CUCS-179-85

A PRIVATE INTERACTIVE TEST OF A BOOLEAN PREDICATE AND MINIMUM-KNOWLEDGE PUBLIC-KEY CRYPTOSYSTEMS-

EXTENDED ABSTRACT.

Zvi Galil, Stuart Haber & Moti Yung

A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems

Extended Abstract

Zvi Galil[1,2,3] Stuart Haber[1,3] Moti Yung[1,3,4]

[1] Department of Computer Science, Columbia University
[2] Department of Computer Science, Tel Aviv University

Summary of Results

We introduce a new two-party protocol with the following properties:

1. The protocol gives a proof of the value, 0 or 1, of a particular Boolean predicate which is (assumed to be) hard to compute. This extends the 'interactive proof systems' of [7], which are only used to prove that a certain predicate has value 1.
2. The protocol is provably **minimum-knowledge** in the sense that it communicates no additional knowledge (besides the value of the predicate) that might be used, for example, to compromise the private key of a user of a public-key cryptosystem.
3. The protocol is **result-indistinguishable:** an eavesdropper, overhearing an execution of the protocol, does not know the value of the predicate that was proved. This bit is cryptographically secure. The protocol achieves this without the use of encryption functions, all messages being sent in the clear.

These properties enable us to define a **minimum-knowledge cryptosystem**, in which each user receives exactly the knowledge he is supposed to receive and nothing more. In particular, the system is provably secure against both chosen-message and chosen-ciphertext attack. Moreover, extending the Diffie-Hellman model, it allows a user to encode messages to other users with his **own** public key. This enables a **symmetric** use of public-key encryption.

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 [7] 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 which 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 *validator*, that an input string x is in fact an element of L. The validator 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 *validator*, which uses y to check that indeed $x \in L$.)

An interactive proof system which releases no additional knowledge — that is, nothing more than the one bit of knowledge given by the assertion that $x \in L$ — is called *minimum-knowledge*. Naturally, such interactive protocols are of particular interest in a cryptographic setting where distrustful users with unequal computing power communicate with each other.

In this paper we extend the ability of interactive proof-system protocols from *validation* that a given string x is in a language L to *verification* whether $x \in L$ or $x \notin L$. That is, we give the first (non-trivial) example of a language L so that both L and its complement have minimum-knowledge interactive proof systems for validating membership, where both the proof of membership in L and the proof of non-membership in L are by means of the *same* protocol.

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. In fact, the protocol releases *no knowledge at all* to an eavesdropper. As usual, we assume that the eavesdropper knows both the prover's and the verifier's algorithms and that his computational resources are polynomially bounded, and we allow him access to all messages passed during an execution of the protocol. In spite of the fact that

---

[3] Supported in part by N.S.F. Grant DCR-83-03139.

[4] Supported in part by an IBM graduate fellowship.

– August 1985 – to appear in FOCS-85

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

If membership or non-membership in L is an intractable computation, then, 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 possibly release any compromising knowledge, either to the verifier or to an eavesdropper. The minimum-knowledge requirement is a property which ensures that each party receives exactly the knowledge he is supposed to receive and nothing more. A cryptosystem whose protocols are minimum-knowledge 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.

Moreover, we show how to extend the use of public keys in encryption systems. As introduced by Diffie and Hellman and studied further by many authors, public-key encryption would seem to be inherently assymmetric: messages sent to user A are encrypted using A's public key. (See, e.g., [4, 9].) When interaction is added to the model, it becomes possible to perform *symmetric* public-key encryption. (See also [5].) Using our protocol, we specify a probabilistic minimum-knowledge public-key cryptosystem, in which A's public key is used to encode messages that are sent to A as well as to encode messages that A sends to others. In some applications, it is useful that only one of a group possess an encryption key which can be used for all communications.

It is by serially composing several minimum-knowledge subprotocols that we formulate the more complex minimum-knowledge proof-system that we introduce in this paper. We remark that this demonstrates the importance of the minimum-knowledge property for modular design of complex protocols.

The predicate that our protocol tests is that of being a quadratic residue or non-residue mod 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 non-membership is in $NP \cap co\text{-}NP$. A conventional membership proof for these languages uses 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. 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. More precisely, the probability that the opponent's computation succeeds in compromising N is essentially the same, whether or not his computation is allowed to see the texts of the messages passed during the executions of the protocol.

2. Interactive Proof Systems

We specify the model for which we will describe our protocol; this is an extension of the model used in [7]. 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 read-only for A and write-only for B, while the other is read-only for B and write-only for A. The two machines take turns being "active". While it is active, a machine can read the appropriate tapes, perform computation using its own work tape, and send a message to the other machine by writing the message on its write-only communication tape. In addition, B has a private output tape; whatever is written on this tape when A and B halt will be the result of their computation.

In what follows, B will be limited to random polynomial time, while we make no limiting assumption about A's computational resources. (For cryptographic applications, A will possess some trapdoor information). Their messages to each other will be in cleartext (though these messages will depend on their private coin flips, which will remain hidden). Our scenario will also include a third probabilistic Turing machine C, limited to polynomial-time computation, which can read the communication tapes of A and B and knows their algorithms. A will be the prover, B will be the verifier, and C will be the eavesdropper.

