exclusive-write communication tape. In addition, B has a private *output tape*; whatever is written on this tape when A and B halt is the *result* of their computation.

In order to model the fact that the system is not memory-less, we also assume that each machine has a *history tape*, with a unidirectional write-head, on which the following records are automatically written:

- When the machine flips a coin, the bit it reads from its random tape is recorded on its history tape.
- At the beginning of each active turn, when the machine reads a new message from the other machine's exclusive-write communication tape, it records this message on its history tape.
- At the end of each active turn, when the machine writes a message to the other machine on its own exclusive-write communication tape, it records this message on its history tape.

- The result written on B's output tape is also recorded on B's history tape.

These records are written on the history tape sequentially in order according to the machine's computation; for example, when the machine flips a coin several times while computing its next message, these random bits are recorded on the history tape immediately before the message. The input tape and communication tapes are public, or shared by the two machines; each machine's random tape, history tape, and work tape are private, as is B's output tape. This is not the only way to model the situation we would like to describe, and some of the records written on the history tape are redundant, but without loss of generality we may assume this mode of operation.

When A and B begin their computation, an infinite bit-string is written on each of their random tapes. The choice of these two bit-strings, independently and uniformly at random from the set of all infinite strings, defines a probability measure on the set of possible computation histories of (A, B) that begin in any particular configuration.

For any strings x, h we say that the interactive pair of Turing machines (A, B) begins its computation with input x and B's *initial history* h if in their initial configuration x is written on the common input tape and h is the written portion of B's history tape. (Throughout this paper, we are not concerned with the contents of A's history tape.) We use $(\mathrm{A}, \mathrm{B})[x, h]$ to denote the set of computations that begin in this configuration. In each of the protocols that we present in this paper, B never consults its history tape. However, in discussing the properties of these protocols, we must be concerned with an arbitrary Turing machine that may take the role of B in an interaction with A, and that may make use of its history tape.

In what follows, B is limited to expected running time that is polynomial in the length of the common input x, while we make no limiting assumption about A's computational resources. (For cryptographic applications, A is also limited to feasible computation but possesses some trapdoor information.) Their messages to each other are in cleartext, though these messages may depend on their private coin flips, which remain hidden. We assume that both the length of B's initial history, as well as the total length of the messages written on the two communication tapes, are polynomial in $|x|$. For any input string x, we introduce the notation $H_x = \{h \mid h \in \{0, 1\}^*, |h| = O(|x|^{O(1)})\}$ for the set of associated initial histories that we allow.

Our scenario also includes a third probabilistic Turing machine, C, limited to expected polynomial-time computation, that can read the input and communication tapes of A and B and knows their algorithms. A is the *prover*, B is the *verifier*, and C is the *eavesdropper.*

2.2. Ensembles of strings. In order to speak precisely of the knowledge transmitted by communicated messages, we need the following definitions [17], [27], [6]. Let $I \subseteq \{0, 1\}^*$ be an infinite set of strings, and for each $x \in I$, let $\pi[x]$ be a probability distribution on a set of bit-strings. We call $\Pi = \{\pi[x] \mid x \in I\}$ an *ensemble* of strings (usually suppressing any mention of I).

For example, if M is a probabilistic Turing machine, then any input string x defines a probability distribution, according to the coin-tosses (i.e., the random tape) of M's computation, on the set $\mathrm{M}[x]$ of possible outputs of M on input x. Thus, for any I, $\{\mathrm{M}[x] \mid x \in I\}$ is an ensemble.

As a second example, suppose that (A, B) is an interactive pair of Turing machines. For any strings x, h, let $VIEW_{\mathrm{B}}\{(\mathrm{A}, \mathrm{B})[x, h]\}$ denote the set of private "histories" that may be written on B's history tape during a computation that begins with input x and B's initial history h; each of these is B's private view of the protocol execution. This set has a natural probability distribution according to the random tapes of A and B. Thus, for any set I, $\{VIEW_{\rm B}\{(\mathrm{A}, \mathrm{B})[x, h]\} | x \in I, h \in H_x\}$ is an ensemble of strings.

As another example, for any string x let $COM\{(\mathrm{A}, \mathrm{B})[x]\}$ denote the set of possible ordered sequences of messages written on the communication tapes of A and B during a computation that begins with input x. Each of these is the public view, and in particular that of the eavesdropper C, of a protocol execution of A and B. This set also has a natural probability distribution. (We assume that the specified computations do not make use of previous private or public history.) Thus, for any set I, $\{COM\{(\mathrm{A}, \mathrm{B})[x]\} | x \in I\}$ is an ensemble of strings that we call the *communications ensemble* produced by the interactive system (A, B).

A *distinguisher* is a family $D = \{D_x | x \in I\}$ of circuits with a single Boolean output; we assume that there is a constant c so that circuit D_x has $|x|^c$ input gates and one output gate. D is *polynomial-size* if there is a constant d so that D_x has at most $|x|^d$ nodes. Suppose that $\Pi = \{\pi[x] | x \in I\}$ and $\Pi' = \{\pi'[x] | x \in I\}$ are ensembles of strings, and that D is a distinguisher (all with respect to the same constant c). Let $p_D(\pi[x])$ be the probability that D_x outputs a 1 when it is given as input a single sample string of length $|x|^c$, randomly selected according to probability distribution $\pi(x)$; and let $p_D(\pi'[x])$, depending on the distribution $\pi'[x]$, be defined similarly. We call the two ensembles (computationally) *indistinguishable* if for any polynomial-size distinguisher D, for all n and sufficiently long x,

$$|p_D(\pi[x]) - p_D(\pi'[x])| < |x|^{-n}.$$

This condition holds, of course, if the two ensembles are exactly identical. In this case, for *any* distinguisher D the difference $|p_D(\pi[x]) - p_D(\pi'[x])|$ is equal to zero.

Let π and π' be two probability distributions on strings, and suppose that the number δ satisfies $0 \leqq \delta \leqq 1$. We say that π *approximates* π' *with error probability* δ if

$$\sum_s |\text{prob}\,(\pi[x] = s) - \text{prob}\,(\pi'[x] = s)| \leqq \delta$$

(where the sum is taken over all strings s in $\{0, 1\}^*$). This implies that the difference $|p_D(\pi[x]) - p_D(\pi'[x])| \leqq \delta$ for any distinguisher D, even if the definition of "distinguisher" is relaxed to allow as inputs to D_x a set of many samples randomly chosen either according to $\pi[x]$ or according to $\pi'[x]$.

2.3. Interactive proof-systems and transfer protocols. This paper is mainly devoted to a special sort of two-party protocol, that of interactively proving or disproving membership in a language L. A protocol that achieves this is called an *interactive proof-system* for L [17]. The prover A and the verifier B share a common input x, the string whose membership is in question. We assume that x belongs to a fixed set I, $I \supseteq L$, of input strings for (A, B). Depending on $k = |x|$, the length of the (binary) representation of the input string, we allow an error probability $\delta(k)$ that vanishes with increasing k. (In fact, all of the examples in this paper satisfy the stronger requirement of an error probability that is exponentially vanishing in k.)

Extending the definition of [17], we distinguish between a *confirming* proof-system for L, whose purpose is that the verifier confirm membership in L for the input string, and a *deciding* proof-system for L, whose purpose is that the verifier decide whether or not the input string is in L. At the end of a confirming protocol, the verifier may either *accept* the proof that $x \in L$, or *reject* the proof; at the end of a deciding protocol, the verifier may either *accept* a proof that $x \in L$, or *accept* a proof that $x \notin L$, or *reject* the proof. The execution ends normally when all of B's messages appear as if B is following the protocol; if this is so, then A ends the execution in a *success* state. A may halt the execution of the protocol if it detects that B is not following the protocol, ending the execution in a *failure* state.

For any input string x, let $k = |x|$. We say that $(\mathrm{A}, \mathrm{B})_k$ is a *confirming interactive proof-system* for L with inputs I and error probability $\delta(k)$ if:

(1) For any $x \in L$ given as input to (A, B), B accepts the proof with probability at least $1 - \delta(k)$.

(2) For any interactive Turing machine A*, and for any $x \in I - L$ given as input to (A*, B), B accepts the proof with probability at most $\delta(k)$.

We say that (A, B) is a *deciding interactive proof-system* for L with inputs I and error probability $\delta(k)$ if:

(1) For any $x \in I$ given as input to (A, B), B accepts the proof, halting with the correct value of the predicate $(x \in L)$ on its output tape, with probability at least $1 - \delta(k)$.

(2) For any interactive Turing machine A*, and for any $x \in I$ given as input to (A*, B), B accepts a proof of the incorrect value of the predicate $(x \in L)$ with probability at most $\delta(k)$.

As part of the definition, we require that these conditions should hold independently of the choice of the initial-history string (of length polynomial in k) that may be written on B's history tape at the beginning of the computation.

In the first definition, we require that (with high probability) B correctly accept the proof for strings $x \in L$, and that no cheating adversary, no matter how powerful, can convince B incorrectly to accept the proof for strings $x \notin L$ (except with vanishingly small probability). In the second definition, we require that (with high probability), given *any* input string $x \in I$, B correctly decide whether $x \in L$ or $x \notin L$, and that no adversary can convince B to accept an incorrect proof (except with vanishingly small probability). The probability is taken over all sequences of coin-tosses (i.e., over all possible random-tape bit-strings) used by the probabilistic computations of the two Turing machines.

The two definitions above describe correctness for protocols that transfer to B the computed value of a Boolean predicate that supplies one bit of "knowledge" about the input string. We can also study a more general sort of *transfer protocol* whose purpose is to transfer the result $F(x)$ of any specified computation depending on the input string x. For example, a deciding interactive proof-system for the language L is a transfer protocol for the function $F(x)$ taking the value 1 or 0 according to whether or not $x \in L$. Because the interacting machines are probabilistic, the intended result may take values in a probability distribution whose value $F(x, r)$ depends on x as well as on a random input string r. As in the case of an interactive proof-system, B may either *accept* or *reject* an execution of an interaction with another Turing machine. We say that a given protocol (A, B) is *correct* for a specified probability distribution of outputs if B's computed result, when it interacts with A, has the intended distribution (with very high probability), and no machine A*, no matter how powerful, can bias the distribution of B's outputs (except with vanishingly small probability).

In order to define "correctness" more precisely, we observe that the computations of any interactive pair of Turing machines (A, B) determine a partial function $f_{\mathrm{A,B}}$ as follows. Given strings x, r_{A}, and r_{B}, we define $f_{\mathrm{A,B}}(x, r_{\mathrm{A}}, r_{\mathrm{B}})$ to be the result written on B's output tape at the end of an accepting computation of (A, B) that begins with input x, when their random-tape strings begin with r_{A} and r_{B} (respectively); this value is well-defined, as long as r_{A} and r_{B} are sufficiently long. Notice that the choice of r_{A} and r_{B} defines a probability distribution $f_{\mathrm{A,B}}(x, \cdot, \cdot)$.

We say that (A, B) is a *correct transfer protocol* for the probability distribution $F(x, r)$, with inputs I and error probability $\delta(\cdot)$, if:

(1) For each $x \in I$, the distribution $f_{A,B}(x, \cdot, \cdot)$ of B's computed outputs approximates, with error probability $\delta(|x|)$, the distribution $F(x, \cdot)$ of intended results.

(2) Let A* be any interactive Turing machine. We require that for any $x \in I$ and for any $s \in \{0, 1\}^*$, the probability that B accepts the computation of (A*, B) on input x and writes out the string s as its output is bounded by the quantity prob $(F(x, \cdot) = s) + \delta(|x|)$.

