The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser
MIT

Silvio Micali
MIT

Charles Rackoff
University of Toronto

1. Introduction

In the first part of the paper we introduce a new theorem-proving procedure, that is a new *efficient method of communicating a proof.* Any such method implies, directly or indirectly, a definition of proof. Our "proofs" are probabilistic in nature. On input an n-bits long statement, we may erroneously be convinced of its correctness with very small probability, say, $\frac{1}{2^n}$, and rightfully be convinced of its correctness with very high probability, say, $1 - \frac{1}{2^n}$. Our proofs are *interactive.* To efficiently verify the correctness of a statement, the "recipient" of the proof must actively ask questions and receive answers from the "prover".

In the second part of the paper, we address the following question:

> *How much knowledge should be communicated for proving a theorem T?*

Certainly enough to see that T is true, but usually much more. For instance, to prove that a graph is Hamiltonian it suffices to exhibit an Hamiltonian tour. This appears, however, to contain much additional knowledge than the single bit "Hamiltonian/non-Hamiltonian".

We give a computational complexity measure of knowledge and measure the amount of additional knowledge contained in proofs.

We propose to classify languages according to the amount of additional knowledge that must be released for proving membership in them.

Of particular interest is the case where this additional knowledge is essentially 0 and we show that is possible to interactively prove that a number is quadratic non residue mod m releasing 0 additional knowledge. This is surprising as no efficient algorithm for deciding quadratic residuosity mod m is known when m's factorization is not given. Moreover, all known *NP* proofs for this problem exhibit the prime factorization of m. This indicates that adding interaction to the proving process, may decrease the amount of knowledge that must be communicated in order to prove a theorem.

2. Interactive Proof Systems

Much effort has been previously devoted to make precise the notion of a theorem-proving procedure. *NP* constitutes a very successful formalization of this notion. Loosely speaking, a theorem is in provable in NP if its proof is easy to verify once it has been found. Let us recall Cook's [C] (and independently Levin's [L]) influential definition of *NP* in this light.

> The *NP* proof-system consists of two communicating Turing machines A and B: respectively, the *prover* and the *verifier.* The prover is exponential-time, the verifier is polynomial-time. Both A and B are deterministic, read a common input and interact in a very elementary way. On

This research was supported in part by IBM Young Faculty Development Award dated September 1983, IBM Young Faculty Development Award dated September 1984, and NSF grant DCR-8413577

input a string x, belonging to an NP language L, A computes a string y (whose length is bounded by a polynomial in the length of x) and writes y on a special tape that B can read. B then checks that $f_L(y)=x$ (where f_l is a polynomial-time computable function relative to the language L) and, if so, halts and accepts. This process is illustrated in figure 1.

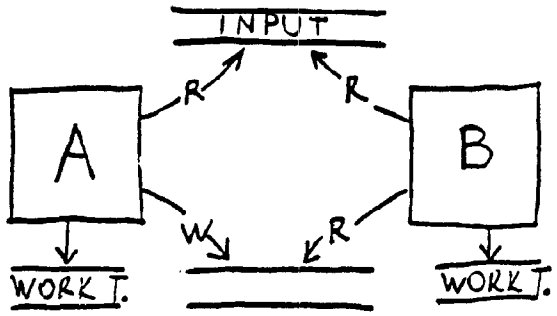

Fig. 1: The NP proof-system(*)

What is intuitively required from a theorem-proving procedure? First, that it is possible to "prove" a true theorem. Second, that it is impossible to "prove" a false theorem. Third, that communicating a proof should be efficient in the following sense. It does not matter how long must the prover compute during the proving process, but it is essential that the computation required from the verifier is easy.

Theorem-proving procedures differ in the underlying definition of a proof. The notion of a proof, like the notion of a computation, is an intuitive one. Intuition, however, may and must be formalized. Computability by (deterministic) Turing machines is an elegant example of formalization of the intuitive concept of a computation. Each formalization, however, cannot entirely capture our original and intuitive notions, exactly because they are intuitive. Following our intuition, probabilistic algorithms [R] [SS] are means of computing, though they are not in the previous formal model. Similarly, NP is an elegant formalization of the intuitive notion of a theorem-proving procedure. However, NP only captures a particular way of communicating a proof. It deals with those proofs that can be "written down in a book". In this paper we introduce interactive proof-systems to capture a more general way of communicating a proof. We deal with those proofs that can be "explained in class". Informally, in a classroom, the lecturer can take full advantage of the possibility of interacting with the "recipients" of the proof. They may ask questions at crucial points of the argument and receive answers. This makes life much easier. Writing down a proof that can be checked by everybody without interaction is a much harder task. In some sense, because one has to answer in advance all possible questions. Let us now formally set up the proper computational model.

(*) (By $----\!>$ we denote a read/write head, by $---R\!->$ a read-only head and by $---W\!->$ a write-only head)

2.1 Interactive Turing machines and interactive pairs of Turing machines

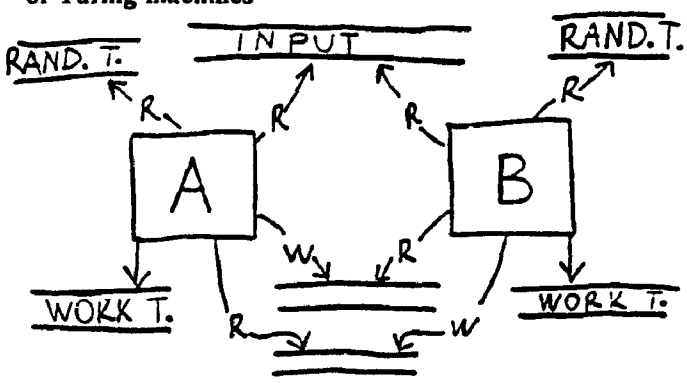

Fig. 2: an interactive pair of Turing machines

An interactive Turing machine (*ITM*) is a Turing machine with a read-only input tape, a work tape and a random tape. The random tape contains an infinite sequence of random bits. The random tape can be scanned only from left to right. When we say that an interactive machine flips a coin we mean that it reads next bit in its own random tape. This tape is the only source of randomness for the machine. In addition an interactive machine has a read-only communication tape and a write-only communication tape. The head writing on the latter tape moves only from left to right, writes only on a blank cell and cannot move to the right without writing.

