INTERACTIVE PROOFS:
A NEW LOOK AT PASSWORDS, PROOFS, AND REFEREEING

László Lovász
Dept. of Computer Science
Eötvös Loránd University, Budapest, and
Princeton University, Princeton, NJ

Motivated by computer science, in particular, by issues like data security, electronic correspondance and cryptography, interactive proofs extend the 2000 year old, well established notion of mathematical proof. We are going to survey some of the recent developments in the field, by showing the general logical framework of what is happening by everyday examples.

In these notes we assume familiarity with some basic notions of complexity theory. (Garey and Johnson 1979) gives an introduction to some aspects of this theory.)

1. How to save the last move in chess?

Alice and Bob are playing chess over the phone. They want to interrupt the game for the night; how can they do it so that the person to move should not get the improper advantage of being able to think about his move whole night? At a tournament, the last move is not made on the board, only written down, put in an envelope, and deposited with the referee. But now the two players have no referee, no envelope, no contact other than the telephone line. The player making the last move (say, Alice) has to tell something; but this information should not be enough to determine the move, Bob would gain undue advantage. The next morning (or whenever they continue the game) she has to give some additional information, some "key", which allows Bob to reconstruct the move.

Surely this is impossible?! If she gives enough information the first time to uniquely determine his move, Bob will know her move; if the information given the first time allows several moves, then she can think about it overnight, and change her mind without being noticed. If we measure information in the sense of classical information theory, then there is no way out of this dilemma. But complexity comes to our help: it is not enough to communicate information, it must also be processed.

To describe a solution, we need a few words on algorithmic number theory. Given a natural number N, it can be tested in polynomial time whether it is a prime. These algorithms are also practically quite efficient, and work well up to several hundred digits. However, all the known prime testing algorithms have the (somewhat strange) feature that if they conclude that N is not a prime, they do not (usually) find a decomposition of N

into a product of two smaller natural numbers (usually, they conclude that N is not a prime by finding that N violates Fermat's "Little" Theorem). It seems impossible to find the prime factorization of a natural number N in polynomial time.

(Of course, very powerful supercomputers and massively parallel systems can be used to find decompositions by brute force for fairly large numbers; the current limit is around a 100 digits, and the difficulty grows very fast (exponentially) with number of digits. To find the prime decomposition of a number with 400 digits is way beyond the possibilities of computers in the foreseeable future.)

Returning to our problem, Alice and Bob can agree to encode every move as a 4-digit number. Alice extends these four digits to a prime number with 200 digits. (She can randomly extend the number and then test if the resulting number is a prime. By the Prime Number Theorem, she will have a success out of every $\ln 10^{200} \approx 400$ trials.) She also generates another prime with 201 digits and computes the product of them. The result is a number N with 400 or 401 digits; she sends this number to Bob. Next morning, she sends both prime factors to Bob. He checks that they are primes, their product is N, and reconstructs Alice's move from the first four digits of the smaller prime.

The number N contains all the information about her move: this consists of the first for digits of the smaller prime factor of N. Alice has to commit herself to the move when sending N. To find out the move of Alice, Bob would have to find the prime factors of N; this is, however, hopeless. So Bob only learns the move when the factors are revealed the next morning.

What Alice and Bob have established is an electronic "envelope": a method to deposit information at a certain time that can be retrieved at a given later time, and cannot be changed in the meanwhile. The key ingredient of their scheme is *complexity*: the computational difficulty to find the factorization of an integer. In this scheme, factoring could be replaced by any other problem in NP that is (probably) not in P.

2. How to check a password — without knowing it?

In a bank, a cash machine works by name and password. This system is safe as long as the password is kept secret. But there is one week point in security: the computer of the bank must store the password, and the programmer may learn it and later misuse it.

Complexity theory provides a scheme where the bank can verify that the costumer does indeed know the password — without storing the password itself. One solution uses the same construction as our telephone chess example. The password is a 200-digit prime number P (this is, of course, too long for everyday use, but it illustrates the idea best). When the costumer chooses the password, he also chooses another prime with 201 digits, forms the product N of the two primes, and tells the bank the number N. When the teller is used, the costumer tells his name and the password P. The computer of the bank checks whether or not P is a divisor of N; if so, it accepts P as a proper password. The division of a 400 digit number by a 200 digit number is a trivial task for a computer.

Let us assume now that a programmer learns the number N stored along with the files of our costumer. To use this in order to impersonate the costumer, he has to find a 200-digit number that is a divisor of N; but this is essentially the same problem as

finding the prime factorization of N, and, as remarked above, is hopelessly difficult. So -- even though the all the necessary information is contained in the number N — the computational complexity of the factoring problem protects the password of the costumer!

