Lecture 14 - Zero Knowledge

Boaz Barak

November 8, 2005

Proofs In mathematics and in life, we often want to convince or prove things to others. Typically, if I know that X is true, and I want to convince you of that, I try to present all the facts I know and the inferences from that fact that imply that X is true.

Example: I know that 26781 is not a prime since it is 113 times 237, to prove to you that fact, I will present these factor and demonstrate that indeed $113 \times 237 = 26781$.

Zero Knowledge Proofs A typical byproduct of a proof is that you gained some knowledge, other than that you are now convinced that the statement is true. In the example before, not only are you convinced that 26781 is not a prime, but you also learned its factorization.

A *zero knowledge proof* tries to avoid it. In a zero-knowledge proof Alice will prove to Bob that a statement X is true, Bob will completely convinced that X is true, but *will not learn anything as a result of this process.* That is, Bob will gain *zero knowledge.*

Zero knowledge proofs were invented by Goldwasser, Micali and Rackoff in 82 (the paper, which we'll call GMR, appeared in FOCS 85). Zero-knowledge proofs (and interactive proofs in general, also introduced in that paper) turned out to be one of the most beautiful and influential concepts in computer science, with applications ranging from practical signature schemes to proving that many NP-complete problems are hard even to approximate.

Motivation One motivation is philosophical: the notion of a proof is basic to mathematics and to people in general. It is a very interesting quesiton whether a proof inherently carries with it some knowledge or not. Another motivation is practical: zero knowledge proofs have found many applications. Most practical applications fall into two types:

- *Protocol design.* A *protocol* is an algorithm for interactive parties to achieve some goal. For example, we saw the Diffie-Hellman *key exchange protocol.* In that protocol, we assume that both parties follow the instructions of the protocol, and the only thing we worried about was a passive easvesdropping adversary Eve.

  However, in crypto we often want to design protocols that should achieve security even when one of the parties is "cheating" and not following the instructions. This is a hard problem since we have no way of knowing the exact way the party will cheat.

  One way to avoid cheating is the following: If Alice runs a protocol with Bob, to show Bob she is not cheating she will send Bob all the inputs she had, and then Bob can verify for himself that if one runs the prescribed instructions on these inputs, you will indeed get the outputs (messages) that Alice sent.

  However, this way will be often unacceptable to Alice: the only reason they are running this protocol is that they don't completely trust each other, and the inputs she had may be secret, and she does not want to share them.

Zero-knowledge offer a solution to this conundrum. Instead of sending her inputs, Alice will prove in zero knowledge that she followed the instructions. Bob will be convinced, but will not learn anything about her inputs he did not know before.

In fact, we will see that it is possible to do this in a very general way, applying essentially to all cryptographic protocols. Thus, a general technique (invented by Goldreich, Micali and Wigderson , GMW) is to design a cryptographic protocol first assuming everyone will follow the instructions, and then "force" them to follow instruction using a zero knowledge proof system.

- *Identification scheme.* A somewhat simpler and more direct application is to identification schemes. Suppose that we want to control access to the CS department. One way to do that is to give authorized people a secret PIN number, and have a box on the door where type the PIN number on that box. (A more convenient but essentially equivalent way is that the authorized people have a card that transmits the PIN number to the box.)

  A drawback of this approach is that the box remains outside all the time, and if someone could examine the box, they would perhaps be able to view its memory and extract the secret keys of all people. Thus, from a security standpoint, it is much better if the box contains no secret information at all, and even if someone installed a "fake box" they would not learn anything about the secret PIN.

  Zero-knowledge proofs help us in the following way.

  1. Have the box contain an *instance* of a hard problem. For example, the box can contain a composite number n without its factorization.
  2. Give the authorized people the *solution* to the instance. For example, they can get the factorization of n to $n = p \cdot q$.
  3. The authorized people will *prove* to the Box they know the factorization in zero knowledge. (Of course, there is a question of how do you prove that you know something, but this was also shown by GMR (and further developed by others.)

Plan: Zero knowledge is an elusive concept in the sense that not only it's not clear how to construct such things, it's also not clear even how to *define* such creatures. We will start by explaining some of the generalizations to the notion of proofs that are needed. Then, we will give an example for a zero knowledge proof for a particular family of statements (or in more standard terms, for a particular language). We will then talk about the definition of zero knowledge proofs. Next lecture we will see that the extremely useful fact, shown by GMW, that *any* NP-statement can be proven in zero knowledge.

Interactive probabilistic proofs. The standard mathematical notion of a proof is the following: you have axioms and inference rules, and the proof for x is a string π that derives x from the axiom using the inference rules.

A proof system is *sound* if you can never derive false statements using it. Soundness is a minimal condition, in the sense that unsound proof systems are not very interesting.

A proof system is *complete* if you can prove all true statements using it. Similarly, we ay it is complete for a family L of true statements, if you can prove all statements in L using it.

Thus the traditional notion assumes that the proof π is a static string that was written down somewhere and anyone can verify. A valid proof gives absolute certainty that the statement is true.