Two ITM's A and B form an *interactive pair of*

Turing machines (A,B) by

1) letting A and B share the same input tape and
2) letting B's write-only communication tape be A's read-only communication tape and vice versa.

The interactive pair (A,B) is ordered and machine B starts the computation. The machines take turns in being active. When, say, A is active, it can perform internal computation, read and write on the proper tapes and send a message to B by writing on the appropriate communication tape. The ith message of A is the entire string that A writes on the communication tape during its ith turn. The ith message of B is similarly defined. Either machine can, during its turn, terminate the computation of the pair. Consider a computation of (A,B) on input x. Let the computation consist of n turns and let a_i be A's ith message and b_i be B's ith message. Then the *text of the computation* is defined to be the sequence $\{b_1,a_1,\cdots,b_n,a_n\}$. ($a_n$ is empty if it is B that halts the computation of (A,B) in its nth turn). The text of all possible computations of A and B on input x will be of relevance to our analysis and it will be denoted by $(A,B)[x]$. This set has the structure of a probability space in the natural way. The probability of each computation in $(A,B)[x]$ is taken over the coin tosses of both machines.

2.2 Interactive proof-systems

Let $L\subseteq\{0,1\}^*$ be a language and (A,B) an interactive pair of Turing machines. We say that (A,B) is an *interactive proof-system* for L if A (the prover) has infinite power, B (the *verifier*) is polynomial time and they satisfy the following properties.

1) For any $x\in L$ given as input to (A,B), B halts and accepts with probability at least $1-\frac{1}{n^k}$ for each k and sufficiently large n.
2) For any ITM A' and for any x not in L given as input to (A',B), B accepts with probability at most $\frac{1}{n^k}$ for each k and sufficiently large n.

Here n denotes the length of the input and the probabilities are taken only over B's own coin tosses.

Condition 1 essentially says that, if $x\in L$, there exist a way to easily prove this fact to B that succeeds with overwhelming probability. This way is A's algorithm. In other words, it is possible to prove a true theorem so that the proofs are easily verified (B is polynomial-time). Condition 2 says that, if x not in L, there exist no strategy, for convincing B of the contrary, that succeeds with non negligible probability. In other words, no one can prove a false theorem. In fact, B needs not to trust (or to know) the machine with which it is interacting. It is enough for B to trust the randomness of its own coin tosses. Notice that, as for NP, the emphasis is on the "yes-instances": if a string is in the language we want to show it, if it is not we do not care. Let us consider an example of an interactive proof-system.

Example 1: Let Z_m^* denote the set of integers between 1 and m that are relatively prime with m. An element $a\in Z_m^*$ is a *quadratic residue* mod n if $a=x^2$ mod m for some $x\in Z_m^*$, else it is a *quadratic nonresidue*. Now let $L=\{(m,x)\mid x\in Z_m^*$ is a quadratic nonresidue $\}$. Notice that $L\in$NP: a prover needs only to compute the factorization of m and send it to the verifier without any further interaction. But looking ahead to zero knowledge proof-systems, we will consider a more interesting interactive proof-system for L. The verifier B begins by choosing $n=|m|$ random members of Z_m^*, $\{r_1,r_2,\ldots,r_n\}$. For each i, $1\le i\le n$, he flips a coin, and if it comes up heads he forms $t_i=r_i^2$ mod m, and if it comes up tails he forms $t_i=x\cdot r_i^2$ mod m. Then B sends $t_1,t_2,\ldots,t_n$ to A. The prover, having unrestricted computing power, finds which of the t_i are quadratic residues, and uses this information to tell B the results of his last n coin tosses. If this information is correct, B accepts.

Why does this work? If $(m,x)\in L$, then A correctly predicts all last n coin tosses of B who will definitely accept. If (m,x) not in L, then the $\{t_i\}$ are just random quadratic residues, and the prover will respond correctly in the last part of the computation with probability $\frac{1}{2^n}$. In fact, for each of the last n coin tosses of B, A has probability exactly 1/2 of guessing it correctly.

A more complex interactive proof-system for L, that releases essentially 0 additional knowledge, can be found in section 4.2.

2.3 Interactive Complexity Classes

We define IP, *Interactive Polynomial-time*, to be the class of languages possessing an interactive proof-system. In this case we may also say that L is interactively provable. To emphasize that the prover has unlimited power, we may write IP_∞ for IP. To closer analyze the role of the prover, we define $IP_{T(n)}$ to be the class of languages having an interactive proof-system whose prover runs in time $T(n)$. To focus on the role of interaction, we let $IP[f(n)]$ denote the class of languages having a proof-system that, on input a string x of length n, halts within $f(n)$ turns. Here f is a non decreasing function from natural numbers to natural numbers.

Interactive proof-systems should be contrasted with the "Arthur-Merlin" games of Babai [B]. In those games Merlin plays the role of A and Arthur the role of B. The big difference is that Merlin sees all results of Arthur's coin tosses. This allows Babai to prove that arbitrary interaction is not necessary in his framework: it is sufficient to allow Arthur to talk to Merlin and have Merlin respond; at least as long they alternate a constant number of times. Actually Arthur's message to Merlin consists exactly of the sequence of its own coin tosses. (See figure 3).

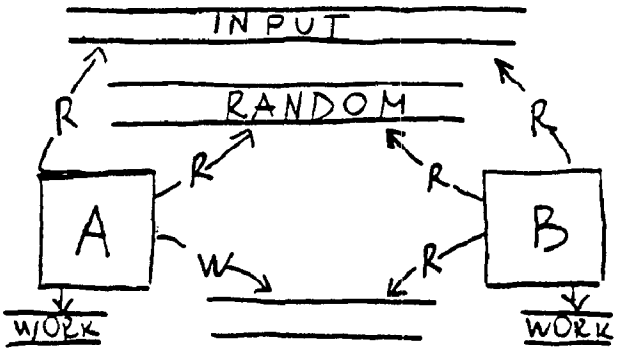

fig. 3: The Arthur-Merlin proof-system

If membership in a language L can be proved by an Arthur-Merlin game ($L \in AM$) then, for any random oracle O, $L \in NP^O$ with probability 1. It is apparent that $AM \subseteq IP$ (actually, $AM \subseteq IP[1]$) and we believe that the inclusion is a strict one. We also believe that our "interactive hierarchy" does not collapse, i.e. that $IP[k]$ is strictly contained in $IP[k+1]$. In any case, interactive proof-systems are the right proof model to both analyze and reduce the knowledge complexity of a language. Next section is devoted to the discussion of this more subtle notion. Let us also mention Papadimitriou' [P] "games against nature". This is an elegant characterization of PSPACE, though not an efficient method of communicating a proof.