There are many other schemes to achieve the same. Let us describe another one that we shall also use later. Recall that a *Hamilton cycle* in the graph is a closed polygon that goes through every node exactly once.

We can use this notion for a password scheme as follows. The costumer chooses a polygon H on 200 nodes (labelled 1,2,...,200) as his password (this means that the password is a certain ordering of these labels, like $(7, 50, 1, \ldots, 197, 20, 3)$). Then he adds some further edges randomly, and gives the resulting graph G to the bank. When the costumer uses the teller, the computer simply checks that every edge of the polygon encoded by the password is an edge of the graph G stored for the given costumer, and if this checks out, it accepts the password.

The safety of this scheme depends on the fact that given a graph G with a few hundred nodes (and with an appropriate number of edges), it is hopelessly difficult to find a Hamilton cycle in it (even if we know that it exists). The number of edges to be added is a difficult issue. If we add many edges to hide the original Hamilton cycle well (say, const$\cdot n^2$) then we create too many new Hamilton cycles and one can find one of them in polynomial time (see Karp***). On the other hand, if we add only const$\cdot n$ new edges (so that we create only a few new Hamilton cycles) then again there is a polynomial time algorithm to find a Hamilton cycle (Frieze). Perhaps the "right" number is const$\cdot n \log n$ (which is the threshold where Hamilton cycles disjoint from the original begin to appear.

Many other problems in mathematics (in a sense, every problem in NP that is not in P) gives rise to a password scheme.

3. How to use your password — without telling it?

The password scheme discussed in the previous section is secure if the program is honest; but what happens if the program itself contains a part that simply reads off the password the costumer uses, and tells it to the programmer? In this case, the password is compromised if it is used but once. There does not seem to be any way out — how could one use the password without telling it to the computer?

It sounds paradoxical, but *there is a scheme which allows the costumer to convince the bank that he knows the password — without giving the slightest hint as to what the password is!* I'll give an informal description of the idea (following Blum 1987), by changing roles: let me be the costumer and you (the audience) play the role of the computer of the bank. I'll use two overhead projectors. The graph G shown on the first projector is the graph known to the bank; I label its nodes for convenience. My password is a Hamilton cycle (a polygon going through each node exactly once) in this graph, which I know but don't want to reveal.

I have prepared two further tranparencies. The first contains a polygon with the same number of nodes as G, drawn with random positions for the nodes and without labels. The second transparency, overlayed the first, contains the labels of the nodes and the edges that

complete it to a drawing of the graph G (with the nodes in different positions). The two transparencies are covered by a sheet of paper.

Now the audience may choose: should I remove the top sheet or the two top sheets? No matter which choice is made, the view contains no information about the Hamilton cycle, since it can be predicted: if the top sheet is removed, what is shown is a re-drawing of the graph G, with its nodes in random positions; if the two top sheets are removed, what is shown is a polygon with the right number of nodes, randomly drawn in the plane.

But the fact that the audience sees what is expected is an evidence that I know a Hamilton cycle in G! For suppose that G contains no Hamilton cycle, or at least I don't know such a cycle. How can I prepare the two transparencies? Either I cheat by drawing something different on the bottom transparency (say, a polygon with fewer nodes, or missing edges), or by having a different graph on the two transparencies together.

Of course, I may be lucky (say, I draw a polygon with fewer edges and the audience opts to view the two transparencies together) and I will not be discovered; but if I cheat, I only have a chance of 1/2 to get away with it. We can repeat this 100 times (each time with a brand new drawing!); then my chance for being lucky each time is less than 2^{-100}, which is much less than the probability that a meteorite hits the building during this demonstration. So if a 100 times in a row, the audience sees what it expects, this is a *proof* that I know a Hamilton cycle in the graph!

To make this argument precise, we have to get rid of non-mathematical notions like overhead projectors and transparencies. The reader may figure out how to replace these by envelopes, which have been mathematically defined in section 1.

The most interesting aspect of the scheme described above is that it extends the notion of a *proof*, thought (at least in mathematics) to be well established for more than 2000 years. In the classical sense, a proof is written down entirely by the author (whom we call the *Prover*), and then it is verified by the reader (whom we call the *Verifier*). Here, there is *interaction* between the Prover and the Verifier: the action taken by the Prover depends on "questions" by the Verifier. The notion of interactive proof systems was introduced independently by Goldwasser and Micali and by Babai around 1985, and has lead to many deep and surprising results in computer science and mathematical logic.

