# The Theory of Minimum-Knowledge Protocals

Stuart Haber

Technical Report
CUCS-264-87

# The Theory of Minimum-Knowledge Protocols

Stuart Haber

Columbia University
Department of Computer Science

April 1987

## Abstract

This paper explores the complexity-theoretic approach to the transmission of knowledge that was introduced by Goldwasser, Micali, and Rackoff, and further studied by a number of authors. Roughly speaking, a protocol designed to solve a given computational problem is said to be *minimum-knowledge* if its outputs give no more information than an oracle for the problem would give to a user whose computational resources are polynomially bounded. This notion has important consequences for the design of cryptographic protocols. A few slightly different definitions have been given in the literature; some of the results included here have been published previously without proofs. This paper proposes a uniform definition, collects the known results and proves them, and describes the problems that are still not understood.

# Table of Contents

# 1. Introduction

"What is knowledge?" is an old philosophical riddle. Recently, theoretical computer scientists have attempted to formulate a theory of the knowledge possessed by the participants in a multi-party protocol (e.g. [14, 21, 13]). From researchers in cryptography and complexity theory have come the notions of "interactive proof-systems" and "zero-knowledge" (or "minimum-knowledge") protocols, which are the subject of this survey [20]. These were first introduced by Goldwasser, Micali, and Rackoff. Among the motivations for this work we mention the following:

- A desire to quantify the "amount of knowledge" transmitted by a message or gleaned from an interaction by one of the participants. The first step, perhaps, is to characterize those situations in which *no* knowledge (or information), or at least the minimum possible knowledge, is communicated.
- Modularity and decomposability of protocols: We would like to be able to combine cryptographic protocols in such a way as to preserve their properties of cryptographic security. Protocols are still not well understood. For example, in many protocols the participants use random bits; the question is, what exactly should we require of a sub-routine (sub-protocol) for producing or distributing these bits?
- Security of cryptographic keys: What does it mean to say that a key is not compromised? How long is it "safe" to use a key?

In this approach, instead of trying to prove theorems directly about the "knowledge" transmitted by a communicated message, or about users' changing "knowledge states" during the course of a protocol execution [13], we take a computational approach and consider certain theoretically more tractable objects. We model the participants involved in a protocol as interacting Turing machines, and we study the sets of strings that may appear on the various tapes of these machines.

In the next section, we give the necessary definitions. In section 3, we describe a minimum-knowledge protocol for proving two given graphs isomorphic, and another protocol for proving them nonisomorphic; these two protocols exemplify the techniques that have been used so far in constructing minimum-knowledge protocols, as well as several of the questions about their construction that are still not well understood. In section 4, we discuss a class of problems, the "random-self-reducible" languages, that seems to be appropriate for cryptographic applications, and all of which can be shown to have minimum-knowledge protocols for proving membership. Section 5 presents minimum-knowledge protocols for several problems in elementary number theory. In section 6, we prove the theorem of Goldreich, Micali, and Wigderson that, under the assumption that one-way functions exist, every language in NP has a minimum-knowledge protocol for proving membership [17]. Finally, in section 7, we briefly discuss the consequences of the results of section 6 for the design of cryptographic protocols.

# 2. Preliminaries

## 2.1. Ensembles of strings

We need the following definitions [20, 30]. Let $I \subseteq \{0,1\}^*$ be an infinite set of strings; for each $x \in I$, let $\pi_x$ be a probability distribution on the set of bit-strings. (Without loss of generality, we often assume that $\pi_x$ assigns positive probability only to strings of length $|x|^k$, where $k$ is a positive constant.) We call $\Pi = \{ \pi_x \mid x \in I \}$ an *ensemble* of strings (usually suppressing mention of $I$ and $k$).

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

A crucial notion for the work surveyed here is that of distinguishing between two different ensembles of strings $\Pi = \{ \pi_x \mid x \in I \}$ and $\Pi' = \{ \pi_x' \mid x \in I \}$. We imagine the following experiment being performed. A string is drawn at random, either with probability distribution $\pi_x$ or with probability distribution $\pi_x'$, and the string is given to a judge, who is charged with the task of deciding which of the two distribution was used. If it can be shown that no judge is able to decide, then the two ensembles may be used interchangeably.

More formally, a *distinguisher* is a probabilistic Turing machine that, given a string as input, outputs a bit. Suppose that $\Pi$ and $\Pi'$ are ensembles of strings, and that $D$ is a distinguisher. For any $x \in I$, 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 $\pi_x$, and let $p_D'(x)$, depending on the distribution $\pi_x'$, be defined similarly. We call the two ensembles (polynomial-time) *indistinguishable* if for any polynomial-time distinguisher $D$, for all $m$ and all sufficiently long $x$,

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

We call the two ensembles *statistically indistinguishable* if there is a constant $c$ such that for *any* distinguisher $D$, no matter how powerful, for sufficiently long $x$,

$$|p_D(x) - p_D'(x)| < 2^{-|x|^c};$$

i.e., for all sufficiently long $x$ the two distributions given by that $x$ differ only on an exponentially quickly vanishing fraction of the set of strings of the appropriate length. This is true, of course, if the two ensembles are exactly identical; in this case the difference $|p_D(x) - p_D'(x)|$ is exactly zero, for any distinguisher $D$.

## 2.2. Model of computation

Two probabilistic Turing machines $M_1$ and $M_2$ form a pair of *communicating 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 $M_1$ and write-only for $M_2$, while the other is read-only for $M_2$ and write-only for $M_1$. In addition, each machine has a private input tape and a private output tape. 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. Whatever is written on each machine's output tape when they both halt is the *result* of its computation. Unless stated otherwise, all Turing machines are assumed to be limited to feasible computations; that is, they are limited to probabilistic polynomial time. Finally, a two-party *protocol* is simply a pair of such machines.

In many of the protocols that we describe below, we use the notation "$M_1 \rightarrow M_2: m$" to indicate the transmission of the message $m$ from Turing machine $M_1$ to $M_2$.

In order to study multi-party protocols, we formalize the interacting parties as a system of communicating probabilistic Turing machines. Every machine has a private input tape, a private output

tape, and one public communication tape for each of the other machines. There is a global clock, and the protocol proceeds in *rounds*; during each round every machine reads its communication tapes, performs some polynomial-time computation, perhaps using some random bits, and then may write messages on the communication tapes. There is also a global input $k$, a parameter of input size, of cryptographic security, and of certainty [16, 20, 12].

By an $n$-party *protocol* we mean an $n$-tuple $M=(M_1,\ldots,M_n)$ of such machines. Given local input $i$ and global input $k$, each machine makes some (probabilistic) computations, sends messages to and receives messages from other machines, and halts in time polynomial in the parameter $k$ with a string $o$ on its output tape.

We are concerned with computational problems as formalized by Yao [31] in the following way. Let $\Sigma=\{0,1\}$. If $k$ is the global input, then we have local inputs $(i_1,\ldots,i_n)$ distributed according to a probability distribution $I_k$ over $(\Sigma^*)^n$. Our aim is to design an $n$-party protocol $M=(M_1,\ldots,M_n)$ whose outputs $(o_1,\ldots,o_n)$ are distributed according to another probability distribution $O_k(i_1,\ldots,i_n)$; here, $o_j$ is the output of $M_j$. For example, the special case in which $o_j=f_j(i_1,\ldots,i_n)$ with probability 1 describes in this framework the problem of designing a protocol whose outputs are $n$ specified functions of the inputs. Briefly, we speak of the *computational problem* $[I_k,O_k]$. A protocol $M$ for the problem $[I_k,O_k]$ is *correct* if its outputs are indeed distributed according to $O_k$ when the input distribution is $I_k$.

In the case of two parties $A$ and $B$, we often write their respective inputs in the form $(k,i_A)$ and $(k,i_B)$, where $(i_A,i_B)$ is an instance distributed according to $I_k$.

We let $(M_A,M_B)[k,i_A,i_B]$ denote the set of possible ordered sequences of messages written on the communication tapes of Turing machines $M_A$ and $M_B$ during their computation on inputs $(k,i_A)$ and $(k,i_B)$. This set has a natural probability distribution induced by the coin tosses of $M_A$ and $M_B$. Thus, the *communications ensemble*

$$\{\,(M_A,M_B)[k,i_A,i_B] \mid (i_A,i_B)\text{ distributed according to }I_k\,\}$$

is another example of an ensemble of strings.

Since we are interested in feasible computation, we assume that both the input distributions $\{I_k\}$ and the output distributions $\{O_k(i_1,\ldots,i_n)\}$ form *polynomial-time computable ensembles*. That is, there is a probabilistic Turing machine that, given input $k$, generates in time polynomial in $k$ an instance $(i_1,\ldots,i_n)$ randomly distributed according to the distribution $I_k$. (Alternatively, there is a uniform family of probabilistic polynomial-sized circuits whose $k^{\text{th}}$ member has output distribution $I_k$.) We make a similar assumption on the output distributions for which we hope to construct protocols: there is a probabilistic Turing machine that, on input $k$ and any $n$-tuple $(i_1,\ldots,i_n)$ to which $I_k$ assigns positive probability, generates an instance distributed according to $O_k(i_1,\ldots,i_n)$.

## 2.3. Interactive proof-systems

Much of this paper is devoted to a special sort of two-party computational problem, that of interactively proving membership in a language $L$. In this case, the desired protocol allows one party, the *prover* $P$, to convince the other party, the *verifier* $V$, that a given input string is in fact (with high probability) in $L$; if

the input string is not in $L$, then even a cheating prover should not (except with vanishing probability) be able to convince the verifier that it is.

Such a protocol is called an *interactive proof-system* for the language $L$. This can be formalized in terms of input distributions by requiring that $i_P = i_V = (k,x)$, where $x$ is the string whose membership is in question and $k$ is a parameter of certainty. (Sometimes the prover's input may be of the form $i_P = (k,x,w)$, where $w$ is a witness or a short proof that $x \in L$.) 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; if in fact $x \notin L$, then $V$ should reject the proof, even if $P$ is replaced by another Turing machine $P^*$ (no matter how powerful). 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 correct answers define the desired output distribution for $V$; we allow a positive error probability that is, for any constant $c$, less than $k^{-c}$ for sufficiently large $k$.[1] (The output distribution for $P$ is null. However, $P$ may halt the protocol when it detects cheating on the part of $V$.)

The definition of a confirming proof-system requires that $V$ correctly accept instances of strings $x \in L$, and that no malevolent adversary can convince $V$ incorrectly to accept strings $x \notin L$, except with vanishingly small probability. The definition of a deciding proof-system requires that, given *any* input string $x$, $V$ correctly decide whether $x \in L$ or $x \notin L$, and that no adversary can convince $V$ of what is not the case, except with vanishingly small probability.