GMR generalized this notion to think of a proof as a game between a prover and a verifier. The game can be interactive, where the verifier asks questions and the prover answers, and the goal of the game is for the prover to convince the verifier that the statement is true.

They even further generalized it to the notion of a *probabilistic* proof system. That is, the verifier does not convinced with absolute certainty that the statement is true but "only" with 99.999% certainty. What is crucial here is that no matter what the prover does and how she tries to cheat, if the statement is false she will fail with this probability.

One example for a probabilistic interactive proof is proving that Alice can distinguish between Coke and Pepsi using the following protocol: Alice turns her back, Bob flips a coin and puts either Coke or Pepsi into a paper cup according the result, Alice tastes and announces whether she thinks it was Coke or Pepsi. If they repeat this k times and Alice always answers correctly then Bob can conclude with $1 - 2^{-k}$ probability that she really can tell the difference.

Example We are going to give an example for a zero knowledge interactive probabilistic proof. The formal definition will be after the example. Similarly to the Coke/Pepsi proof, we will give a basic proof that has soundness error equal to 1/2, but can be amplified to a proof with soundness error 2^{-k} by repeating it k times.

Recall that if n is a number, then $x \in \mathbb{Z}_n^*$ is a *quadratic residue* mod n if there is some s such that $x = s^2 \pmod{n}$. It is believed to be hard to tell whether x is a quadratic residue mod n without knowing the factorization of n.

Some other facts: if n is prime, then $\mathbb{Z}_n^*$ has a generator g and x is a residue iff $x = g^i$ for an even i. This means that if x and y are residues then xy is a residue, but if x is a non-residue and y is a residue then xy will be equal to g^j for an odd j and so will be a non-residue. Because of the chinese remainder theorem the same holds also if n is a composite.

Another thing is that the quadratic residues are a *group*. This means that if x and y are residues then xy is also a residue. This also means that x is a residue and y is a random residue then xy is a random residue. That is, for every $z \in QR_n$, the probability that $xy = z$ is equal to $1/|QR_n|$. The reason is that this is the same as the probability that $y = x^{-1}z$ and since y is chosen at random, this probability is exactly $1/|QR_n|$.

Protocol QR

Statement x is a quadratic residue mod n.

Public input x , n

Prover's (Alice) private input. w such that $x = w^2 \pmod{n}$.

P $\rightarrow$ **V** Alice chooses random $u \leftarrow_{\mathrm{R}} \mathbb{Z}_n^*$ and sends $y = u^2$ to Bob.

P $\leftarrow$ **V** Bob chooses $b \leftarrow_{\mathrm{R}} \{0, 1\}$

P $\rightarrow$ **V** If $b = 0$, Alice sends u to Bob. If $b = 1$, Alice sends $w \cdot u \pmod{n}$.

Verification. Let z denote the number sent by Alice. Bob *accepts* the proof in the case $b = 0$, $z^2 = y \pmod{n}$. In the case $b = 1$, Bob accepts the proof if $z^2 = xy \pmod{n}$.

Analysis We will want to prove three properties about Protocol QR: **completeness**, **soundness** and **zero knowledge**. We will present the formal definition as we go along.

Completeness: Completeness means that whenever x is really a quadratic residue, and Alice is given s such that $x = s^2 \pmod{n}$ and both Alice and Bob follow the instructions, then Bob will accept with probability one. Completeness is often easy to see and this is also the case here.

Soundness: Soundness means that if x is *not* a quadratic residue, then regardless of what Alice does, Bob will reject the proof with probability at least 1/2.

To define and prove soundness, we will need to formalize a bit what it means "regardless of what Alice does". Suppose that Alice is untrusted and possibly cheating. This means that she uses a different *strategy* than the instructions she is supposed to follow in the protocol. We will model the strategy she uses as a function P^* that computes the messages. (Alice can also use of course randomness, but we'll "hardwire" any random coins she needs into the description of P^* to make it into a deterministic function. We can also hardwire into P^* the public and private inputs.)

That is, we think of P^* as follows: on input the empty word, it gives a string y (which is Alice's first message) and on input b it gives a string z (which is Alice's second and last message).

We want to prove the following:

Lemma 1. *For every (possibly not efficiently computable) P^*, and (x, n) such that x is* not *a QR mod n, we have that*

$$\Pr_{b \leftarrow \{0,1\}} [\mathsf{out}_V \langle P^*, V_{x,b} \rangle = \mathsf{accept}] \leq \frac{1}{2}$$

Notation: We use P and V to denote the "honest" (i.e., non-cheating / following instructions) algorithms. We use subscripts to denote the inputs they use which can include public and private inputs and the random coin tosses.

If two interactive algorithms A and B are running a protocol, we denote this execution by $\langle A, B \rangle$. We also use the following notations:

- $\mathsf{out}_A \langle A, B \rangle$ - the output of A after this interaction is finished. Similarly we define $\mathsf{out}_B \langle A, B \rangle$.
- $\mathsf{view}_A \langle A, B \rangle$ - the *view* of A during the interaction: all the messages it received.

