COM S 687 Introduction to Cryptography November 7, 2006

Lecture 18: Zero-knowledge proofs

Instructor: Rafael Pass *Scribe: Ari Rabkin*

Proving Identity

Suppose you wanted to prove your identity to a serve. You certainly don't want to just send a password across the wire, because any adversary who intercepts the message can impersonate you. It would be much better if we could prove identity in such a way that a passive adversary cannot subsequently impersonate us. One way to solve the problem is to use a signature. Consider the following protocol.

User	$\rightarrow$	Server	Login
Server	$\rightarrow$	User	"Server name", r
User	$\rightarrow$	Server	$P_k, S_k\{$"Server name", r$\}$

Here, "User" is trying to prove to "Server" that she holds the private key S_k corresponding to public key P_k; r is a nonce chosen at random from $\{0,1\}^n$. We are implicitly assuming that the signature scheme resists chosen-plaintext attacks. Constraining the text to be signed in some way (requiring it to start with "server") helps.

This protocol has a subtle consequence. The server can prove that the user with public key P_k logged in, since the server has, and can keep, the signed message $S_k\{$"Server name", r$\}$. This property is sometimes undesirable. Imagine that the user is accessing a politically subversive website. With physical keys, there's no way to show afterwards whether the key was used, and it might be nice to have a digital version of that property.

Zero-knowledge in general

This seems almost paradoxical, to prove something in such a way that the thing proved can't be established subsequently. However, zero-knoweldge proofs are a way of achieving exactly this. The key insight is for the server to ask the client to solve a puzzle the server knows the answer to. Thus, the server cannot convince a skeptical third party of anything afterwards, since the server could have created a transcript of the whole conversation.

Until now in the course we've worried about an honest Alice who wants to talk to an honest Bob, in the presence of a malicious Eve. Here, we're worried about an Alice and Bob who don't trust each other.

Zero-knowledge proofs are a way for Alice (the prover) to convince Bob (the verifier) that

a string x is in language L, but in such a way that Bob learns nothing else. For instance, it might be useful in a cryptographic protocol for Alice to show Bob that a number N is the product of exactly two primes, but without revealing anything about the two factors.

How can this be possible? Let's think about the following toy example. Do any of you remember those "Where's Waldo?" children's books? Each page is a large complicated illustration, and somewhere in it there's a small picture of Waldo, in his sweater and hat, and the reader is invited to find him. Sometimes, you wonder if he's there at all.

You might want to be able to prove that Waldo is in the image, without showing where. There is indeed a zero-knowledge proof for Waldo-presence. Take a large sheet of newsprint, cut a Waldo-sized hole, and overlap it on the "Where's waldo" image, so that Waldo shows through the hole. This shows he's somewhere in the image, but there's no context to show where.

Here's a slightly less toy example. Suppose you want to prove that two pictures or other objects are distinct, without revealing anything about the distinction. Then you can have the verifier give the prover one of the two, selected at random. If the two really are distinct, then the prover can reliably say "this one is object ", or "this is #2". If they were identical, this would be impossible.

Zero-knowledge for graph isomorphism

Let's see a more mathematical version of a zero-knowledge proof, in particular, this is a zero-knowledge protocol for graph isomorphism.

Suppose there are two graphs G_1 and G_2, and that the prover wants to convince the verifier that they are isomorphic; the prover has access to the permutation σ such that $\sigma(G_1) = G_2$. The prover generates a new graph H by permuting the vertexes of G_1 according to the random permutation π^{-1} (so $\pi(H) = G_1$.

Prover	$\rightarrow$	Verifier	H
Verifier	$\rightarrow$	Prover	$b \in \{0, 1\}$
Prover	$\rightarrow$	Verifier	if $b = 1$, π else $\sigma\pi$

The verifier then applies the given permutation to h and checks if the result is identical to G_i. If the two graphs are isomorphic, this will always be the case, because $\pi(H) = G_1$ and $\sigma(\pi(H)) = \sigma(G_1) = G_2$. If the graphs are not isomorphic, then this protocol will succeed with probability at most one half; we'll prove this next time, but the following argument will give some intuition. If G_1 and G_2 are not isomorphic, then H can be isomorphic to at most one of them. Thus, since b is selected at random after H is fixed, then with probability $\frac{1}{2}$ it will be the case that H and G_i are not isomorphic. This protocol can be repeated many times (provided a fresh H is generated), to drive the probability of error as low as desired.

Interactive proof

We will now formally define zero-knowledge interactive proof; but let's first remind ourself how non-interactive proof is defined.

Definition 1 (NP-verifier) *V is an NP-verifier for a language L if:*

1. *Completeness: $x \in L \Rightarrow$ there exists a proof pi so that $V(x, \pi) = 1$*
2. *Soundness: $x \notin L \Rightarrow \forall \pi V(x, \pi) = 0$*
3. *V is polynomial time in $|x|$.*

It's not too hard to show that this represents exactly the complexity class NP as defined in terms of nondeterministic turing machines.

An Interactive proof differs in two key respects. First, it is produced by the interaction between two machines, and second, because we allow a small soundness error: the error has some low probability over the space of interactions.

Definition 2 (Interactive proof) *P, V is an interactive proof for a language L if:*

1. *Completeness: $\forall x \in L \exists y \in \{0,1\}^*$ so that $Pr[Out_v[P(x, y) \leftrightarrow V(x)] = 1] = 1$*
2. *Soundness: There exists a negligible ϵ so that $\forall x \notin L$ and for all turing machines P^* and $\forall y \{0,1\}^*$ it holds that*

$$Pr[Out_v[P*(x, y) \leftrightarrow V(x)] = 0] > 1 - \epsilon(|x|)$$

3. *V is PPT in $|x|$.*

If we are only interested in small x, we can arrange to define x with some sort of padding to drive the probability of error own. We also can relax the definition and replace the $1 - \epsilon(|x|)$ with some constant like $\frac{1}{2}$ and repeat the interactive proof many times.

We don't stop to formalize $\leftrightarrow$; it represents the two machines exchanging bit strings. The definition is a bit tricky since it needs to exclude infinite rounds of message exchange and so that $P*$ always halts.

The set of languages for which interactive proofs exist is the complexity class IP. Trivially, $NP \subset IP$, since if a non-interactive proof exists, P can just give V the NP proof string.

However, it is believed that $IP \not\subset NP$, since there is an interactive proof for graph non-isomorphism, which is a problem in co-NP. The interactive proof is just the one mentioned above for distinguishing objects.

Suppose we have two non-isomorphic graphs G_1 and G_2

Verifier	$\rightarrow$	Prover	H isomorphic to G_i, $i \in \{0, 1\}$
Prover	$\rightarrow$	Verifier	b' where H is isomorphic to G'_b

The verifier checks if $b = b'$, and if so, accepts. If the graphs really are isomorphic, this will only be true half the time. Note that the prover is able to determine which of the graphs H is isomorphic to, since the prover can do arbitrary computation.

Conclusion

This shows that IP includes things that are believed to not be in NP. We're not going to prove it, but it turns out that IP is equal to $PSPACE$.

Next time, we're going to focus on the case where the prover is constrained to be polynomial time, but add the zero-knowledge criterion. We're ultimately going to show how to do zero-knoweldge for graph 3-coloring (similar to isomorphism). Since 3-coloring is NP-complete, this establishes that there are zero-knowledge proofs for every language in NP.