One of the well-known characterizations of NP may be rephrased in these terms. Every language $L \in$ NP has a confirming interactive proof-system, consisting of a single interaction: given $x \in L$ as input, the prover simply sends to the verifier a proof $w$; after checking that $w$ does indeed prove that $x \in L$ (a polynomial-time computation), the verifier accepts the proof. (For example, if $L$ is the language of Hamiltonian graphs, then the witness $w$ could be a Hamiltonian circuit in the input graph $x$.) If the input string $x$ is not in $L$, then no such $w$ exists.

## 2.4. Minimum-knowledge: definitions

Suppose that $M = (M_A, M_B)$ is a protocol that solves the two-party computational problem $[I_k, O_k]$. We say that the protocol $M$ is *minimum-knowledge for B* if, given any probabilistic polynomial-time Turing machine $M_B^*$, there exists another probabilistic polynomial-time Turing machine $S$ such that:

1. Given any input $(k, i_B)$ whose second coordinate comes from a pair $(i_A, i_B)$ to which $I_k$ assigns positive probability, $S$ has one-time access to an oracle which returns a value $o_B$ randomly distributed according to $O_k^B(i_A, i_B)$.
2. $S$ can use $B^*$ as a subroutine, as described below.
3. The ensembles $\{ S[k,i_B] \}$ and $\{ (M_A, M_B^*)[k,i_A,i_B] \}$ are indistinguishable.

If the ensembles are statistically indistinguishable, we say that the protocol is *statistically*

[1] In fact, all of the examples we give satisfy the stronger requirement of an error probability which is exponentially vanishing in $k$.

*minimum-knowledge*; if they are exactly identical, we will say that the protocol is *perfectly minimum-knowledge.*

The simulating machine $S$ can use $M_B^*$ as a subroutine in the following way. We model the probabilistic nature of $M_B^*$ by providing it with a random read-only tape. While carrying on its computation, the machine $S$ may back up a few steps in the simulated protocol, re-setting both the read-head of $M_B^*$'s random tape as well as the write-head of its own output tape to where they had been earlier, and then to proceed with the protocol.

In order to motivate this definition, 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 that is the result of a feasible computation that one could just as well have carried out by oneself. In particular, a message that consists of a randomly chosen element of a set conveys no knowledge to the receiver, since the receiver is able to use its own coin-flips in order to choose an element at random. This fact is used many times in the design of minimum-knowledge protocols.

If the purpose of a protocol followed by two interacting parties $A$ and $B$ is that $A$ transmit to $B$ a value $o_B(i_A,i_B)$, 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 $M_B$ is replaced by another (polynomial-time, but possibly cheating) machine $M_B^*$. The simple transmission of the value $o_B(i_A,i_B)$ can be modelled by a single oracle query. If the provision of this oracle query makes it possible, by means of a feasible computation, to simulate the entire "conversation" that $M_A$ and $M_B^*$ would have had on input $(k,i_A,i_B)$, then we can say that when $A$ and $B$ actually have a conversation (i.e. follow the protocol) with these inputs (and $B$ follows the program $M_B^*$), there is no *additional* knowledge transmitted to $B$ besides the value $o_B(i_A,i_B)$. Anything that a cheating party is able to compute using what it learns from $A$ (by running the program $M_B^*$ when it interacts with $A$), it would be able to compute for itself by first running the simulation $S$.

Note that if $o_B(i_A,i_B)$ is feasibly computable just from the input $(k,i_B)$, then the oracle adds no power to the machine $S$. In this case $S$ (or, for that matter, $B$) can compute $o_B(i_A,i_B)$ without the assistance of $A$.

In the literature, a confirming interactive proof-system for a language that is minimum-knowledge for the verifier has often been called a *zero-knowledge* interactive proof-system. This definition was first given by Goldwasser, Micali, and Rackoff [20]; Galil, Haber, and Yung proposed the above generalization to any two-party protocol [15]. The reader who would like to see an example of a minimum-knowledge protocol, without wading through any more definitions, should turn immediately to Section 3.1 below.

## 2.5. Number theory

Many of the protocol examples discussed in this paper involve problems that come from number theory; therefore, we assume that the reader is familiar with the following notions from elementary number theory. (See, for example, [22, 27] for the number theory, and [24] for a computational point of view.)

We use $\mathbf{Z}_N^*$ to denote the multiplicative group 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 $N$. It be convenient to define, for any positive integer $N$ and any element $z \in \mathbf{Z}_N^*$, the predicate

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

If $N=p$ is prime, then $\mathrm{RES}_p(z)$ is given by the *Legendre symbol* $\left(\frac{z}{p}\right) = (-1)^{\mathrm{RES}_p(z)}$, which happens to be equal to $z^{(p-1)/2} \bmod p$. If $N$ has prime factorization $N = \prod_{i=1}^{l} p_i^{e_i}$, then the *Jacobi symbol* of any $z \in \mathbf{Z}_N^*$ (generalizing the Legendre symbol modulo a prime) is given by

$$\left(\frac{z}{N}\right) = \prod_{i=1}^{l} \left(\frac{z}{p_i}\right)^{e_i}.$$

Without using the factorization of $N$, the Jacobi symbol $\left(\frac{z}{N}\right)$ can be computed efficiently (i.e. in time polynomial in $\log N$) by using the law of quadratic reciprocity, 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. (However, given the prime factorization of $N$, it is easy to determine whether or not $z$ is a quadratic residue, because $z$ is a quadratic residue mod $N$ if and only if it is a quadratic residue modulo every prime that divides $N$.) Several cryptographic schemes have been proposed that base their claim to security on the assumed difficulty of distinguishing between residues and nonresidues modulo an integer $N$ that is hard to factor [19, 6, 26].

If $p_1 < p_2 < \ldots < p_l$ are the primes that divide $N$, and $z \in \mathbf{Z}_N^*$, then we may call the list $[\left(\frac{z}{p_1}\right), \ldots, \left(\frac{z}{p_l}\right)] \in \{-1,+1\}^l$ the *Legendre list* of $z$ mod $N$. Two elements $y, z \in \mathbf{Z}_N^*$ have the same Legendre list if and only if their product $yz \bmod N$ is a quadratic residue mod $N$.

## 3. Minimum-knowledge interactive proof-systems

Our first examples of minimum-knowledge protocols are confirming interactive proof-systems for the languages of graph isomorphism and of graph nonisomorphism. These examples do not require much deep mathematics; they depend only on the fact that two graphs $G$ and $G'$ are either isomorphic or nonisomorphic, and that if a given map from the vertices of $G$ to the vertices of $G'$ is claimed to be an isomorphism, then this assertion can be verified or refuted quickly. These two examples illustrate most of the issues that arise in the study of minimum-knowledge proof-systems: the varying computational power that may be required of the prover, whether or not the protocol is perfectly minimum-knowledge, and whether or not the protocol can be executed in parallel while preserving its minimum-knowledge properties.

To *compute a random isomorphism* of a given graph $G = (V, E)$ is to choose a permutation $\pi$ of the set $V$ of vertices of $G$, uniformly and at random from the set of these permutations, and then to compute the permuted edge-set $E' = \{ (\pi(u), \pi(v)) \mid (u, v) \in E \}$; the resulting graph $G' = (V, E')$ is isomorphic to $G$. For convenience, we may speak simply of choosing a random isomorphism $\pi: G \to G'$.

### 3.1. Graph isomorphism

We begin with a confirming interactive proof-system for graph isomorphism, i.e. for the language

$GI = \{ (G_0, G_1) \mid G_0 \text{ and } G_1 \text{ are isomorphic graphs} \}$.

The protocol is perfectly minimum-knowledge. Here (along with the usual input parameter $k$) the verifier's input is a pair of graphs $(G_0, G_1)$, and the prover's input is $(G_0, G_1, \phi)$, where $\phi: G_0 \to G_1$ is an isomorphism. Both prover and verifier are probabilistic polynomial-time machines.

Repeat $k$ times:

1. $P$ computes $\psi: G_0 \to H$, a random isomorphism of $G_0$;
   $P \to V$: $H$
2. $V$ chooses $\varepsilon \in \{0,1\}$ at random (and will ask to be convinced that $G_\varepsilon$ is isomorphic to $H$);
   $V \to P$: $\varepsilon$
3. $P$ computes: if $\varepsilon=0$ then $\pi := \psi$ else if $\varepsilon=1$ then $\pi := \psi\phi^{-1}$ else (if $\varepsilon \notin \{0,1\}$) $P$ HALTs;
   $P \to V$: $\pi$
4. $V$ checks that $\pi: G_\varepsilon \to H$ is an isomorphism; if it is not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

To see that this is a confirming interactive proof-system, first note that if the two input graphs are indeed isomorphic, and both $P$ and $V$ follow the protocol, then $V$ will accept the proof. On the other hand, if the two graphs are *not* isomorphic, the only way for a cheating prover to convince $V$ that they are isomorphic is to guess ahead of time, in each of the $k$ iterations, whether $V$ will ask him to show that $H$ is isomorphic to $G_0$ or to $G_1$; his chance of doing this successfully is $2^{-k}$.

To prove that this protocol is minimum-knowledge for the verifier, let $V^*$ be any communicating Turing machine that can interact with $P$; we have to specify the computation of a Turing machine $S$ that uses $V^*$ as a subroutine and whose output, on input $(k, G_0, G_1)$, is a simulation of the communications that $P$ and $V^*$ would have had on the same input (with $P$'s input augmented by an isomorphism between the two graphs). We make no assumptions about the internal computations of $V^*$. However, if $V^*$ departs so far from the specified protocol that appropriate messages are not sent when they are assigned --- e.g., if $V^*$ rejects the proof (and halts the protocol) even though the two graphs *are* isomorphic and $P$ has been acting correctly --- then $P$ can detect this and halt execution of the protocol. Since this departure is easily detectible in polynomial time, $S$ can simulate the communications between $P$ and such a $V^*$. Thus, without loss of generality we can assume that $V^*$ behaves ''reasonably''.

In this case, the communications ensemble that $S$ must simulate consists of $k$ triples $(H, \varepsilon, \pi)$, where $\pi: G_\varepsilon \to H$ is an isomorphism. $S$ will operate by simulating the actions of $P$ in its ''interactions'' with $V^*$; this is how $S$ ''uses $V^*$ as a subroutine''. $S$ proceeds as follows.

Repeat $k$ times:

i. choose $\varepsilon \in \{0,1\}$ at random;

ii. compute $\pi: G_\varepsilon \to H$, a random isomorphism of $G_\varepsilon$;
   ''send'' $H$ to $V^*$ (simulating step 1 of the protocol);

iii. ''receive'' $\alpha$ from $V^*$ (simulating step 2)

iv. (if $\alpha \notin \{0,1\}$ then HALT; else)
if $\alpha=\varepsilon$ then "send" $\pi$ to $V^*$ (simulating step 3) and append the triple $(H,\varepsilon,\pi)$ to the output tape else
($\alpha\neq\varepsilon$) reset $V^*$ to its configuration at the beginning of the current iteration and go to step i.