Proof. Suppose that $x \notin QR_n$. And let P^* be any interactive strategy for the prover. Denote by y the output of P^* on the empty input. We note P^* is cheating and so need not select y in the same way the honest prover does. We can make the verifier always reject if y is not of the proper length and does not satisfy $gcd(y, n) = 1$. Thus without loss of generality we can assume $y \in \mathbb{Z}_n^*$. What we can *not* assume is that y is a quadratic residue (as would be in the case of the honest prover). However, we know that y is output before P^* sees the query b, and so it is independent of b. We split into two cases:

- **Case 1:** y is in QR_n. That is, $y = u^2 \pmod{n}$ for some $u \in \mathbb{Z}_n^*$. In this case, with probability 1/2 we have that $b = 1$. Denote $z = P^*(1)$ be the prover's last message. The verifier will accept only if $z^2 = xy$. We claim that this is impossible since if $z^2 = xy$ then $zu^{-1} = z^2u^{-2} = xyy^{-1} = x$ but x is not in QR_n! Thus the verifier will reject with probability $\geq 1/2$.

- **Case 2:** y is not in QR_n. In this case with probability $1/2$ we have that $b = 0$. However, if $b = 0$ then the prover has to come up with some z such that $z^2 = y$, which is impossible. Hence also in this case the verifier will reject with probability $\geq 1/2$.

$\square$

Zero knowledge. The zero knowledge property is more tricky to define. For zero knowledge we now think of a possibly cheating verifier V^*. However, the power of the verifier is very limited: he can only send either $b = 0$ or $b = 1$. We want to show that in both cases he gets a random element in $\mathbb{Z}_n^*$, which gives him no information about the quadratic residue of x.

The spirit of the definition is that a proof is zero knowledge if whatever Bob learns, he could have learned by himself without any interaction with Alice. The idea to formalize this is using the notion of a *simulator*. That is, we make the following definition:

Definition 1. A prover strategy P is (T, ϵ)-*zero knowledge* if for every T-time cheating strategy V^* there exists a poly(T)-time non-interactive algorithm S (called the *simulator* for V^*) such that for every valid public input x and private input w, the following two random variables are (T, ϵ)-computationally indistinguishable:

- $view_{V^*}\langle P_{U_m,x,w}, V^*\rangle$. (Where m is the number of random coins P uses
- $S(x)$. (Note that S can be probabilistic and so this is a random variable).

That is, S only gets the *public* input and has no interaction with P, but still manages to output something indistinguishable from whatever V^* learned in the interaction.

Lemma 2. *The prover of Protocol QR is* $(\infty, 2^{-|x|})$*-zero knowledge.*

Proof. Let V^* be a possibly cheating verifier. The simulator S will do the following: (Note that the simulator can depend on V^* and hence in particular can use the strategy V^* in its computation)

1. **Input:** x, n such that $x \in QR_n$. Note that the simulator does not get w such that $x = w^2 \pmod{n}$.
2. Choose $b' \leftarrow_{\text{R}} \{0, 1\}$.
3. Choose $z \leftarrow_{\text{R}} QR_n$.
4. If $b' = 0$, compute $y = z^2$. Otherwise (if $b' = 1$) compute $y = z^2 x^{-1}$.
5. Invoke V^* on the message y to obtain a bit b.
6. If $b = b'$ then output $\langle y, z\rangle$. Otherwise, go back to Step 2.

Initially, it is not even clear this algorithm doesn't loop forever. We make the following claims:

Claim 2.1. *In both cases $b' = 0$ and $b' = 1$, the message y is a random element in QR_n.*

Proof. In the case $b' = 0$ this is obvious. In the case $b' = 1$, y is x^{-1} multiplied by a random quadratic residue, but since x^{-1} is also in QR_n, the result (as we said above) is a random element in QR_n. $\square$

This implies that y is *independent* of b' (since it is the same distribution regardless of whether $b' = 0$ or $b' = 1$). Hence we have that $b = V^*(y)$ is also independent of b' and hence we have $\Pr[b' = b] = \frac{1}{2}$. This means that if we run the procedure for k steps, we will halt with very high $(1 - 2^{-k})$ probability.

We now make the following claim:

Claim 2.2. *The output of the simulator is distributed identically to the view of V^* in an interaction with the honest prover.*

Proof. We already know that the first message y is a random quadratic residue in both cases. Now, let $b = V^*(y)$ and condition on the case that $b' = b$ (which happens independently of y with probability $1/2$) then for both the prover and simulator if $b = 0$ then z is a random root of y and if $b = 1$ then z is a random root of xy. □

This algorithm has small probability of running for a long time. To make into an algorithm that at the worst case makes at most $|x|$ invocations of V^*, we can just stop and output an arbitrary value after more than $|x|$ iterations. We'll introduce at most $2^{-|x|}$ statistical distance this way.

□