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 (see below) — does not convey knowledge, since we can flip coins by ourselves. Our formal definitions follow.

2.1. Interactive Proof Systems

Suppose that (A, B) is an interactive pair of Turing machines, and let $I \subseteq \{0, 1\}^*$ denote the set of legal inputs to the computations of A and B. Suppose also that $L \subseteq I$ is a language, for which A is able to compute the predicate $(x \in L)$. Suppose in addition that their total communication time is polynomial in the length of the input and in $1/\delta$, where their computation is correct with probability $1-\delta$ (i.e., "with high probability"). Following the authors of [7], we will say that (A, B) is a *validating interactive proof-system* for L if:

1. For any $x \in L$ given as input to (A, B), with high probability B halts and accepts x.
2. For any Turing machine A^* which can interact with B, and for any string $x \notin L$ given as input to (A^*, B), with high probability B halts and rejects x.

Extending the above definition, we will say that (A, B) is a *verifying interactive proof-system* for L if:

- For any Turing machine A^* which can interact with B, and for any string $x \in I$ given as input to (A^*, B), with high probability B halts with the correct value of the predicate $(x \in L)$.

In the first definition, we require that B correctly accept instances of strings $x \in L$, and that no malevolent adversary can convince B to incorrectly accept strings $x \notin L$. In the second definition, we require that, given *any* string x, B correctly verify whether $x \in L$ or $x \notin L$.

2.2. Ensembles of Strings

In order to speak precisely of the "knowledge" transmitted by communicated messages, we will need the following definitions [7, 10]. Let $I \subseteq \{0, 1\}^*$ be an infinite set of strings, and let c be a positive constant; for each $x \in I$ of length n, let π_x be a probability distribution on the set of n^c-bit strings. We will call $\Pi = \{ \pi_x : x \in I \}$ an *ensemble* of strings (usually suppressing any mention of I and c).

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

As another example, suppose that (A, B) is an interactive pair of Turing machines. For any string x, let (A, B)[x] denote the set of possible ordered sequences of messages written on the communication tapes of A and B during their computation on input x. This set has a natural probability distribution according to the coin tosses of A and B. Thus, $\{ (A, B)[x] : x \in I \}$ is an ensemble of strings.

A ***distinguisher*** is a probabilistic polynomial-time Turing machine that, given a string as input, outputs a bit. 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. Let $p_D(x)$ be the probability that D outputs a 1 when it is given as input a string of length $|x|^c$, randomly selected according to probability distribution π_x; and let $p_D'(x)$, depending on the distribution π_x', be defined similarly. We will call the two ensembles (polynomial-time) *indistinguishable* if for any distinguisher D, for all k and sufficiently long x,

$$|p_D(x) - p_D'(x)| < |x|^{-k}.$$

2.3. Minimum Knowledge

Suppose that (A, B) is a validating interactive proof-system for a language L, taking inputs from the set I. Following the definition in [7], we will say that the system (A, B) is *minimum-knowledge* if, given any probabilistic polynomial-time Turing machine B^*, there exists another probabilistic polynomial-time Turing machine M such that:

1. M can use B^* as a subroutine, in the strong sense described below.
2. The ensembles $\{ M[x] : x \in L \}$ and $\{ (A, B^*)[x] : x \in L \}$ are indistinguishable.

M's output, on input $x \in L$, is a simulation of the communications that A and B^* would have on the same input. Note that, in this definition, we do not care about inputs not belonging to L. An eavesdropper who overhears a successful execution of the protocol (A, B) with input x learns that (with high probability) $x \in L$; however, he gains no more knowledge than this.

M can use B^* as a subroutine in the following way. We model the probabilistic nature of B^* by providing it with a random read-only tape. In the course of (simulating A and) communicating with B^*, M is allowed to back up a few steps in the simulated protocol, re-setting B^*'s random read-head to where it had been earlier, and then to proceed with the protocol.

We now show how to extend this definition so as to be able to say when a more general sort of protocol should be called "minimum-knowledge". The computations of any interactive pair of Turing machines (A, B) define a partial function $f_{A,B}$ as follows. Given a string x as input, suppose that A and B use a total of at most k random bits during the course of their probabilistic computations (k polynomial in $|x|$). For any k-bit string r, let $f_{A,B}(x, r)$ denote the result of the computation of A and B (i.e., what is left on B's output tape) on input x, when the sequence of their coin flips is given by r.

In this paper, $f_{A,B}(x, r)$ will only take on Boolean values. For example, if (A, B) is a proof-system for the language L, then $f_{A,B}(x, r)$ is (with very high probability, i.e. for most r) equal to the membership-bit $(x \in L)$.

We will say that the interactive system (A, B) is *minimum-knowledge* if, given any probabilistic polynomial-time Turing machine B*, there exists another probabilistic polynomial-time Turing machine M such that:

1. Given any input x, M has one-time access to an oracle which returns the value $f_{A,B}(x, r)$ for a random $r \in \{0, 1\}^k$.
2. M can use B* as a subroutine, as described above.
3. The ensembles $\{ M[x] : x \in I \}$ and $\{ (A, B^*)[x] : x \in I \}$ are indistinguishable.