If the two input graphs are isomorphic, then in each iteration of the loop above, the graph $H$ is a random variable that is uniformly distributed over the set of isomorphic copies (according to the graph-representation used), regardless of whether $S$ has chosen $\varepsilon=0$ or $\varepsilon=1$. It follows that the event $\alpha=\varepsilon$ will occur with probability exactly 1/2, so that the expected number of times (per iteration) that the loop will be repeated is two. The ensemble $\{S[k,G_0,G_1] \mid G_0 \approx G_1\}$ is identical to the ensemble $\{(P,V^*)[k,(G_0,G_1,\phi),(G_0,G_1)] \mid \phi: G_0 \rightarrow G_1$ is an isomorphism $\}$, and therefore the protocol above is perfectly minimum-knowledge.

The version of this protocol in which all $k$ iterations are executed in parallel, while it is still an interactive proof-system for *GI*, does not appear to be minimum-knowledge for $V$. At least, the proof just given does not generalize to the parallelized version, since the expected number of repetitions that the simulator would have to repeat its guess would be $2^k$, which is no longer polynomial-time.

Next, we discuss a perfectly minimum-knowledge confirming interactive proof-system for a variant of graph isomorphism, namely the language

$$\textit{1of2-GI} = \{(G_0,G_1,F) \mid \text{either } F \approx G_0 \text{ or } F \approx G_1\}.$$

We will use this as a sub-protocol in the interactive proof-system for graph nonisomorphism. For this protocol, the verifier's input is a triple of graphs $(G_0,G_1,F)$, and the prover's input is $(G_0,G_1,F,\varepsilon,\rho)$, where $\varepsilon \in \{0,1\}$ and $\rho: G_\varepsilon \rightarrow F$ is an isomorphism. Both prover and verifier are probabilistic polynomial-time machines.

Repeat $k$ times:

1. $P$ chooses $\varepsilon' \in \{0,1\}$ at random and computes random isomorphisms $\sigma_0: G_{\varepsilon'} \rightarrow H_0$ and $\sigma_1: G_{1-\varepsilon'} \rightarrow H_1$;
   $P \rightarrow V$: $(H_0,H_1)$
2. $V$ chooses $\alpha \in \{0,1\}$ at random;
   $V \rightarrow P$: $\alpha$
3. if $\alpha=0$ then $P \rightarrow V$: $(\varepsilon',\sigma_0,\sigma_1)$;
   else if $\alpha=1$ then $P$ computes $e := \varepsilon \oplus \varepsilon'$ and the map $\tau := \sigma_e \rho^{-1}$ (an isomorphism $F \rightarrow H_e$), and
   $P \rightarrow V$: $\tau$;
   else (if $\alpha \notin \{0,1\}$) $P$ HALTs (detecting cheating)
4. $V$ checks that $\sigma_0: G_{\varepsilon'} \rightarrow H_0$ and $\sigma_1: G_{1-\varepsilon'} \rightarrow H_1$ are isomorphisms if $\alpha=0$ or that (for at least one choice of $e \in \{0,1\}$) $\tau: F \rightarrow H_e$ is an isomorphism if $\alpha=1$;
   if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

A straightforward generalization of the proof above for *GI* shows that this protocol is a perfectly minimum-knowledge confirming interactive proof-system for *1of2-GI*; as before, the parallelized version of the same protocol may not be minimum-knowledge. Note that after an execution of this protocol, the

verifier, while convinced that the triple $(G_0, G_1, F)$ is of the required form --- i.e., that $F$ is isomorphic either to $G_0$ or to $G_1$ --- has learned nothing that would help him decide *which* of these is the case.

### 3.2. Graph nonisomorphism

In this section we give a statistically minimum-knowledge confirming interactive proof-system for graph nonisomorphism, i.e. for the language

$GNI = \{ (G_0, G_1) \mid G_0 \text{ and } G_1 \text{ are not isomorphic graphs} \}$.

Here the shared input is of the form $(k, G_0, G_1)$. For this protocol, the prover must be able to carry out certain computations that are not (known to be) feasible. To be precise, given a third graph, the prover must be able to recognize whether it is isomorphic to $G_0$, to $G_1$, or to neither. For example, the prover may be a probabilistic polynomial-time Turing machine with a *GNI*-oracle (or with a *GI*-oracle). The verifier is simply a probabilistic polynomial-time machine.

We begin with an interactive proof-system that is not minimum-knowledge for $V$.

Repeat $k$ times:

1. $V$ chooses $\varepsilon \in \{0, 1\}$ at random, and computes a random isomorphism $\rho: G_\varepsilon \to F$;
   $V \to P$: $F$
2. $P$ chooses $\varepsilon' \in \{0, 1\}$ so that $G_{\varepsilon'} \approx F$;
   $P \to V$: $\varepsilon'$
3. $V$ checks that $\varepsilon' = \varepsilon$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

In step 2, the prover uses its ability to tell which of the two input graphs is isomorphic to any given graph $F$. If the input graphs are indeed nonisomorphic, and both prover and verifier follow the protocol, then the verifier will accept the proof. On the other hand, if the input string is not in the language *GNI*, i.e. if $G_0$ and $G_1$ are isomorphic, then the $k$ graphs $F$ sent by the verifier in step 1 of each iteration are all isomorphic. Even a prover with unlimited computational power would be unable to tell, in each round, whether $F$ was computed after a choice of $\varepsilon=0$ or of $\varepsilon=1$. The only way a cheating prover can convince the verifier to accept the proof incorrectly is by guessing, $k$ consecutive times, the verifier's coin-flip $\varepsilon$; thus the probability that $V$ will accept the cheater's proof is at most $2^{-k}$. Hence, the above protocol is indeed a confirming interactive proof-system for *GNI*.

But the protocol is not minimum-knowledge for $V$, since in step 1 the verifier may ask about a graph $F$ whose isomorphism type he does not know, and in this case $P$'s answer in step 2 gives him additional knowledge. However, before $P$'s answer $V$ can now use the interactive proof-system given above for the language *1of2-GI* in order to prove to $P$ that the triple $(G_0, G_1, F)$ is of the required form, i.e. that he *does* know the isomorphism type of $F$. Note that $P$ and $V$ must reverse the roles they played before; now, as a sub-protocol of our proof-system for *GNI*, it is $V$ that is proving something to $P$. A subtle point is that, because of this reversal of roles, we are able to use the parallelized version of the sub-protocol. (The parallelized sub-protocol is still a proof-system; we don't need it to be minimum-knowledge for $P$, since $P$ is presumed to be able to decide whether $F$ is isomorphic to $G_0$ or to $G_1$. The purpose of the sub-

protocol is to assure $P$ that $V$ "knows" which is the case.) The full protocol is as follows.

Repeat $k$ times:

1. $V$ chooses $\varepsilon \in \{0,1\}$ at random, and computes a random isomorphism $\rho: G_\varepsilon \to F$;
   $V \to P$: $F$
   - 1.1 for $i=1 \ldots k$, $V$ chooses $\varepsilon_i \in \{0,1\}$ at random and computes random isomorphisms $\sigma_{i0}: G_{\varepsilon_i} \to H_{i0}$ and $\sigma_{i1}: G_{1-\varepsilon_i} \to H_{i1}$;
     $V \to P$: $(H_{i0}, H_{i1})_{i=1 \ldots k}$
   - 1.2 $P$ chooses $I \subseteq \{1, \ldots, k\}$ at random;
     $P \to V$: $I$
   - 1.3 for each $j \notin I$, $V$ computes $e_j = \varepsilon \oplus \varepsilon_j$ and the map $\tau_j = \sigma_{je_j} \rho^{-1}$;
     $V \to P$: $(\varepsilon_i, \sigma_{i0}, \sigma_{i1})_{i \in I}$, $(\tau_j)_{j \notin I}$
   - 1.4 $P$ checks that $\sigma_{i0}: G_{\varepsilon_i} \to H_{i0}$ and $\sigma_{i1}: G_{1-\varepsilon_i} \to H_{i1}$ are isomorphisms for $i \in I$, and that (for at least one choice of $e_j \in \{0,1\}$) $\tau_j: F \to H_{je_j}$ is an isomorphism for $j \notin I$;
     if any of these conditions does not hold, then $P$ HALTs the protocol (detecting cheating)
2. $P$ chooses $\varepsilon' \in \{0,1\}$ so that $G_{\varepsilon'} \simeq F$;
   $P \to V$: $\varepsilon'$
3. $V$ checks that $\varepsilon' = \varepsilon$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

As before, if the input graphs are nonisomorphic, and both prover and verifier follow the protocol, then the verifier accepts the proof. On the other hand, if $G_0 \simeq G_1$, then the sub-protocol refinement (steps 1.1 through 1.4) will not help the prover --- even a prover with unlimited computing power --- to distinguish an iteration in which $V$ has chosen $\varepsilon=0$ from one in which $V$ has chosen $\varepsilon=1$; in either case, the prover only receives a list of random isomorphic copies of $G_0$. Thus, again as before, the only way a cheating prover can convince the verifier to accept the proof incorrectly is by guessing, $k$ consecutive times, the verifier's coin-flip $\varepsilon$; thus the probability that $V$ will accept the cheater's proof is at most $2^{-k}$. Hence, the refined protocol is a confirming interactive proof-system for *GNI*.

To prove that the protocol is minimum-knowledge for the verifier, let $V^*$ be an arbitrary probabilistic polynomial-time Turing machine that interacts with $P$. We must specify the computation of a machine $S$ whose output, on input $(k, G_0, G_1)$ satisfying $(G_0, G_1) \in GNI$, will be a simulation of the "conversation" recorded on the communication tapes of $P$ and $V^*$ when they are given the same input. Assume, without loss of generality, that $V^*$ behaves "reasonably", i.e. that with high probability $P$ does not detect cheating on the part of $V^*$ and halt the protocol.

In each round, $S$ carries on the protocol through step 1.4 in a straightforward manner: $S$ "interacts" with $V^*$, and easily simulates $P$'s role, choosing the set $I$ in step 1.2 and then checking several graph isomorphisms in step 1.4. The difficulty comes in simulating $P$'s communication in step 2; our polynomial-time machine $S$ has no way (given our present knowledge) of computing whether $V^*$ has sent a graph $F$ which is isomorphic to $G_0$ or to $G_1$. $S$ accomplishes this by saving the messages "sent" so far,

and resetting the random read-head of $V$ and its own output head so as to restart the computation at step 1.2. Continuing its probabilistic computation, $S$ flips new coins in order to choose a new set $I'$ that is not a superset of $I$, and "sends" it to $V^*$. Since we are assuming that (with high probability) the messages "sent" by $V^*$ would pass $P$'s checks in step 1.4, there must be an index $j \in I-I'$, so that $V^*$ "sent" $(\varepsilon_j, \sigma_{j0}, \sigma_{j1})$ in the first repetition of step 1.3 and $\tau_j$ the second time. These isomorphisms enable $S$ to recover the isomorphism $\rho: G_\varepsilon \rightarrow F$, and therefore to continue with step 2 of the round.