3. Knowledge Complexity

Communication is a tool for transferring or exchanging knowledge. Knowledge has received a lot of attention in a model-theoretic framework [FHV], [HM]. In this context, roughly speaking,

1) *All participants are considered to have infinite computing power.* (E.g. each participant "knows" all logical consequences of the information in his hands) and

2) *The object they try to "know better" is not an available public input.* (Rather some event occurs that is witnessed or noticed by some but not all participants. To give an elementary example, one participant flips a coin and tells the outcome to a few others who now "know" it. The remaining participants do not "know" what the outcome was and they have to decide between two possible worlds: one in which "heads" came up and one in which "tails" came up).

This scenario may not be realistic in many practical contexts. In physics, for example, scientists have *bounded resources* and the object they try to know better is a *public input*: nature. Our point of view is that

1) *Knowledge is a notion relative to a specific model of computation with specified computing resources and*

2) *One studies and gains knowledge about available objects.*

In this paper we measure the amount of knowledge that can be gained from a communication by a participant with polynomially bounded resources and

investigate how much knowledge must be communicated for proving a theorem.[2] Our computational complexity measure of knowledge is, however, of wider applicability. For example, as sketched in section 6, it constitutes a powerful tool for developing a mathematical theory of cryptographic protocols. The following concept will be crucial to our analysis.

3.1 Degrees of distinguishability for probability distributions

Let I be an infinite set of strings and c a positive constant. For each $x \in I$ with length n, let Π_x be a probability distribution over the n^c-bit strings. Then we say that $\Pi = \{\Pi_x \mid x \in I\}$ is a *I-c-ensemble*. By saying that Π is an *ensemble* or a *I-ensemble* we mean, respectively, that there exist I and c or simply c such that Π is a I-c-ensemble.

A *distinguisher* is a probabilistic polynomial-time algorithm D that on input a string s outputs a bit b. Let $\Pi_1 = \{\Pi_{1,x} \mid x \in I\}$ and $\Pi_2 = \{\Pi_{2,x} \mid x \in I\}$ be two I-c-ensembles. Let $p^D_{x,1}$ denote the probability that D outputs 1 on input a $|x|^c$-bit long string randomly selected with probability distribution $\Pi_{1,x}$. Symmetrically, $p^D_{x,2}$ denotes the probability that D outputs 1 on input a $|x|^c$-bit long string randomly selected with probability distribution $\Pi_{2,x}$. Let $p: N \to [0,1]$. We say that the ensembles Π_1 and Π_2 are *at most p-distinguishable* if for all distinguishers D, $|p^D_{x,1} - p^D_{x,2}| < p(|x|) + \frac{1}{|x|^k}$ for all k and sufficiently long x.

Of particular interest will be the notion of at most 0-distinguishability (or indistinguishability). In this case the two ensembles are "equal" with respect to any polynomial-time computation. In section 4.2 we will present an interesting example of indistinguishable ensembles. In this example, the $\Pi_{1,x}$ and $\Pi_{2,x}$ are indistinguishable in a stronger sense. In fact the probability that they assign to each $|x|^c$-bit string is identical except for a set of strings strings whose total probability does not exceed $\frac{1}{2^{d|x|}}$ for some constant d between 0 and 1. Such strong indistinguishability is a luxury not always available and, in any case, is not necessary to develop our theory.

Notice that our distinguishers are fed with a single $|x|^c$-bit string at a time. One may consider distinguishers that are fed with more strings of length $|x|^c$ at the same time. In this case, if two ensemble are 0-distinguishable, they will remain undistinguishable (as long "more" $< poly(|x|)$). If the two ensembles are at most p-distinguishable, they may remain at most p-distinguishable or the probability of "distinguishing" them may become much higher. (This plays a role for deciding whether a certain cryptographic protocol may be played securely more than once using the same secret key).

Related notions of indistinguishability have been previously considered in [GM] in the context of probabilistic encryption and then in [Y] and [GGM] in the context of pseudo-random number generation.

3.2 The knowledge computable from a communication

Which communications convey knowledge? Informally, those that transmit the output of an unfeasible computation, a computation that we cannot perform ourselves. For example, if A sends to B n random bits, this will be n bits of information. We would say this contains no *knowledge*, however, because B could generate random bits by himself. Similarly, the result of any probabilistic polynomial-time computation will not contain any knowledge. With this in mind we would like to derive an upper bound (expressed in bits) for the *amount* of knowledge that a polynomially bounded B can extract from a communication.

First a bit of notation. Notice that any probabilistic Turing machine M generates the ensemble $M[\cdot] = \{M[x]\}_{x \in I}$, where $M[x]$ denotes the set of possible outputs of M (on input $x \in I$) taken with the probability distribution induced by M's coin tosses. Similarly, we will denote by $(A,B)[\cdot]$ the ensemble associated to an interactive pair of Turing machines

(2) Our definitions may be given with respect to any time bound, but we restrict our attention to polynomial-time both to simplify the matter a bit and because we believe that it constitutes the most important case.

(A,B). We are now ready to introduce our definition.

Definition: Let (A,B) be an interactive pair of Turing machines and I the set of its inputs. Let B be polynomial-time and $f:N\rightarrow N$ be non decreasing. We say that A *communicates at most* $f(n)$ *bits of knowledge to* B if there exists a probabilistic polynomial-time machine M such that the I-ensembles $M[\cdot]$ and $(A,B)[\cdot]$ are at most $1-\frac{1}{2^{f(n)}}$-distinguishable. We say that A *communicates at most* $f(n)$ *bits of knowledge* if for all polynomial-time ITM's B' A communicates at most $f(n)$ bits of knowledge to B'.

Remark 1: Assume M, on input x, tries to select a string "as undistinguishable as possible" from a computation randomly selected in $(A,B)[x]$. Note that in this attempt no information is hidden from M: A's program, B's program and x are all inputs of M. M may have "built in" the description of A. This, however, is not of great help, as A's algorithm may be absolutely inefficient.