There is another feature which distinguishes this scheme from a classical proof: it also makes use of *lack of information*, namely, that I cannot possibly know in advance the choice of the audience. (If I am telling, say, Gauss' proof that the regular 7-gon cannot be constructed by compass and ruler, then it remains a valid proof even if I anticipate the questions from the audience, or even have a friend asking the questions I wish.) In a sense, a password scheme is also an interactive proof: my password is a proof that I have the right to access the account — and again, a certain "lack of information" is a key assumption, namely, that no unauthorized person can possibly know my password.

The kind of interactive proof described in this section, where the Verifier gets no information other than a single bit telling that the claim of the Prover is correct, is called a *zero-knowledge proof*. This notion was introduced by Goldwasser and Micali.

4. How to prove non-existence?

Everybody familiar with the notion of the class NP knows that it is easy to prove that a graph is Hamiltonian: it suffices to exhibit a Hamiltonian cycle. But how to prove if it is *not* Hamiltonian?

There are many famous problems (e.g, the existence of a perfect matching, or embeddability in the plane), when there is also an easy way to prove non-existence; in other words, these problems are in NP∩co-NP. For these problems, the existence of such a proof often depends on rather deep theorems, such as the Marriage Theorem of Frobenius and König, or Tutte's theorem, or Kuratowski's Theorem on planarity. But for the Hamilton cycle problem no similar result is known; in fact, if NP≠co-NP then no NP-complete problem can have such a "good characterization".

It turns out that with interaction, things get easier, and a Prover (being computationally very powerful himself) can convince a Verifier (who works in polynomial time and, in particular, is able to follow proofs only if they have polynomial length relative to the input size) that a given graph has no Hamiltonian cycle. We shall sketch the protocol, due to Nisan, that they have to follow in the case of another NP-complete problem: *given a polynomial $p(x_1, \ldots, x_n)$ with integral coefficients (say, of degree n), is there a point $y \in \{0,1\}^n$ such that $f(y) \neq 0$?* Trivially, the problem is in NP. But how to prove if the polynomial vanishes on $\{0,1\}^n$?

The property of f the Prover wants to prove is equivalent to saying that

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} p^2(x_1, \ldots, x_n) = 0. \tag{1}$$

Of course, the sum on the left hand side has an enormous number of terms, so the Verifier cannot check it directly. So the Prover suggests to consider the one-variable polynomial

$$f_1(x)p^2(x, x_2, \ldots, x_n),$$

and offers to reveal its explicit form:

$$f_1(x) = a_0 + a_1 x + \ldots + a_d x^d, \tag{2}$$

where $d \leq n$. The Verifier checks, using this form, that $f(0) + f(1) = 0$; this is easy. But how to know that (2) is the true form of $f_1(x)$? To verify the coefficients one-by-one would mean to reduce an instance in n variables to $d+1$ instances in $n-1$ variables, which would lead to a hopelessly exponential protocol with more than d^n steps.

Here is the trick: the Verifier chooses a random value $\xi \in \{0, \ldots, n^3\}$, and requests a proof of the equality

$$f_1(\xi) = a_0 + a_1 \xi + \ldots + a_d \xi^d. \tag{2}$$

Note that if the polynomials on both sides of (2) are distinct, then they agree on at most $d \leq n$ places, and hence the probability that the randomly chosen integer ξ is one of these places is at most $n/n^3 = 1/n^2$. So if the Prover is cheating, the chance that he is lucky at this point is at most $1/n^2$.

Now (3) can be written as

$$\sum_{x_2,\ldots,x_n \in \{0,1\}} p^2(\xi, x_2 \ldots, x_n) = b, \tag{4}$$

where the verifier easily calculates the value $b = a_0 + a_1\xi + \ldots + a_d\xi^d$. This is of the same form as (1) (except that the right hand side is not 0, which is irrelevant), and the Prover can prove that (4) holds by recursively. The amount of exchange is $O(d \log n)$ bits per iteration, which gives a total $O(dn \log n)$ bits; the total computational time used by the verifier is clearly also polynomial.

By the theory of NP-completeness, it follows that every problem in co-NP has polynomial-time interactive proofs. But the truth is much more interesting. This protocol has been improved by Lund, Fortnow, Karloff and Nisan (1990) and Shamir (1990) to show that *every problem in PSPACE has polynomial-time interactive proofs.* (It is easy to see that, conversely, every polynomial-time interactive proof translates into a polynomial-space algorithm.)

5. How to handle submissions that keep the main result secret?

Our last two examples come from a field which causes much headache to many of us: editing scientific journals. These are "fun" examples and their purpose is to illuminate the logic of interactive proofs rather then propose real-life applications (although with the fast development of e-mail, electronic bulletin boards, on-line reviewing, and other forms of electronic publications — who knows?).