The above computations are repeated $k$ times, once for each round. By construction, the output ensemble computed by $S$ is statistically indistinguishable from the communication ensemble generated by $P$ and $V$, and therefore the protocol is indeed statistically minimum-knowledge for the verifier. (The only difference between the two ensembles occurs when $V^*$ attempts to cheat during the refinement sub-protocol by guessing ahead of time the subset $I$. There is a positive, though vanishingly small, probability that $P$ does not catch him in a particular execution of the protocol; however, because of the backtracking in our simulation, the simulator $S$ insists on catching $V^*$ cheating and then halts the protocol --- and this happens very quickly.)

If all $k$ iterations are performed in parallel, then the resulting protocol is still an interactive proof-system for *GNI*. Moreover, the simulator can also perform in parallel all $k$ iterations of the simulating procedure just described; thus the parallel version of this protocol is also statistically minimum-knowledge.

The assertion that both *GI* and *GNI* possess minimum-knowledge interactive proof-systems for confirming membership relies on no unproved assumptions (such as the existence of the one-way functions required by the protocols of Sections 6 and 7 below). Of course, it may be the case that there exists a polynomial-time algorithm for graph isomorphism, in which case this assertion is only vacuously true, since a polynomially limited verifier would not need the assistance of a prover in order to ascertain whether two given graphs were or were not isomorphic.

Both protocols rely on what may be called the random-self-reducibility of the graph-isomorphism problem. This is made precise in the next section.

We call attention to several important differences between the two interactive proof-systems described above. The protocol for graph isomorphism is perfectly minimum-knowledge for the verifier when it is performed sequentially, but the parallel version might not be minimum-knowledge; the prover is a probabilistic polynomial-time machine --- no more powerful than the verifier --- which is provided with a witness to the fact that the two input graphs are isomorphic. (The protocol may be regarded as a proof to $V$ that $P$ *knows* this witness.) On the other hand, the protocol for graph nonisomorphism is statistically minimum-knowledge, even when it is performed in parallel; the prover must possess some additional computational power in order to perform its role in the protocol. The reader will notice, as we present more examples of minimum-knowledge protocols, that these two sets of characteristics tend to occur together.

## 4. Random-self-reducible problems

The notion of random-self-reducibility was mentioned by Angluin and Lichtenstein as a possible explanation of why certain number-theoretic functions are good candidates for use in cryptographic applications [3]. The notion was formalized by [28], who proved the theorem described in this section, and by [1], whose results imply that there probably do not exist many examples of random-self-reducible languages.

Following both [28] and [1], we say that a language $L$ is *random-self-reducible* if there is a polynomial-time mapping $\rho: L \to L$ that uses a source of independent random bits, satisfying the following conditions. Without loss of generality, assume that $\rho$ uses $k^c$ bits on inputs of length $k$; we write $\rho(x,r)$ for the output of $\rho$ on input $x \in L$ and $r \in \{0,1\}^{|x|^c}$. For convenience, we write the elements of $L$ as pairs $x=\langle u,v\rangle$.

1. For all $x \in L$ and $r \in \{0,1\}^{|x|^c}$, $|\rho(x,r)|=|x|$, and if $x=\langle u,v\rangle$ then $\rho(x,r)=\langle u,v'\rangle$.
2. For every $x=\langle u,v\rangle \in L$, when $r$ is chosen with uniform distribution in $\{0,1\}^{|x|^c}$, the outputs $\rho(x,r)$ occur with the uniform distribution on the set of instances $\langle u,v'\rangle \in L$ of length $|x|$.

If $L$ is a set in NP, then there is a polynomial-time predicate $p_L$ such that a string $x$ is in $L$ if and only if there is a witness-string $w$ (of size polynomial in $|x|$) satisfying $p_L(x,w)=1$; let $W(x)$ denote the set of all witnesses for $x$. We will say that $L$ is *sampleable*[2] if:

> Given an integer $l$, we can generate pairs $(x,w)$ in time polynomial in $l$, with $x$ uniformly distributed over strings in $L$ of length $l$ and $w$ uniformly distributed over $W(x)$.

In our examples, we would like the self-reduction $\rho$ to interact well with witness-strings, in the following way. We say that the self-reduction $\rho$ is *witness-respecting* if it has the following properties.

1. Let $\rho(x,r)=x'$ and $w \in W(x)$. Given $x, r$ and $w$ it is possible to compute (in polynomial time) a witness $w' \in W(x')$. If $r$ is chosen with uniform distribution in $\{0,1\}^{|x|^c}$, then each $w' \in W(x')$ should occur with equal probability.
2. Let $\rho(x,r)=x'$ and $w' \in W(x')$. Given $x, r$ and $w'$ it is possible to compute (in polynomial time) a witness $w \in W(x)$.

**Example 1:** The language $GI = \{ (G_0,G_1) \mid G_0 \simeq G_1 \}$ of graph isomorphism is also random-self-reducible. With a slight abuse of notation, let $\rho(G_0,G_1,r)=(G_0,G')$, where $\phi_r$ is a randomly chosen permutation of the vertices of $G_0$ and $G'=\phi_r(G_0)$ is the resulting isomorphic copy of $G_0$. (In the notation of our definition, we have $u=G_0$ $v=G_1$, and $v'=G'$.) If the (single) witness for an instance $(G_0,G_1)$ is the isomorphism between the two graphs, then $GI$ is sampleable, and this random-self-reduction is easily seen to be witness-respecting.

**Example 2:** The language $QR = \{ (N,z) \mid N>1, z \text{ a quadratic residue mod } N \}$ of quadratic residues has a random-self-reduction defined as follows. Abusing notation again, let $\rho(N,z,r)=(N,z')$, where $r$ is a randomly chosen element of $\mathbf{Z}_N^*$ and $z'=zr^2 \bmod N$. (Here we have $u=N$, $v=z$, and $v'=z'$.) In this case, with witness sets defined in the obvious way by $W(N,z)=\{ w \in \mathbf{Z}_N^* \mid z \equiv w^2 \bmod N \}$, $QR$ is sampleable; the equivalence $w' \equiv wr \bmod N$ shows that the mapping $\rho$ is witness-respecting.

---

[2] E. Allender and J. Feigenbaum, personal communication.

Example 3: The language

$$\textit{SameQ} = \{ (N,y,z) \mid N>1, y \text{ and } z \in \mathbf{Z}_N^* \text{ have the same Legendre list}^3 \bmod N \}$$

has a random-self-reduction defined as follows. Let $\rho(N,y,z,r)=(N,y,z')$, where $r$ is a randomly chosen element of $\mathbf{Z}_N^*$ and $z'=zr^2 \bmod N$. (In this example we have $u=(N,y)$, $v=z$, and $v'=z'$.) The witness set is $W(N,y,z)=\{ w \in \mathbf{Z}_N^* \mid z \equiv yw^2 \bmod N \}$. The equivalence $w' \equiv wr \bmod N$ shows that the mapping $\rho$ is witness-respecting. This language is sampleable.

By taking $y$=1, which is a quadratic residue for any modulus $N$, we see that ***QR*** is (isomorphic to) a witness-respecting sub-language of ***SameQ***.

Example 4: Consider the language

$$L = \{ (p,a,x) \mid p \text{ prime}, a,x \in \mathbf{Z}_p^* \text{ such that } \exists w \in [1 \ldots p-1],\ a^w \equiv x \bmod p \}.$$

The exponent $w$ is the witness that $x$ belongs to the multiplicative subgroup of $\mathbf{Z}_p^*$ generated by the element $a$.[4] The mapping defined by $\rho(p,a,x,r)=(p,a,x')$, where $r \in [1 \ldots p-1]$ is chosen at random and $x'=xa^r \bmod p$, is a random-self-reduction. Since $x'$ has witness $w'=w+r \bmod p-1$, $\rho$ is witness-respecting.

The following theorem was proved by Tompa and Woll [28].

> **Theorem 1:** If the sampleable NP language $L$ has a witness-respecting random-self-reduction $\rho$, then there is a perfect minimum-knowledge confirming interactive proof-system for $L$; the prover is a probabilistic polynomial-time Turing machine that is given as additional input a witness for the input string.

The theorem is proved by exhibiting an interactive proof-system for $L$. The verifier's input is of the form $i_V=(k,x)$, while the prover's input is of the form $i_P=(k,x,w)$, where $w \in W(x)$.

Repeat $k$ times:

1. $P$ chooses $r \in \{0,1\}^{|x|^c}$ at random, and computes $x'=\rho(x,r)$ and $w' \in W(x')$;
   $P \rightarrow V$: $x'$
2. $V$ chooses $\varepsilon \in \{0,1\}$ at random;
   $V \rightarrow P$: $\varepsilon$
3. $P$ computes: if $\varepsilon$=0 then $z:=r$ else if $\varepsilon$=1 then $z:=w'$ else (if $\varepsilon \notin \{0,1\}$) $P$ HALTs;
   $P \rightarrow V$: $z$
4. $V$ checks that $\rho(x,z)=x'$ if $\varepsilon$=0 or that $z \in W(x')$ if $\varepsilon$=1; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

To see that this is a confirming interactive proof-system, observe that when the input is of the required form and both prover and verifier follow the protocol, the verifier will in fact accept the proof. On the other hand, if the input string $x$ is not in $L$ then there is no witness-string $w$ for $x$. The only way for a

---

[3] I.e. $y$ and $z$ have the same quadratic character modulo every prime dividing $N$.

[4] Using the best known algorithm for primality testing, this language can only be said to be *statistically sampleable*.

cheating prover to convince $V$ that $x \in L$ is by correctly guessing, before each iteration, whether $V$ will choose $\varepsilon=0$ --- in which case the cheater can compute $x' = \rho(x, r)$ as specified, and send $z=r$ in step 3 --- or will choose $\varepsilon=1$ --- in which case the cheater can deceive $V$ by generating a a random instance $x' \in L$ along with a witness $w'$, sending $x'$ in step 1 and $z=w'$ in step 3; the fact that $\rho$ is witness-respecting implies that the cheater is unable to prepare simultaneously for both possibilities. The probability of cheating the verifier by guessing successfully in $k$ consecutive iterations is $2^{-k}$.

To prove that this protocol is minimum-knowledge, let $V^*$ be any communicating Turing machine that interacts with $P$. We have to specify the computation of a Turing machine $S$ that, on input $(k,x)$ with $x \in L$, computes a simulation of the communication tapes of $P$ and $V^*$ on input $(k,x,w)$ and $(k,x)$. Here, the communications transcript of an accepting computation consists of $k$ triples $(x', \varepsilon, z)$ that satisfy the conditions of step 4 above. $S$ proceeds as follows.