A non mathematical discussion: Let us try to illustrate the above definitions. Assume that a crime x has happened, B is a reporter and A a police officer. A understands the rights of the press but, for obvious reasons, also tries not to communicate too much knowledge. Should reporter B call the police officer A to know more about x? It depends. If he has probability essentially equal to 1 of generating at home, in front of his typewriter, the "same" conversations about this specific x that he might have with A, he should not bother to call. A will give him essentially 0 knowledge about x. If, instead, say, he may generate an honest conversation about x with probability 1/4 (i.e. what he generates is at most 3/4-distinguishable from the "real" conversations), then the officer may tell him something that he does not know. This knowledge however, will not exceed two bits and may not be of the "useful" kind! Still, it may pay off to call. If, finally, B has only chance 1 in 2^{100} of generating the possible conversations about x with the police officer, then A is a real gossiper and B should rush to the telephone!

Assume now that B is so news-hungry that is ready to become dishonest during the phone conversation, i.e. he is ready to transform himself to B'. Despite this, if the officer is so skillful to be one who communicates, say, at most 2 bits of knowledge, no matter how tricky questions B' asks and how much he cheats, he will not get out of him more than two bits about x. (Here we are implicitly assuming that a cheating reporter still remains a polynomial-time one!)

Example 2: Consider the ITM (A,B) of example 1. Restrict its inputs only to the strings in L. Then A communicates at most 0 bits of knowledge to B. In fact, there exists a probabilistic polynomial-time machine M such that (for those inputs) generates exactly the same ensemble that (A,B) does. Essentially, M can simulate B, as B is polynomial-time, and simulates A by looking at B's coin tosses as follows. When B sends t_i computed by squaring r_i, M will answer "quadratic residue". When B sends t_i computed by squaring r_i and then multiplying it by x, M answers "quadratic nonresidue".

Notice, however, that, if the problem of deciding quadratic residuosity is not in probabilistic polynomial-time, A does not communicate at most 0 bits of knowledge. In fact, some machine B', interacting with A, may decide to create the t_i's in a different way. For instance, such a B may send the sequence of integers $t_i=i$ and therefore receive an answer about their quadratic residuosity that it may not be able to compute by itself.

An interesting ITM A that communicates at most 0 bits of knowledge may be found in section 4.2.

3.3 The knowledge complexity of a language

How much knowledge should be communicated to provide a proof of a theorem T? Certainly enough to verify that T is true. Usually, much more. For example, to prove that a certain $a\in Z_m^*$ is a quadratic residue, it is sufficient to communicate an x such that $a\equiv x^2 \bmod m$. This communication, however, contains more knowledge than just the fact that a is a quadratic residue. It communicates a square root of a. We intend to measure the additional knowledge that a prover gives to a verifier during a proof, and

investigate whether this additional knowledge may be essentially 0.

Definition: Let L be a language possessing an interactive proof-system (A,B). Let $f:N\rightarrow N$ be non decreasing. We say that L has *knowledge complexity* $f(n)$ if, when restricting the inputs of (A,B) to the strings in L A communicates at most $f(n)$ bits of knowledge. We denote this fact by $L\in KC(f(n))$.

An informal discussion. Let us recall that we are concentrating on the "yes-instances". When a string x is not in the language the prover "gives up" and we do not measure knowledge. When, instead, $x\in L$, what is the verifier's point of view at the end of an interactive proof? First, it is "convinced" (correctly with overwhelming probability) that $x\in L$. This was the goal of the proof-system in the first place. Second, it possesses the text of the entire computation with the prover on input x. This text, has been used to verify that $x\in L$, but does not contain more than $f(n)$ bits of additional knowledge. In fact, on input $x\in L$, we are guaranteed to be able to easily generate such texts with probability distribution at most $(1-\frac{1}{2^{f(n)}})$-distinguishable from the "real" texts, no matter with which machine B' A is interacting. The special case $L\in KC(0)$ is of particular interest. In this case, by interacting with A and from the text of the computation, B can verify that $x\in L$, but, with respect to polynomial-time computation, the text is irrelevant for any other purpose, no matter with which B` A is interacting. In fact, on input a guaranteed $x\in L$, such texts can be easily selected with essentially the right probability distribution and without A.

We believe that knowledge complexity is one of the fundamental parameters of a language or, equivalently, of a theorem-proving procedure. Theorem-proving procedures are intended to communicate knowledge and it is very natural to classify them according to the amount of knowledge they communicate.

Note that knowledge complexity is also defined for NP proof-systems as they are a special type of interactive proof-system. However, their knowledge complexity tends to be very high.

A very important application of knowledge complexity is that it enables proving correctness of cryptographic protocols in a *modular way* (see section 6).

4. Languages in KC(0)

Every language in P or RP or BPP has trivially knowledge complexity 0. If L is not in probabilistic polynomial-time, no NP proof-system for L can release 0 additional knowledge. However, there may be a more interactive proof-system for L that does release 0 additional knowledge. A natural question arises. Do meaningful examples of languages in $KC(0)$ exist or is $KC(0)$-BPP a fancy way to define the empty set? A similar question could be asked for, say, RP. Namely, is RP-P a fancy name for the empty set? The best sign of a possible negative answer to the latter question is constituted by the fact that primality testing is in RP [SS] [R] and, while the problem of deterministically deciding primality has received a lot of attention for centuries, no polynomial-time algorithm is currently known. Similarly, it is of great interest to find candidates for languages in $KC(0)$ but not in, say, BPP. This is the best one can do, given our current knowledge about proving lower-bounds.

We know of two interesting languages that have knowledge complexity 0. Both are algebraic. The first one is the following language BL proposed by Blum in [Bl1] where he gives all the essential ingredients to prove $BL\in KC(0)$. Let n be an integer with prime factorization $n=p_1^{h_1}\cdots p_k^{h_k}$. Then $n\in BL$ if the number of different p_is congruent to 3 mod 4 is even. The other language that is known to belong to $KC(0)$ is the well known quadratic non-residuosity language. We give a proof of this fact in this section.

For $y\in Z_m^*$ we define

$$Q_m(y)=\begin{cases}0 & \text{if } y \text{ is a quadratic residue mod } m\\ 1 & \text{otherwise}\end{cases}$$

Then $L=\{(y,m)\mid Q_m(y)=1\}$ is the quadratic non-residuosity language.

Our proof that $L\in KC(0)$ does not depend on any unproved computational complexity assumptions.

We first review what is known about the complexity of deciding membership in this language.