Note that, in this definition, the simulation M[x] is defined for any $x \in I$.

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. For us, "feasible" means probabilistic polynomial-time.

If the purpose of a protocol followed by two interacting parties A and B is that A transmit to B a value $f_{A,B}(x, r)$, 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 $f_{A,B}(x, r)$ 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 polynomial-time computation, to simulate the entire "conversation" that A and B* would have had on input x, then it seems reasonable to say that when A and B* actually have a conversation (i.e. follow the protocol) with input x, there is no *additional* knowledge transmitted to B*.

When a user performs A's role in a minimum-knowledge protocol, relying on the security guaranteed by the minimum-knowledge property, it is the user's responsibility not to cheat (i.e. to carry out A's instructions exactly), since the guarantee of security may not hold if A is replaced by a cheating A*.

Note that if $f_{A,B}$ is computable in probabilistic polynomial time, then the $f_{A,B}$-oracle adds no power to the machine M. In this case M (or, for that matter, B) can compute $f_{A,B}$ without the assistance of A. (The protocols that we present in this paper are all minimum-knowledge; however, they may be only trivially so if it turns out that integer factorization, for example, is a tractable problem.)

Suppose that we are given two protocols: the first, taking inputs from I_1, is a validating interactive proof-system for L_1, and the second, taking inputs from I_2, is a validating interactive proof-system for L_2. Suppose in addition that $L_1 \subseteq I_2$. We will use in what follows the simple observation that the concatenation of the two protocols is then a validating interactive proof-system for L_2, taking inputs from I_1. If the two given protocols are minimum-knowledge, then so is their concatenation.

2.4. Result Indistinguishability

We will call an interactive proof-system (A, B) *result-indistinguishable* if an eavesdropper (as described above) who has access to the communications of A and B on input x gains *no* knowledge. More formally, the system (A, B) is *result-indistinguishable* if there exists a probabilistic polynomial-time Turing machine M such that the ensembles $\{ M[x] : x \in I \}$ and $\{ (A, B^*)[x] : x \in I \}$ are indistinguishable.

Observe that unlike the machine M in the definition of the minimum-knowledge property, this machine M does not have access to an oracle for $f_{A,B}$; in other words, M can simulate the communications of A and B on input x, regardless of the value $f_{A,B}(x, r)$ (even if calculating $f_{A,B}$ is an intractable computation). Since this simulation is by means of a feasible computation that an eavesdropping adversary could carry out for himself, the adversary gains no knowledge if he is given the text of a "conversation" belonging to the communications ensemble (A, B)[x].

3. Specification of the Language

3.1. Preliminaries

We assume that the reader is familiar with the following notions from elementary number theory. We will be working in the multiplicative group Z_N^* of integers relatively prime to N. Any element $z \in Z_N^*$ 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 $z \in Z_N^*$, the quantity called the *Jacobi symbol* of z with respect to N, denoted $\left(\frac{z}{N}\right)$, can be efficiently computed (in time polynomial in log N) and takes on the values $+1$ and -1. If $\left(\frac{z}{N}\right) = -1$, then z must be a quadratic nonresidue mod N. On the other hand, if $\left(\frac{z}{N}\right) = +1$, 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 which base their security on the assumed difficulty of distinguishing between residues and nonresidues modulo a hard-to-factor integer N [6, 1, 8].

The protocol introduced in [7] is a minimum-knowledge validating interactive proof-system for the language

$$\{ (N, z) : z \in Z_N^*, z \text{ a nonresidue mod } N \}.$$

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

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

Our protocol will be concerned with integers of a special form, namely integers 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 **BL** (for Blum, who pointed out their usefulness in cryptographic protocols) denote the set of such integers. There are two alternate (equivalent) formulations of membership in **BL**: (1) $N \in$ **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) $N \in$ **BL** iff 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 **BL**, namely

$$\mathcal{N} = \{ N : N \in \mathbf{BL}, N \equiv 1 \bmod 4, \nu(N) = 2 \}.$$

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

$$\mathcal{N} = \{ p^i q^j : p \neq q \text{ prime}, i, j \geq 1, p^i \equiv q^j \equiv 3 \bmod 4 \}.$$

Finally, we define the languages

$$I = \{ (N, z) : N \in \mathcal{N}, z \in Z_N^*, \left(\frac{z}{N}\right) = +1 \}$$

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

Taking I as the set of inputs, this paper gives a verifying interactive proof-system for L.

3.2. Outline of the Protocol

Throughout the description of our protocol, we will be speaking of picking elements of a set "at random". Unless otherwise specified, this will mean that the element in question should be chosen at random according to the uniform probability distribution on the set.

Our protocol has two parts. The first part is a validating 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 verifying 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. (Thus, an eavesdropper who overhears a successful execution of the protocol learns that, with high probability, $N \in \mathcal{N}$. But he gains no more knowledge than this — in particular, he does not learn $\mathrm{RES}_N(z)$.)

The first part, the validation that an input string (N, z) belongs to I, in turn requires three stages, which are carried out in the following order; each stage validates a property of N or of z.