Note that, according to the second part of this definition, it may be possible for a malicious adversary A* to bias the distribution of the set of conversations of (i.e., the set of sequences of messages exchanged by) A* and B on a particular input string x. But A* cannot significantly increase the probability that any given result string is accepted by B; in particular, A* cannot force B to accept an erroneous result (one that occurs with probability zero in the distribution $F(x, \cdot)$) except with probability $\delta(|x|)$.

Observe that the probability threshold δ occurs twice in the above definition. In general, there may be protocols for which it makes sense to define correctness with two different δ's. In all our examples, the function $\delta(k)$ is exponentially vanishing in k; therefore, for simplicity, we use the same δ in both places.

3. Knowledge. In the setting of complexity theory, what do we mean by "knowledge"? Informally, a message conveys knowledge if it communicates the result of an intractable computation. A message that consists of the result of a computation that we can easily carry out by ourselves does not convey knowledge. In particular, a string of random bits—or a string of bits that is "indistinguishable" from a random string (as defined above)—does not convey knowledge, since we can flip coins by ourselves.

3.1. Minimum knowledge. Suppose that (A, B) is a confirming interactive proof-system for a language L, taking inputs from the set I. Following the definition in [17], we say that the system (A, B) is *minimum-knowledge* if, given any expected polynomial-time probabilistic Turing machine B*, there exists another probabilistic Turing machine M_{B^*}, running in expected polynomial time, such that the ensembles $\{M_{B^*}[x, h] \mid x \in L, h \in H_x\}$ and $\{VIEW_{B^*}\{(A, B^*)[x, h]\} \mid x \in L, h \in H_x\}$ are (computationally) indistinguishable. If the ensembles are identical, we say that the proof-system is *perfectly minimum-knowledge.*

The output of M_{B^*}, on input $x \in L$ and initial history h, is a simulation of B*'s view of the computation that A and B* would have on the same input and the same initial history. Note that, in this definition, we are not concerned with inputs that do not belong to L. When it takes part in a successful execution of the protocol with input x, B* learns that (with high probability) the predicate of language-membership associated with the protocol, $x \in L$, is true; however, it gains no more knowledge that this. Note that in our examples, B (the machine that acts according to the protocol specifications) does not use its initial history string at all; however, when we worry about the "knowledge" that a cheating machine B* may try to extract from A we have to consider the fact that B* can use its history string.

The authors of [17] called a confirming proof-system satisfying the above properties "zero-knowledge." We now show how to extend this definition so as to be able to say when a more general sort of protocol—for example, a two-party protocol whose purpose is to transfer to one of the parties the result of a hard computation—should be called "minimum-knowledge."

Let (A, B) be an interactive pair of Turing machines which constitute a correct transfer protocol for the probability distribution $F(x, r)$, with inputs I and error probability δ. We say that (A, B) is *minimum-knowledge* if, given any expected polynomial-time probabilistic Turing machine B*, there exists another probabilistic Turing machine M_{B^*}, running in expected polynomial time, such that:

(1) M_{B^*} has one-time access to an F-oracle, as follows. Given any input x and initial history h, M_{B^*} queries the oracle with input x; the oracle returns a value distributed according to $F(x, \cdot)$.

(2) The ensembles $\{M_{B^*}[x, h] \mid x \in I, h \in H_x\}$ and $\{VIEW_{B^*}\{(A, B^*)[x, h]\} \mid x \in I, h \in H_x\}$ are indistinguishable.

If the ensembles are identical, we say that the proof-system is *perfectly minimum-knowledge.*

In order to motivate this definition, we recall that we are trying to formalize the notion of the amount of knowledge transmitted by a sequence of messages. Speaking informally, one gains no knowledge from a message which is the result of a feasible computation that one could just as well have carried out by oneself. If the purpose of a protocol followed by two interacting parties A and B is that A transmit to B a value v chosen according to the probability distribution $F(x, r)$, we would like to be able to say exactly when the protocol transmits no more knowledge than this value. We might also demand that the protocol accomplish this even if B somehow tries to cheat—that is, even if the Turing machine B is replaced by another (polynomial-time, but possibly "cheating") machine B*. The simple transmission of the value v can be modelled by a single oracle query (with input x). If the provision of this oracle query makes it possible, by means of a feasible computation, to simulate B*'s view of the "conversation" that A and B* would have had on input x, then we can say that when A and B* actually have a conversation (i.e., follow the protocol) with the same input, there is no *additional* knowledge transmitted to B* besides the value v.

Note that if F is computable in expected polynomial time, then the F-oracle adds no power to the machine M_{B^*}. In this case M_{B^*} can compute F without the assistance of A.

In all our examples, the simulating machine M_{B^*} uses the program of B* as a subprogram or subroutine. This subprogram makes use of the simulator's input tape (containing the input string x), a virtual history tape (which is initialized to contain the given initial history h), a virtual random tape, a virtual work tape, two virtual communication tapes, and a virtual output tape. Without loss of generality we supply the probabilistic machine M_{B^*} with two random tapes; one of these is B*'s virtual random tape. On its output tape—which is also the virtual history tape for the subprogram B*—the simulator uses the subprogram to write records that correspond to B*'s view of the simulated protocol execution.

While carrying on its computation, the machine M_{B^*} may back up a few steps in the simulated protocol and restore a previous machine configuration: It recovers the old state of B* and the old content of the virtual work tape, and resets both the virtual read-head of B*'s random tape and the write-head of its own output tape (the virtual history tape) to where they had been earlier; then it proceeds with its simulation, starting again from the old configuration but "flipping new coins" in its probabilistic computation.

The virtual communication tapes are used to simulate the communication activities of the simulated protocol. The simulator "sends " a message to B* by writing it on the appropriate virtual communication tape and then activating the subprogram. The subprogram operates for (the simulation of) one active turn and then writes a message on the other virtual communication tape; this is the next message "received" from B*. Just as in the interaction of B* with A, the simulator's subprogram B* records random bits, messages read and written, and the computed result on the virtual history tape. The operation of the subprogram B* during a simulated active turn, beginning in a certain state with a certain configuration of the virtual tapes, is identical to the operation of the interactive Turing machine B* during an active turn, beginning in the same state with the same configuration of the actual tapes, of an actual protocol execution with A. This matter of the difference in B*'s operation, either as a subprogram of the simulator or as a Turing machine interacting with A, is discussed further in the remark at the end of the next section.

3.2. Concatenation of protocols. Next, we investigate how protocols may be concatenated in order to achieve modularity in protocol design and how properties of the resulting protocol can be derived from the properties of its subprotocols. The protocol presented in this paper is an example of such a modular design.

We write $s \cdot s'$ for the concatenation of the two strings s and s'.

Suppose that we are given two protocols $P_1 = (\mathrm{A}_1, \mathrm{B}_1)$ and $P_2 = (\mathrm{A}_2, \mathrm{B}_2)$. We define the *concatenation* of the two protocols, denoted $P = P_1; P_2$, to be the following two-stage protocol: Its first stage is P_1. If at the end of this stage A_1 is not in a failure state and B_1 has not rejected, the protocol continues with P_2; otherwise the protocol halts. We write $\mathrm{A}_1; \mathrm{A}_2$ and $\mathrm{B}_1; \mathrm{B}_2$ for the interacting machines of the concatenated protocol. At the end of an execution, the history tape of $\mathrm{B}_1; \mathrm{B}_2$ contains the initial history-string, followed by B_1's private view of the execution of P_1, followed by B_2's private view of the execution of P_2.

Assume that P_1 and P_2 are two transfer protocols for the probability distributions F_1 and F_2, respectively, both taking inputs from the set I. Then the concatenated protocol, on input $x \in I$, transfers to $\mathrm{B}_1; \mathrm{B}_2$ the combined result $[F_1(x, \cdot), F_2(x, \cdot)]$. As a special case, suppose that P_1 is a confirming interactive proof-system for L_1 with inputs I, and that P_2 is a confirming interactive proof-system for L_2 with inputs L_1. Then the concatenated protocol is a confirming interactive proof-system for $L_1 \cap L_2$, with inputs I.

It may not be surprising that the concatenation of two correct protocols gives the correct combined result. The more important observation is that, as we prove below, the concatenated protocol is minimum-knowledge if P_1 and P_2 are both minimum-knowledge.

LEMMA. *Given two protocols P_1 and P_2 as above, with error probabilities $\delta_1(k)$ and $\delta_2(k)$, respectively. Then the concatenation $P = P_1; P_2$ is a correct transfer protocol for the combined result $[F_1(x, \cdot), F_2(x, \cdot)]$ with error probability $\delta(k) = \delta_1(k) + \delta_2(k) + \delta_1(k) \cdot \delta_2(k)$. Furthermore, if P_1 and P_2 are both minimum-knowledge (or, respectively, both perfectly minimum-knowledge), then so is their concatenation.*

Proof. First we show that correctness of protocols is preserved by concatenation. It is clear that if the output distribution of $(\mathrm{A}_1, \mathrm{B}_1)$ approximates the intended distribution F_1 with error probability δ_1, and the output of $(\mathrm{A}_2, \mathrm{B}_2)$ approximates the intended distribution F_2 with error probability δ_2, then (A, B) approximates $[F_1, F_2]$ with error probability at most $\delta_1 + \delta_2$.

To show that the second correctness condition holds, let $x \in I$ and let s_1, s_2 be a pair of possible output strings, occurring with probabilities p_1 and p_2 in probability distributions $F_1(x, \cdot)$ and $F_2(x, \cdot)$, respectively. The pair (s_1, s_2) occurs with probability $p_1 p_2$ in the combined result distribution $[F_1(x, \cdot), F_2(x, \cdot)]$. Let us write $\delta_1 = \delta_1(|x|)$ and $\delta_2 = \delta_2(|x|)$. By the correctness of protocols P_1 and P_2, if A* is any interactive

Turing machine that interacts with $\mathrm{B}=\mathrm{B}_1;\mathrm{B}_2$, then the probability that B writes out (s_1, s_2) is at most

$$(p_1+\delta_1)(p_2+\delta_2)=p_1p_2+\delta_1p_2+\delta_2p_1+\delta_1\delta_2\leqq p_1p_2+(\delta_1+\delta_2+\delta_1\delta_2),$$

as required.

Next we show that concatenation maintains the minimum-knowledge property. Assume that P_1 and P_2 are both minimum-knowledge, and let B* be any probabilistic interactive Turing machine, running in expected polynomial time, that interacts with $\mathrm{A}_1;\mathrm{A}_2$. We may write $\mathrm{B}^*=\mathrm{B}_1^*;\mathrm{B}_2^*$ to denote the two parts of B*'s program. For convenience, let us write $V_1[x,h]=VIEW_{\mathrm{B}_1^*}\{(\mathrm{A}_1,\mathrm{B}_1^*[x,h]\}$ and $V_2[x,h]=VIEW_{\mathrm{B}_2^*}\{(\mathrm{A}_2,\mathrm{B}_2^*)[x,h]\}$. Thus, for any input string x and any initial history h, we have $VIEW_{\mathrm{B}^*}\{(\mathrm{A},\mathrm{B}^*)[x,h]\}=\{v_1\cdot v_2\,|\,v_1\in V_1[x,h],\ v_2\in V_2[x,h\cdot v_1]\}$.

To show that the concatenated protocol P is minimum-knowledge we have to show the existerce of a simulating expected polynomial-time probabilistic Turing machine $\mathrm{M}=\mathrm{M}_{\mathrm{B}^*}$ whose output ensemble $\{\mathrm{M}[x,h]\,|\,x\in I,\ h\in H_x\}$ is indistinguishable from the ensemble $\{VIEW_{\mathrm{B}^*}\{(\mathrm{A},\mathrm{B}^*)[x,h]\}\,|\,x\in I,\ h\in H_x\}$.