4.1 The Quadratic Residuosity Problem

The quadratic residuosity problem with parameters $m \in N$ and $x \in Z_m^*$ consists of computing $Q_m(x)$. If the factorization of m is known, it is trivial to compute Q_m. If the factorization of m is unknown, then there is no known efficient procedure for computing Q_m. This decision problem is one of the four main problems discussed by Gauss in "Disquisitiones Arithmeticae" (1801) (along with primality testing, integer factorization and Solvability of Diophantine Equations). A polynomial time solution for it would imply a probabilistic polynomial time solution for other open problems in Number Theory such as deciding whether a composite integer m is a product of 2 or 3 primes.

The Jacobi symbol $(\frac{x}{m})$ for $m \in N$ and $x \in Z_m^*$ is a polynomial time computable function that evaluates to 1 and -1 and provides some information about $Q_m(x)$. Namely, if $(\frac{x}{m}) = -1$ then $Q_m(x) = 1$. However, when $(\frac{x}{m}) = 1$ then computing $Q_m(x)$ is a hard problem. n fact, it is not even known how to efficiently produce a single "guaranteed" quadratic non-residue mod m with Jacobi symbol 1.

4.2 A "0" Knowledge Interactive Proof System for L

In the proof system, (A,B), that we exhibit for $(y,m) \in L$ the prover A is only required to be a probabilistic polynomial time Turing machine with the additional power of being able to evaluate Q_m. (Of course, it remains true that no infinitely powerful A' can convince B that y is a quadratic non-residue mod m if that is not the case).

For simplicity, we only consider proving that $(y,m) \in L$ when the Jacobi symbol $(\frac{y}{m}) = 1$. The case where $(\frac{y}{m}) = -1$ is uninteresting. We specify A and B by giving their explicit program at each step of the interaction.

The basic idea is that B generates numbers of two types: $x = r^2 \bmod m$ (type 1) and $x = y \cdot r^2 \bmod m$ (type 2) where r is randomly chosen, and quizzes A about them. If indeed (y,m) is in L, then A can tell the types of these numbers. If (y,m) is not in L, they look all the same to A and it will fail the quizzes with very high probabiilty. The danger with this basic idea arises when indeed (y,m) is in L as A , when answering the quizzes, may release some knowledge other than $(y,m) \in L$ (e.g. the quadratic residuosity of specific other $x \in Z_n^*$ chosen by a cheating B'). We overcome this danger, by having A make sure that the machine with which it is interacting "knows" what are the types of the numbers it quizzes A about.

A and B's Interactive Program

Input: $(y,m) \in L$ such that $(\frac{y}{m}) = 1$ and $n = \log_2 m$.

Initialize *iteration* $= 0$.

Step 1:

B first chooses a random r_0 from Z_m^*, and then tosses a coin C_x. If $C_x = 0$, then B sets $x = r_0^2 \bmod n$, else if $C_x = 1$, B sets $x = y \cdot r_0^2 \bmod n$. B sends x to A.

Then, B chooses two random sets, each of size n,

$$T = \{ t_1, t_2, \ldots, t_n \mid t_i = r_i^2 \bmod m \}$$

and,

$$S = \{ t_{n+1}, t_{n+2}, \ldots, t_{2n} \mid t_i = y \cdot r_i^2 \bmod m \}$$

B sends to A the elements in $T \bigcup S$ in random order.

Step 2:

A picks a random subset $Z \subseteq T \bigcup S$ of size n and sends it back to B.

Step 3:

For each $z \in Z$, B sends to A r such that $z = r^2 \bmod m$ or $z = y \cdot r^2 \bmod m$.

Suppose that the sizes of $T - Z$ and $S - Z$ differ by d. Then, B chooses d random elements from the larger set, $t_{i_1}, \ldots, t_{i_d}$ and sends their respective $r_{i_1}, \ldots, r_{i_d}$ to A. (i.e $t_{i_j} = r_{i_j}^2$ or

$t_{i_j} = y \cdot r_{i_j} \bmod m$ for some $1 \leq i_j \leq 2n$).

B sets $X = T - Z - \{t_{i_1}, \ldots, t_{i_d}\}$, and $Y = S - Z - \{t_{i_1}, \ldots, t_{i_d}\}$.

If $x = r_0^2 \bmod m$, B lets:

$$X' = \{r_0 \cdot r_i = \sqrt{x \cdot t_i} \bmod n \mid t_i \in X\}$$
$$Y' = \{y \cdot r_0 \cdot r_i = \sqrt{y \cdot x \cdot t_i} \bmod n \mid t_i \in Y\}.$$

else if $x = y \cdot r_0^2 \bmod m$, B lets:

$$X' = \{y \cdot r_0 \cdot r_i = \sqrt{y \cdot x \cdot t_i} \bmod n \mid t_i \in X\}$$
$$Y' = \{y \cdot r_0 \cdot r_i = \sqrt{x \cdot t_i} \bmod n \mid t_i \in Y\}$$

B then sends the elements in $X' \bigcup Y'$ to A in random order.

Step 4:

A checks that $X' \bigcup Y'$ is of the form specified in step 3 (i.e for all $w \in X' \bigcup Y'$, $w^2 = t_i x \bmod m$ or $w^2 = t_i \cdot x \cdot y \bmod m$ for some $t_i \in X \bigcup Y$) and that $|X' \bigcup Y'| > \frac{n}{3}$. If this is not the case, A halts detecting cheating. Otherwise, A sends B the value $v = Q_m(x)$.

Step 5:

If $v \neq C_x$ then B halts detecting cheating, otherwise *iteration* = *iteration* + 1 (this is the end of an iteration).

If *iteration* $\geq n$, then B accepts $(y,m) \in L$, otherwise B goes back to step 1.

Let us first prove that (A,B) constitutes an interactive proof-system for L.

Remark 2: Note that if A, B both operate according to specification, then each iteration of the program will be completed with probability $> 1 - \frac{1}{2^{cn}}$ for $0 < c \leq 1$.

The following claims 1&2 hold for each completed iteration.

Claim 1: If (y,m) is not in L, then A (or any other A') correctly guessed C_x (i.e sends $v = C_x$), with probability exactly $\frac{1}{2}$.

proof: The proof follows from the fact that $C_x = 0$ with probability exactly $\frac{1}{2}$ and that even with infinite computation power A' can't distinguish between a computation with B in which $C_x = 0$ from one in which $C_x = 1$. The latter can be seen as follows.