1. $N \equiv 1 \bmod 4$, $\nu(N) > 1$, $z \in Z_N^*$, and $\left(\frac{z}{N}\right) = +1$.
2. $N \in$ **BL**.
3. $\nu(N) \leq 2$.

While proving that our protocol has the properties that we desire, we will 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 will have chosen N along with its prime factorization.)

We now give the details of our protocol: the validating first part in section 4, and the verifying second part in section 5.

4. Validation of the Input

In each of the protocols that we describe, we will use the notation "A → B: . . . " to indicate the transmission of a message from A to B.

4.1. Blum's Coin-Flip Protocol

Our validation protocol will require that A and B jointly generate a sequence of unbiased random bits. They will do this by following a protocol due to Blum [2].

An integer $N \in$ **BL**, $N \equiv 1 \bmod 4$, is given.

A and B generate a random bit b:

1. B chooses $u \in Z_N^*$ at random, computes $v = u^2 \bmod N$;
   B → A: v
2. A chooses $\sigma = +1$ or -1 at random, his guess for $\left(\frac{u}{N}\right)$;
   A → B: σ
3. B → A: u
4. if $\sigma = \left(\frac{u}{N}\right)$ then $b := 1$ else $b := 0$

If factoring N is an intractable problem, then the first alternate characterization of **BL** implies that this protocol generates random bits. In fact, as long as at least one of A and B does not cheat — i.e. as long as either A picks σ at random or B picks u at random — b is a random bit.

This protocol is also minimum-knowledge. To prove this, we fix a (possibly cheating) Turing machine B* that interacts with A. We have to specify the (polynomial-time) computation of a Turing machine M whose output, on input N (satisfying $N \in \mathbf{BL}$ and $N \equiv 1 \bmod 4$), is a simulation of the communications ensemble $(\mathrm{A}, \mathrm{B}^*)[N]$, namely a triple (v, σ, u) as specified above. M can use B^* as a subroutine, and has access to an oracle that returns the intended result of this protocol, namely a random bit.

M begins by consulting the oracle. Having received the random bit b, M executes the protocol with B^* (letting B^* "send" v, simulating A's choice of σ in step 2 by flipping a coin, and then letting B^* "send" u). If the bit generated by this execution is b, then M outputs the triple (v, σ, u). Otherwise, M re-sets B^*'s random read-head, goes back to step 2, "sends" $-\sigma$ instead of σ to B^*, and lets B^* "send" u'; now M outputs the triple $(v, -\sigma, u')$. In either case, the output triple corresponds to the bit b, and the distribution of its possible values is indistinguishable from $(\mathrm{A}, \mathrm{B}^*)[N]$. (Note that if B^* does not follow the protocol exactly as specified, it may happen that the numbers u and u' are not the same; but they must have the same Jacobi symbol mod N, because the ability to compute two square roots (mod N) of v with opposite Jacobi symbols would enable B^* to factor N.)

4.2. The Validation Protocol

This is a minimum-knowledge validating interactive proof system by which A proves to B that the input string (N, z) is in the language I defined above. It consists of three sub-protocols, each of which takes, as legal input, a string that has been validated by the preceding sub-protocol.

First Stage: The trivial properties of N and z

This stage involves no communications between A and B. B checks that $N \equiv 1 \bmod 4$, that N is not a prime power, that $z \in \mathbf{Z}_N^*$, and that $\left(\frac{z}{N}\right) = +1$. Each of these is easily accomplished in time polynomial in $\log N$. If any one of these conditions does not hold, then B rejects the input and halts the protocol.

Second Stage: N belongs to BL

The following protocol is also due to Blum [2]. Its correctness depends on the alternate characterizations of membership in BL. In order to guarantee that the protocol is correct with probability at least $1-\delta$, the parameter k should be chosen so that $k \geq \log 1/\delta$.

This stage does not concern itself with z at all. N must satisfy $N \equiv 1 \bmod 4$; this condition will hold if the first stage was successfully completed.

Repeat k times:

1. A chooses a quadratic residue $r \in \mathbf{Z}_N^*$ at random;
   A → B: r
2. B chooses $\sigma = +1$ or -1 at random;
   B → A: σ
3. A computes s such that $s^2 \equiv r \bmod N$ and $\left(\frac{s}{N}\right) = \sigma$;
   A → B: s
4. B checks to make sure that s satisfies the above conditions; if not, then B rejects the input and halts the protocol.

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 $\nu(N) \leq 2$. If N is not a prime power, then $\nu(N)$ 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 $N \in \mathbf{BL}$ and $N \equiv 1 \bmod 4$. These conditions will hold if the second stage was successfully completed.

In order to guarantee that the protocol is correct with probability at least $1-\delta$, the parameter k' should (according to the weak law of large numbers) be chosen so that $k' \geq 16/\delta$.