Our hypothesis on P_1 implies that, given B_1^*, there is a simulating machine M_1, running in expected polynomial time, with access to an F_1-oracle, so that the ensembles $\{\mathrm{M}_1[x,h]\,|\,x\in I,\ h\in H_x\}$ and $\{V_1[x,h]\,|\,x\in I,\ h\in H_x\}$ are indistinguishable. Similarly, our hypothesis on P_2 implies that, given B_2^*, there is a simulating machine M_2, running in expected polynomial time, with access to an F_2-oracle, so that the ensembles $\{\mathrm{M}_2[x,h]\,|\,x\in I,\ h\in H_x\}$ and $\{V_2[x,h]\,|\,x\in I,\ h\in H_x\}$ are indistinguishable. We specify M to be the machine that operates as follows, given any input string $x\in I$ and initial history $h\in H_x$. First, M runs machine M_1 on (x,h) to produce an output h_1. Second, if h_1 is the simulation of a successful execution of P_1, then M runs M_2 on (x,h_1) to produce its final output; otherwise, M simply writes out h_1.

For any $x\in I$, $h\in H_x$ we define the sets of strings

$$E_1[x,h]=VIEW_{\mathrm{B}^*}\{(\mathrm{A},\mathrm{B}^*)[x,h]\}=\{v_1\cdot v_2\,|\,v_1\in V_1[x,h],\ v_2\in V_2[x,h\cdot v_1]\},\text{ and}$$

$$E_2[x,h]=\mathrm{M}[x,h]=\{m_1\cdot m_2\,|\,m_1\in \mathrm{M}_1[x,h],\ m_2\in \mathrm{M}_2[x,h\cdot m_1]\}.$$

(As usual, the choices of the bit-strings that are written on these probabilistic machines' random tapes define a probability distribution on both of these sets.) We need to show that the ensembles $\bar{E}_1=\{E_1[x,h]\,|\,x\in I,\ h\in H_x\}$ and $\bar{E}_2=\{E_2[x,h]\,|\,x\in I,\ h\in H_x\}$ are indistinguishable. For this purpose, we introduce the intermediate ensemble $\bar{E}_3=\{E_3[x,h]\,|\,x\in I,\ h\in H_x\}$, where

$$E_3[x,h]=\{v_1\cdot m_2\,|\,v_1\in V_1[x,h],\ m_2\in \mathrm{M}_2[x,h\cdot v_1]\}.$$

Assume that $\bar{E}_1$ and $\bar{E}_2$ are not computationally indistinguishable. Then there is a polynomial-size distinguisher $D=\{D_{x,h}\,|\,x\in I,\ h\in H_x\}$ that distinguishes between the two ensembles. In other words, in the notation of § 2.2, for some n and for infinitely many pairs (x,h) (with $x\in I$, $h\in H_x$)

$$|p_D(E_1[x,h])-p_D(E_2[x,h])|\geqq|x|^{-n}.$$

This implies, by the triangle equality, that at least one of the inequalities

$$(1)\qquad |p_D(E_2[x,h])-p_D(E_3[x,h])|\geqq\tfrac{1}{2}|x|^{-n}$$

$$(2)\qquad |p_D(E_1[x,h])-p_D(E_3[x,h])|\geqq\tfrac{1}{2}|x|^{-n}$$

holds infinitely often, i.e., that the circuit-family D distinguishes either between $\bar{E}_2$ and $\bar{E}_3$ or between $\bar{E}_1$ and $\bar{E}_3$ (or both.) We next show that either of these possibilities leads to a contradiction.

First, we show that if D distinguishes between $\bar{E}_2$ and $\bar{E}_3$ then we can construct a distinguisher D_1 that distinguishes between the ensembles $\{\mathrm{M}_1[x, h] \mid x \in I, h \in H_x\}$ and $\{V_1[x, h] \mid x \in I, h \in H_x\}$, contradicting the hypothesis that P_1 is minimum-knowledge. Let I_1 be the infinite set of pairs (x, h) for which inequality (1) holds. Given as input a string s, chosen either from $\mathrm{M}_1[x, h]$ or from $V_1[x, h]$, let $C_{x,h}$ be the (probabilistic) circuit that does the following: It simulates the operation of M_2 on input (x, s) for a suitable multiple of its expected running time to produce either a string m_2 or (for those few sequences of coin-flips which may cause M_2 to run too long) a null output, then passes $h \cdot m_2$ to the circuit $D_{x,h}$, which gives its output. Since the simulation of M_2 is polynomial in length, and D is polynomial-size, the circuit-family C is also polynomial-size. Inequality (1) shows that for all pairs $(x, h) \in I_1$, the circuit $C_{x,h}$ distinguishes between $\mathrm{M}_1[x, h]$ and $V_1[x, h]$. Therefore, $C_{x,h}$ can be converted into a deterministic polynomial-size circuit $(D_1)_{x,h}$ that distinguishes between the same two sets.

Second, we show that if D distinguishes between $\bar{E}_1$ and $\bar{E}_3$ then we can construct a distinguisher D_2 that distinguishes between the ensembles $\{\mathrm{M}_2[x, h] \mid x \in I, h \in H_x\}$ and $\{V_2[x, h] \mid x \in I, h \in H_x\}$, contradicting the hypothesis that P_2 is minimum-knowledge. Let I_2 be the infinite set of pairs (x, h) for which inequality (2) holds, and consider the infinite set $I_2' = \{(x, v) \mid v \in V_1[x, h], (x, h) \in I_2\}$. We define $(D_2)_{x,h}$ to be the circuit whose output, on input s (chosen either from $\mathrm{M}_2[x, h]$ or from $V_2[x, h]$) is the output of $D_{x,h}$ on input $h \cdot s$. Since D is polynomial-size, it is clear that D_2 is polynomial-size, too. Inequality (2) shows that $(D_2)_{x,h}$ distinguishes between $\mathrm{M}_2[x, h]$ and $V_2[x, h]$ for infinitely many pairs (x, h)—namely, for all pairs $(x, v) \in I_2'$.

We therefore conclude that the concatenated protocol is minimum-knowledge. Analogous arguments show that the concatenated protocol is perfectly minimum-knowledge if the same is true of both component protocols. □

REMARK. We mention here a special case of the above lemma that we use implicitly throughout the proofs in §§ 5 and 6. Suppose that a protocol (A, B) is given, and consider a certain point in the protocol execution when A has just sent a message and B is about to perform its next active turn. Let P_1 be the protocol up to this point, and let P_2 consist just of B's next active turn. The lemma implies that if P_1 and P_2 are minimum-knowledge, then so is the given protocol through the end of B's next turn. This allows us to specify a machine $\mathrm{M}_{\mathrm{B}^*}$ for our proofs below, simply by having the machine activate a subprogram B* as explained at the end of the previous section: As long as the subprogram, when activated, has access to a virtual history tape whose contents are indistinguishable from the history tape of an actual protocol execution carried on with A, its operation within $\mathrm{M}_{\mathrm{B}^*}$ is identical to its operation during an actual interaction.

3.3. Result indistinguishability.

Next we introduce the eavesdropper C, as described above. Recall that $COM\{(\mathrm{A}, \mathrm{B})[x]\}$ is the set of possibilities for C's view of the computation of A and B on input x. In all our examples of interactive pairs of Turing machines (A, B), neither machine uses its history tape. Thus, without loss of generality we can assume that A and B begin their computation with their history tapes initially empty.

We call an interactive pair of Turing machines (A, B) result-indistinguishable if an eavesdropper that has access to the communications of A and B on input x gains *no* knowledge. More precisely, the system (A, B) is *result-indistinguishable* if there exists a probabilistic polynomial-time Turing machine M such that the ensembles $\{M[x] \mid x \in I\}$ and $\{COM\{(A, B)[x]\} \mid x \in I\}$ are indistinguishable. If the ensembles are exactly identical, we say that the proof-system is *perfectly result-indistinguishable.*

Suppose that (A, B) is a transfer protocol for the probability distribution $F(x, r)$. Observe that unlike the simulating machine in the definition of the minimum-knowledge property, this machine M does not have access to an oracle for F. In other words, M can simulate the communications of A and B on input x, regardless of the value $F(x, r)$ (even if computing F is intractable). Since this simulation is by means of a feasible computation that an eavesdropping adversary could carry out for itself, the adversary gains no knowledge if it is given the text of a "conversation" belonging to the set $COM\{(A, B)[x]\}$.

We remark that if two protocols are result-indistinguishable, then so is their concatenation. The simulating machine for the concatenated protocol is simply the concatenation of the two simulators for the component protocols; neither the interacting parties nor the simulator makes any computation that depends on the history tapes.

4. Specification of the language

4.1. Preliminaries.

We assume that the reader is familiar with the following notions from elementary number theory. (See, for example, [19], [23] for the number theory, and [21] for a computational point of view.) We will be working in the multiplicative group $\mathbf{Z}_N^*$ of integers relatively prime to N. Any element $z \in \mathbf{Z}_N^*$ is called a *quadratic residue* if it is a square mod N (i.e., if the equation $x^2 \equiv z \bmod N$ has a solution); otherwise, z is a *quadratic nonresidue* mod N. Given N and $z \in \mathbf{Z}_N^*$, the quantity called the *Jacobi symbol* of z with respect to N, denoted $(\frac{z}{N})$, can be efficiently computed (in time polynomial in log N) and takes on the values +1 and −1. If $(\frac{z}{N}) = -1$, then z must be a quadratic nonresidue mod N. On the other hand, if $(\frac{z}{N}) = +1$, then z may be either a residue or a nonresidue. Determining which is the case, without knowing the factorization of N, appears to be an intractable problem, namely the *quadratic residuosity* problem. (However, given the prime factorization of N, it is easy to determine whether or not z is a quadratic residue.) Several cryptographic schemes have been proposed that base their security on the assumed difficulty of distinguishing between residues and nonresidues modulo an integer N that is hard to factor [16], [3], [22].

We also make use of Bernstein's law of large numbers [25], [21]: Suppose that the event E occurs with probability p, and let $F_k(E)$ denote the frequency with which E occurs in k independent trials. Then for any $k \geqq 1$ and any positive $\varepsilon \leqq p(1-p)$,

$$\text{Prob}\,\{|F_k(E) - p| \geqq \varepsilon\} \leqq 2\, e^{-k\varepsilon^2}.$$

4.2. The language.

The protocol introduced in [17] is a minimum-knowledge confirming interactive proof-system for the language

$$\{(N, z) \mid z \in \mathbf{Z}_N^*,\ z \text{ a quadratic nonresidue mod } N\}.$$

The protocol that we present below is a deciding interactive proof-system, which is both minimum-knowledge and result-indistinguishable, for a language based on the same problem.

We use the notation $\nu(N)$ to represent the number of distinct prime factors of an integer N.

Our protocol is concerned with integers of a special form, namely those with prime factorization $N = \prod_{i=1}^{k} p_i^{e_i}$ such that for some i, $p_i^{e_i} \equiv 3 \bmod 4$. Let $\boldsymbol{BL}$ (for Blum, who pointed out their usefulness in cryptographic protocols) denote the set of such integers. There are two equivalent formulations of membership in $\boldsymbol{BL}$: (1) $N \in \boldsymbol{BL}$ if and only if for any quadratic residue mod N, half its square roots (mod N) have Jacobi symbol $+1$ and half its square roots have Jacobi symbol -1. (2) $N \in \boldsymbol{BL}$ if and only if there exists a quadratic residue mod N which has two square roots with different Jacobi symbols [2].