Repeat $k$ times:

i. choose $\varepsilon \in \{0,1\}$ at random; if $\varepsilon=0$ then choose $r \in \{0,1\}^{|x|^c}$ at random, and compute $x' = \rho(x, r)$ and $w' \in W(x')$;
else generate a random pair $(x', w')$ with $x' \in L$, $w' \in W(x')$, $|x|=|x'|$

ii. "send" $x'$ to $V^*$

iii. "receive" $\alpha$ from $V^*$

iv. if $\alpha \notin \{0,1\}$ then halt; else
if $(\alpha \neq \varepsilon)$ then reset $V^*$ to its configuration at the beginning of the current iteration and go to step i; else
if $\alpha=\varepsilon=0$ then set $z := r$ else $(\alpha=\varepsilon=1)$ set $z := w'$;
"send" $z$ to $V^*$ and append $(x', \varepsilon, z)$ to the output tape

In each iteration of the loop above, the element $x'$ is uniformly distributed over $L \cap \{0,1\}^{|x|}$, whether $S$ has chosen $\varepsilon=0$ or $\varepsilon=1$. It follows that the event $\alpha=\varepsilon$ occurs with probability exactly 1/2, so that the expected number of times (per iteration) that the loop will be repeated is two. The ensemble $\{ S[k,x] \mid x \in L \}$ is identical to the ensemble $\{ (P,V^*)[k,(x,w),x] \mid x \in L, w \in W(x) \}$, and therefore the protocol above is perfectly minimum-knowledge.

As with the example of *GI* presented in Section 3.1 above, the version of this protocol in which all $k$ iterations are executed in parallel, while it is still an interactive proof-system for $L$, does not appear to be minimum-knowledge for the verifier.

## 5. Number-theoretic examples

### 5.1. Quadratic residues and nonresidues

These were the first examples that motivated the original definition of minimum-knowledge protocol [20].

By the results of Section 4 above, the language of quadratic residues, defined by

$QR = \{ (N,z) \mid N>1, z \text{ a quadratic residue mod} N \}$,

has a confirming interactive proof-system that is perfectly minimum-knowledge. For completeness, we give the protocol below. The prover is a probabilistic polynomial-time Turing machine that receives, in addition to the input string $(N, z)$, a witness $w$ satisfying $w^2 \equiv z \bmod N$.

Repeat $k$ times:

1. $P$ chooses $r \in \mathbf{Z}_N^*$ at random, and computes $z' = r^2 \bmod N$;
   $P \rightarrow V$: $z'$
2. $V$ chooses $\varepsilon \in \{0,1\}$ at random;
   $V \rightarrow P$: $\varepsilon$
3. $P$ computes: if $\varepsilon=0$ then $y := r$ else if $\varepsilon=1$ then $y := wr \bmod N$ else (if $\varepsilon \notin \{0,1\}$) $P$ HALTs;
   $P \rightarrow V$: $y$
4. $V$ checks that $y^2 \equiv z' \bmod N$ if $\varepsilon=0$ or that $y \equiv zz'$ if $\varepsilon=1$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

Just as we defined the language ***1of2-GI***, a variant of the graph isomorphism language, in order to specify a protocol for graph nonisomorphism, we now define the language

$$\textit{1of2-Q} = \{ (N, y_1, y_2, x) \mid N > 1, \text{ either } (N, y_1, x) \text{ or } (N, y_2, x) \in \textit{SameQ} \}.$$

A simple adaptation of the interactive proof-system for ***QR*** (or for ***SameQ***) --- analogous to the adaptation of the protocol for ***GI*** to give a protocol for ***1of2-GI*** --- results in a confirming interactive proof-system for ***1of2-Q*** that is perfectly minimum-knowledge for the verifier. Similarly, we could give a confirming interactive proof-system for the language

$$\textit{1of3-Q} = \{ (N, y_1, y_2, y_3, x) \mid N > 1, (N, y_1, x) \text{ or } (N, y_2, x) \text{ or } (N, y_3, x) \in \textit{SameQ} \}$$

that is perfectly minimum-knowledge for the verifier.

Next, we give a statistically minimum-knowledge confirming interactive proof-system for the language of quadratic nonresidues, defined by

$$\textit{QNR} = \{ (N, y) \mid N > 1, y \text{ a quadratic nonresidue mod } N \}.$$

The prover and the verifier both have inputs of the form $(k, N, y)$. As was true in the case of ***GNI***, the prover must possess some additional computational power. Namely, the prover must be able to distinguish between quadratic residues and quadratic nonresidues modulo $N$; for example, it can receive, as additional input, $N$'s prime factorization.

We begin with an interactive proof-system that is not minimum-knowledge for the verifier.

Repeat $k$ times:

1. $V$ chooses $r \in \mathbf{Z}_N^*$ and $\varepsilon \in \{0,1\}$ at random, and computes $x := y^{\varepsilon} r^2 \bmod N$ (so that $\mathrm{RES}_N(x) = \varepsilon$);
   $V \rightarrow P$: $x$
2. $P$ computes $\varepsilon' := \mathrm{RES}_N(x)$;
   $P \rightarrow V$: $\varepsilon'$
3. $V$ checks that $\varepsilon' = \varepsilon$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

In step 2, the prover uses its power to distinguish between residues and nonresidues mod $N$. If $y$ is indeed a quadratic nonresidue, and both prover and verifier follow the protocol, then the verifier accepts the proof. On the other hand, if $y$ is a quadratic residue, then the numbers $x$ sent in step 1 of each round form a list of $k$ randomly chosen quadratic residues mod $N$. Even a prover with unlimited computational power would be unable to tell, in each round, whether $x$ was computed after a choice of $\varepsilon=0$ or of $\varepsilon=1$. The only way a cheating prover can convince the verifier to accept the proof incorrectly is by guessing, $k$ consecutive times, the verifier's coin-flip $\varepsilon$; thus the probability that $V$ accepts the cheater's proof is at most $2^{-k}$. Hence, the above protocol is indeed a confirming interactive proof-system for ***QNR***.

But the protocol is not minimum-knowledge for $V$, since in step 1 the verifier may ask about a number $x$ whose quadratic character he does not know, so that $P$'s answer in step 2 gives him additional knowledge. However, before $P$'s answer $V$ can now use a variation of the interactive proof-system for ***1of2-Q*** in order to prove that $x$ was constructed as specified, i.e. either as a random quadratic residue or as a random element of the same quadratic character as $y$. As in the protocol for graph isomorphism, we will be able to use a parallelized version of this sub-protocol step. The full protocol is as follows.

Repeat $k$ times:

1. $V$ chooses $r\in \mathbf{Z}_N^*$ and $\varepsilon\in\{0,1\}$ at random, and computes $x:=y^{\varepsilon}r^2 \bmod N$ (so that $\mathrm{RES}_N(x)=\varepsilon$);
   $V\rightarrow P$: $x$

   1.1 for $i=1\ldots k$, $V$ chooses $\varepsilon_i\in\{0,1\}$ and $r_{i0}, r_{i1}\in \mathbf{Z}_N^*$ at random and computes $x_{i0}:=r_{i0}^2 y^{\varepsilon_i}$ and $x_{i1}:=r_{i1}^2 y^{1-\varepsilon_i} \bmod N$;
   $V\rightarrow P$: $(x_{i0}, x_{i1})_{i=1\ldots k}$

   1.2 $P$ chooses $I\subseteq\{1,\ldots,k\}$ at random;
   $P\rightarrow V$: $I$

   1.3 for each $j\notin I$, $V$ computes $e_j=\varepsilon\oplus\varepsilon_j$ and $w_j:=y^{\varepsilon}r r_{j\varepsilon_j} \bmod N$ ($w_j$ is a witness that $\mathrm{RES}_N(x_{j\varepsilon_j})=\mathrm{RES}_N(x)$);
   $V\rightarrow P$: $(\varepsilon_i, r_{i0}, r_{i1})_{i\in I}, (w_j)_{j\notin I}$

   1.4 $P$ checks that $x_{i0}\equiv r_{i0}^2 y^{\varepsilon_i}$ and $x_{i1}\equiv r_{i1}^2 y^{1-\varepsilon_i} \bmod N$ for $i\in I$, and that $w_j^2\equiv x x_{j\varepsilon_j} \bmod N$ (for at least one choice of $e_j\in\{0,1\}$) for $j\notin I$;
   if any of these conditions does not hold, then $P$ HALTs the protocol (detecting cheating)

2. $P$ computes $\varepsilon':=\mathrm{RES}_N(x)$;
   $P\rightarrow V$: $\varepsilon'$

3. $V$ checks that $\varepsilon'=\varepsilon$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

We omit the proof that this protocol is a confirming interactive proof-system for ***QNR*** that is statistically minimum-knowledge for the verifier, since it is exactly analogous to the proof given in section 3.2 for the graph nonisomorphism protocol.

The fact that the number $w_j$ computed in step 1.3 is a witness that $\mathrm{RES}_N(x)=\mathrm{RES}_N(x_{j\varepsilon_j})$ --- in fact, a

witness that $x$ and $x_{je_j}$ have the same quadratic character modulo every prime dividing $N$ --- follows from the calculation:

$$w_j^2 \equiv (y^{\varepsilon} r r_{je_j})^2 \equiv (y^{\varepsilon} r^2)(y^{\varepsilon} r_{je_j}^2) \equiv x x_{je_j} \bmod N, \text{ since } y^{\varepsilon} \cdot r_{je_j}^2 \equiv \begin{cases} y^{\varepsilon} \cdot r_{j0}^2 \equiv y^{\varepsilon_j} \cdot r_{j0}^2 \equiv x_{j0} & \text{if } e_j = 0, \\ y^{\varepsilon} \cdot r_{j1}^2 \equiv y^{1-\varepsilon_j} \cdot r_{j1}^2 \equiv x_{j1} & \text{if } e_j = 1. \end{cases}$$

As with the protocol for ***GNI***, if all $k$ iterations are performed in parallel, then the simulator can also perform in parallel all $k$ iterations of the simulating procedure; thus the parallel version of this protocol is also statistically minimum-knowledge for the verifier.

In the original presentation of this protocol, the verifier uses a slightly different sub-protocol refinement (of similar cost) in order to convince the prover that each number $x$ sent in step 1 is correctly constructed [20]. The version above was chosen for ease of explanation.

Of course it may turn out that there is a polynomial-time (probabilistic or deterministic) algorithm for distinguishing residues from nonresidues mod $N$, in which case the protocols described above are only trivially minimum-knowledge. Another minimum-knowledge protocol involving quadratic residues and nonresidues is described in section 5.5 below.