1. A and B use Blum's coin-flip protocol to generate k' random elements $r_1, \ldots, r_{k'} \in \mathbf{Z}_N^*(+1)$. Elements of $\mathbf{Z}_N^*$ are generated bit by bit, and those with Jacobi symbol -1 are thrown away.
2. For each $i = 1, \ldots, k'$ such that r_i is a quadratic residue, A computes s_i such that $r_i \equiv s_i^2 \bmod N$;
   A → B: s_i
3. B checks that at least 3/8 of the r_i are quadratic residues; if so he accepts the input, and otherwise he rejects it and halts the protocol.

Theorem: This protocol is a minimum-knowledge validating interactive proof system for the language I.

Proof: We will treat each of the three sub-protocol stages separately. It will then follow immediately that the concatenation of the three has the required properties.

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

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

because each of these conditions can be validated by B without interacting with A at all.

Given an integer $N \equiv 1 \bmod 4$ (in particular, given input that has been validated in the first stage), the **second stage** is a minimum-knowledge validating interactive proof system for the language $I_2 = \{ (N, z) : N \in \mathrm{BL} \}$.

If $N \in \mathrm{BL}$, 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. On the other hand, if $N \notin \mathrm{BL}$ then no quadratic residue mod N has two square roots with Jacobi symbols of opposite sign. With high probability, for some i A will be unable to send an appropriate s_i, so that B will halt the protocol. The only way for a cheating A* to convince B that $N \in \mathrm{BL}$ (by sending the appropriate elements s_i) is by guessing beforehand the entire sign-sequence $\sigma_1, \ldots, \sigma_k$; such a guess will only be correct with probability 2^{-k}. Thus, this protocol is indeed a validating interactive proof system for BL.

To prove the minimum-knowledge property, we fix a Turing machine B* that interacts with A; we have to specify the computation of a Turing machine M whose output, on input $N \in \mathrm{BL}$, will be a simulation of the communications that A and B* would have had on input N: namely, the ensemble $(\mathrm{A}, \mathrm{B}^*)[N]$ which consists of triples (r_i, σ_i, s_i) satisfying the conditions implicitly defined by the specification of the protocol. If B* departs so far from the protocol that these triples are not produced, then A will quickly halt the protocol; simulating the communications between A and such a B* is easy to do. In other words, without loss of generality we may assume that B* behaves "reasonably".

```
M repeats the following loop $k$ times:
  1. choose $s \in Z_N^*$ at random
  2. $r := s^2 \bmod N$
  3. "send" $r$ to B*, and receive $\sigma$ in return
  4. if $\left(\frac{s}{N}\right) \neq \sigma$ then re-set the random read-head of B* and
     go back to step 1; else output $(r, \sigma, s)$
```

For each iteration, the expected number of times this loop will have to be repeated is 2, since for any value of r the probability that $\left(\frac{s}{N}\right) = \sigma$ is exactly 1/2. The triples produced by M do satisfy the required conditions, and so the ensembles $\mathrm{M}[N]$ and $(\mathrm{A}, \mathrm{B}^*)[N]$ are indistinguishable. This completes the proof for the second stage.

Given an integer from the set

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

(in particular, given input that has been validated in the second stage), the **third stage** is a minimum-knowledge validating interactive proof system for the language $I_3 = \{ (N, z) :\ \nu(N) = 2 \}$.

Consider the experiment of choosing a random element of $Z_N^*(+1)$, 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. As mentioned above, the probability of success in one trial is exactly $1/2^{\nu(N)-1}$. (Since N is known to have at least two prime factors, this probability is at most 1/2.) If $\nu(N)$ is exactly 2, then the probability that B does not accept N is, by the weak law of large numbers,

$$\mathrm{Prob}\{ F_{k'}(N) < 3/8 \} \le$$
$$\mathrm{Prob}\{ |F_{k'}(N) - 1/2| \ge 1/8 \} \le \frac{1}{4k'(1/8)^2} = 16/k'.$$

On the other hand, if N has more than two prime factors, the probability of success in one trial is at most 1/4, and thus the probability that B will incorrectly accept N is

$$\mathrm{Prob}\{ F_{k'}(N) \ge 3/8 \} \le$$
$$\mathrm{Prob}\{ |F_{k'}(N) - 1/4| \ge 1/8 \} \le 16/k'.$$

(Note that these estimates of probabilities are also correct for B's interactions with another Turing machine A*, because even a cheating A* cannot bias the bits generated by Blum's coin-flip.) Hence this protocol is indeed a validating interactive proof system for $\{ N:\ \nu(N) = 2 \}$.

To prove the minimum-knowledge property, we have to specify the computation of a simulating Turing machine M. (Once again, as discussed above, we can assume that B* behaves "reasonably".) The communications ensemble $(\mathrm{A}, \mathrm{B}^*)[N]$ that M must simulate consists of many Blum coin-flips which were used to generate random elements of Z_N^*. The difficulty for M is that those elements which are quadratic residues must be randomly generated along with their square roots.

```
Given $N$ such that $\nu(N)=2$, M proceeds as follows:
  1. $i := 0$; $A :=$ the empty list
  2. do until $i=k'$:
       choose a random bit $b$ (to decide the Jacobi symbol of
     the next element generated);
       if $b=0$ then adjoin to $A$ a random element of $Z_N^*(-1)$,
     else:
       a. $i := i+1$
       b. choose $s_i \in Z_N^*$ at random
       c. choose a random bit $b_i$;
          if $b_i=0$ then $r_i := s_i^2$ (a random residue in
          $Z_N^*(+1)$)
          else $r_i := -s_i^2 \pmod N$ (a random non-residue in
          $Z_N^*(+1)$)
       d. adjoin $r_i$ to $A$
  3. simulate Blum's coin-flip in order to 'generate' the
     sequence of bits in the resulting list $A$ (as in section 4.1)
  4. ignore the elements in $A$ with Jacobi symbol $-1$
```