Suppose $C_x = 0$.

Then, in step 3 for all $t_i \in X$, A receives $r_0 r_i = \sqrt{t_i x} = \sqrt{r_i^2 r_0^2} \bmod m$. Note that $e_i = t_i \cdot x \bmod m$ is a random square, (as t_i is) and $r_0 r_i$ is a random square root of $e_i \bmod m$.

for all $t_i \in Y$, A receives $y \cdot r_0 \cdot r_i = \sqrt{y \cdot t_i \cdot x} = \sqrt{y^2 r_i^2 r_0^2} \bmod m$. Note that $f_i = y \cdot t_i \cdot x = y^2 \cdot r_i^2 \cdot x \bmod m$ is a random square, (as r_i^2 is) and $y \cdot r_0 \cdot r_i$ is a random square root of $f_i \bmod m$.

Suppose $C_x = 1$.

Then, in step 3, for all $t_i \in X$, A receives $y \cdot r_0 \cdot r_i = \sqrt{y \cdot t_i \cdot x} = \sqrt{y^2 r_i^2 \cdot r_0^2} \bmod m$. Note that $\tilde{e}_i = y \cdot t_i \cdot x \bmod m$ is a random square, (as both y and t_i are now squares and t_i is a random square) and $y \cdot r_0 r_i$ is a random square root of $\tilde{e}_i \bmod m$.

for all $t_i \in Y$, A receives $y \cdot r_0 \cdot r_i = \sqrt{t_i \cdot x} \bmod m$. Note that $\tilde{f}_i = t_i \cdot x = y^2 \cdot r_i^2 \cdot r_0^2 \bmod m$ is a random square, (as r_i^2 is) and $y \cdot r_0 \cdot r_i$ is a random square root of $\tilde{f}_i \bmod m$.

Thus, for both $C_x = 0$ and $C_x = 1$ A will still receive random square roots of random squares. Therefore A can't have any advantage in predicting C_x.

Claim 2: If (y,m) in L, then A correctly computed C_x in step 4.

Theorem 1: (A,B) is an interactive proof-system for L.

Proof: For every $(y,m) \in L$ given as input to (A,B), B halts and accepts with probability greater than $(1 - \frac{1}{2^{cn}})$ for all constants $0 < c \leq 1$ and sufficiently large n. This follows by claim 2. For any machine A' and for any (y,m) not in L, given as input to (A',B), B accepts with probability at most $\frac{1}{2^n}$ by claim 1 and remark 3.

We now proceed to show that L has knowledge complexity 0.

Theorem 2: L has knowledge complexity 0.

Proof: To show that (A,B) constitutes a 0 knowledge proof-system for L, we must show that for each polynomial-time ITM B', there exists a probabilistic polynomial-time Turing Machine M, such that the two ensembles $M[\cdot]$ and $(A,B')[\cdot]$ are indistinguishable. The basic idea is that M can easily simulate B', as B' runs in polynomial time. On the other hand, M will succeed in simulating A, by running B' twice with the same coin tosses.

A more precise description of M is the following: On input $(y,m)\in L$, M randomly fills the random tape of B' with a sufficiently long string R, and makes B' perform "its own version" of step 1. (B' may in fact execute a different algorithm than B during step 1.) Simulating A in step 2 is easy for M, as all A does here is picking a random subset. Next, M makes B' perform its own version of step 3. Now, M must simulate A in step 4. Notice that it is easy to check whether A will halt in step 4. Therefore it will be easy for M to simulate A in a computation with B' in which A halts in step 4. Difficulties arise if A won't halt but continue. This implies that M must compute $Q_m(x)$ correctly as A does. This is easy to do for A who has enough power to decide the quadratic residuosity of x. Notice that this would also be easy for M if B', either generated x by squaring mod m an r_0 that M may observe (in which case M knows that $Q_m(x)=0$), or if B' generated x by squaring mod m an r_0 and multiplying by y (in which case M knows that $Q_m(x)=1$). However, life may be not so easy. B' might have generated x in some other way (e.g. at random) which would make it hard for M to compute $Q_m(x)$. We overcome this difficulty as follows. By $c_1,c_2,c_3,...$ we denote fixed, positive constants depending on A and B'. Without loss of generality, we may assume that on input (y,m), A will halt in step 4 with probability less than $1 - \frac{1}{2^{c_1 n}}$. (Otherwise by simulating A and B' for steps 1,2 and 3, as above, and having A halt in step 4, we trivially generate computations which are indistinguishable from $(A,B')[(y,m)]$.

At the end of step 3, M saves all messages sent so far by B' and the "virtual" A. M now runs B' again with the same input (y,m) and the same content R in the random tape of B'. For this second compuation, M simulates A anew, by flipping new coins. Four things will happen in this second computation.

1) B' sends in step 1 the same sets S and T, as in its first computation.
2) In step 2, A will select a random subset $\tilde{Z}\subseteq T\bigcup S$. With probability greater than $1-\frac{1}{2^{c_2 n}}$, $\tilde{Z}\neq Z$ (where Z denotes the set chosen in the first computation).
3) In step 3, B sends the sets $\tilde{X}$ and $\tilde{Y}$. (The respective sets in the first compuation were X' and Y'). With probability $> 1-\frac{1}{2^{c_3 n}}$, $\tilde{X}$ and $\tilde{Y}$ are of the right form (i.e could not cause the legal A to halt).
4) With probability $> 1-\frac{1}{2^{c_4 n}}$, $\tilde{X}\neq X'$ and $\tilde{Y}\neq Y'$.

M now selects an element $t_i\in(T-X')\bigcap\tilde{X}$. As $t_i\in T-X'$, in the first computation B' sent its corresponding r_i. As $t_i\in\tilde{X}$, in the second computation B' sends $\sqrt{xt_i}$ mod m or $\sqrt{xt_i y}$ mod m. Now, in whatever case, it is just a matter of algebra for M to easily compute r_0 such that $r_0^2=x$ mod m or $r_0^2\cdot y=x$ mod m. If $(y,m)\in L$, exactly one of these cases may occur. Therefore M, having computed r_0, can simulate A by sending a $v=Q_m(x)$.
QED

5. A parenthetical section.

Remark 3: A stronger way of saying that A communicates at most $f(n)$ bits of knowledge with respect to polynomial-time computation, is the following.

