Proceedings of the International Congress of Mathematicians
Berkeley, California, USA, 1986

How to Prove a Theorem So No One Else Can Claim It

MANUEL BLUM

Goldwasser, Micali, and Rackoff [**GMR**] define for us what it means for a theorem to have a "zero-knowledge proof." In brief, a *zero-knowledge proof* is an interactive probabilistic protocol that gives highly convincing (but not absolutely certain) evidence that a theorem is true and that the prover knows a proof (a "standard" proof in a given logical system), while providing *not a single additional bit of information* about the proof. GMR formalize this idea. We do not. Nevertheless, we hope that the reader who has not read their paper will still understand our proofs.

Goldreich, Micali, and Wigderson [**GMW**] take another leap forward. They show that if one makes a reasonable assumption (that one-way functions[1] exist), then it is possible to convert any standard constructive proof of any of the theorems in a large natural class of theorems[2] into a zero-knowledge proof that the theorem is true. GMW start by considering a particular NP-complete problem:

Graph 3-Colorability.

Instance. A graph G.

Question. Can G be "properly" 3-colored (each node colored by one of 3 given colors so that no two adjacent nodes receive the same color).

GMW show that a "prover" who knows how to 3-color a particular graph G can convince a verifier that (1) G is 3-colorable, and (2) the prover knows a 3-coloring, without giving away any additional information. In particular, the prover does not give away the slightest clue *how* to 3-color G.

Supported, in part, by National Science Foundation Grant DCR 85-13926.

[1]One-way functions are 1-1 functions from n-bit integers to n-bit integers that, informally, are easy to compute in the forward direction, but hard to invert on all but a small fraction of n-bit integers.

[2]These theorems, which arise frequently in mathematics and computer science, are the *yes*-instances of decision problems π in NP. A good reference to NP and the theory of NP-completeness is: Michael Garey and David Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman and Company, 1979.

The essence of the GMW proof is to show the prover how to break up his proof into several pieces in such a way that

(1) the verifier can tell, by looking at any one piece of the proof, whether or not that piece has been properly constructed. Moreover, it should be clear to the verifier that if all the pieces are properly constructed, then the proof is valid, and

(2) the prover will not reveal any information about how the proof was constructed when he reveals any single piece of the proof.

To start, the prover hides each piece of the proof in its own locked safe, in reality a one-way function applied to the piece of proof. The verifier is permitted to point to any safe and ask the prover to open it. The fact that the piece of proof inside the safe is properly constructed is evidence to the verifier that *all* pieces are properly constructed, so the proof is valid. It will be evident to the verifier that the pieces can all be properly constructed by any prover who knows how to properly 3-color G, but at least one piece must be improperly constructed by a prover who does not.

Now, proofs can be broken up into pieces in many ways. The prover must select a sequence of breakups such that a piece from the first breakup plus another from the second, and so on, does not accumulate evidence to provide the slightest hint to the verifier about how to prove the theorem. It is even possible to continue the process indefinitely without ever providing a single additional bit of information about how to prove the theorem.

By repeatedly breaking up the proof and opening just one safe each time—whichever the verifier requests—the prover convinces the verifier that he is not cheating unless he is very *very* lucky.

GMW point out that because Graph 3-Colorability is an NP-complete problem, any problem in NP can be given a zero-knowledge proof, i.e., anyone who knows a polynomial length proof of a *yes*-instance of an NP problem can give a zero-knowledge proof of this fact. (For the reader familiar with the concepts of NP-completeness, this result is a consequence of Cook's theorem that *satisfiability* is NP-complete, the NP-completeness of 3-*colorability*, and the fact that the transformations used in these proofs are (many-one) Karp-reductions.)

Outline of the talk. In this talk we show the following:

(1) How a prover can give a zero-knowledge proof that he knows a Hamilton cycle in a graph. Since the proof is zero-knowledge, the prover does not give the verifier the slightest idea how to construct that cycle. The zero-knowledge proof is interactive (the prover breaks his proof up into pieces and the verifier requests to see a particular piece) along the lines of GMW's proof. It is, however, more efficient than GMW in terms of the number of requests the verifier must make to achieve any fixed level of confidence. To ensure that a cheater will pass the test with probability of cheating $\leq 1/2^k$, we require just k requests rather than the $k \cdot E$ requests required by GMW for graphs with E edges.