5. for each r_i in A such that $b_i = 0$ output s_i

M generates lists of elements of Z_N^* with the same distribution as do A and B*, so that the communications ensemble (A, B*)[N] and the output ensemble M[N] are indistinguishable. This completes the proof for the third stage.

It is worth remarking that, with 3/8 replaced by suitable constants, the third stage is a minimum-knowledge interactive proof system for the *value* of $\nu(N)$. That is, if the fraction of the r_i that an honest A shows to be quadratic residues is at least 3/8, then B should accept that $\nu(N) \leq 2$; if this fraction is between, say, 3/16 and 3/8, then B should accept that $\nu(N) \leq 3$; and so on.

Finally, we see that, given any input string at all, the concatenation of the three stages is a minimum-knowledge validating interactive proof system for the language $I_1 \cap I_2 \cap I_3 = I$.

5. The Minimum-Knowledge Test of Residuosity

If the validating 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 $\left(\frac{z}{N}\right) = +1$; these are the properties that are required by the next part of the protocol.

This part is a verifying interactive proof system for L, taking inputs from I. A pair (N, z) which 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. Thus, since it is also result-indistinguishable, this part may be regarded as a private interactive test of quadratic residuosity mod N.

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

Let $y \equiv -1 \bmod N$. Everything that follows will hold for any non-residue $y \in Z_N^*$ which has Jacobi symbol $+1$. As long as $N \in \mathrm{BL}$ and $N \equiv 1 \bmod 4$, we can take $y = -1$. If another non-residue y is desired, A can prove to B (as an additional sub-protocol stage) that y is a non-residue by following the minimum-knowledge interactive proof-system of [7].

Let us fix some notation. For any $z \in Z_N^*$ we define the predicate

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

Recall that $Z_N^*(+1)$ denotes the set of elements of Z_N^* with Jacobi symbol $+1$; half of these are quadratic residues mod N, and half of them are non-residues.

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

In order to guarantee that the protocol is correct with probability at least $1-\delta$, the parameter k should be chosen so that $k = \Omega(\log 1/\delta)$.

First version

Iterate $3k$ times:

```
1. B chooses $r \in Z_N^*$ at random
2. B chooses $c \in \{1, 2, 3\}$ at random; case $c$ of:
    1: $z := r^2 \bmod N$
    2: $z := yr^2 \bmod N$
    3: $z := zr^2 \bmod N$
3. B → A: $z$
4. A computes $b = \mathrm{RES}_N(z)$;
   A → B: $b$
5. B checks that if $c = 1$ then $b = 1$, if $c = 2$ then $b = 0$,
   and if $c = 3$ then $b$ is consistent with any previous
   iterations for which $c$ was $= 3$; if not then B halts the
   protocol
```

If the protocol is not halted by B, then the consistent value of b for case-3 iterations is equal to $\mathrm{RES}_N(z)$.

As explained above, if z is a quadratic residue then z's constructed in case 1 are indistinguishable from z'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 z, which of these were constructed in case 1 and which of them in case 3. The probability of successful cheating is less than 2^{-Ck}, for a suitable constant C. Similarly if z is a quadratic non-residue. Hence the above version is a verifying 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 4 of each iteration.

Second version

But a simple modification of the above protocol does hide the result from an eavesdropper. At the beginning, A flips a fair coin in order to decide whether to use $R(z) = \mathrm{RES}_N(z)$ or $R(z) = 1 - \mathrm{RES}_N(z)$ as the bit b to be sent to B in step 4 of each iteration throughout the protocol. $R(z)$ can be regarded as an encoding, chosen at random, of

$\mathrm{RES}_N(x)$.

In step 5, 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 x is a quadratic residue. As before, if in step 5 of any iteration B finds that the value of b is not consistent then B halts the protocol (detecting cheating).

With this modification, the protocol is still --- arguing as above --- a verifying interactive proof-system for L. Furthermore, it is result-indistinguishable. An eavesdropper expects to overhear one bit about 2/3 of the time during step 4 of each iteration and the complementary bit the remaining 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 3 instead of an element x constructed at random according to steps 1 and 2. A's response in step 4 will convey to B^* the value $\mathrm{RES}_N(17)$, which is something that B^* could not have computed by himself.

Third version

We can make this a minimum-knowledge protocol by refining step 3 of the version just presented; the refinement consists of several interactive sub-steps by which B proves to A that the element x which he sends was constructed as specified. (The rest of the protocol is unchanged.)

All computations are modulo N.