## 5.2. Blum integers

Our next example concerns the set of integers with prime factorization $N = \prod_{i=1}^{l} p_i^{e_i}$ 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 language consisting of these integers. If $N \equiv 3 \bmod 4$, an easy condition to check, then clearly $N$ is in this language. However, if $N \equiv 1 \bmod 4$, it is not clear how to tell whether or not its prime factorization is of this form. There are two 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*** if and only if there exists a quadratic residue mod $N$ that has two square roots with different Jacobi symbols [4].

The following protocol, due to Blum, may have been the first protocol in the literature that satisfied the definition of minimum-knowledge [4]. It is a perfectly minimum-knowledge confirming interactive proof-system for ***BL***. The prover and the verifier share the input $(k, N)$; the additional input for $P$ is the prime factorization of $N$ (or, equivalently, any means of computing square roots mod $N$).

Repeat $k$ times:

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

*V* ACCEPTs the proof

The correctness of this protocol --- i.e. the fact that it is a proof-system --- depends on the alternate characterizations of membership in ***BL***. If $N \in BL$, then each quadratic residue $r$ sent by $P$ 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 $\sigma$ $V$ chooses, $P$ can respond with a square root of the appropriate sign. On the other hand, if $N \notin BL$ then no quadratic residue mod $N$ has two square roots with Jacobi symbols of opposite sign. In this case, it is highly probable that there is some $i$ for which $P$ will be unable to send an appropriate $s_i$, and $V$ will halt the protocol. The only way for a cheating $P^*$ to convince $V$ that $N \in BL$ (by sending the appropriate elements $s_i$) is by guessing 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 confirming interactive proof-system for ***BL***.

To prove the minimum-knowledge property, we have to specify the computation of a simulating Turing machine $S$. (Once again, as discussed in Section 3.1 above, we can assume that $V^*$ behaves "reasonably".) In this case, $S$ must simulate a communications ensemble that consists of triples $(r, \sigma, s)$ satisfying the conditions implicitly defined by the specification of the protocol.

On input $(k,N)$, $S$ repeats the following loop $k$ times:

1. choose $s \in \mathbf{Z}_N^*$ at random
2. $r := s^2 \bmod N$
3. "send" $r$ to $V^*$, and "receive" $\sigma$ in return
4. if $\left(\frac{s}{N}\right) \neq \sigma$ then re-set the random read-head of $V^*$ and go back to step 1;
   else output $(r, \sigma, s)$

Assume now that $N \in BL$. 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 random triples produced by $S$ do satisfy the required conditions, and so the two ensembles are indeed identical. This completes the proof that the protocol is perfectly minimum-knowledge.

This is another example in which it is not clear how to perform the simulation required for the minimum-knowledge proof, if all $k$ iterations of the protocol are executed in parallel.

## 5.3. Blum's coin-flip

There are many two-party protocols that require random bits. Each of the parties, $A$ and $B$, has to be sure that the other cannot bias these bits. They can do this by following a protocol due to Blum [4].

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

$A$ and $B$ generate a random bit $\beta$:

1. $B$ chooses $u \in \mathbf{Z}_N^*$ at random, computes $v = u^2 \bmod N$;
   $B \rightarrow A$: $v$
2. $A$ chooses $\sigma = +1$ or $-1$ at random, a guess for the value of $\left(\frac{u}{N}\right)$;

$A \rightarrow B$: $\sigma$

3. $B \rightarrow A$: $u$

4. if $\sigma = \left(\frac{u}{N}\right)$ then $\beta := 1$ else $\beta := 0$

The protocol indeed achieves its task: The first alternate characterization of ***BL*** (Section 5.2) implies that, no matter what its computational power, $A$ cannot bias the bit produced, since $A$ cannot guess the Jacobi symbol of the square root of $v$ chosen by $B$ and be correct with probability greater than 1/2.

If $B$ picks $u$ at random, then the bit $\beta$ chosen by this protocol is random. 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 for $B$. The reason is that $A$'s only task is to transmit a guess, $\sigma = +1$ or $-1$, for a sign, a task that can be performed easily by a simulator interacting with the verifier $B^*$. We show how to formalize this argument, when the coin-flip is used as a sub-protocol in another protocol, in the next section.

## 5.4. Number of prime factors

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

Given an integer from the set

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

the protocol below is a perfectly minimum-knowledge confirming interactive proof-system for the language $L \cap \{ N \mid \nu(N) = 2 \}$. Note that it is easy to determine that a given integer $N$ is not a prime power.

Let us use $\mathbf{Z}_N^*(\pm 1)$ to denote the set of elements of $\mathbf{Z}_N^*$ with Jacobi symbol $\pm 1$ (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. $P$ and $V$ jointly pick random elements of $\mathbf{Z}_N^*(+1)$. If $P$ can show that about half of them are residues (e.g. by producing their square roots mod $N$), then $V$ should be convinced that $\nu(N) \leq 2$. Since $N$ is not a prime power, $\nu(N)$ must be equal to 2.

In order to pick a list of random elements of $\mathbf{Z}_N^*(+1)$, $P$ and $V$ follow Blum's coin-flip protocol, which requires that $N \in \mathbf{BL}$ and $N \equiv 1 \bmod 4$.

1. $P$ and $V$ use Blum's coin-flip protocol to generate $k$ random elements $r_1, \ldots, r_k \in \mathbf{Z}_N^*(+1)$:
   $i := 0$;
   do until $i = k$:
   a. generating it bit by bit using Blum's coin-flip protocol, $P$ and $V$ choose a number $a$, $0 < a < N$
   b. if g.c.d.$(a, N) \neq 1$ (which will happen very rarely) then HALT the protocol
   c. if $\left(\frac{a}{N}\right) = +1$ then $\{ i := i+1 ; r_i := a \}$
2. For each $i = 1, \ldots, k$ such that $r_i$ is a quadratic residue, $P$ computes $s_i$ such that $r_i \equiv s_i^2 \bmod N$;

$P \rightarrow V$: $(i, s_i)$

3. $V$ checks that at least 3/8 of the $r_i$ are quadratic residues; if so it ACCEPTs the proof, and otherwise it REJECTs the proof

During the above protocol, $P$ and $V$ together choose random elements of $Z_N^*(+1)$. Since they do this by means of Blum's coin-flip protocol, and no Turing machine $P^*$ 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 $Z_N^*(+1)$, 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 $V$ accepts the proof if the frequency $F_k(N) \geq 3/8$. 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 $V$ does not accept is, by Bernstein's law of large numbers,

$$\text{Prob}\{ F_k(N) < 3/8 \} \leq \text{Prob}\{ |F_k(N) - 1/2| \geq 1/8 \} \leq 2\, e^{-k(1/8)^2},$$

which is exponentially vanishing in $k$. 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 $V$ will incorrectly accept the proof (when interacting with a cheating $P^*$) is

$$\text{Prob}\{ F_k(N) \geq 3/8 \} \leq \text{Prob}\{ |F_k(N) - 1/4| \geq 1/8 \} \leq 2\, e^{-k(1/8)^2}.$$

To prove the minimum-knowledge property, we have to specify the computation of a simulating Turing machine $S$. (Once again, as discussed in Section 3.1 above, we can assume that $V^*$ behaves "reasonably".) The communications ensemble $(P, V^*)[k, N]$ that $S$ must simulate includes a sequence of Blum coin-flips, so we begin by showing that Blum's coin-flip protocol is perfectly minimum-knowledge for $V$. To prove this, we must specify the computation of a probabilistic polynomial-time Turing machine $S_{\text{coin}}$ whose output, on input $N$ (satisfying $N \in \boldsymbol{BL}$ and $N \equiv 1 \bmod 4$), is a simulation of the communications ensemble $(P, V^*)[N]$, namely a triple $(v, \sigma, u)$ that encodes a bit as described in Section 5.3 above.

Given a random bit $\beta$ (the result of a fair coin flip), $S_{\text{coin}}$ proceeds as follows:

a. $S_{\text{coin}}$ executes the protocol with $V^*$: letting $V^*$ "send" $v$, simulating $P$'s action in step 2 by flipping a coin to choose $\sigma$ and "sending" it to $V^*$, and then letting $V^*$ "send" $u$

b. if the bit generated by this execution is $\beta$, then $S_{\text{coin}}$ outputs the triple $(v, \sigma, u)$

c. otherwise, $S_{\text{coin}}$ re-sets $V^*$'s random read-head, goes back to step 2, "sends" $-\sigma$ instead of $\sigma$ to $V^*$, and lets $V^*$ "send" $u'$; now $S_{\text{coin}}$ outputs the triple $(v, -\sigma, u')$

Note that if $V^*$ 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 $\beta$ and this has no effect on the output distribution (since $V^*$, when interacting with $P$, can decide to send either $u$ or $-u$). On the other hand, if they have opposite Jacobi symbols mod $N$, then the outcome bit $1-\beta$ has been determined by $V^*$ and not chosen at random. As noted above, this can only happen if $V^*$ 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 $P$ or serving $S_{\text{coin}}$ as a subroutine.

Whether the output triple was generated by $S_{coin}$ in step b or step c, the distribution of its possible values (and thus the probability distribution of the bit it encodes) is identical to that of $(P, V^*)[N]$, and thus the coin-flip protocol is perfectly minimum knowledge for $V$.

Next we describe the simulation by $S$ of the protocol above. The communications ensemble $(P, V^*)[N]$ that $S$ must simulate begins with a sequence of Blum coin-flips, which are used to generate random elements of $\mathbf{Z}_N^*$. The simulation of these coin-flips can be performed as just described; the difficulty for $S$, a polynomial-time machine that may not be able to factor $N$, is that those elements that are quadratic residues must be randomly generated along with their square roots. Therefore $S$ uses fair coin-flips in order to pick the quadratic character of the elements of $\mathbf{Z}_N^*$ that are being chosen. An element of $\mathbf{Z}_N^*(-1)$ should be chosen with probability 1/2, a quadratic nonresidue in $\mathbf{Z}_N^*(+1)$ should be chosen with probability 1/4, and a quadratic residue (which is necessarily an element of $\mathbf{Z}_N^*(+1)$) should be chosen with probability 1/4.

