1 Lecture 8: Intro to Zero-Knowledge Proofs (ZKP)

The central idea of a zero-knowledge proof is to prove that you know something, or that something is true, without revealing any additional details, beyond the fact that the statement is true, or that you know it.

As a toy example, we might consider that someone asserts they can tell the difference between coke and pepsi, or that there is a difference between coke and pepsi. They take a blind taste test of either coke or pepsi multiple times, and if they pass it every time, we might be convinced that there is a difference between coke and pepsi, and that they are able to tell the difference. Furthermore, we will have gained no knowledge about what the difference is (in terms of taste), or how it is that this person is able to tell the difference.

1.1 Background: Interactive Proofs (IP)

An interactive proof is an interaction between a prover P (who may be infinitely powerful) and a verifier $V \in \mathbf{PPT}$, in which P does something to convince V of some statement. Formally, we consider statements to be of the form $x \in L$ or $x \notin L$, where L is some language, and x is a string.

We say that a language L has an interactive proof if there is some method of interaction that satisfies the following two criteria:

1. **Completeness**: If P is honest and is actually trying to convince V of a true statement, then there exists some strategy they may use to convince V.
2. **Soundness**: If P is dishonest and is trying to convince V of a false statement, then they will not succeed, except with negligible probability.

If V accepts with probability 1 when P is honest and is using the correct strategy, then we say that the interactive proof has "*perfect completeness*".

Similarly, if V rejects with probability 1 when P is dishonest, then we say that the interactive proof has "*perfect soundness*".

If the completeness or soundness has a probability less than 1, but bounded away from 1/2 (i.e., 2/3), we call it "*computational completeness/soundness*" (respectively).

We use **IP** to denote the set of languages L for which there exists an interactive proof protocol with perfect completeness and computational soundness.

Shamir proved that $\mathbf{IP} = \mathbf{PSPACE}$, where **PSPACE** is the set of all decision problems which can be solved using a polynomial amount of space, and $\mathbf{PPT} \subset \mathbf{PSPACE}$.

Allowing for some leeway in the soundness and/or completeness, we can come up with some simple interactive proofs for types of statements which might be much more difficult to prove classically (non-interactively).

1.1.1 IP for Graph Non-Isomorphism (GNI)

We say that two graphs are *isomorphic* if there is a bijection/permutation of their vertices (together with the induced mapping of the edges) that takes one graph to the other. If no such permutation exists, then the two graphs are *non-isomorphic*.

Non-interactively, a proof of isomorphism can be done simply by providing a permutation, but proof of non-isomorphism is not quite as straightforward.

We consider the following IP, between P and V, where P asserts that $G_0 \not\sim G_1$.

1. V selects a random bit b
2. V generates a random permutation π of G_b
3. V sends the graph $H = \pi(G_b)$
4. P sends back a bit b'
5. V accepts if $b' = b$.

Note that this IP protocol exhibits perfect completeness, since the ideal strategy for the prover would be to figure out which graph H is isomorphic to, and send back the corresponding bit. If it does this, then it should be correct 100% of the time.

However, as written, it does not have very strong soundness as a dishonest prover could get away with a false statement 1/2 of the time. We may fix this by repeating the listed steps k times, so that a dishonest prover can only get away with asserting a false statement $1/2^k$ of the time (a negligible probability). Note that this repetition does not affect the perfect completeness.

1.1.2 IP for Graph Isomorphism (GI)

As noted above, proving Graph Isomorphism is very straightforward if we are working non-interactively, so we may simply consider doing the same thing in our interactive proof. Of course, this would give us an interactive proof with perfect completeness and perfect soundness, since the prover convinces the verifier if and only if he is able to send an isomorphism between two graphs G_0 and G_1.

But this costs the prover something: the prover had to figure out what the isomorphism was, and in using this interactive proof, they gave away the isomorphism for free!

This is the motivation behind Zero-Knowledge proofs: what if some party has spent a lot of time and/or computational power figuring something out, and they want to let someone else know that it is true or false, without revealing anything else.

With this in mind, we consider the following interactive proof that $G_0 \sim G_1$:

1. P picks a random bit b, and generates a permutation π on G_b
2. P sends the graph $H = \pi(G_b)$
3. V sends back a random bit b'
4. P computes and sends back a permutation $\pi' : H \to G_{b'}$
5. V accepts if π' does actually map H to $G_{b'}$

As with the interactive proof from above, we may repeat this k times to improve soundness.