3.0 B → A: x

3.1 B chooses $s_i \in Z_N^*$ at random ($i = 1, \ldots, 4k$)

3.2 B computes
$T_1 = \{ t_1, \ldots, t_k : t_i \equiv s_i^2 \bmod N \}$,
$T_2 = \{ t_{k+1}, \ldots, t_{2k} : t_i \equiv ys_i^2 \}$,
$T_3 = \{ t_{2k+1}, \ldots, t_{3k} : t_i \equiv zs_i^2 \}$,
$T_4 = \{ t_{3k+1}, \ldots, t_{4k} : t_i \equiv yzs_i^2 \}$,
T, a random permutation of the elements of $T_1 \cup T_2 \cup T_3 \cup T_4$;
B → A: T

3.3 A chooses S, a random subsequence of T of size $2k$;
A → B: S

3.4 B adds to S randomly chosen elements of $T-S$, forming an enlarged set S' such that the 4 subsets $S' \cap T_j$ are the same size;
for each $t_i \in S'$, B → A: s_i

3.5 A verifies (for each $t_i \in S'$) 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 by 1/4 of the elements of S'; if not, then A halts the protocol

3.6 for each $t_i \in T-S'$, B computes w_i according to the table below: if x was chosen as case c of step 2 and $w_i \in T_j$, then $w_i :=$ the table-entry in the j^{th} row and c^{th} column;
for each $t_i \in T-S'$, B → A: w_i

3.7 A verifies (for each $t_i \in T-S'$) 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 by 1/4 of the elements of $T-S'$; if not, then A halts the protocol

The protocol now continues as before. A sends $b = R(x)$ to B (step 4), and B checks b for consistency (step 5); and then they continue with step 1 of the next iteration.

Table for Step 3.6

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

The idea is that B must follow the protocol, because if he tries to cheat A will, with overwhelming probability, detect his cheating either in Step 3.5 or in Step 3.7. This is formalized in the following proof.

Theorem: Given input belonging to I, this protocol is a result-indistinguishable minimum-knowledge verifying interactive proof-system for L.

Proof: First we prove that the protocol is a verifying proof-system for L. Since we have already shown that this is true of the second version presented above, it will suffice to show that the refinement of step 3 preserves this property.

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

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

Similarly for non-residues t_i of the form ys_i^2 or yzs_i^2. A like analysis holds if z is a non-residue mod N. Hence the protocol is indeed a verifying proof-system for L.

In order to prove the minimum-knowledge property, we fix a Turing machine B* that interacts with A; we must describe the computation of a simulating machine M. As before, we assume that B* behaves "reasonably" enough so that (with high probability) A does not halt the protocol.

M has access to an oracle for the result of the protocol. M begins by querying the oracle on the input string (N, z), and learns (with high probability) the value of $\mathrm{RES}_N(z)$. The rest of the simulation is quite similar to that of the proof that the protocol of [7] is minimum-knowledge.

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

In each iteration, M carries on the protocol through the end of (the refinement of) step 3 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 subsequence S in step 3.3 and checking several congruences mod N in steps 3.5 and 3.7. The difficulty comes in simulating A's communication in step 4, which consists of the bit $R(z)$; how can M quickly calculate the correct value of $\mathrm{RES}_N(z)$? M accomplishes this by saving the messages "sent" so far, re-setting the random read-head of B*, and re-starting the iteration. The second time through the iteration, B* "flips the same coins" and therefore "sends" the same number z (Step 3.0) and the same sequence of numbers T (Step 3.2) as the first time; however, M (continuing its probabilistic computation) "flips new coins" while simulating A in step 3.3. Thus, with high probability the simulated A chooses a subsequence $\overline{S}$ which is different from the subsequence S chosen the first time; the enlarged sets S' and $\overline{S}'$ will also be different. Now M can choose an index i for which $t_i \in S'$ (so that B* sent s_i in step 3.4 the first time through the iteration) and $t_i \notin \overline{S}'$ (so that B* sent w_i in step 3.6 the second time through the iteration). M can use s_i to see which row of the table B* used (i.e. which set T_j contains t_i); then M can use w_i to see which column c of the table B* used. The choice of column gives M the value of $\mathrm{RES}_N(z)$, which can now be used to simulate M's communication of $R(z)$ in step 4 of the iteration. B* then completes the iteration.

By construction, the output ensemble computed by this machine M is indistinguishable from the communication ensemble generated by A and B*, and therefore the protocol is indeed minimum-knowledge.

In order to prove that the protocol is result-indistinguishable, we must specify the computation of a probabilistic Turing machine M′ which will simulate the communications ensemble (A, 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 2, M′ chooses $z := zr^2 \bmod N$ with probability 2/3 and $z := yzr^2 \bmod N$ with probability 1/3. In (simulated) step 4, M′ outputs $b = R(z)$ if $z = zr^2$ and $b = 1-R(z)$ if $z = yzr^2$. (Here the simulation of step 4 is much simpler than in the minimum-knowledge proof above, since M′ "knows" how each z was constructed.) In (simulated) step 3.6, M′ outputs w_i computed according to the following table:

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

The numbers z output by M′ have the same distribution as the numbers z output by B; the same is true of the s_i and the w_i. Hence, as required, the output ensemble M′[N, z] is indistinguishable from the communications ensemble $(A, B)[N, z]$.