The special integers that we require form a subset of $\boldsymbol{BL}$, namely

$$\boldsymbol{N} = \{N \mid N \in \boldsymbol{BL},\ N \equiv 1 \bmod 4,\ \nu(N) = 2\}.$$

It is not hard to see that this set may be defined equivalently as

$$\boldsymbol{N} = \{p^i q^j \mid p \neq q \text{ prime},\ i, j \geqq 1,\ p^i \equiv q^j \equiv 3 \bmod 4\}.$$

Finally, we define the languages

$$I = \{(N, z) \mid N \in \boldsymbol{N},\ z \in \mathbf{Z}_N^*,\ (\tfrac{z}{N}) = +1\} \quad \text{and}$$

$$L = \{N, z) \in I \mid z \text{ a quadratic residue mod } N\}.$$

Taking I as the set of inputs, this paper gives a deciding interactive proof-system for L. Notice that $I - L = \{(N, z) \in I \mid z \text{ a quadratic nonresidue mod } N\}$.

4.3. Outline of the protocol. Our protocol is the concatenation of two subprotocols. The first part is a confirming interactive proof-system for I. If the first part is completed successfully (i.e., if A proves to B that the input string is in I), then A and B perform the second part of the protocol. The second part, taking inputs from the set I, is a deciding interactive proof-system for the language L; A proves to B either that the input string is in L or that it is not in L. Both parts are minimum-knowledge, and the second part is result-indistinguishable as well. The eavesdropper learns that, with high probability, the input is in I. But he gains no more knowledge than this—in particular, he gains no computational advantage in deciding whether the input is in L or in $I - L$, i.e., whether or not z is a quadratic residue mod N.

The confirmation that an input string (N, z) belongs to I in turn requires three stages, each of which confirms a property of N or of z; these stages are carried out in the following order:

(1) $N \equiv 1 \bmod 4$, $\nu(N) > 1$, $z \in \mathbf{Z}_N^*$, and $(\frac{z}{N}) = +1$.

(2) $N \in \boldsymbol{BL}$.

(3) $\nu(N) \leqq 2$.

While proving that our protocol has the properties that we desire, we make no limiting assumption about the computational power of Turing machine A. However, we remark that the protocol can be performed by a probabilistic polynomial-time Turing machine A which is given the factorization of the relevant integers N. (In the cryptographic applications that we discuss later, the party that performs A's role in our protocol chooses N along with its prime factorization.)

We now give the details of our protocol: the confirming first part in § 5, and the deciding second part in § 6.

5. Interactive confirmation of the input language. In each of the protocols that we describe, we use the notation "A→B: · · · " to indicate the transmission of a message from A to B.

5.1. Blum's coin-flip protocol. Our confirmation protocol requires that A and B jointly generate a sequence of bits. The verifier B has to be sure that A cannot bias these bits. They do this by following a protocol due to Blum [2].

An integer $N \in \boldsymbol{BL}$, $N \equiv 1 \bmod 4$, is given.

A and B generate a random bit b:

1. B chooses $u \in \mathbf{Z}_N^*$ at random, and computes $v := u^2 \bmod N$;
   B→A: v

```
    2. A chooses $\sigma := +1$ or $-1$ at random, its guess for $(\frac{u}{N})$;
       A → B: $\sigma$
    3. B → A: $u$
    4. if $v \not\equiv u^2 \bmod N$, then A halts the protocol in the FAILURE state;
       otherwise, if $\sigma = (\frac{u}{N})$ then $b := 1$ else $b := 0$
```

The message triple (v, σ, u) may be regarded as an encoding of the bit $b = \frac{1}{2} + \frac{1}{2}(\frac{u}{N})\sigma$.

This protocol is correct: Since B picks u at random and A picks the sign σ at random, the bit b chosen by the protocol is random. Furthermore, the first alternate characterization of ***BL*** (§ 4.2) implies that no interactive Turing machine A*, no matter what its computational power, can bias the bit produced, since it cannot guess the Jacobi symbol of the square root of v chosen by B with probability greater than $\frac{1}{2}$.

We remark that a cheating Turing machine B* could bias the bit solely by using its ability to produce two numbers u and u', both square roots (mod N) of v, with opposite Jacobi symbols; this capacity would enable B* to factor N simply by computing the greatest common divisor $(u - u', N)$.

The protocol is perfectly minimum-knowledge. The reason is that A's only task is to transmit a guess, $\sigma = +1$ or -1, for a sign, a task that may easily be carried out by a simulator interacting with B*. We formalize this argument below.

5.2. The confirmation protocol. This is a minimum-knowledge confirming interactive proof-system by which A proves to B that the input (N, z) is in the language I defined above. It consists of the concatenation of three subprotocols, each of which takes, as legal input, a string that has been confirmed (with high probability) by the preceding subprotocol. Let k denote the length of the input string.

First Stage: The easy properties of N and z

This stage involves no communications between A and B. Given (N, z) as input, B checks that $N \equiv 1 \bmod 4$, that N is not a prime power, and that $(\frac{z}{N}) = +1$. Each of these is easily accomplished in time polynomial in $\log N$ [21]. If any one of these conditions does not hold, then B REJECTS the proof (and halts the entire protocol).

Second Stage: N belongs to ***BL***

The following protocol is due to Blum [2]. The error probability of this proof-system is $\delta_2(k) = 2^{-k}$. This stage does not concern itself with z at all. The integer N must satisfy $N \equiv 1 \bmod 4$; this condition holds if the first stage has been completed successfully.

```
1. repeat $k$ times:
    1.1 A chooses a quadratic residue $r \in \mathbf{Z}_N^*$ at random;
        A → B: $r$
    1.2 B chooses $\sigma := +1$ or $-1$ at random;
        B → A: $\sigma$
    1.3 if $\sigma \notin \{-1, +1\}$, then A halts the protocol in the FAILURE state;
        otherwise, A computes $s$ such that $s^2 \equiv r \bmod N$ and $(\frac{s}{N}) = \sigma$;
        A → B: $s$
    1.4 B checks to make sure that $s$ satisfies the above conditions; if not, then
        B REJECTS the proof (and halts the entire protocol).
2. B ACCEPTS the proof.
```

Third Stage: N has two prime factors

This stage also does not concern itself with z.

Let us use $\mathbf{Z}_N^*(\pm 1)$ to denote the set of elements of $\mathbf{Z}_N^*$ with Jacobi symbol ± 1 (respectively). This protocol relies on the fact that if N has exactly i prime factors (i.e., $\nu(N)=i$), then exactly $1/2^{i-1}$ of the elements of $\mathbf{Z}_N^*(+1)$ are quadratic residues. A and B jointly pick random elements of $\mathbf{Z}_N^*(+1)$. If A can show that about half of them are residues (e.g., by producing their square roots mod N), then B should be convinced that $\nu(N) \leqq 2$. Since N is not a prime power, $\nu(N)$ must be equal to 2.

In order to pick a list of random elements of $\mathbf{Z}_N^*(+1)$, A and B follow Blum's coin-flip protocol, which requires that $N \in \boldsymbol{BL}$ and $N \equiv 1 \bmod 4$. These conditions hold (with very high probability) if the second stage has been completed successfully.

This proof-system has error probability $\delta_3(k) = 2e^{-k(1/8)^2}$.

```
1. A and B use Blum's coin-flip protocol to generate $k$ random elements
   $r_1, \cdots, r_k \in \mathbf{Z}_N^*(+1)$:
   $i := 0$;
   do until $i = k$:
      a. generating it bit by bit using Blum's coin-flip protocol, A and B choose
         a number $a$, $0 < a < N$
      b. if g.c.d. $(a, N) \neq 1$ (which happens with vanishingly small probability)
         then HALT the protocol
      c. if $(\frac{a}{N}) = +1$ then $i := i + 1$; $r_i := a$
2. for each $i = 1, \cdots, k$ such that $r_i$ is a quadratic residue, A computes $s_i$ such
   that $r_i \equiv s_i^2 \bmod N$;
   A → B: $(i, s_i)$
3. B checks that at least $\frac{3}{8}$ of the $r_i$ are quadratic residues; if so then B ACCEPTS
   the proof (and otherwise B REJECTS the proof and halts the entire protocol).
```

THEOREM 1. *This protocol is a perfectly minimum-knowledge confirming interactive proof-system for the language* $I = \{(N, z) \mid N \equiv 1 \bmod 4,\ N \in \boldsymbol{BL},\ \nu(N) = 2,\ (\frac{z}{N}) = +1\}$.

Proof. We treat each of the three subprotocol stages separately. As a consequence of the lemma of § 3.2, it then follows immediately that the concatenation of the three has the required properties.

First Stage

The first stage is, trivially, a confirming proof-system for the language

$$I_1 = \{(N, z) \mid N \equiv 1 \bmod 4,\ \nu(N) > 1,\ z \in \mathbf{Z}_N^*,\ (\tfrac{z}{N}) = +1\},$$

since each of these conditions is validated by B in polynomial time without interacting with A at all.

Second Stage

Given an integer $N \equiv 1 \bmod 4$ (in particular, given input that has been confirmed in the first stage), the second stage is a perfectly minimum-knowledge confirming interactive proof-system for the language $I_2 = \{(N, z) \mid N \in \boldsymbol{BL}\}$ with error probability $\delta_2(k) = 2^{-k}$.

This stage requires $O(k)$ communication rounds, during which $O(k^2)$ bits are exchanged.

The correctness of this stage depends on the alternate characterizations of membership in $\boldsymbol{BL}$ (§ 4.2). If $N \in \boldsymbol{BL}$, then each quadratic residue r sent by A has at least one square root mod N with Jacobi symbol $+1$ and at least one square root mod N with Jacobi symbol -1; no matter which sign σ B chooses, A can respond with a square root of the appropriate sign. B accepts the proof with probability 1. On the other hand, if $N \notin \boldsymbol{BL}$ then no quadratic residue mod N has two square roots with Jacobi symbols of opposite sign. In this case, it is very likely that there is some i for which A will be unable to send an appropriate s_i, and B will halt the protocol. The only way for a cheating A* to convince B that $N \in \boldsymbol{BL}$ (by sending the appropriate elements s_i) is by guessing the entire sign-sequence $\sigma_1, \cdots, \sigma_k$; the probability that such a guess will be correct is at most $2^{-k} = \delta_2(k)$. Thus, this protocol is indeed a confirming interactive proof-system for ***BL***.

To prove the perfect minimum-knowledge property, choose any interactive Turing maching B*; we have to specify the computation of a Turing machine M_{B^*} whose output, on input $N \in \boldsymbol{BL}$ and initial history h, is a simulation of B*'s view of the computation that A and B* would have performed on the same input. This view includes a message-history that consists of triples (r, σ, s) satisfying the conditions implicitly defined by the specification of the protocol. As described above in § 3.1, M_{B^*} uses the program of B* as a subroutine. After initializing B*, M_{B^*} operates as follows:

```
1. repeat $k$ times:
   1.1 save the current configuration of B*;
   1.2 choose $s \in \mathbf{Z}_N^*$ at random, compute $r := s^2 \bmod N$, "send" $r$ to the simulated $B^*$, and "receive" $\sigma$ in return;
   1.3 if $\sigma \notin \{-1, +1\}$ then append HALT to A's message-record in B*'s virtual history, write out the updated virtual history, and halt;
       otherwise if $(\frac{s}{N}) \neq \sigma$ then restore the saved configuration of B* and go back to step 1.1;
       (else $(\frac{s}{N}) = \sigma$ and the most recent exchange of messages recorded in B*'s virtual history is the triple $(r, \sigma, s)$)
2. write out B*'s virtual history.
```

For each of the k iterations, the expected number of times the loop has to be repeated is 2, since for any value of r the probability that $(\frac{s}{N}) = \sigma$ is exactly $\frac{1}{2}$; thus the expected running time of M_{B^*} is polynomial in k.