This protocol then has perfect soundness, since an honest prover either sends π if $b = b'$, or composes π with an isomorphism between G_0 and G_1 (which they should know if they are being honest) to create π' if $b \neq b'$.

If the prover is being dishonest (i.e., G_0 is not actually isomorphic to G_1), then they should not be able to come up with an isomorphism if $(b \neq b')$. A dishonest prover should only be able to provide an isomorphism $1/2$ of the time, which means that they can only get away with being dishonest with probability $1/2^k$ (negligible).

Intuitively, it seems like this interactive proof hides the permutation π^* between G_0 and G_1. But we still need to formalize the notion of something being "**Zero-Knowledge**".

1.2 Defining Zero-Knowledge

The basic intuition is that if something is truly zero-knowledge, then it could be faked by someone who doesn't actually know whether the statement being proven is true. If this is the case, then there certainly must not be any other information being communicated throughout the proof.

Formally, an interactive proof for L is "*zero-knowledge*" if, $\forall\ x \in L$ and verifiers $V^* \in \mathbf{PPT}$, $\exists$ a simulator $S_{V^*} \in \mathbf{PPT}$ such that the distribution of possible transcripts generated by the simulator S_{V^*} is identical to the distribution of transcripts between an honest prover P and the verifier V^*. We write this as $[S_{V^*}(x)] \cong [P \leftrightarrow V^*(x)]$.

We use V^* to denote that the verifier might be dishonest. Note that the whole point of zero-knowledge is to help defend against dishonest verifiers, especially those who break the protocol in an attempt to learn something that they should not know or be able to find out. So, if we don't account for dishonest verifiers in our treatment of zero knowledge, we miss a very important consideration for the whole premise.

As we consider simulators, it is important to note that the verifier is simply a piece of code, and it sends messages using some pre-determined next-message function. Throughout the simulation, the simulator has access to the verifier and may rewind it, start over, and mess around with the protocol as much as it wants –as long as the final transcript of the interaction ends up looking like something between a verifier and an honest prover.

1.2.1 GI is ZK, for honest verifier

Now, in the protocol above, if the verifier V is honest, that means it actually generates a random bit b each time it is run. So, it doesn't matter whether the simulator actually sends the graph H first, or receives the bit b first. That is, the simulator may do the following:

1. Have V generate a random bit b, and S_V generates a permutation π on G_b
2. send the graph $H = \pi(G_b)$
3. Have V send back b
4. send π

Now, it's clear that the resulting transcript (1. S_V sends H, 2. V sends b, 3. S_V sends (correct) π), repeated some k times, is indiscernible from a transcript between an honest prover and verifier. And, the protocol takes k tries so the simulator is **PPT**, as required.

So, by the reasoning above, this protocol is certainly zero-knowledge when used between an honest prover and an honest verifier.

But this doesn't address a dishonest verifier at all.

1.2.2 GI is ZK, for arbitrary verifier

If a verifier V^* is being dishonest in this protocol, we must consider what it could possibly do to be dishonest. The only thing it does is generate and send a bit b, and the only information that it has at the point where it does this is the graph H. So, we may assume that the verifier takes H into account when it chooses b, and we revamp our simulator to take care of this. The most important part of the verifier cheating in this way is that it forces H to be sent before we see the bit, but as we will see, this won't be a problem.

1. generate a random bit b, and a permutation π on G_b
2. send the graph $H = \pi(G_b)$
3. (Repeat steps 1 and 2 until) V^* sends back b (2 repeats on average)
4. send π

And since the simulator has control over the final transcript, it can simply cut out the trials where V^* sent back the wrong value for b. So, this will end up giving us the same distribution of transcripts as before, which means that it is still zero knowledge. And, we have $2k$ trials on average, meaning that the simulator is still **PPT**, as required.

1.3 Batched Zero-Knowledge

When considering efficiency, I/O and communication is almost always a bottleneck. And for a reliable level of soundness, we find ourselves having to communicate back and forth a linear-order number of times. So that brings us to the question of whether or not we can do these communications in parallel and still manage to have an interactive, zero-knowledge proof.

1.3.1 GI Batched ZK

For GI, the batched-ZK protocol looks like the following:

1. P sends k graphs $H_1, \ldots, H_k$, all of which satisfy $H_i \sim G_0 \sim G_1$.
2. V sends k randomly generated bits $b_1, \ldots, b_k$.
3. P sends k isomorphisms $\pi_1, \ldots, \pi_k$, all of which satisfy $\pi_i(H_i) = G_{b_i}$.