(2) How a prover can give a zero-knowledge proof that a graph G is 3-colorable. This serves to show that the Hamilton cycle problem is not special so far as zero-

knowledge is concerned. Again, the standard proof of 3-colorability is broken into just 2 pieces in every round, though at the price of polynomial growth in each piece. The reader who knows the GMW proof may find it instructive to compare their proof with ours.

(3) How the proof of any theorem whatsoever (e.g., Fermat's Last Theorem) whose proof has been formalized in a standard logical system (such as Whitehead and Russell's *Principia Mathematicae*), together with any integer upper bound on the length of the proof, can be translated into a zero-knowledge proof. The zero-knowledge proof shows that the theorem is very probably true and that the prover almost certainly knows a proof in the given logical system. It gives away no other information whatsoever.

Some zero-knowledge proofs. Let G be a graph. A *Hamilton cycle* in G is a cycle that passes through all the nodes of G exactly once. We show how a prover can convince a verifier that he knows a Hamilton cycle in graph G without giving the slightest additional clue about how to construct that cycle.

The theorem to be given a zero-knowledge proof is one of a class of theorems asserting the existence of a Hamilton cycle in a graph. Although we do not formalize it, the logical proof system in which the theorem is proved is one in which each proof is just a sequence of edges in the graph. If the edges form a cycle through all the nodes of G, then the proof (that G has a Hamilton cycle) is valid; otherwise, it is not.

In the following protocols, we assume that lockable boxes are available to the prover, and that only the prover has the key. Instead of locking information in a box, however, one can encrypt it. One-way functions serve this purpose, providing us the equivalent of digital lockable boxes. The one-way functions make it possible to pursue the following interactive protocol entirely on paper rather than by using the hardware of lockable boxes and keys.

A zero-knowledge protocol for proving that a graph G has a Hamilton cycle. The protocol is interactive and probabilistic. It is probabilistic because (1) both prover and verifier must have the capability of generating sequences of independent unbiased random bits, and (2) the successful outcome of the protocol ensures to the verifier that the prover is *probably* not cheating. On the other hand, the protocol absolutely—not just probably—guarantees the prover that no hint of the proof is divulged to the verifier.

Begin.

The n nodes of G are labeled $N_1, \ldots, N_n$.

Prover: Fix one Hamiltonian cycle.

The protocol has k rounds. Each round proceeds as follows:

Begin.

Prover: In secret (i.e., without letting the verifier know what you are doing), encrypt G with the boxes. Do this by randomly mapping n labeled nodes $N_1, \ldots, N_n$ 1-1 into n labeled boxes $B_1, \ldots, B_n$, in such a way that every

one of the $n!$ permutations of the nodes into the boxes is equally probable. For every pair of boxes (B_i, B_j) prepare a box labeled B_{ij}. This box is to contain a 1 if the node placed in B_i is adjacent (linked by an edge) to the node in B_j; 0 otherwise. All $n + \binom{n}{2}$ boxes are then to be locked and presented to the verifier.

The verifier receives $n + \binom{n}{2}$ labeled boxes. He is now given a choice:

(1) If he wishes, the prover will unlock *all* the boxes. In this case, the verifier may check that the boxes contain a description of G. (For example, if N_1 is adjacent to both N_2 and N_5 but to no other nodes of G, and if N_1 is in B_i, N_2 in B_j, and N_5 in B_k, then there should be a 1 in both B_{ij} and B_{ik}, and a 0 in B_{ix} for every other value of x.)

(2) On the other hand, if the verifier so chooses, the prover will open exactly n boxes $B_{ij}, B_{jk}, B_{kl}, \ldots, B_{l'i}$ (note the cyclic subscripts), those containing the Hamilton cycle that the prover selected in G, and show that these boxes all contain a 1. This proves the existence of a Hamilton cycle (in whatever graph, if any, is represented by the boxes). Since the B_i are not opened, the sequence of node numbers defining the Hamilton cycle in G is *not* revealed.