The simulated messages "sent" to B* are drawn from the same probability distribution as the messages sent by A in an actual execution of the protocol, and the random communications triples (r, σ, s) produced by M_{B^*} satisfy the conditions $s^2 \equiv r \bmod N$ and $(\frac{s}{N}) = \sigma$. As explained in § 3.2, these messages are interleaved on the virtual history tape with the random-tape bits used by B*, exactly as they would be in an actual interaction with A. Therefore the sets $M_{B^*}[N, h]$ and $VIEW_{B^*}\{(A, B^*)[N, h]\}$ are identical. This completes the proof for the second stage.

Third Stage

Given an integer from the set

$$\{N \mid N \in \boldsymbol{BL},\ N \equiv 1 \bmod 4,\ \nu(N) > 1\}$$

(in particular, given input that has been confirmed in the first and second stages), the third stage is a perfectly minimum-knowledge confirming interactive proof-system for the language $I_3 = \{(N, z) \mid \nu(N) = 2\}$ with error probability $\delta_3(k) = 2e^{-k(1/8)^2}$.

This stage requires $O(k^2)$ communication rounds, during which $O(k^3)$ bits are exchanged.

During the third stage, A and B together choose random elements of $\mathbf{Z}_N^*(+1)$. Since they do this by means of Blum's coin-flip protocol, and no Turing machine A* can bias the bits produced by Blum's procedure, these elements are indeed produced at random. In order to prove that this stage is a proof-system, consider the experiment of choosing a random element of $\mathbf{Z}_N^*(+1)$, where the experiment is a success if the chosen element is a quadratic residue mod N; let $F_k(N)$ denote the frequency of successes in k independent trials. Recall that B accepts N if the frequency $F_k(N) \geqq \frac{3}{8}$. As mentioned above, the probability of success in one trial is exactly $(\frac{1}{2})^{\nu(N)-1}$. (Since N is known to have at least two prime factors, this probability is at most $\frac{1}{2}$.) If $\nu(N)$ is exactly 2, then the probability that B does not accept N is, by Bernstein's law of large numbers,

$$\text{Prob}\,\{F_k(N) < \tfrac{3}{8}\} \leqq \text{Prob}\,\{|F_k(N) - \tfrac{1}{2}| \geqq \tfrac{1}{8}\} \leqq 2e^{-k(1/8)^2} = \delta_3(k).$$

On the other hand, if N has more than two prime factors, the probability of success in one trial is at most $\frac{1}{4}$, and thus the probability that B incorrectly accepts N (when interacting with a cheating A*) is

$$\text{Prob}\,\{F_k(N) \geqq \tfrac{3}{8}\} \leqq \text{Prob}\,\{|F_k(N) - \tfrac{1}{4}| \geqq \tfrac{1}{8}\} \leqq 2e^{-k(1/8)^2} = \delta_3(k).$$

To prove the minimum-knowledge property, given an interactive Turing machine B* we have to specify the computation of a simulating Turing machine $\mathrm{M}_{\mathrm{B}^*}$. The ensemble that $\mathrm{M}_{\mathrm{B}^*}$ must simulate includes a sequence of Blum coin-flips, so we begin by showing that Blum's coin-flip protocol is perfectly minimum-knowledge. To prove this, we must specify the computation of a probabilistic polynomial-time Turing machine M_{coin}, whose output, on input N (satisfying $N \in \boldsymbol{BL}$ and $N \equiv 1 \bmod 4$), and initial history h, provides a simulation of the ensemble $\mathit{VIEW}_{\mathrm{B}^*}\{(\mathrm{A}, \mathrm{B}^*)[N, h]\}$, which includes a message triple (v, σ, u) encoding a bit as described in § 5.1 above. Modelling the result oracle for the protocol, M_{coin} is given as additional input a (presumably random) bit b.

Given any bit b, M_{coin} (initializes B* and) proceeds as follows:

```
a. execute the protocol with B*:
    1. let B* "send" $v$ (simulating step 1)
    2. save the current configuration of B*
    3. simulate A's action in step 2 by choosing $\sigma := \pm 1$ at random and
       "sending" it to B*
    4. let B* "send" $u$ (simulating step 3)
b. if the bit encoded by $(v, \sigma, u)$ is $b$, then write out B*'s virtual history (which
   includes the triple $(v, \sigma, u)$) and halt; otherwise:
    1. restore the saved configuration of B*
    2. simulating step 2 again, "send" $-\sigma$ (instead of $\sigma$) to B*
    3. let B* "send" $u'$
    4. write out B*'s virtual history (which includes the triple $(v, -\sigma, u')$)
```

Note that if B* does not follow the protocol, it may happen that the numbers u and u' are not the same; if their Jacobi symbol is the same the outcome of the protocol is the same random bit b and this has no effect on the output distribution of M_{coin} (since B*, when interacting with A, can decide to send either u or $-u$). On the other hand, if they have opposite Jacobi symbols mod N, then the outcome bit $1-b$ has been determined by B* and not chosen at random. As noted above, this can only happen if B* can factor N, in which case it indeed has the ability to dictate the outcome of the protocol, regardless of whether it is interacting with A or acting as a subroutine for M_{coin}.

Whether the virtual history of B* written out by M_{coin} was generated in step a or step b of the simulation, the distribution of its possible values (and thus the probability distribution of the bit encoded by the message triple) is identical to that of $\mathit{VIEW}_{\mathrm{B}^*}\{(\mathrm{A}, \mathrm{B}^*)[N, h]\}$. Thus the coin-flip protocol is perfectly minimum-knowledge.

Next we describe the simulation by $\mathrm{M}_{\mathrm{B}^*}$ of the third stage of our protocol. The set $\mathit{VIEW}_{\mathrm{B}^*}\{(\mathrm{A}, \mathrm{B}^*)[N, h]\}$ that $\mathrm{M}_{\mathrm{B}^*}$ must simulate begins with a sequence of Blum coin-flips, which are used to generate random elements of $\mathbf{Z}_N^*$. This simulation can be performed by following the program $\mathrm{M}_{\mathrm{coin}}$ as just described; the difficulty for $\mathrm{M}_{\mathrm{B}^*}$, a polynomial-time machine that may not be able to factor N, is that those elements which are quadratic residues must be randomly generated along with their square roots.

Given as input an integer N that has been confirmed by the first two stages and that satisfies $\nu(N)=2$, and given an initial history h, $\mathrm{M}_{\mathrm{B}^*}$ proceeds as follows:

```
1. $i := 0$; $\Lambda :=$ the empty list
2. do until $i = k$:
     choose a random number $a$, $0 < a < N$;
     if g.c.d. $(a, N) \neq 1$ (which happens with vanishingly small probability) then
     FLAG the number $a$, adjoin it to $\Lambda$, and go to step 3;
     else:
     choose a random bit $b$ (to decide the Jacobi symbol of the next element
     generated);
     if $b = 0$ then adjoin to $\Lambda$ a random element of $\mathbf{Z}_N^*(-1)$;
     else:
         a. $i := i + 1$
         b. choose $s_i \in \mathbf{Z}_N^*$ at random
         c. choose a random bit $b_i$ (to decide whether the next element generated
            should be a quadratic residue);
            if $b_i = 0$ then $r_i := s_i^2 \bmod N$ (a random residue in $\mathbf{Z}_N^*(+1)$)
            else $r_i := -s_i^2 \bmod N$ (a random nonresidue in $\mathbf{Z}_N^*(+1)$)
         d. adjoin $r_i$ to $\Lambda$
3. (simulate as many executions as needed of Blum's coin-flip in order to generate
   the sequence of bits in the list $\Lambda$)
   for each bit $b$ in the representation of each number in $\Lambda$:
     follow the procedure above for $\mathrm{M}_{\mathrm{coin}}$ (using B* as a subroutine), recording
   the numbers $u$ (and possibly $u'$) "sent" by B*;
     if the outcome of the coin-flip simulation is indeed $b$, then continue with the
   next bit in $\Lambda$;
     otherwise B* has "forced" the complementary outcome $1 - b$ by "sending"
   $u$ and $u'$ with $(\frac{u}{N}) \neq (\frac{u'}{N})$, in which case:
       a. use $u$ and $u'$ to factor $N$
       b. discard the rest of $\Lambda$
       c. repeatedly execute Blum's coin-flip with B* (as originally specified,
          without backtracking to restore previous configurations of B*) in order
          to choose elements of $\mathbf{Z}_N^*$, bit by bit, until the resulting list contains $k$
          elements $(r_1, \ldots, r_k$, say) with Jacobi symbol $+1$; again let $\Lambda$ denote
          the new list
4. if the last number in $\Lambda$ is FLAGGED then halt
5. discard the elements in $\Lambda$ with Jacobi symbol $-1$
6. if B* has not "forced" the outcome of any of the coin-flip simulations of step
   3, then for each $r_i$ in $\Lambda$ such that $b_i = 0$ "send" $(i, s_i)$ to B*;
   otherwise, use the factorization of $N$ to test each $r_i$ in $\Lambda$ to see whether it is a
   quadratic residue; if it is, then compute $s_i$ such that $r_i \equiv s_i^2 \bmod N$ and "send"
   $(i, s_i)$ to B*
7. write out the virtual history of B*.
```

If $\nu(N)=2$, then a randomly chosen element of $\mathbf{Z}_N^*$—in particular, one that has been chosen by A interacting with a machine B* that does not "force" the outcome of any Blum coin-flips—will have Jacobi symbol +1 with probability $\frac{1}{2}$; among these, quadratic residues will occur with probability $\frac{1}{2}$. If B*, as a subprogram of the simulator M_{B^*}, does not "force" any (simulated) Blum coin-flips, then the simulator generates elements of $\mathbf{Z}_N$ with exactly the same probabilities, and then perfectly simulates the generating coin-flips; on the other hand, if B* does "force" a coin-flip, then M_{B^*} simply performs with it a sequence of Blum coin-flips, exactly as in the specification of the protocol. Either way, M_{B^*} generates lists of elements of $\mathbf{Z}_N$ with the same distribution as do A and B*, and B* makes identical use of bits from its random tape, so that the sets $VIEW_{B^*}\{(A, B^*)[N, h]\}$ and $M[N, h]$ are identical. This completes the proof for the third stage.

Finally, to conclude the proof of Theorem 1, we see by the concatenation lemma that, given any input string at all, the concatenation of the three stages is a perfectly minimum-knowledge confirming interactive proof-system for the language $I_1 \cap I_2 \cap I_3 = I$. □

6. Interactive decision of quadratic residuosity. If the confirming part of our protocol has been successfully completed, then with high probability the input string (N, z) is in the language I. In particular, we know that $\nu(N) = 2$, that $z \in Z_N^*$, and that $(\frac{z}{N}) = +1$; these are the properties that are required of the inputs to the next part of the protocol.

This part is a deciding interactive proof-system for L, taking inputs from I. The proof-system is both perfectly minimum-knowledge and perfectly result-indistinguishable. As noted above, a pair (N, z) that is known to belong to I either is or is not also a member of L according to whether or not z is a quadratic residue mod N.

To make the exposition clearer, we present three successive versions of our protocol.

