Zero Knowledge Protocols, 15-451, Fall 2000

Daniel Sleator

- Zero Knowlege Proofs
- Graph Isomorphism
- Sharmir's authentication protocol
- Hamiltonian Cycles

In the 80s, theorists explored the power of "interaction". They developed a rich addition to complexity theory. Here we'll just scratch the surface of what came out of that work.

1 Zero Knowledge

Intuitively, the general framework of Zero Knowledge is this this: Suppose somebody named P (the prover) who has the solution S to a problem. And we have somebody else named V (the verifier). They engage in a protocol, after which V is convinced that P knows the solution to S. (This is kind of like proving that a problem is in NP.) Here's the interesting thing: the protocol is zero-knowlege, that is, V knows there's a solution and knows that P has the solution but V has absolutely no new information about what that solution is.

In this lecture, I'll explain several simple examples of zero-knowledge proofs. My exposition will be pretty informal and intuitive. This will not be terribly rigorous. All this can be made completely rigorous (and somewhat tedious) by setting up proper definitions.

In order for such a protocol to work correctly it must have two properties:

(1) If P does not have the solution S to a problem, then he will not be able to successfully carry out the protocol (V will catch him trying to cheat with high probability)

(2) The protocol reveals nothing to V. Roughly speaking, the way we'll show this is as follows. Suppose V engages in the protocol. V writes down a transcript of the protocol. This transcript involves randomness, and can be simulated by a random process. The key point is this: There's a way for V to generate transcripts taken from that same distribution, ALL BY HERSELF. This guarantees that V learns nothing about the hidden information.

2 Graph Isomorphism

I'll start by explaining how this works for the problem of graph isomorphism. Graph isomorphism is simply this: Given two undirected graphs G_1 and G_2, are the two graphs isomorphic? (Two graphs are isomorphic if there is a one to one, onto mapping $f()$ that maps vertices x of G_1 to vertices $f(x)$ of G_2, and preserves edges. That is, (x, y) is an edge of G_1 if and only if $(f(x), f(y))$ is an ege of G_2.)

Graph isomorphism is clearly in NP. However, currently there is no known polynomial time algorithm to solve this problem, and the probelm is not known to be NP-complete either. (Actually, there are known polynomial time algorithms if the degree of the graph is bounded by a constant.)

Suppose for some graph isomorphism problem involving G_1 and G_2 that P has the isomorphism showing that they are isomorphic. Suppose P wants to prove to V that P has knows this isomorphism, but does not want to reveal it to V. Here is a protocol to do this:

(1) P randomly permutes the vertex names of G_2, and constructs a new graph G_3 using these names. P sends this graph to V.

(2) V now flips a coin. If the result is heads, he asks P to prove that G_3 is isomorphic to G_1. If it comes up tails, he asks P to prove that G_3 is isomorphic to G_2.

(3) Now P complies with V's request. This is easy to do. If he's asked to show that G_3 is isomorphic to G_2 he just supplies the map that he generated in step (1). If he is asked to show that G_3 is isomorphic to G_1 he just computes the necessary map by composing $f()$ (which maps G_1 to G_2) with the map he used in step (1).

(4) Now V checks to make sure that the map supplied by P in step (3) works.

Let's quickly show that this is a zero-knowlege protocol. First of all, if P does not know the isomorphism between G_1 and G_2, then he will be caught by V with probability at least 1/2. It's easy to see this. If he knows both isomorphism mapping G_1 to G_3 and the isomorphism mapping G_2 to G_3, then he can easily compute the isomorphism mapping G_1 to G_2. Therefore he cannot know those two isomorphisms if he's cheating, and he will be caught with probability at least 1/2.

On the other hand, V does not learn anything. How do we know this? We know this because she can simulate transcripts of the protocol from exactly the same distribution. Here's how she does this. She first flips the coin, and remembers the result. If the result is heads, then she generates G_3 by randomly permuting G_2, otherwise she generates G_3 by randomly permuting G_1. (She remembers the permutation used.) Note that in either case, G_3 is just being generated by sampling exactly the same distribution. The rest of the protocol is now easy for V to write down. (The use of the coin in step (1) only allows V to supply the later parts of the protocol (step (3)). The key point is that it does not change the distribution from which the protocol transcript is being sampled.)

3 Shamir's Authentication Protocol

Here's another very nice application of zero knowledge. This is a protocol proposed by Adi Shamir (of RSA fame) to do authentication. Authentication is simply the task of proving your identity.

Here is a summary of the Shamir's authentication protocol.

A trusted government agency chooses primes p and q, computes $n = pq$, publishes n and self-destructs. This is a one-time initialization for the entire system. (All numbers used throughout this section are modulo n. The symbol $\sqrt{x}$ denotes any number y such that $y^2 \equiv x \pmod{n}$)

Every user U of the system generates a number s_U, which is always kept secret. User U computes $r_U = (s_U)^2$ and publishes this number. This initialization that is done once for each user.

In a typical application of the system there are two users P and V having a conversation, and user V wants to be sure he is talking to P. In order for P to convince V that she is indeed talking to P, P will prove that he knows s_P without revealing it by using the following protocol:

(1) P generates a random number x, computes $y = x^2$, sends y to V.

(2) V flips a coin. If it's heads she asks P for $\sqrt{y}$, if it's tails he asks P for $\sqrt{y \cdot r_P}$.

(3) P answers with x or $x \cdot s_P$, depending on V's request.

(4) V verifies that P correctly answered, by squaring P's answer.