> For all ITM B' there exist a polynomial-time ITM M that by interacting with B' (but also reading the random tape of B'!) produces an ensemble at most $(1-\frac{1}{2^{f(n)}})$-distinguishable from $(A,B')[\cdot]$.

This notion is stronger as it allows B' not to be bound to polynomial-time computation while A needs not to know what the computing power of B' is. Full details will be given in the final paper. Interestingly, the interactive proof-system for quadratic non-residuosity of section 4.2 releases 0 additional knowledge even with respect to this stronger definition.

An informal definition: One advantage of the point of view of Remark 3 is that it allows one to express in a clean way notions like "the polynomial-time machine B knew x at some point of its computation". Let us consider a particular example. Assume that machine B started computing on input k and outputs a k-bit integer m. B may have randomly selected two primes p_1 and p_2, multiplied them together to produce m, then "erased" p_1 and p_2 and output m. What could one mean by saying that B knew the factorization of m? A natural choice is that B is able to compute it. In a narrow sense, this may mean that, in performing next instruction, B will output m's factorization or that it was written, say, at the beginning of B's work-tape at some point in time. In a broader sense it may mean that if a probabilistic polynomial-time machine M "monitors" the sequence of istantaneous descriptions of B's computation, then M outputs m's factorization with very high probability in poly(k) time. This, however, may not be general enough. In fact, "extracting" m's factorization may not be easy for M, and still B had enough "potential" to efficiently compute it (though B's program may never explicitly do so). We believe that the following (informal) definition achieves the right level of generality. Let M be a probabilistic polynomial-time machine that monitors B's computation from the start till it outputs m. In particular, M reads all the inputs (random and not) of B and all its outputs. Informally we say that *B knew m's factorization* if M can now use B to compute m's factorization. This use of B may be very general. For example, M may run B more than once after altering the content of its tapes. An example of this is implicit in section 4.2. Full details will be given in the final paper.

6. Applications to Cryptographic Protocols

Given our current state of knowledge about lower bounds, the security of a cryptographic protocol must be proved based on the intractability assumption of some candidate hard problem. Thus one must accept that further analysis may reveal some candidate hard problems to be efficiently solvable. What is not acceptable is that a protocol may be broken without violating the relative intractability assumption.

In traditional computational complexity or communication complexity, the goal is to communicate as much knowledge as possible as efficiently as possible. Since all participants are considered good friends, no one cares if more knowledge than necessary is communicated. The situation with respect to cryptographic protocols is very different. In this case there is generally no problem at all communicating the knowledge efficiently, but the whole problem is making sure not *too much* knowledge has been communicated.

Model theoretic knowledge has been used to analyze protocols. For example, in [HR] it has been used to prove Rabin's "Oblivious Transfer" correct in some setting. However, as pointed out in [FMR], Rabin's oblivious transfer still lacks a proof of correctness in a complexity theoretic framework.

We believe that knowledge complexity provides the right framework to discuss the correctness of crytographic protocols. Applying these ideas, [FMR] modified Rabin's oblivious transfer so that it can be proved correct. A sketch of this can be found in section 6.1.

Knowledge complexity helps in proving or disproving the correctness of cryptographic protocols as these are based on the secrecy of some private information and should preserve this secrecy. The privacy of some information is what gives us an advantage over our adversaries. Let A(lice) possess the prime factorization of an integer n (say $n = p_1 \cdot p_2$), while B(ob) only knows n. During a protocol with B, A must protect the privacy of her information. Assume that A can perform each step of the protocol without having even to look at the value of p_1 and p_2. Then it is easy to show that the protocol did not

compromise the privacy of n s factorization. It is also easy to see, however, that the protocol could not have accomplished any interesting task. In fact A has not made use of her "advantage"! The protocol may accomplish a non-trivial task if, in at least one step of it, A performs a computation c that depends on p_1 and p_2. This raises the question:

Will $c(p_1,p_2)$ betray to much information about p_1 and p_2?

Classical information theory does not provide an answer to this question. Knowledge complexity can. In particular,

1) We can quantify the amount of knowledge about p_1 and p_2 that c conveys and

2) We can design protocols so to minimize this amount of knowledge.

If (A,B) is a 0 knowledge interactive proof-system for L, we already saw that, on input $x \in L$, A gives B at most *one* bit of knowledge, namely $x \in L$. (That is 0 additional knowledge). More generally however, we define an upper bound, measured in bits, on the amount of knowledge A gives to B in a particular protocol (to appear in the final paper).

We use this to give an upper bound on the number of times a single protocol or a combination of protocols can be played, using a common secret key, without giving away too much information about the secret key. In addition, trying to measure the amount of knowledge revealed during the execution of a protocol about the secret, may pin point weaknesses in the design of the protocol. For example the amount of knowledge revealed in a protocol of [BD] appeared to be unreasonably large. Further analysis by [H] showed that this protocol could be broken if the encryption function used in the protocol is RSA with low exponents or Rabin's function.

A most important application of these ideas is that it allows us to prove correctness of protocols in a *modular* way. Complex protocols are usually composed of sub-protocols. For instance, many protocols use a sub-protocol for "coin tossing over a telephone" (Blum [Bl1]). However, it is not clear how to use a "normal" definition of correctness of "coin tossing" to prove the correctness of the main protocol. In general, it appears that much stronger definitions for these sub-protocols are needed in order to fit them modularly and cleanly inside larger protocols. Full details will be given in the final paper.

6.1 A Modification of the Oblivious Transfer That Is Provably Equivalent to Factoring

This section is joint work of [FMR]. The notion of an Oblivious Transfer (OT) has been introduced by Rabin [HR] who also proposed the first protocol implementing it. OT appears useful as a design tool. See for example Blum [Bl2] and Even Goldreich and Lempel [EGL]. Rabin introduced OT (to be described below) in a number theoretic setting. More generally the OT can be viewed as a protocol for transferring a *large amount of knowledge* with probability 1/2 [EGL]. Berger, Peralta and Tedrick [BPT] present a correct protocol for "obliviously transferring" a random number. Different from OT, this protocol transfers no knowledge.

The notion of an OT involves two parties A and B and an integer n (product of two large distinct primes) whose factorization is only known to A. A would like to send the factorization of n to B with the following constraints:

1) B must have 50% chance of receiving the factorization of n and the other half of the time B should not know any information at all about the factors of n.

2) A should not have any idea whether or not B received the factorization of n.