Verifier: Select one of the 2 options (graph or Hamilton cycle) at random in such a way that both choices are equally likely.

Prover: Open the appropriate boxes.

End.

Verifier: *Accept* the proof if the prover complies and, in every case, correctly exhibits either the requested G or the requested Hamilton cycle. Otherwise, *reject* the proof.

End.

THEOREM 1 (PROVER PROBABLY CANNOT CHEAT VERIFIER). *If the prover does not know a proof of the theorem, his chances of convincing the verifier that he* does *know a proof are* $\leq 1/2^k$ *when there are k rounds.*

PROOF. If the prover does not know a proof, then to pass the test, he must guess in advance what the provee will request. He fails the test if he guesses wrong even once. Q.E.D.

THEOREM 2 (VERIFIER CANNOT CHEAT PROVER). *The verifier gets not the slightest hint of the proof (other than that "the theorem is true and the prover knows a proof in the given logical proof system"). In particular, the verifier cannot turn around and prove the theorem to anyone else without proving it from scratch himself.*

PROOF. (1) When the prover reveals G, what does the verifier get? Just one of the $n!$ random mappings of the n nodes of G into n labeled boxes, each instance

of G having exactly the same probability as any other. The verifier could have constructed such instances for himself with the same probability distribution. So the prover is not giving the verifier any additional information.

(2) When the prover reveals the Hamilton cycle, what does the verifier get? Just a random n-cycle, every n-cycle being exactly as likely as any other. This is because (a) the prover is required to select a particular Hamilton cycle in G and to always reveal *this particular* cycle when so requested, and (b) every permutation of the n nodes into the n boxes is equally likely. Thus the verifier is being shown a random cycle. He could have created random cycles with this uniform distribution himself. Q.E.D.

The above theorem does *not* prove that the protocol is zero-knowledge. The formal definition of zero-knowledge requires one to show that a verifier can simulate the prover, that is, take the prover's part in the dialogue with the verifier. If so, then any efficient[3] probabilistic algorithm that enables the verifier to extract useful information from his conversation with the prover could just as well be used *without* the prover to obtain that information efficiently. Here is how a proof of zero-knowledge would go:

THEOREM 3. *The protocol above for proving that a graph G has a Hamilton cycle is zero-knowledge.*

PROOF. Suppose the verifier has an efficient probabilistic algorithm A to extract useful information from his conversation with the prover. Then the verifier can use his algorithm to extract the information even without the aid of the prover. In each round he does the following:

Begin.

Verifier simulates the *prover*: The verifier flips a fair coin and, according to the outcome of the coin, encrypts either the graph G or an arbitrary n-cycle. G is (randomly) encrypted the same way the prover would have done so. A cycle is (randomly) encrypted just the way the prover would have encrypted an n-cycle (in G). Then, acting as prover, he presents the encrypted information to the verifier. Now he takes the other side.

Verifier simulates the *verifier*: The verifier uses his algorithm A to compute (perhaps probabilistically) whether to request a graph or a cycle. Because the algorithm has no way to guess with any advantage whether the boxes contain a graph or a cycle, there is a 50% chance that A requests an option (graph or cycle) that the verifier, in the guise of prover, can supply. If not, the verifier backs up algorithm A to the state it was in at the start of this round and restarts the entire round (verifier simulating the *prover*).

End.

In an expected 2 passes through each round, the verifier will obtain the benefit of algorithm A *without* the help of the prover. Thus the algorithm does

[3] An *efficient* (probabilistic) algorithm is one that computes its output in (expected) time polynomial in the length of the input.

not help the verifier do something *with* the prover in expected polynomial time that he could not as well have done *without* the prover in expected polynomial time. Q.E.D.