Let $y \equiv -1 \bmod N$. Everything that follows holds for any nonresidue $y \in \mathbf{Z}_N^*$ that has Jacobi symbol +1. As long as $N \in \boldsymbol{BL}$ and $N \equiv 1 \bmod 4$, we can take $y = -1$. (*Remark.* If another nonresidue y is desired, A can prove to B, as a preliminary subprotocol stage, that y is a nonresidue by following the minimum-knowledge interactive proof-system of [17].)

Let us fix some notation. For any $x \in \mathbf{Z}_N^*$ we define the predicate:

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

Recall that $\mathbf{Z}_N^*(+1)$ denotes the set of elements of $\mathbf{Z}_N^*$ with Jacobi symbol +1. Since $\nu(N) = 2$, half of these are quadratic residues mod N, and half of them are nonresidues.

Our protocol relies on the fact that if $r \in \mathbf{Z}_N^*$ is chosen at random, then $r^2 \bmod N$ is a random quadratic residue in the set $\mathbf{Z}_N^*(+1)$ and $yr^2 \bmod N$ is a random quadratic nonresidue in $\mathbf{Z}_N^*(+1)$; similarly, $zr^2 \bmod N$ is either a random residue or a random nonresidue in $\mathbf{Z}_N^*(+1)$ according to whether or not z is a residue mod N.

This interactive proof-system has error probability $\delta(k) = 2e^{-4k/81}$.

First version: A deciding proof-system

```
i. Repeat $k$ times:
    1. B chooses $r \in \mathbf{Z}_N^*$ and $c \in \{1, 2, 3\}$ at random, and computes CASE $c$ of:
        1: $x := r^2 \bmod N$
        2: $x := yr^2 \bmod N$
        3: $x := zr^2 \bmod N$
    B → A: $x$
```

2. A computes $b := \mathrm{RES}_N(x)$;
   $\mathrm{A} \to \mathrm{B}$: b
3. B checks that if $c = 1$ then $b = 0$, if $c = 2$ then $b = 1$, and if $c = 3$ then b is consistent with any previous case-3 iterations; if not then B REJECTS the proof and halts the protocol

ii. B ACCEPTS the proof that $\mathrm{RES}_N(z)$ is equal to the consistent value of b for case-3 iterations.

As explained above, if z is a quadratic residue then x's constructed in case 1 are indistinguishable from x's constructed in case 3. If A acts as specified, then when the protocol finishes B will be convinced that z is a residue. The only way that a cheating A* can convince B that z is not a residue is by correctly guessing, among all iterations during which B has sent a residue x, which of these were constructed in case 1 and which of them in case 3; if there are ck such iterations in a particular execution of the protocol, then the probability of successful cheating is 2^{-ck}. Since c is very likely to be close to $\frac{2}{3}$, a simple calculation using Bernstein's law of large numbers shows that the probability of successful cheating is at most $2e^{-4k/81}$. Similarly if z is a quadratic nonresidue. Hence the above version is a deciding interactive proof-system for L.

However, this version is not result-indistinguishable. An observer of an execution of the protocol can easily tell whether he is watching an interactive proof that $\mathrm{RES}_N(z) = 1$ or a proof that $\mathrm{RES}_N(z) = 0$ by keeping a tally of the bits b sent by A in step 2 of each iteration.

Second version: A result-indistinguishable proof-system

A simple modification of the above protocol does hide the result from an eavesdropper. The only change is that at the beginning (before step i), A flips a fair coin in order to decide whether to use $R(x) = \mathrm{RES}_N(x)$ or $R(x) = 1 - \mathrm{RES}_N(x)$ as the bit b to be sent to B in step 2 of each iteration throughout the protocol. $R(x)$ can be regarded as an encoding, chosen at random for the entire protocol, of $\mathrm{RES}_N(x)$.

In step 3, B checks for consistency in the obvious way: B should receive the same bit b in all case-1 iterations and the complementary bit in all case-2 iterations; B should receive a consistent bit b in all case-3 iterations, and its value indicates to B whether or not z is a quadratic residue. As before, if in step 3 of any iteration B finds that the value of b is not consistent then B halts the protocol, REJECTing the proof.

With this modification, the protocol is still—arguing as above—a deciding interactive proof-system for L. Furthermore, it is result-indistinguishable. An eavesdropper expects to overhear one bit about $\frac{2}{3}$ of the time during step 2 of each iteration and the complementary bit the remaining $\frac{1}{3}$ of the time; whether the majority bit in a particular execution of the protocol is 0 or 1 gives him no knowledge. A formal proof of result-indistinguishability of the full protocol is presented below.

However, the version so far presented is not minimum-knowledge. For example, a cheating B* that wanted to find out whether a particular number—17, say—is a quadratic residue mod N could, during one of the iterations, send $x = 17$ in step 1 instead of an element x constructed at random according to B's program. A's response in step 2 will convey to B* the value $\mathrm{RES}_N(17)$, which is something that B* could not have computed by itself.

Third version: A minimum-knowledge result-indistinguishable proof-system

We can make this a minimum-knowledge protocol by refining step 1 of the version just presented; the refinement consists of several interactive substeps by which B gives to A what amounts to a minimum-knowledge proof that the element x that it sends was constructed in one of the three ways specfied (without giving A any knowledge about which of the three ways). The rest of the protocol is unchanged.

1.0 B chooses $r \in \mathbf{Z}_N^*$ and $c \in \{1, 2, 3\}$ at random, and computes CASE c of:
1: $x := r^2 \bmod N$
2: $x := yr^2 \bmod N$
3: $x := zr^2 \bmod N$
$\mathrm{B} \to \mathrm{A}$: x

1.1 B chooses $s_i \in \mathbf{Z}_N^*$ at random $(i = 1, \cdots, 4k)$ and computes:
$T_1 = \{t_1, \cdots, t_k \mid t_i \equiv s_i^2 \bmod N\}$,
$T_2 = \{t_{k+1}, \cdots, t_{2k} \mid t_i \equiv ys_i^2 \bmod N\}$,
$T_3 = \{t_{2k+1}, \cdots, t_{3k} \mid t_i \equiv zs_i^2 \bmod N\}$,
$T_4 = \{t_{3k+1}, \cdots, t_{4k} \mid t_i \equiv yzs_i^2 \bmod N\}$;
taking this to be matrix of 4 rows $[T_1, T_2, T_3, T_4]$ and k columns (where column j contains the elements t_j, t_{k+j}, t_{2k+j}, t_{3k+j}), B randomly permutes each column of the matrix, resulting in a matrix T;
$\mathrm{B} \to \mathrm{A}$: T

1.2 A chooses $S \subseteq \{1, \cdots, k\}$ at random (a *query* indicating a random subset of T's columns);
$\mathrm{A} \to \mathrm{B}$: S

1.3 for each $j \in S$, for each $t_i \in$ column j, $\mathrm{B} \to \mathrm{A}$: s_i
(These numbers show A that B has correctly computed the jth column of T for each $j \in S$ and convince A that it is very likely that at least one other column of T was also computed correctly.)

1.4 A verifies (for each such t_i) that $t_i \equiv$ either s_i^2, ys_i^2, zs_i^2, or $yzs_i^2 \bmod N$, with each congruence being satisfied once in each column $j \in S$;
if not, then A halts the protocol in the FAILURE state

1.5 for each $j \notin S$, for each $t_i \in$ column j, B computes w_i according to Table 1: if x was chosen as case c of step 1.0 and $t_i \in T_l$, then $w_i :=$ the table-entry in the lth row and cth column;

TABLE 1. (Step 1.5.)
(*All computations of table entries are modulo N.*)
$x = \cdots$

$t_i = \cdots$	r^2 $(c=1)$	yr^2 $(c=2)$	zr^2 $(c=3)$
$s_i^2 \in T_1$	$rs_i = \sqrt{(xt_i)}$	$yrs_i = \sqrt{y(xt_i)}$	$zrs_i = \sqrt{z(xt_i)}$
$ys_i^2 \in T_2$	$yrs_i = \sqrt{y(xt_i)}$	$yrs_i = \sqrt{(xt_i)}$	$yzrs_i = \sqrt{yz(xt_i)}$
$zs_i^2 \in T_3$	$zrs_i = \sqrt{z(xt_i)}$	$yzrs_i = \sqrt{yz(xt_i)}$	$zrs_i = \sqrt{(xt_i)}$
$yzs_i^2 \in T_4$	$yzrs_i = \sqrt{yz(xt_i)}$	$yzrs_i = \sqrt{z(xt_i)}$	$yzrs_i = \sqrt{y(xt_i)}$

(For each $j \notin S$, these four numbers show A that if B has correctly computed the jth column of T, then B has correctly computed x.)
for each such t_i, $\mathrm{B} \to \mathrm{A}$: w_i.

1.6 A verifies (for each such t_i) that $w_i^2 \equiv$ either (xt_i), $y(xt_i)$, $z(xt_i)$, or $yz(xt_i) \bmod N$, with each congruence being satisfied once in each column $j \notin S$;
if not, then A halts the protocol in the FAILURE state.

The protocol now continues as before. A sends $b = R(x)$ to B (step 2), and B checks b for consistency (step 3); and then they continue with step 1 of the next iteration. Note that it is in A's "interest" to choose S at random in step 1.2, so that with overwhelming probability both S and $\{1, \cdots, k\} - S$ are reasonably large (and thus the probability that any particular column of T will be queried is close to $\frac{1}{2}$).

The idea is that any machine playing the role of B (and desiring that the protocol succeed) must follow the protocol, because if it tries to cheat during any iteration—either by sending a number x in step 1.0 for which it does not "know" the corresponding number r, or by sending numbers t_i in step 1.1 for which it does not "know" the corresponding numbers s_i—then A will, with overwhelming probability, detect its cheating either in step 1.6 or in step 1.4. This is formalized in the following proof.

THEOREM 2. *Given input belonging to* $I = \{(N, z) \mid N \equiv 1 \bmod 4, N \in \boldsymbol{BL}, \nu(N) = 2, (\frac{z}{N}) = +1\}$, *this protocol is a perfectly minimum-knowledge and perfectly result-indistinguishable deciding interactive proof-system for* $L = \{(N, z) \in I \mid z$ *a quadratic residue* mod $N\}$.

Proof. First we prove that the protocol is a deciding proof-system for L. Since we have already shown that the second version presented above is a proof-system, it suffices to show that the refinement of step 1 preserves this property.

Suppose that z is a quadratic residue. The question is whether a cheating A*—even if it does not choose S at random in step 1.2—can use the numbers sent by B during step 1 to distinguish correctly between case-1 iterations ($x = r^2 \bmod N$ for a random r) and case-3 iterations ($x = zr^2 \bmod N$). Since B has chosen them at random, A* is unable to distinguish between residues t_i of the form s_i^2 and residues t_i of the form zs_i^2. Table 2, the subtable of Table 1 corresponding to these four possibilities, has rows that are permutations of each other, and thus A* is not able to tell whether B is using column $c = 1$ or column $c = 3$ of the whole table.

TABLE 2
A subtable of Table 1.

		$c=1$	$c=3$
$t_i = \cdots$	s_i^2	$\sqrt{(xt_i)}$	$\sqrt{z(xt_i)}$
	zs_i^2	$\sqrt{z(xt_i)}$	$\sqrt{(xt_i)}$

Similarly for nonresidues t_i of the form ys_i^2 or yzs_i^2. A like analysis holds if z is a nonresidue mod N. Hence the protocol is indeed a deciding interactive proof-system for L.

In order to prove the minimum-knowledge property, we choose an interactive Turing machine B* that runs in expected polynomial time; we must describe the computation of a simulating machine $\mathrm{M} = \mathrm{M}_{\mathrm{B}^*}$.