Given as input an integer $N$ (satisfying $N \in \boldsymbol{BL}$, $N \equiv 1 \bmod 4$, $\nu(N) > 1$), $S$ 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 will happen very rarely) then FLAG the number $a$, adjoin it to $\Lambda$, and go to step 3;
   else:

   choose a random bit $\varepsilon$ (to decide the Jacobi symbol of the next element generated);
   if $\varepsilon=0$ 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 $\varepsilon_i$ (to decide whether the next element generated should be a quadratic residue);
   if $\varepsilon_i=0$ 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 resulting list $\Lambda$)
   for each bit $\beta$ in the representation of each number in $\Lambda$:

   follow the procedure above for $S_{coin}$ (using $V^*$ as a subroutine), recording the numbers $u$ (and possibly $u'$) "sent" by $V^*$;

   if the outcome of the coin-flip simulation is indeed $\beta$, then continue with the next bit in $\Lambda$;

   otherwise $V^*$ has "forced" the complementary outcome $1-\beta$ by "sending" $u$ and $u'$ with $\left(\frac{u}{N}\right) \neq \left(\frac{u'}{N}\right)$, 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 $V^*$ (as originally specified, without re-

setting the random read-head of $V^*$) 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 $V^*$ has not "forced" the outcome of any of the coin-flip simulations of step 3, then for each $r_i$ in $\Lambda$ such that $\varepsilon_i=0$ output $(i, s_i)$;
   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 output $(i, s_i)$

$S$ generates lists of elements of $\mathbf{Z}_N$ with the same distribution as do $P$ and $V^*$, so that the communications ensemble $(P, V^*)[k, N]$ and the output ensemble $S[k, N]$ are identical.

Note that the above protocol can be adapted slightly so that instead of being an interactive proof-system for the language of suitable integers $N$ satisfying $\nu(N)=2$, it would be a protocol for communicating the *value* of $\nu(N)$. In the last step of the protocol, $V$ counts the proportion of the randomly chosen elements of $\mathbf{Z}_N^*(+1)$ that are shown to be quadratic residues. If this proportion is at least 3/8, then $V$ accepts a proof that $\nu(N)=2$; if it is between 3/16 and 3/8, then $V$ accepts a proof that $\nu(N)=3$; and so on.

### 5.5. Quadratic residues and nonresidues, result-indistinguishably

Before describing the next number-theoretic example, we give a general definition. We call a deciding interactive proof-system $(P, V)$ for a language $L$ (with input language $I$) "result-indistinguishable" if an eavesdropper that has access to the communications of $P$ and $V$ gains no knowledge. More formally, the protocol is *result-indistinguishable* if there exists a probabilistic polynomial-time Turing machine $S$ such that the ensembles $\{ S[k,x] \mid x \in I \}$ and $\{ (P,V)[k,x] \mid x \in I \}$ are indistinguishable. If the ensembles are statistically indistinguishable, we say that the proof-system is *statistically result-indistinguishable*; if they are exactly identical, we say that the proof-system is *perfectly result-indistinguishable.*

Observe that unlike the machine $S$ in the definition of the minimum-knowledge property, this machine $S$ does not have access to an oracle for the result of the computation, i.e. for the membership bit ($x \in L$); in other words, $S$ can simulate the communications of $P$ and $V$ on input $(k, x)$, regardless of whether or not $x \in L$ (even if recognizing membership in $L$ 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 $\{ (P,V)[k,x] \mid x \in I \}$.

In this section, we present a protocol that deals with Blum integers of a special form, namely the set $N = \{ N : N \in \boldsymbol{BL}, N \equiv 1 \bmod 4, \nu(N)=2 \}$. It is not hard to see that this set may equivalently be defined as $N = \{ p^i q^j : p \neq q \text{ prime}, i,j \geq 1, p^i \equiv q^j \equiv 3 \bmod 4 \}$. The protocols presented in Sections 5.3 and 5.4 may be concatenated so as to give a perfectly minimum-knowledge confirming interactive proof-system for $N$.

We define the languages

$$I = \{ (N,z) : N \in \boldsymbol{N}, z \in \mathbf{Z}_N^*, \left(\frac{z}{N}\right) = +1 \} \text{ and } L = \{ (N,z) \in I : z \text{ a quadratic residue mod } N \}.$$

Taking $I$ as the set of inputs, this section gives a deciding interactive proof-system for $L$. Observe that 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 $N$, i.e. according to the value $\mathrm{RES}_N(z)$. Thus, instead of regarding the protocol below as deciding membership, it can be useful to look at it as transmitting the result of the computation of the value $\mathrm{RES}_N(z)$.

As in the case of the protocol for the language $QNR$, the prover must be able to distinguish between quadratic residues and quadratic nonresidues modulo $N$; for example, it can receive, as additional input, $N$'s prime factorization. The verifier is simply a probabilistic polynomial-time machine.

Let $y \equiv -1 \bmod N$. Everything that follows holds for any nonresidue $y \in \mathbf{Z}_N^*$ which has Jacobi symbol $+1$. As long as $N \in \boldsymbol{BL}$ and $N \equiv 1 \bmod 4$, we can take $y = -1$.

This 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 $N$.

Repeat $k$ times:

1. $V$ 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: $x := zr^2 \bmod N$
   $V \rightarrow P$: $x$
2. $P$ computes $\varepsilon = \mathrm{RES}_N(x)$;
   $P \rightarrow V$: $\varepsilon$
3. $V$ checks that if $c = 1$ then $\varepsilon = 1$, if $c = 2$ then $\varepsilon = 0$, and if $c = 3$ then $\varepsilon$ is consistent with any previous iterations for which $c$ was = 3; if not then $V$ HALTs the protocol

$V$ ACCEPTs the proof that $\mathrm{RES}_N(z)$ = the consistent value of $\varepsilon$ 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 $P$ acts as specified, then when the protocol finishes $V$ is convinced that $z$ is a residue. The only way that a cheating $P^*$ can convince $V$ that $z$ is not a residue is by correctly guessing, among all iterations during which $V$ 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 $2^{-ck}$. Because $c$ is very likely to be close to 2/3, a simple calculation using Bernstein's law of large numbers shows that the probability of successful cheating is exponentially vanishing in $k$. Similarly if $z$ is a quadratic nonresidue. Hence the above version is a deciding interactive proof-system for $L$.

However, the protocol is not minimum-knowledge for the verifier, since $V$ can ask about numbers

$x \in Z_N^*$ whose quadratic character it does not know. In order to make the protocol minimum-knowledge, we can add, as a refinement sub-protocol to step 1, a (parallelized) variant of the interactive proof-system for *1of3-Q* sketched above. $V$ uses this sub-protocol to prove to $P$ that $x$ has been chosen in one of the three specified ways, without transmitting to him any knowledge about which of the three ways. The resulting protocol is a statistically minimum-knowledge deciding interactive proof-system for $L$. We omit the proof, since it is exactly analogous to the proof given in section 3.2 for the graph nonisomorphism protocol.

If all $k$ iterations of the protocol are performed in parallel, then the simulator can also perform in parallel all $k$ iterations of the simulating procedure; thus the parallel version of this protocol is also statistically minimum-knowledge for the verifier.

The above protocol is not result-indistinguishable, however, because 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 $\varepsilon$ sent by $P$ in step 2 of each iteration.

But a simple modification of the protocol does hide the result from an eavesdropper. The only change is that at the beginning, $P$ 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 $\varepsilon$ to be sent to $V$ in step 2 of each iteration throughout the protocol. $R(x)$ can be regarded as an encoding, chosen at random, of $\mathrm{RES}_N(x)$. In step 3, $V$ checks for consistency in the obvious way: $V$ should receive the same bit $\varepsilon$ in all case-1 iterations and the complementary bit in all case-2 iterations; $V$ should receive a consistent bit $\varepsilon$ in all case-3 iterations, and its value indicates to $V$ whether or not $z$ is a quadratic residue. As before, if in step 3 of any iteration $V$ finds that the value of $\varepsilon$ is not consistent then $V$ halts the protocol, detecting cheating.

With this modification, the protocol is still --- arguing as above --- a minimum-knowledge deciding 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 2 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. In order to prove that the protocol is result-indistinguishable, we must specify the computation of a probabilistic Turing machine $S$ that simulates the communications ensemble $\{(P,V)[k,N,z]\}$. (Recall that $S$ does not have access to an oracle for $\mathrm{RES}_N(z)$.) $S$ 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 $S$ simulates the specified computations of $P$ and $V$, except for the following changes. In (simulated) step 1, $S$ chooses $x := zr^2 \bmod N$ with probability 2/3 and $x := yzr^2 \bmod N$ with probability 1/3. In (simulated) step 2, $S$ outputs $\varepsilon = R(z)$ if $x = zr^2$ and $\varepsilon = 1-R(z)$ if $x = yzr^2$. We omit here the details of the simulation by $S$ of the *1of3-Q* sub-protocol refinement of step 1. The output ensemble $S[k,N,z]$ is identical to the communications ensemble $(P,V)[k,N,z]$, and therefore the protocol is perfectly result-indistinguishable.

## 6. Protocols based on cryptographic assumptions

In this section we discuss a number of protocols that can be shown to be minimum-knowledge, but only under the assumption that a particular computational problem is intractable. Our "cryptographic" assumption (so called because much of the recent work in cryptography depends on assumptions of this

nature) is that there exist *one-way functions*, functions that are easy to compute but intractably hard to invert. (The weakest complexity-theoretic assumption that suffices for the theorem of this section is that of Levin [25]; see also [8, 30, 24, 9, 6, 2, 7, 29].)

Specifically, we assume the existence of a sequence $\{f_k\}$ of probabilistic encryption functions. The $k^{\text{th}}$ of these functions may be used to encrypt the bit $\beta$ by choosing a bit-string $r$ at random in the appropriate domain[5] and computing the encryption $e := f_k(\beta, r)$. Given $e$, no polynomial-time machine has any computational advantage in guessing the value of the bit $\beta$. To *open* the encryption $e$ is to reveal the string $r$ (and thus the encrypted bit $\beta$) that was used to compute it. It is then a simple matter to check that indeed $f_k(\beta, r) = e$.

## 6.1. Minimum-knowledge interactive proof-systems for languages in NP

Goldreich, Micali, and Wigderson have shown that, under the assumption that one-way functions exist, every language in NP has a minimum-knowledge confirming interactive proof-system [17, 5].

We begin by exhibiting a minimum-knowledge confirming interactive proof-system for the (NP-complete) language of graphs containing a Hamiltonian cycle. Prover and verifier share the input $(k, G)$; the prover has as additional input a Hamiltonian cycle in $G$. Let the nodes of $G$ be $\{1, \ldots, n\}$, and let the given Hamiltonian cycle pass through the nodes $[1, i_1, i_2, \ldots, i_{n-1}, 1]$, in that order. We assume that both the prover and the verifier have a sequence $\{f_k\}$ of probabilistic encryption functions, indexed by the security parameter $k$.