What is the difference between Theorems 2 and 3? Theorem 2 asserts that the verifier gets no hint of the proof of a theorem from the protocol (though he may get other information). Theorem 3 asserts that the verifier not only gets no hint of the proof but actually gets no information (that he couldn't equally well have generated efficiently for himself) whatsoever. It may be helpful to observe that a proof of zero-knowledge will be difficult if not impossible to obtain (i.e., I do not know how to obtain it) if the protocol for proving that G has a Hamilton cycle is modified so that its rounds are executed in parallel. In *parallel* means that the prover first presents all k graphs to the verifier, then the verifier makes his k requests all at once, and finally the prover opens the requested boxes. To prove that this parallel protocol is zero-knowledge is difficult because it is unclear how the verifier can simulate the prover's role in this interaction efficiently.

The Hamilton cycle problem is not the only one with zero-knowledge proofs. In fact, any logical proof of length n can be split into two pieces of length polynomial(n) along the lines shown above, so that k rounds will catch all but 1 in 2^k attempts to cheat. Moreover, as GMW have shown, and as we indicate in our own way in Theorem 4, the process of transforming logical proofs into zero-knowledge proofs can be entirely mechanized so that a computer program could do it efficiently. We now give another simple example of how to transform a standard proof, in this case a proof of 3-colorability, into a zero-knowledge proof. The reader who knows GMW's method for breaking up a proof of 3-colorability into E pieces, E being the number of edges in G, may find it interesting to compare that protocol to ours, which breaks a proof into just 2 pieces.

A zero-knowledge protocol for proving that a graph G is 3-colorable. The n nodes of G are labeled $N_1, \ldots, N_n$. The colors of the nodes will be red, white, and blue. To start, the prover knows a proper 3-coloring of G, which we call the "standard" 3-coloring. If node N_i is colored red, we call it N_i^R; if white, N_i^W; if blue, N_i^B. A triangle might therefore have the proper coloring scheme N_1^R, N_2^B, N_3^W.

Begin.

The protocol has k rounds. Each round proceeds as follows:

Begin.

Prover: Prepare $3n$ *pairs* of boxes $\langle B_1^c, B_1\rangle, \langle B_2^c, B_2\rangle, \ldots, \langle B_{3n}^c, B_{3n}\rangle$. Without revealing to the verifier what you are doing, randomly map $3n$ nodes $N_1^R, \ldots, N_n^R, N_1^W, \ldots, N_n^W, N_1^B, \ldots, N_n^B$ 1-1 onto the $3n$ pairs of boxes. Do this in such a way that every one of the $(3n)!$ permutations mapping the $\{N_i^x\}$ onto the $\{\langle B_j^c, B_j\rangle\}$ is equally probable. Next, insert $N_{node-number}^{color}$ into the associated $\langle B_j^c, B_j\rangle$ by putting *color* into B_j^c and *node-number* into B_j.

For every pair of number-containing boxes (B_i, B_j), prepare a box labeled B_{ij}. This box is to contain a 1 if the prover's proper 3-coloring of G has colored the node of G in B_i with the color in B_i^c, the node of G in B_j with the color in B_j^c, and if the node in B_i is adjacent in G to the node in B_j; 0 otherwise.

All boxes are then to be locked and presented to the verifier.

The verifier is now given a choice:

(1) If the verifier so wishes, the prover will unlock *all* the boxes B_{ij} and *all* the number-containing boxes B_i, but *none* of the color-containing B_i^c. In this way, the prover reveals the graph G without revealing its coloring. The verifier checks that the boxes contain a correct description of G.

(2) On the other hand, if the verifier so chooses, the prover will open the $3n$ boxes $\{B_i^c\}$ to reveal the colors they contain, and then open just those boxes B_{ij} (joining B_i to B_j) such that B_i^c contains the same color as B_j^c. The opened boxes B_{ij} will all contain a 0 if and only if any 2 nodes that are colored the same are *not* adjacent in the graph represented by the boxes. This allows the verifier to check correct 3-coloring.

Verifier: The correct thing to do is select one of the 2 options at random in such a way that both choices are equally likely.

Prover: Open the requested boxes.