The proof that this is zero-knowledge is very similary to the one above. First let's prove that if P can carry out this protocol without getting caught with probability 1/2 then he must know the secret s_P. Let's say P generates y by some means other than what's listed in step (1). Now he's going to be asked for either $\sqrt{y}$ or $\sqrt{y \cdot r_P}$. He better know both of these, or he's going to be caught with probability at least 1/2. But if he knows both of these, he can simply invert $\sqrt{y}$ and compute $\sqrt{y^{-1}}$ and multiply this by $\sqrt{y \cdot r_P}$. The result is a square root of r_P, which is the secret. (Well, it might be a different square root of r_P, but that doesn't matter. To know this he either broke into Ps computer or he managed to compute a new square root, which is as hard as factoring.)

We also must prove that V can generate transcripts taken from the same distribution all by herself. As in graph isomorphism, V generates the coin flip first. If it's heads, she does step (1) just as the protocol describes. If it's tails, she generates a random x and lets $y = r_P^{-1}x^2$. Step (3) is now easy for her to simulate. In case of either heads or tails, she returns x. It's easy to see that this satisfies the correctness of the protocol.

Why does this simulation generate transcripts with the same distribution as a real instance of the protocol? This is because no matter which method is used by V to simulate step (1), the distribution of numbers generated is exactly the same. Specifically, it's just a random number from the field Z_n^*.

4 Hamiltonian Cycles

The Hamiltonian cycle problem is the following: Given an undirected graph G is there a cycle that uses each vertex exactly once. We'll show that under a slightly different scenario, a prover who knows a Hamiltonian cycle can prove it to a verifier without revealing anything.

A one-way permutation can be converted into a cryptosystem with the following properties: (1) e(x,r) is a polynomial time computable function of x and r with the property that e(x,s)=e(y,s) implies that x=y. (2) We also need to be able to compute e(x,r) in polynomial time, (3) and we need it to be difficult to compute x given e(x,r), and (4) given a number M, we need it to be hard to find any x and r such that e(x,r)=M.

It is useful to view the function e() as a sort of envelope. It has the property that Alice can put a message m into an envelope and give it to Bob (by giving him e(m,r) for random value of r). Alice may later open the envelope for Bob by revealing m and r. However she cannot change the contents of the envelope, for this would require finding an m' and an r' such that e(m,r) = e(m',r') which is contrary to the definition of e. It is also impossible for Bob to open the envelope by himself, because he is a polynomial time bounded machine. From now on when we say "put a message into an envelope" we mean doing just such a transformation on the message.

[Note that it's now easy to play rock-scissors-paper over the phone]

Suppose there's a graph G, and P knows a Hamiltonian cycle of G. We want to give a zero-knowledge protocol by which P can show V that there is such a cycle, without revealing it. Here's the protocol:

(1) P generates a random permutation of G and calls it G'. P now encrypts G' in the following way. For every pair of vertices (i,j) an envelope m(i,j)=e("(i,j) is in G'", r) (where r is random) or m(i,j)= e("(i,j) is not in G'", r) is generated, whichever is

appropriate. Let M be the encryption of the entire graph G'. M is transmitted to V. In other words, all the edges of G' are put in envelopes and given to V.

(2) V now flips a coin, and asks one of the following questions depending on the outcome: "show me that M is an encrypted version of a permuted version of G" or "show me the subset of edges in M that form a Hamiltonian cycle".

(3) If the first question is asked, then P reveals G', the permutation that relates it to G, and all of the random keys (values of r) used to encrypt G'. (P opens all of the envelopes.) If the second question was asked, P reveals the subset of the m(i,j) that are involved in the cycle, and also random keys for these encryptions. (P opens a subset of the envelopes corresponding to the cycle.)

(4) V checks that P answers are valid.

As usual, there are two things to prove. Actually, it's not possible to give a rigorous proof without more definitions. For example, we need the notion of when two distributions are polynomially indistinguishable. I won't do this. All of these issues can be brushed under the rug by thinking in terms of the envelope analogy.

Suppose the bogus prover P' is trying to cheat, and the graph is non-hamiltonian or P does not know the hamiltonian cycle. The bogus prover must make a decision of what to encrypt. Suppose it follows the protocol and encrypts a permuted version of G. In this case with probability 1/2 V will ask to see the Hamiltonian cycle, and P' will not be able to successfully reveal it. (That is, P' will not be able to supply the subset of the m(i,j) that form a cycle along with the appropriate decryption keys, since there is no such cycle.) Finally, suppose P' violates the protocol and the message it sends is not a correct encryption of a permuted version of G. Then with probability (1/2) V will ask to see the entire graph, and will discover the deception.

Now, we need to verify that information is not being leaked from anybody who is following the protocol (P or V) to anybody else (an evesdropper, or a bogus player).

Say that P is playing with a bogus verifier V'. To show that no information is being leaked to V', we'll show that V' can be modified to efficiently generate random instances of the protocol transcript from the same distribution, without any help from P. V' will be made to play the roll of the prover, as well as itself. First it flips a coin. The outcome tells it if it is going to ask its simulated P for a cycle, or for the whole graph. In the latter case, it does exactly what P would have done, and sends itself the result. In the former case it generates a graph that is a cycle, permutes the vertices randomly, and encrypts this graph (in the manner used by P to encrypt G). Its simulated version of P sends this graph. [The distribution of transcripts generated by this scheme is not the same as the real thing. However, they are the same as far as the computation limited V is concerned, because of the cryptographic properties of e(). If V could distinguish between transcripts generated by these two different methods, then this can be shown to violate the cryptographic assumption.] The rest of the simulated protocol is straightforward.

Once we accept the assertion that that the transcripts generated in this manner are indistinguishable from the real thing, the proof is complete. This proves that no information could possibly have been gained by V'. It also proves that an evesdropper cannot obtain any information.