M has one-time access to an oracle for the result of the protocol, as explained in § 3.1. M begins by querying the oracle on the input string (N, z), and learns (with very high probability) the value of $\mathrm{RES}_N(z)$. The rest of the simulation is similar to that of the proof that the protocol of [17] is minimum-knowledge.

As its next step, M flips a coin to simulate A's choice of whether to compute $R(x) = \mathrm{RES}_N(x)$ or $R(x) = 1 - \mathrm{RES}_N(x)$ during the protocol.

In each iteration, M carries on the protocol through the end of (the refinement of) step 1 in a straightforward manner: M uses B* to perform its own version of B's role, and M easily simulates A's role, choosing a random query S in step 1.2 and checking several congruences mod N in steps 1.4 and 1.6. If these congruences do satisfy the check, the difficulty comes in simulating A's communication in step 2, which consists of the bit $R(x)$; how can M quickly calculate the correct value of $\mathrm{RES}_N(x)$? M accomplishes this by following the EXTRACTION procedure described below.

After B* has performed its computations in the simulation of step 1.1 and "sent" the matrix T, M saves the current configuration C_0 of B*. At this point, given C_0 (which includes a fixed random-tape string) and any fixed query-set $S \subseteq \{1, \cdots, k\}$ that A might choose in step 1.2, the lists of numbers that B* would "send" in steps 1.3 and 1.5 in answer to the query S are determined. Let us call S a *satisfiable* query if these answers would satisfy A's verifying checks of steps 1.4 and 1.6, causing the protocol to continue with step 2. (A query that is not satisfiable would cause A to halt the protocol in its failure state.) It is easy to check whether or not a query S is satisfiable, by setting B*'s configuration to C_0, "sending" S to B*, and checking the numbers that B* "sends" in return.

In its simulation, M makes use of an auxiliary matrix T' that contains two data fields for each entry t_i of the matrix T, one for the number s_i and one for the number w_i (where s_i and w_i are related to t_i as in the specification of the protocol). Note that if M succeeds in filling both fields for any single entry t_i, then M can easily deduce the value $R(x)$ that it needs in order to simulate step 2: M can use s_i to see how t_i was computed in step 1.1 (i.e., which set T_j contains t_i, and hence which row of the table B* must use in step 1.5); next M can use w_i to see which column c of the table B* must use; and then the choice of column gives M the value of $\mathrm{RES}_N(x)$, and hence of $R(x)$.

Next we describe the EXTRACTION procedure that M performs in each iteration following the simulation of step 1.1.

```
1. save the current machine configuration $C_0$
2. choose a query $S \subseteq \{1, \cdots, k\}$ at random, store it, and "send" it to B* (simulating step 1.2)
3. let B* "send" its answers to $S$ (the numbers $s_i$ of step 1.3 and $w_{i'}$ of step 1.5),
   and check the congruences of steps 1.4 and 1.6;
   if the congruences do not check, then halt the simulation;
   otherwise, store B*'s answers in the auxiliary matrix $T'$ and repeat the following
   two loops concurrently until success:
     a. sampling the query space (without repetition):
          i. restore configuration $C_0$
         ii. choose a new query $S' \subseteq \{1, \cdots, k\}$ at random that has not
             already been chosen (if there is one; if none exists, then halt
             the sampling loop);
             store $S'$ and "send" it to B*
        iii. let B* "send" its answers to $S'$
         iv. for each $j = 1 \cdots k$ if B*'s answers for column $j$ of the matrix
             $T$ satisfy the congruences of 1.4 (if $j \in S'$) or of 1.6 (if $j \notin S'$),
             then enter them in the auxiliary matrix $T'$;
             if any of these new entries is an $s_i$ for which $T'$ already contains
             $w_i$ or vice versa, then (as explained above) use $s_i$, $w_i$ to compute
             $R(x)$ and set success := TRUE
     b. use any factoring algorithm $F$ to factor $N$:
          i. until (success or {$F$ has successfully factored $N$}) do the next
             step of $F$
         ii. use the prime factors of $N$ to compute $\mathrm{RES}_N(x)$ and $R(x)$,
             and set success := TRUE
```

4. restore configuration C_0 and "send" to B* the original query S
5. let B* "send" its answers and update its history tape (exactly as it did the first time it received the query S)

Simulating step 2, M sets $b := R(x)$ (as computed either in a or b of the last inner loop) and "sends" b to B*, which performs its version of step 3 of the protocol. If B* is following the instructions of step 3, then it is indeed "expecting" the computed value of b, and the simulation continues with the next iteration.

We need to show that M's expected running time is polynomial (in k, the length of the input) and that its output ensemble is identical to B*'s view. To bound the running time, it suffices to prove a polynomial bound on the expected time required by each of the k iterations of M's program. First observe that the outer loop of the EXTRACTION procedure takes polynomial time. The same is true for any single execution of the inner loop: queries may be stored in a lexicographically ordered tree (so that choosing a new one costs $O(k)$); the rest of the sampling loop is polynomial-time, and in each inner loop only one step of the factoring algorithm is performed. Therefore, it is enough to show that, for any fixed configuration C_0, the expected number of repetitions of the inner loop is polynomial in k.

In fact, we show that this number is constant. In configuration C_0, let p $(0 \leqq p \leqq 2^k)$ be the number of queries that are satisfiable. When M performs the EXTRACTION procedure, with probability $1 - p/2^k$ its first query S will not be satisfiable, in which case the inner loop is not executed at all. If $p = 0$, we have no other case to consider; so assume $p \geqq 1$. With probability $p/2^k$, S is satisfiable, and the inner loop is repeated until *success* is set to TRUE (either in the sampling process or after factoring N). Each sampling loop begins with the choice of a new random query. Of the $2^k - 1$ possible queries, at least the $p-1$ satisfiable queries (besides S) lead to *success*; that is, the probability of a successful inner loop is at least $(p-1)/(2^k-1)$. Hence if $p > 1$, the expected number of attempts after choosing a satisfiable original query is at most $(2^k-1)/(p-1)$; overall, the expected number of repetitions of the inner loop is at most $(p/2^k) \cdot (2^k-1)/(p-1) \leqq 2$. We consider below the special case $p = 1$.

Next, we show that the sets $VIEW_{B^*}\{(A, B^*)[(N, z), h]\}$ and $M[(N, z), h]$ are identical; following the Remark of § 3.2, it suffices to show that this is so for any one of the k iterations of the protocol. Consider, therefore, an iteration—either of an actual protocol execution by A and B* or of the simulation by M—at the beginning of which B* sends the matrix T, and let p $(0 \leqq p \leqq 2^k)$ be the number of satisfiable queries. With probability $1 - p/2^k$ a randomly chosen query is not satisfiable, causing either the protocol execution or the simulation to halt; in this case, the actual history and the virtual history are identical. If $p = 0$, then this is the only case that occurs. Otherwise, with probability $p/2^k$, the original query S is satisfiable, and both the actual protocol and the simulation continue with step 2. As long as $p \geqq 2$, there is at least one other query that leads to *success* in the inner loop of M's EXTRACTION procedure, enabling M to "send" in its simulation of step 2 the correct value of $b = R(x)$, the same one that A would send during an actual execution. The factoring algorithm may be faster then the sampling process, in which case the correct value of b is computed directly. Either way, the actual history and the virtual history are identical.

If $p = 1$, then the probability that M's original query is satisfiable is only 2^{-k}. In this rare case, the sampling process in the inner loop of the EXTRACTION procedure might never lead to *success*; the inner loop might not terminate until after N has been factored. Since the cost of factoring N is less than $O(2^k)$, the total expected number of repetitions of the inner loop when $p = 1$ is less than $2^{-k} \cdot 2^k = 1$. In this case, as before, the actual history and the virtual history are identical. This concludes the proof that the protocol is perfectly minimum-knowledge.

In order to prove that the protocol is result-indistinguishable, we must specify the computation of a probabilistic Turing machine M′, running in expected polynomial time, that simulates the communications ensemble $COM\{(\mathrm{A}, \mathrm{B})[N, z]\}$. (Recall that M′ does not have access to any oracle.) M′ begins by flipping a coin to decide whether to simulate the choice $R(z)=0$ or the choice $R(z)=1$. Then in each iteration M′ simulates the specified computations of A and B, except for the following changes. In (simulated) step 1.0, M′ chooses $x := zr^2 \bmod N$ with probability $\frac{2}{3}$ and $x := yzr^2 \bmod N$ with probability $\frac{1}{3}$. In (simulated) step 2, M′ outputs $b = R(z)$ if $x = zr^2$ and $b = 1 - R(z)$ if $x = yzr^2$. (Here the simulation of step 2 is much simpler than in the minimum-knowledge proof above, since M′ "knows" how each x was constructed.) In (simulated) step 1.5, M′ outputs w_i computed according to Table 3.

TABLE 3

	$x = \cdots$	
$t_i = \cdots$	zr^2	yzr^2
s_i^2	$zrs_i = \sqrt{z(xt_i)}$	$yzrs_i = \sqrt{yz(xt_i)}$
ys_i^2	$yzrs_i = \sqrt{yz(xt_i)}$	$yzrs_i = \sqrt{z(xt_i)}$
zs_i^2	$zrs_i = \sqrt{(xt_i)}$	$yzrs_i = \sqrt{y(xt_i)}$
yzs_i^2	$yzrs_i = \sqrt{y(xt_i)}$	$yzrs_i = \sqrt{(xt_i)}$

The numbers x output by M′ have the same distribution as the numbers x output by B; the same is true of the s_i and the w_i. Hence, as required, the set of outputs $\mathrm{M}'[N, z]$ is identical to the set $COM\{(\mathrm{A}, \mathrm{B})[N, z]\}$, so the protocol is perfectly result-indistinguishable.

As presented, the protocol takes $O(k)$ communication rounds during which $O(k^3)$ bits are exchanged. However, all k iterations of the main loop can be performed in parallel, taking $O(1)$ rounds. The simulator M can perform in parallel all k iterations of its main loop, and its expected running time is still polynomial in k. Similarly, M′ can operate in parallel. Thus the parallelized version of the protocol is also perfectly minimum-knowledge and perfectly result-indistinguishable. This concludes the proof of Theorem 2. □

We note that there is another modification of the first version of our protocol that also achieves result-indistinguishability. A can always respond in step 2 with the true value of $\mathrm{RES}_N(x)$ if B computes each x in step 1 according to a random choice among four varieties: to the types r^2, yr^2, and $zr^2 \bmod N$ we add the fourth type $yzr^2 \bmod N$. If the protocol is to be minimum-knowledge as well, we can refine step 1 as in the third version of our protocol, adding an appropriate fourth column to the table used to compute w_i.

7. Cryptographic applications. In all our applications, we let N be the public key of a user A who knows its factorization. Within the set $\mathcal{N}$, it is most advantageous to

A to choose N to be of the form $N=pq$, with p and q of approximately the same size. A can follow our confirming protocol in order to prove to any other user that $N \in \boldsymbol{BL}$ and $\nu(N)=2$. For these applications, we assume that the residuosity problem is intractable.

When A communicates with another user B, any element $z \in \mathbf{Z}_N^*(+1)$ can serve as an encoding of the bit $\mathrm{RES}_N(z)$, as soon as A has used our protocol to prove to B the value of this bit. According to need, z can be chosen by A or by both A and B together (say, by flipping coins). Because of the result-indistinguishability of the protocol, this encoding is cryptographically secure.