As several people have pointed out, there is another modification of the first version of our protocol which also achieves result-indistinguishability. A can always respond in step 4 with the true value of $\mathrm{RES}_N(z)$ if B computes each z in step 2 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 3 as in the third version of our protocol, adding an appropriate fourth column to the table used to compute w_i.

6. Cryptographic Applications

In all our applications, we will let N be the public key of a user A who knows its factorization. Within the set $\mathcal{N}$, it will be 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 validating protocol in order to prove to any other user that $\nu(N) = 2$.

In communicating with another user B, any element $z \in Z_N^*(+1)$ 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.

Thus a sequence of random numbers $z_1, z_2, \ldots, z_k$ can serve as an encryption of the bit-sequence $\mathrm{RES}_N(z_1), \mathrm{RES}_N(z_2), \ldots, \mathrm{RES}_N(z_k)$, which in turn can be used as a one-time pad, sent either from A to B or from B to A.

Instead of using the z_i directly to encrypt the bits $\mathrm{RES}_N(z_i)$, we can define a much more efficient scheme by using $\mathrm{RES}_N(z_1), \mathrm{RES}_N(z_2), \ldots, \mathrm{RES}_N(z_k)$ as the random seed for a cryptographically secure pseudom-random bit generator based on N (e.g. [1], [3]). 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. 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 pseudom-random 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.

Another characteristic of these schemes is worth noting. Unlike the usual assymmetric use of public keys for encryption, in which only messages sent to user A may be encrypted using A's public key, our scheme is *symmetric*; messages sent to A as well as messages that A sends to others are encoded using A's public key. This capability is useful in a number of situations. For example, it enables secure communication with casual users (who are not registered in the public key directory). It also enables A to transfer the same random bits to a group of users, so that each member of the group will be able to broadcast secret messages to all members. (See also [5].)

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 validating proof that N is of the required form. Suppose in fact that N has more than two prime factors. For any $z \in Z_N^*(+1)$, A can carry out the verifying protocol as before. Now, however, if y and z --- both quadratic non-residues in $Z_N^*(+1)$ --- have different quadratic character modulo one 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 non-residue in $Z_N^*(+1)$ is a non-residue modulo both prime factors of N.) Thus A can, at will, use our verifying 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. In fact, A can use the same number z as a private bit-encoding with each of two different users, neither of whom will know the value of the other's bit.

7. Conclusions and Open Problems

Goldwasser, Micali, and Rackoff give a minimum-knowledge protocol for proving that a number is a nonresidue mod N, where N may be any integer at all; there is also a minimum-knowledge protocol for proving that a number *is* a residue mod N, again for general N [7]. However, the two protocols do not have the same form. Our protocol for proving the value of $\mathrm{RES}_N(z)$ has the same form, whatever that value is. On the other hand, we require a minimum-knowledge proof that N is of a special form. There remains the problem of constructing a single protocol for $\mathrm{RES}_N(z)$ that holds for general N.

More generally, one would like to see examples of minimum-knowledge interactive proof systems for other languages.

As a formalization of the notion of security for cryptosystems, the minimum-knowledge property seems to be the strongest possible. Therefore, its absence is easier to demonstrate than other types of cryptanalytic vulnerabilities. Indeed, several of the cryptographic schemes that have been proposed are easily seen not to have the minimum-knowledge property.

Acknowledgements

We gratefully acknowledge the friendly encouragement and pointed criticism of Silvio Micali. We would also like to thank Shafi Goldwasser, Paul Beame, David Lichtenstein, Gilles Brassard and Charles Rackoff for their helpful remarks and discussions.

References

[1] L. Blum, M. Blum, and M. Shub.
A simple secure pseudo-random number generator.
In *Crypto '82*. 1982.

[2] M. Blum.
Coin flipping by phone.
In *COMPCON*, pages 133-137. IEEE, February, 1982.

[3] M. Blum and S. Goldwasser.
An efficient probabilistic public-key encryption scheme which hides all partial information.
In *Crypto '84*. 1984.

[4] W. Diffie and M.E. Hellman.
New directions in cryptography.
IEEE Trans. on Inform. Theory IT-22:644-654, November, 1976.

[5] Z. Galil, S. Haber, and M. Yung.
Symmetric public-key encryption.
1985.
Presented at Crypto '85.

[6] S. Goldwasser and S. Micali.
Probabilistic encryption and how to play mental poker keeping secret all partial information.
In *Proc. 14th STOC*, pages 365-377. ACM, 1982.

[7] S. Goldwasser, S. Micali, and C. Rackoff.
The knowledge complexity of interactive proof systems.
In *Proc. 17th STOC*, pages 291-304. ACM, 1985.

[8] M. Luby, S. Micali, and C. Rackoff.
How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin.
In *Proc. 24th FOCS*, pages 11-22. IEEE, 1983.

[9] G.J. Simmons.
Symmetric and assymmetric encryption.
Computing Surveys 11:305-330, December, 1979.

[10] Yao, Andrew C.
Theory and applications of trapdoor functions.
In *Proc. 23rd FOCS*, pages 80-91. IEEE, 1982.