It is easy to check that this protocol still has perfect completeness, and has soundness that can only be broken with probability $1/2^k$. But what if we try to simulate this transcript in the same way we did before?

We notice that we quickly run into an issue if the verifier is dishonest. On average, we would expect to have to reset it and try again 2^k times, which is exponential. This means that a **PPT** simulator would not be able to handle this, and would not be able to generate a transcript. So, we aren't going to be able to show that this protocol is zero-knowledge.

(That's not to say that this protocol *isn't* zero-knowledge; we just don't know whether or not it is, since we can't prove it in the usual way. Whether or not this is zero knowledge remains an open problem, as of 2010.)

But remember: our problem came from the dishonest verifier choosing its bits based on the H's. So if we somehow force it to choose the bits before we provide the H's, then we should have no problem running the simulator. We make this guarantee by adding an extra step to the batched ZK protocol to include a commitment of the bits b.

1. V commits k bits $b_1, \ldots, b_k$.
2. P sends k graphs $H_1, \ldots, H_k$, all of which satisfy $H_i \sim G_0 \sim G_1$.
3. V opens the k bits
4. P sends k isomorphisms $\pi_1, \ldots, \pi_k$, all of which satisfy $\pi_i(H_i) = G_{b_i}$.

Now, the key thing to realize/remember to make this work for the simulator is that the verifier is just a piece of code, and the simulator can control its environment, setup, and almost everything about it. In particular, since the verifier must now actually pick

the k bits based on some randomness, the simulator can control this randomness to ensure that the verifier picks the same k bits every time. Now, we run the simulator, which does the following:

1. S_V runs steps 1-3 of the protocol to figure out the k bits $b_1, \ldots, b_k$.
2. S_V resets V with the same randomness, and V recommits the same k bits
3. S_V generates k permutations $\pi_1, \ldots, \pi_k$, such that π_i is a permutation on G_{b_i}
4. S_V sends k graphs $H_1, \ldots, H_k$, all generated from $H_i = \pi_i(G_{b_i})$
5. V opens the k bits
6. S_V sends the k permutations $\pi_1, \ldots, \pi_k$.

And voila! It would seem that all of our problems are solved! Unfortunately, we do need to iron out one last detail. For all of zero-knowledge, we assume that the prover is infinitely powerful and that the verifier is **PPT**. So in order to maintain the soundness of this protocol, we need to make sure that the commitment scheme has information-theoretic-strength hiding.

The idea that we will settle on, to guarantee that our commitment scheme hides information from an infinitely-powerful prover within the scope of this proof, is the following:

We can have the commitment scheme work by providing $A_0 \sim G_0 \sim A_1$. Then, if the verifier wants to commit a bit b, they create some isomorphism π of A_b, and send the graph $com(b) = \pi(A_b)$. As long as $A_0 \sim A_1$ (which we will have the prover confirm later on in the proof), no prover, no matter how powerful, will ever be able to figure out what bit was committed.

Recall that in terms of interactive proof properties, we don't care whether the verifier is being honest or not. So as far as interactive proof is concerned, we don't really care about the binding property of the commitment. However, the binding does affect zero knowledge.

Suppose that the verifier finds a way to break the binding of the commitment. If the prover is being honest, getting both A_0 and A_1 from G_0 is identical to getting A_0 from G_0 and A_1 from G_1, since $G_0 \sim G_1$. So a verifier that can break the binding can do the equivalent of finding an isomorphism between G_0 and G_1. And since all of the proof is concerned with such an isomorphism, there's no way the verifier could have gained any information from the exchange anyways.

So with this particular commitment protocol, we end up with the final form of our batched, zero-knowledge graph isomorphism proof:

1.3.2 5-Round ZKP for GI

1. P sends A_0, A_1, both of which are isomorphic to G_0.
2. V uses the commitment described above to commit k bits $b_1, \ldots, b_k$.
3. P sends back k graphs $H_1, \ldots, H_k$, all of which are isomorphic to G_0, G_1 (if P is honest)
4. V opens the k bits
5. P sends back isomorphisms mapping A_0 and A_1 to G_0, as well as k isomorphisms $\pi_i : H_i \to G_{b_i}$

It is easy to see that this is perfectly complete, and since P must provide isomorphisms showing $A_0 \sim A_1 \sim G_0$, the commitment is hiding and we have soundness that can only be broken with probability $1/2^k$. Now, how would we run a simulator with this protocol?