In contrast, the conventional approach to hiding knowledge from an eavesdropper is to use encryption. (For example, given two different protocols, one for membership in a language L and the other for nonmembership in L, one could "pad" the protocols so that they both caused messages of the same length to be sent at each round of communications, and then encrypt all messages.) However, in this approach, proving a theorem about the security of the protocol against eavesdroppers usually requires an assumption about the security of the encryption scheme used.

The result-indistinguishability of our protocol suggests two different ways that it can be used in a public-key encryption scheme that is secure against chosen-ciphertext attack.

(1) A sequence of random elements $z_1, z_2, \cdots$ can serve as a probabilistic encryption [16] of the bit-sequence $\mathrm{RES}_N(z_1), \mathrm{RES}_N(z_2), \cdots$, which in turn can be used as a one-time pad to encode a message sent either from A to B or from B to A.

(2) Instead of using the z_i directly to encrypt the bits $\mathrm{RES}_N(z_i)$, we can define a much more efficient scheme for probabilistic encryption by using a short bit-sequence $\mathrm{RES}_N(z_1), \mathrm{RES}_N(z_2), \cdots$ as the random seed for a cryptographically secure pseudorandom bit generator [5], [27], [6] whose security may be based on the unknown factorization of N (e.g., [3], [4]). Sharing the seed, A and B can efficiently generate polynomially many bits and use them as a (very long) one-time pad with which to send messages back and forth. The pad bits alone are secure against any polynomially bounded adversary; furthermore, the adversary gains no computational advantage in guessing any pad bit when he is given probabilistic encryptions of the bits of the seed, nor when he is allowed to overhear the protocol interactions that define these encrypted bits. Because our protocol is only used in order to initialize the system, this scheme has low amortized cost.

Whether the bits $\mathrm{RES}_N(z_i)$ are used directly or to form the seed of a pseudorandom bit generator, the resulting schemes have the minimum-knowledge property with respect to B as well as with respect to an eavesdropper C. In particular, they are provably secure against both chosen-message and chosen-ciphertext attack. For precise definitions of levels of cryptographic security, and for further study of the power that interaction seems to add to public-key cryptography, see [13], [10].

Another application of our protocol gives a new private unbiased coin-flip, generated jointly by A and B. The two users simply choose z at random—for example, choosing its bits by means of Blum's coin-flip. Note that the bits of z are public; it is $\mathrm{RES}_N(z)$, the result of the coin-flip, which is private.

In certain applications we can omit the confirming proof that N is of the required form. Suppose in fact that N has more than two prime factors. For any $z \in \mathbf{Z}_N^*(+1)$, A can carry out the deciding protocol as before. Now, however, if y and z—both quadratic nonresidues in $\mathbf{Z}_N^*(+1)$—have different quadratic character modulo several of the prime factors of N, then A can distinguish numbers of the form r^2 from numbers of the form $yr^2 \bmod N$ and can distinguish each of these from numbers of the form $zr^2 \bmod N$. (This is not true if $\nu(N)=2$; recall that for such N any nonresidue in $\mathbf{Z}_N^*(+1)$ is a nonresidue modulo both prime factors of N.) Thus A can, at will, use our deciding protocol to "prove" to B either that z is a residue or that z is a nonresidue. In either case, the interactively proved value of $\mathrm{RES}_N(z)$—whether or not it is the true value—is cryptographically secure. This value gives B no knowledge whatever. The "proof" only convinces B that A can distinguish between numbers with different quadratic characters mod N, without releasing to B any information about the quadratic character mod N of any particular number. (This can be formalized in terms of a simulator $\mathrm{M}=\mathrm{M}_{\mathrm{B}^*}$ for any given verifier B*. Note that at the beginning of the program for M given in the proof of Theorem 2, we can replace the oracle query for $\mathrm{RES}_N(z)$ with a simple coin-flip; then exactly as in that proof, the two sets

$$VIEW_{\mathrm{B}^*}\{(\mathrm{A},\mathrm{B})[(N,z),h]\}$$

and

$$\mathrm{M}[(N,z),h]$$

are identical.) Thus, we may say that in this case, the protocol is result-indistinguishable even with respect to B.

In this situation, when N has more than two prime factors, we can define the following game: A picks a random nonresidue z with quadratic character different from that of y. A then "proves" to user B_1 that $\mathrm{RES}_N(z)=b_1$, and "proves" to user B_2 that $\mathrm{RES}_N(z)=b_2$. The "proven" value of $\mathrm{RES}_N(z)$ in each execution of the protocol is shared only by the prover A and the verifier B_1 or B_2. In fact, user B_1 has absolutely no computational advantage in deciding whether or not $b_1=b_2$, and neither does user B_2.

8. Conclusions. Approaching knowledge from the point of view of computational complexity, we have studied the interactive transmission of computational results. The protocol that we introduce gives a proof of the value, 0 or 1, of a number-theoretic predicate, $\mathrm{RES}_N(\cdot)$. In a sense that we make precise (extending the definitions of [17]), the verifier gains no more knowledge from an execution of the protocol than this value; this is the "minimum-knowledge" property of the protocol. Furthermore, we are able to analyze the difference between the knowledge gained by the active verifier and that gained by a passive eavesdropper of equal computational power; the protocol is "result-indistinguishable," in that an eavesdropper gains no knowledge at all by overhearing the messages passed during an execution.

Recent work on minimum-knowledge protocols has taken several different directions. Feige, Fiat, and Shamir adapted the result-indistinguishable protocol of this paper (originally presented in [11]) and the protocols of [17] in order to give an efficient minimum-knowledge (and therefore cryptographically secure) identification scheme [9]. Their paper proposes a formalization, similar to that of Tompa and Woll [26], of the notion that a protocol can supply a "proof" that the prover knows some fact or possesses some computational ability, while completely hiding this piece of knowledge. (For example, in case N has more than two prime factors, our deciding proof-system for $\mathrm{RES}_N(\cdot)$ may be regarded as demonstrating the prover's ability to distinguish between numbers with different quadratic characters mod N; see § 7.)

Goldreich, Micali, and Wigderson proved that, under the assumption that one-way functions exist, every language in NP has a minimum-knowledge confirming interactive proof-system; this result has important consequences for the design of cryptographic protocols [14]. Under the assumption that certain number-theoretic computations are infeasible, a similar result was proved by Brassard and Crepeau, both for prover and verifier as described in this paper [7], and for the dual situation in which a resource-bounded prover interacts with a verifier of unlimited computational power [8]. (Our formalization of the requirement that a two-party transfer protocol be minimum-knowledge applies to protocols that depend on such "cryptographic assumptions;" and under the appropriate assumption, the concatenation lemma of § 3.2 holds in the cryptographic setting.) Impagliazzo and Yung gave a construction for the direct minimum-knowledge transfer of the result of any given computation (both for the usual and for the dual model of the computational power of the prover and the verifier); the dual protocol is implemented under the more general assumption that any of a large class of one-way functions exist [20]. Their construction applies to probabilistic as well as deterministic computations, and in particular it provides a minimum-knowledge interactive proof-system for any language possessing a confirming interactive proof-system at all (i.e., for any language in the complexity class IP [1], [18]).

In a recent paper, instead of considering only protocols for transferring a computational result from one party to another, Yao studied a broad class of two-party protocols for what may be called "crytographic computation," in which the (polynomially bounded) users combine their private inputs in order to compute private outputs in a minimum-knowledge fashion, preserving the privacy of these inputs and outputs and hiding partial computational results as much as possible; it may also be required that both users compute their final results simultaneously [28]. Under the asssumption that factoring is hard, Yao showed how to design such a protocol for any given cryptographic computation problem. Continuing this work, Goldreich, Micali, and Wigderson proved similar results for multiparty protocols, assuming that one-way trapdoor functions exist, and showed how such protocols could be made to tolerate faults [15]. Galil, Haber, and Yung simplified and extended these constructions for cryptographic computation, giving new methods for the design of fault-tolerant multiparty cryptographic protocols [12].

In summary, the complexity-theoretic approach to measuring and controlling the knowledge transmitted in various distributed and cryptographic settings has proved to be a useful tool in protocol design.

Acknowledgments. We would like to thank Silvio Micali and Charles Rackoff for their helpful discussions, and Paul Beame, Gilles Brassard, Joan Feigenbaum, Shafi Goldwasser, David Lichtenstein and Adi Shamir for their insightful remarks.

Note added in proof. The concatenation lemma of § 3.2 was proved independently in Y. Oren, *On the cunning power of cheating verifiers: Some observations about zero-knowledge proofs*, Proc. 28th Annual IEEE Symposium on the Foundations of Computer Science, 1987, pp. 462–471, and in [26].

REFERENCES

[1] L. Babai, *Trading group theory for randomness*, Proc. 17th Annual ACM Symposium on the Theory of Computing, 1985, pp. 421–429.

[2] M. Blum, *Coin flipping by phone*, Proc. IEEE COMPCON, 1982, pp. 132–137.

[3] L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM J. Comput., 15 (1986), pp. 364–383.

[4] M. Blum and S. Goldwasser, *An efficient probabilistic public-key encryption scheme which hides all partial information*, in Proc. Crypto '84, Springer-Verlag, New York, Berlin, 1984, pp. 289–299.

[5] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850-864.

[6] R. B. BOPPANA AND R. HIRSCHFELD, *Pseudorandom generators and complexity classes*, in Advances in Computer Research, Volume on Randomness and Computation, JAI Press, to appear.

[7] G. BRASSARD AND C. CREPEAU, *Zero-knowledge simulation of Boolean circuits*, Proc. Crypto '86, Springer-Verlag, New York, Berlin, 1987, pp. 223-233.

[8] ———, *Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for* SAT *and beyond*, in Proc. 27th Annual IEEE Symposium on the Foundations of Computer Science, 1986, pp. 188-195.

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

[10] Z. GALIL, S. HABER, AND M. YUNG, *Symmetric public-key encryption*, Proc. Crypto '85, Springer-Verlag, New York, Berlin, 1985, pp. 128-137.

[11] ———, *A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems*, Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 360-371.

[12] ———, *Cryptographic computation: Secure fault-tolerant protocols and the public-key model*, Proc. Crypto '87, Springer-Verlag, New York, Berlin, 1988, pp. 135-155.

[13] ———, *Symmetric public-key cryptosystems*, 1989, submitted.

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

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

[16] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, JCSS, 28 (1984), pp. 270-299.

[17] S. GOLDWASSER, S. MICALI, AND C. RACKOFF, *The knowledge complexity of interactive proof systems*, Proc. 17th Annual ACM Symposium on the Theory of Computing, 1985, pp. 291-304; SIAM J. Comput., 18 (1989), pp. 186-208.

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

[19] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Oxford University Press, London, 1954.

[20] R. IMPAGLIAZZO AND M. YUNG, *Direct minimum-knowledge computations*, Proc. Crypto '87, Springer-Verlag, New York, Berlin, 1988, pp. 40-51.

[21] E. KRANAKIS, *Primality and Cryptography*, John Wiley, New York, 1986.

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

[23] I. NIVEN AND H. S. ZUCKERMAN, *An Introduction to the Theory of Numbers*, John Wiley, New York, 1972.

[24] C. H. PAPADIMITRIOU, *Games against nature*, Proc. 24th Annual IEEE Symposium on the Foundations of Computer Science, pp. 446-450.

[25] A. RENYI, *Foundations of Probability*, Holden-Day, New York, 1970.

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

[27] A. C. YAO, *Theory and applications of trapdoor functions*, Proc. 23rd Annual IEEE Symposium on the Foundations of Computer Science, 1982, pp. 80-91.

[28] ———, *How to generate and exchange secrets*, Proc. 27th Annual IEEE Symposium on the Foundations of Computer Science, 1986, pp. 162-167.