Rabin's protocol relies on the computational difficulty of factoring. However, as described below, there is a potential flow in his protocol: it is *possible* that B can cheat and factor n with probability much higher than 1/2 even if the intractability assumption of factoring holds. Although we cannot prove that B can really cheat, no one has yet been able to prove that B cannot. Before proceeding any further, let us describe Rabin's proposed protocol. We assume that A and B both know n and that A knows its factorization.

Step 1: B chooses a random x, $1 \leq x \leq n$, relatively prime with n. Then B computes $y \equiv x^2 \bmod n$ and sends y to A.

Step 2: A computes a random square root (mod n) z of y and sends z to B. (If no square root exists, A does nothing).

Step 3: B checks that $z^2 \equiv y \bmod n$. (If not, B halts detecting cheating). Let us assume that $z^2 \equiv y \bmod n$. It is well known that y has four square roots mod n that can be written as $\{x, -x, w, -w\}$, where B knows x. With probability 50% z will be x or $-x$ and B receives no knowledge. With probability 50%, however, z will be w or $-w$, in which case $gcd(n, x+z)$ will be a factor of n, allowing B to compute the factorization of n.

Party A cannot cheat by sending back some cleverly chosen square root z of of n: no matter what A does, $z \in \{x, -x\}$ with probability 50% and $z \in \{w, -w\}$ with probability again 50% and A cannot know which is the case.

Is it clear, however, that B cannot cheat? We wish it to be the case that at the end of the protocol B cannot factor with probability (much) bigger than 1/2, even if B cheats, and we wish to prove this assuming only that factoring is hard. What happens if B does not square any x at all, but instead picks a particular cleverly chosen square mod n y to send? Perhaps knowing any square root mod n of y will allow B to factor n. That is, perhaps there is a polynomial time algorithm that given n produces a "special" square mod n y, and another polynomial time algorithm that given y, n and any square root of y mod n factors n. The point is not that we have such algorithms, but that no one has proved that the existence of such algorithms contradicts the assumption that factoring is hard. Hence, the proof that Rabin's protocol is correct relies not only on the assumption that factoring is hard, but on an additional complicated and unnatural assumption, essentially that the above algorithms do not exist.

We have been able to prove that a modified version of Rabin's OT is correct. I.e. the probability (taken over the possible choices of n and all possible random choices of B) that B can factor n in k steps at the end of the protocol, equals 1/2 + the probability that B can factor n in k steps before the protocol starts. The heart of the modified protocol is that in addition to y, B gives A a minimum knowledge interactive proof that he possesses a square root of y following the ideas in section 4.2. In particular, such interactive proof will not reveal any information about which square root B knows. Now that we have made sure that B knows one square root of y, when A will give him one of them at random, it is easy to prove that B's probability of factoring n at the end of the protocol equals 1/2 + the probability that he had of factoring n before the start of the protocol.

7. Open Problems

Many open problems arise. We only list a few of them.

1. Is NP strictly contained in IP?
2. Is KC(0) contained in NP?
3. Is $KC(0)$ contained in $IP[1]$?
4. Is $IP[k]$ strictly contained in $IP[k+1]$?
5. Are there NP Complete languages in $KC(\Omega(n))$?
6. For what time-bound $T(n)$, if any, $IP_{infinite} \subseteq IP_{T(n)}$?

Acknowledegements

Mike Sipser greately helped in focusing on this problem.

We highly benefited from the encouragement and the ideas of Dena Angluin, Manuel Blum, Steve Cook, Mike Fischer, Oded Goldreich, Ravi Kannan, Dick Karp, David Lichtenstein, Albert Meyer, Gary Miller, Ron Rivest and Paul Weiss.

To all our most sincere thanks.

References

[B] Babai L., *Trading Group Theory for Randomness*

[Bl1] M. Blum, *Coin flipping by telephone*, IEEE COMPCON 1982.

[Bl2] M. Blum, *Three applications of the oblivious transfer*, Unpublished manuscript, 1981

[BPT]Berger, Peralta, Tedrick, *On fixing the Oblivious Transfer*, Presented in Eurocrypt 1983. These Proceedings

[C] S.Cook, *The Complexity of Theorem-Proving Procedures", Proc. of 3rd STOC, 1971.*

[DB] D. Dolev, A. Broder, *Flipping Coins in Many Pockets*, Proc. of 25th FOCS, 1984.

[EGL]Even, Goldreich Lempel, *A randomized protocol for Signing Contracts*, Advances in Cryptology: proceedings of Crypto 1982, Plenum press, 1983, 205-210.

[FHV]R. Fagin, J. Halpern, M. Vardi, A model-theoretic analysis of knowledge, Proc. of 25th FOCS, 1984.

[FMR]M. Fischer, S. Micali and C. Rackoff, *A Secure Protocol for the Oblivious Transfer*, Eurocrypt 1984.

[HM] J. Halpern, Y. Moses, *Knowledge and Common Knowledge in a Distributed Environment*, Proc. of 3rd PODC, 1984.

[H] J. Hastad, *On Solving A System of Simultaneous Modular Polynomial Equations of Low Degree*, In preparation.

[HR] J. Halpern and M.O. Rabin, *A Logic to reason about likehood*, Proc. of 15th STOC, 1983.

[HS] J. Hastad, A. Shamir, *On the Security of Linearly Truncated Sequences*, this proceedings.

[GM] S. Goldwasser, and S. Micali, *Probabilistic Encryption*, JCSS Vol. 28, No. 2, April 1984.

[GM] S. Goldwasser, and S. Micali ,*Proofs with Untrusted Oracles*, Unpublished Manuscript 1983.

[GGM]O. Goldreich, S. Goldwasser, and S. Micali, *How to Construct Random Function*, 25th FOCS, 1984.

[L] L.A.Levin, *Universal Sequential Search Problems*, Probl. Inform. Transm. 9/3 (1973), pp. 265-266.

[P] C. Papadimitriou, *Games against nature*, Proc. 24th ann. Symp. on Foundations of Computer Science, 1983, pp 446-450.

[PS] Papadimitriou and Sipser, *Communication Complexity*, 14th STOC, 1982.

[Y] A.C. Yao, *Some Complexity Questions Related to Distributive Computing*, Proc. of 11th STOC, 1979.

[Y] A.C. Yao, *They and Applications of Trapdoor Functions*, Proc. of 23rd FOCS, 1982.