The first thing we can do is simply do what we did before the commitment was added: run the initial steps until the bits are opened, and then rewind the simulator knowing these bits, recreate graphs based on these bits, and run the simulation. If the verifier is honest, the same bits will be decommitted, and everything will go as planned.

However, chances are that the simulator will have to change the H^0's that it sends, which will give the simulator an opportunity to open different bits (if it is able to break the commitment protocol).

If this is the case, the simulator rewinds the verifier once more and starts over, instead sending over $A_0 \sim G_0$ and $A_1 \sim G_1$. Just as before, it runs the verifier to see what bits it committed, creates a set of H^1's based on those bits, rewinds it, and runs it again. If the verifier continues to be dishonest and once again breaks its commitment to some bit b, it will reveal an isomorphism $A_{\bar{b}} \sim G_b$, which will give the simulator an isomorphism $G_b \sim G_{\bar{b}}$. This makes the rest of the simulator's job incredibly easy, as it now knows an isomorphism from G_0 to G_1, so it can simply run the protocol as originally intended.

However, if the verifier now decides to be honest, the simulator is in trouble, since it won't be able to show an isomorphism $A_0 \sim A_1 \sim G_0$. Now, it seems like we could get caught in an infinite back-and-forth with this.

However, it is important to note that $G_0 \sim G_1$, which we assume to be true since we are simulating an honest conversation (because we have guaranteed a high level of soundness). So because of this, the verifier can't actually tell whether we got A_1 from G_0 or G_1, so the probability that it breaks its commitment when $A_1 \tilde{\leftarrow} G_0$ and keeps its commitment when $A_1 \tilde{\leftarrow} G_1$ is at most $1/4$.

Since the verifier cannot discern where A_1 comes from, breaking its commitment and $A_1 \tilde{\leftarrow} G_1$ are independent. So letting p be the probability that $A_1 \tilde{\leftarrow} G_1$ and q be the

probability that V breaks its commitment, we have a problem with probability

$$p(1-q)\cdot q(1-p) \leq \frac{p+1-p}{2}\frac{q+1-q}{2} = 1/4.$$

So in fact we only expect to have to run $\leq 4/3$ trials as we do this, which means that the simulation is still doable in **PPT**.

1.4 ZKP for GNI

Now, we return to graph non-isomorphism. Recall the interactive proof from above:

1. V generates a random bit b and a permutation π on G_b
2. V sends $H = \pi(G_b)$
3. P sends back a bit b'
4. V accepts if $b = b'$

And this is repeated k times.

Now, this might seem like zero-knowledge, but we must remember that zero-knowledge is trying to account for a dishonest verifier, so we have to consider all the ways the verifier might be dishonest, especially if the prover is being honest.

In particular, what if H is not generated properly. What if H is found somewhere else, and the verifier just sends it to see what it gets back from the prover? Well, if the prover is being honest and tests it against each of G_0 and G_1, and sends back a bit b', the verifier has obviously just learned something, since he didn't know whether H was even isomorphic to either of the two graphs before!

So, we're gonna have to add an intermediate step before P sends back b', where V proves that it knows what b is, from which it will follow that H was generated properly, so P sending b' will not be giving away any new information to V.

1.5 An extra note on simulators

I noted above that simulators can control the environment, and the randomness, of the verifier while creating a simulation. At the time, it was important for making our proofs work, but it actually turns out to be a necessary consideration for our proofs of zero-knowledge to be correct. For example, consider the following protocol:

1. V selects a random bit b and a permutation on the graph G_b.
2. V sends $H = \pi(G_b)$.
3. P selects a random bit b', and provides an isomorphism $\pi' : H \to G_{b'}$
4. V accepts if $b \neq b'$.

Now, this protocol is clearly not zero-knowledge, since a verifier who knows nothing about these graphs walks away with an isomorphism between G_0 and G_1 with probability $1/2$.

With this in mind, suppose that S_V did not have to manage the environment and/or randomness of V during the simulation. Then, the simulator could most certainly generate this transcript without any issues at all. This seems to cause issues, since it would appear that we have proven a non-ZK protocol to be ZK.

However, if we make it so that S_V must manage the randomness of V, then the only type of transcript that S_V can generate is one where $b = b'$, and as a result, we will have that the distribution of possible transcripts of the simulator is not the same as the distribution of possible transcripts from an honest prover, since an honest prover could still make this work when $b \neq b'$. Thus, we do not reach a contradiction in this case.

So, in order for our proofs of zero-knowledge to remain true, we need S_V to manage the environment and randomness of the verifier, or else we will run into a contradiction, or will have inconsistency in our proofs, as we showed above.