End.

Verifier: *Accept* if the prover correctly complies with all requests; *reject* otherwise.

End.

This protocol is zero-knowledge, and the probability that a fake prover can cheat a verifier is $1/2^k$.

A zero-knowledge protocol for proving any theorem. Impagliazzo [I] has given direct zero-knowledge protocols along the lines shown above for several problems including the *subset sum* problem, *satisfiability*, and the very general problem of proving that a given polynomial-time nondeterministic Turing machine accepts a given input.

THEOREM 4. *Given any logical* proof system (*such as Russell and Whitehead's very general system within which it is generally acknowledged that all mathematical theorems can be formulated and proved*), *given any* theorem *provable in that system, and given an* upper bound, L, *on the length of some proof of the theorem in the system, it is possible to efficiently transform that proof into a* zero-knowledge proof *of the theorem. This is an interactive probabilistic protocol*

whereby the prover persuades the verifier that with high probability,

(1) *the theorem has a proof in the given proof system of length* $\leq L$, *and*

(2) *the prover knows such a proof.*

The probability that a cheater, i.e., a prover who does not know a proof, will pass this test $\leq 1/2^k$ *for a protocol with* k *rounds.*

IDEA OF PROOF. The *proof system* is defined by a nondeterministic TM (Turing machine) which, on input ⟨statement of *theorem*, 1^n⟩, guesses a proof of the *theorem* of length $\leq n$, checks if it is a valid proof within the system, and *accepts* if it is, *rejects* if not.

The prover gives the verifier a zero-knowledge proof that he, the prover, knows an accepting path for this TM for some n. The protocol for this is along the same lines as for *Hamilton cycle* in a graph [I]: one splits the computations into two pieces. The integer n must be chosen by the prover to be an upper bound on the length of his proof in the system. Q.E.D.

REFERENCES

[**GMR**] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, *The knowledge complexity of interactive proof-systems*, Proc. 17th ACM Sympos. on Theory of Computing,[4] 1985, pp. 291–304.

[**GMW**] Oded Goldreich, Silvio Micali, and Avi Wigderson, *Proofs that yield nothing but the validity of the assertion, and the methodology of cryptographic protocol design*, presented at a Workshop on Probabilistic Algorithms (Marseille, March 1986) organized by C. P. Schnorr; Proc. 27th IEEE Sympos. on the Found. of Computer Science,[5] 1986, pp. 174–187.

[**I**] Russell Impagliazzo, *A collection of direct zero-knowledge protocols for NP-complete problems*, Berkeley Computer Science Division Rept., Univ. of Calif., Berkeley, Calif., 1986.

[**BC**] Gilles Brassard and Claude Crepeau, *Non-transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond*, Proc. 27th IEEE Sympos. on the Found. of Computer Science,[5] 1986, pp. 188–195.

[**BCR**] Gilles Brassard, Claude Crepeau, and Jean-Marc Robert, *Information theoretic reductions among disclosure problems*, Proc. 27th Sympos. on the Found. of Computer Science,[5] 1986, pp. 168-173.

[**Y**] Andrew Yao, *How to generate and exchange secrets*, Proc. 27th Sympos. on the Found. of Computer Science,[5] 1986, pp. 162–167.

UNIVERSITY OF CALIFORNIA, BERKELEY, CALIFORNIA 94720, USA

[4]The ACM Symposium on Theory of Computing (STOC) is held yearly in May. Its proceedings, which are refereed by a distinguished 10-member committee, contain the best of the previous six months computer science research in all areas of theoretical computer science. Proceedings may be ordered from the ACM Order Dept., P.O. Box 64145, Baltimore, MD 21264.

[5]The IEEE Symposium on Foundations of Computer Science (FOCS) is held yearly in October. Its proceedings, like STOC, are refereed by a distinguished 10-member committee. They contain the best of the previous six months computer science research in all areas of theoretical computer science. Proceedings may be ordered from IEEE Comp. Soc., P.O. Box 80452, Worldway Postal Center, Los Angeles, CA 90080.