Repeat $k$ times:

1. $P$ chooses a permutation $\pi: \{1, \ldots, n\} \to \{1, \ldots, n\}$ at random; let $[1, j_1, j_2, \ldots, j_{n-1}, 1]$ be the corresponding Hamiltonian cycle in the permuted graph.
   For each $i, j \in [1 \ldots n]$ $P$ chooses $r_{ij}$ at random in the domain of $f_k$ and computes $e_{ij}$, a probabilistic encryption of either 1 or 0, according to whether $(\pi^{-1}(i), \pi^{-1}(j))$ is or is not an edge of $G$;
   $P \to V$: $(e_{ij})_{i,j \in [1 \ldots n]}$
2. $V$ chooses $\varepsilon \in \{0, 1\}$ at random;
   $V \to P$: $\varepsilon$
3. if $\varepsilon=0$ then $P \to V$: $\pi$, $(r_{ij})_{i,j \in [1 \ldots n]}$
   else if $\varepsilon=1$ then $P \to V$: $[j_1, j_2, \ldots, j_{n-1}]$, $[r_{1j_1}, r_{j_1 j_2}, \ldots, r_{j_{n-1}1}]$
4. $V$ checks that the values $e_{ij}$ correctly encode the permuted graph if $\varepsilon=0$, or that the sequence $[e_{1j_1}, e_{j_1 j_2}, \ldots, e_{j_{n-1}1}]$ correctly encodes a Hamiltonian cycle (in the permuted graph) if $\varepsilon=1$; if not, then $V$ REJECTs the proof

$V$ ACCEPTs the proof

If the inputs to $P$ and $V$ are as specified, and both prover and verifier follow the protocol, then $V$ accepts the proof. On the other hand, if the input graph $G$ is not Hamiltonian, then the only way for a cheating prover to convince the verifier that it is Hamiltonian is by guessing, in each iteration, whether

---

[5] $r$ will be of length polynomial in $k$.

the verifier will send him a 0 or a 1 in step 2. If $V$ is about to send him $\varepsilon$=0, then the prover can satisfy $V$ by following the instuctions of step 1 above, sending a probabilistic encryption of the adjacency matrix of a random isomorphic copy of $G$; if $V$ is about to send him $\varepsilon$=1, then the prover can satisfy him by choosing a sequence of indices $[1, j_1, j_2, \ldots, j_{n-1}, 1]$ at random and then sending him values $(e_{ij})_{i,j \in [1 \ldots n]}$ in which $e_{1j_1}, e_{j_1 j_2}, \ldots, e_{j_{n-1}1}$ are all encryptions of 1, and the other $n^2-n$ entries are all encryptions of 0. If the graph does not have a Hamiltonian cycle, then the cheater cannot prepare simultaneously for both possibilities. Thus the probability that any communicating Turing machine $P^*$ will convince $V$ to accept the proof incorrectly is bounded by $2^{-k}$, and the above protocol is indeed a confirming interactive proof-system for the language of Hamiltonian graphs.

The proof that the above protocol is minimum-knowledge for the verifier is very similar to the proofs for *GI* and for sampleable random-self-reducible languages. In each simulated round, the simulator $S$ guesses $\varepsilon$, the bit to be "sent" in step 2 by $V^*$, in order to decide whether to compute a good encryption of (a permutation of) the input graph or to follow the cheater's second option and prepare a random encoding which contains an encrypted Hamiltonian cycle. If $V^*$ "sends" the wrong bit, then $S$ resets $V^*$ and begins the iteration again. By the hypothesized properties of the probabilistic encryption function $f_k$, the polynomial-time machine $V^*$ can have no computational advantage in guessing whether $S$ has predicted $\varepsilon$=0 or $\varepsilon$=1, given the $n^2$ values $e_{ij}$ "sent" by $S$. Hence the simulator guesses the wrong bit with probability (computationally indistinguishable from) 1/2, so that the expected number of times per iteration that the loop will be repeated is two. Again by the properties of the encryption function, the simulator's output ensemble and the communications ensemble are indistinguishable. (However, they are not statistically indistinguishable, because that would require a much stronger assumption on the encryption function.)

Another similarity with the protocol for *GI* is that the version of this protocol in which all $k$ iterations are executed in parallel, while it is still an interactive proof-system for the language of graphs containing a Hamiltonian cycle, does not appear to be minimum-knowledge for the verifier.

We have just seen that one particular NP-complete language has a minimum-knowledge confirming interactive proof-system. In order to prove the following theorem, we need only note in addition that the polynomial transformations between NP languages map witness strings to witness strings.

> **Theorem 2:** If there exists a secure family of probabilistic encryption functions, then every language in NP has a minimum-knowledge confirming interactive proof-system in which both the prover and the verifier are probabilistic polynomial-time Turing machines, and the prover receives as additional input a witness that the input string is indeed in the language.

Another way interactively to prove membership in an NP language is by means of minimum-knowledge circuit simulation, as follows. Every NP language $L$ has a polynomial-time computable predicate $p_L$ so that

$$L = \{ x \mid \exists w, |w| \text{polynomial in} |x|, p_L(x,w)=1 \}.$$

The idea is that the prover should interactively lead the verifier through a minimum-knowledge simulation of the circuit that computes $p_L$ --- a simulation that proves to the verifier that $p_L(x,w)$=1 for

some witness string $w$, without revealing any knowledge of the value of $w$. Brassard and Crepeau have shown how to do this by using the properties of quadratic residues (and assuming the intractability of distinguishing between residues and nonresidues modulo an integer $N$ with unknown factorization) [10, 11]. Yung and Impagliazzo have shown how to do this, using any one-way function [32, 23].

## 7. Applications for cryptographic computation

The fact that membership in any NP language can be proved interactively in a minimum-knowledge fashion has important consequences for the design of cryptographic protocols [17]. Assume, for example, that at a certain point in the course of executing a protocol, after receiving a string of messages $x$, a party $P$ is supposed to flip several coins to obtain a random bit-string $r$, compute a value $y=f(x,r)$, and then send this value as a message (perhaps keeping $r$ secret). Even though $f$ may be hard to invert, as long as $f$ is polynomial-time computable, the set

$$L = \{ y \mid \exists r, f(x,r) = y \}$$

of next messages which are legal for $P$ at this point in the protocol, given the history of messages sent so far, is an NP language. Thus $P$'s message $y$ can be accompanied by an interactive proof that it is an appropriate one. If $P$, not following its program, sends an illegal message $y' \notin L$, then (with overwhelming probability) it will be unable to prove to the other parties that $y' \in L$.

Thus any protocol $M$ for which all required computations (such as $f$ in the above example) are publicly specified may be transformed into a *validated* protocol $M'$ in which every message is accompanied by an interactive minimum-knowledge proof to all parties that it is indeed a correct message. The resulting protocol will have the property that a Byzantine user --- one who tries to send an erroneous message --- is almost certain to be caught when he tries to cheat. The transformed protocol is polynomial in all relevant parameters, and the transformation from $M$ to $M'$ is by means of a polynomial-time algorithm. Studying different models of faulty behavior in multi-party cryptographic protocols, researchers have recently described several different fault-recovery procedures that can be invoked when an illegal message is detected [17, 18, 16].

## Acknowledgements

It has been a great pleasure to work together with Zvi Galil and Moti Yung, with whom I did some of the research described in this paper. I would also like to thank Joan Feigenbaum for many helpful discussions of this work, and for her critical reading of an earlier draft.

# References


[1] M. Abadi, J. Feigenbaum, and J. Kilian.
On hiding information from an oracle.
In *Proc. 19th STOC*. ACM, 1987.
To appear.

[2] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr.
RSA/Rabin bits are 1/2 + 1/poly(log *N*) secure.
In *Proc. 25th FOCS*, pages 449-457. IEEE, 1984.

[3] D. Angluin and D. Lichtenstein.
*Provable security of cryptosystems: a survey.*
Technical Report YALEU/DCS/TR-288, Yale University, October, 1983.

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

[5] M. Blum.
How to prove a theorem so no one else can claim it.
In *Proc. of the International Congress of Mathematicians 1986*. 1987.

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

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

[8] M. Blum and S. Micali.
How to generate cryptographically strong sequences of pseudo-random bits.
*SIAM J. Comput.* 13(4):850-864, Nov., 1984.

[9] R.B. Boppana and R. Hirschfeld.
Pseudorandom generators and complexity classes.
*Advances in Computer Research. Volume on Randomness and Computation.*
JAI Press, 1987.
To appear.

[10] G. Brassard and C. Crepeau.
Zero-knowledge simulation of Boolean circuits.
In *Proceedings of Crypto '86*. 1987.

[11] G. Brassard and C. Crepeau.
Non-transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond.
In *Proc. 27th FOCS*, pages 188-195. IEEE, 1986.

[12] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch.
Verifiable secret sharing and achieving simultaneity in the presence of faults.
In *Proc. 26th FOCS*, pages 383-395. IEEE, 1985.

[13] R.A. DeMillo, N. Lynch, and M. Merritt.
Cryptographic protocols.
In *Proc. 14th STOC*, pages 383-400. ACM, 1982.

[14] R. Fagin, J. Halpern, and M. Vardi.
A model-theoretic analysis of knowledge: preliminary report.
In *Proc. 25th FOCS*, pages 268-278. IEEE, 1984.

[15] Z. Galil, S. Haber, and M. Yung.
A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems.
In *Proc. 26th FOCS*, pages 360-371. IEEE, 1985.

[16] Z. Galil, S. Haber, and M. Yung.
Cryptographic computation: secure fault-tolerant protocols in the public-key model.
1987.
Extended abstract.

[17] O. Goldreich, S. Micali, and A. Wigderson.
Proofs that yield nothing but their validity and a methodology of cryptographic protocol design.
In *Proc. 27th FOCS*, pages 174-187. IEEE, 1986.

[18] O. Goldreich, S. Micali, and A. Wigderson.
How to play any mental game.
In *Proc. 19th STOC*, pages 218-229. ACM, 1987.

[19] 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.

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

[21] J. Halpern and Y. Moses.
Knowledge and common knowledge in a distributed environment.
In *Proc. 3rd PODC*, pages 50-61. ACM, 1984.

[22] G.H. Hardy and E.M. Wright.
*An Introduction to the Theory of Numbers.*
Oxford University Press, 1954.

[23] R. Impagliazzo.
*A collection of direct zero-knowledge protocols for NP-complete problems.*
Technical Report, University of California at Berkeley, Computer Science Division, 1986.

[24] E. Kranakis.
*Primality and Cryptography.*
John Wiley & Sons, 1986.

[25] L. Levin.
One-way functions and pseudorandom generators.
In *Proc. 17th STOC*, pages 363-365. ACM, 1985.

[26] 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.

[27] I. Niven and H.S. Zuckerman.
*An Introduction to the Theory of Numbers.*
John Wiley & Sons, New York, 1972.

[28] M. Tompa and H. Woll.
1986
Personal communication.

[29] U. Vazirani and V. Vazirani.
Efficient and secure pseudo-random number generation.
In *Proc. 25th FOCS*, pages 458-463. IEEE, 1984.

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

[31] A. C. Yao.
How to generate and exchange secrets.
In *Proc. 27th FOCS*, pages 162-167. IEEE, 1986.

[32] M. Yung.
Direct zero-knowledge computation.
1986.
Manuscript in preparation.