A typical impasse situation in journal editing is caused by the following letter: "I have a proof of Fermat's Last Theorem, but I won't send you the details because I am afraid you'd steal it. But I want you to announce the result." All we can do is to point out that the policy of the Journal is to publish results only together with their proofs (whose correctness is verified by two referees). There is no way to guarantee that the editor and/or the referee, having read the proof, does not publish it under his own name (unfortunately, this does seem to happen occasionally).

The result sketched in the section 3, however, can resolve the situation. One feels that the situation is similar: the author has to convince the editor that he has a proof, without giving any information about the details of the proof. Now the theory of NP-completeness gives the following: for every mathematical statement T and number k one can construct a graph G such that T has a proof of length at most k if and only if G has a Hamilton cycle. So the editor can come up with the following suggestion upon receiving the above letter: "Please construct the graph corresponding to Fermat's last theorem and the length of your proof, and prove me, using Blum's protocol, that this graph has a Hamilton cycle." This takes some interaction (exchange of correspondence), but in principle it can be done.

6. How to referee exponentially long papers?

But of course, the real difficulty with editing a Journal is to find referees, especially for longer papers. Who wants to spend months to read a paper of, say, 150 pages, and look for errors in the complicated formulas filling its pages? And, unfortunately, the devil is often hidden in the little details: a "-" instead of a "+" on the 79-th page may spoil the whole paper...

Recent results of Babai et al (1990) offer new possibilities. Let us give, first, two informal descriptions of what turns out to be the same fact.

Assume that instead of one, we have two Provers. They are isolated, so that they cannot communicate with each other, only with the Verifier. It is clear that this makes it more difficult for them to cheat (see any crime drama), and so they may succeed in convincing the Verifier about some truth that they could not individually convince him about. Does this really happen? Babai at al. (1990) gave a protocol that allows two provers to give a polynomial time *interactive* proof for every property that has an "ordinary" proof of exponential length.

It is interesting to point out that this is the first result where we see that inteeraction really helps, without any unproven complexity hypothesis like P≠NP: it is known that there are properties whose shortest proofs grow exponentially with the input size of the instance.

The protocol of Babai et al. is a very involved extension of the protocol desribed in section 4, and we cannot even sketch it here.

There is another interpretation of these results. It is commonplace in mathematics that sometimes making a proof longer (including more detail and explanation) makes it easier to read. Can this be measured? If a proof is written down compactly, without redundancy, then one has to read the whole proof in order to check its correctness. One way of interpreting the results mentioned above is that there is a way to write down a proof so that a referee can check the correctness of the theorem by reading only a tiny fraction of it. The proof becomes longer than necessary, but not much longer. The number of characters the referee has to read is only about the *logarithm* of the original proof length! To be precise, if the original proof has length N then the new proof can be checked by reading $O(\log N)$ characters; see Feige at al (1991). So a 2000-page proof (and such proofs exist!), can be checked by reading a few lines! What a heaven for referees and editors!

This modified write-up of the proof may be viewed as an "encoding"; the encoding protects against "local" errors just like classical error-correcting codes protect against "local" errors in telecommunication. The novelty here is the combination of ideas from error-correcting codes with complexity theory.

References.

L. Babai (1985): Trading group theory for randomness, in: *Proc. 17th ACM Symp. on Theory of Computing*, 421–429.

L. Babai, L. Fortnow, L. Levin and M. Szegedy (1991): Checking computations in poly-logarithmic time, in: *Proc. 23th ACM Symp. on Theory of Computing*,

L. Babai, L. Fortnow and C. Lund (1990): Non-Deterministic Exponential Time has Two-Prover Interactive Protocols, in: *Proc. 31st IEEE FOCS*, 16-25.

M. Blum (1987):

S.A. Cook (1971): The complexity of theorem proving procedures, *Proc. 3rd Annual ACM Symposium on Theory of Computing* (Shaker Heights, 1971), ACM, New York, 151-158.

U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy (1991): Approximating Clique is Almost NP-complete, in: *32nd Annual Symposium on Foundations of Computer Science* IEEE Computer Society Press, New York.

M.R. Garey and D.S. Johnson (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, San Francisco, 1979.

S. Goldwasser and S. Micali (19):

R. M. Karp (1972), Reducibility Among Combinatorial Problems, in: *Complexity of Computer Computations*, (ed. R. E. Miller and J. W. Thatcher), Plenum Press, 85–103.

C. Lund, L. Fortnow, H. Karloff and N. Nisan (1990), Algebraic methods for interactive proof systems, in: *Proc. 22nd ACM Symp. on Theory of Computing*, 2–10.

A. Shamir (1990), IP=PSPACE, in: *Proc. 22nd ACM Symp. on Theory of Computing*, 11–15.