0368.4170: Cryptography and Game Theory Ran Canetti, Alon Rosen

Lecture 3: Interactive Proofs and Zero Knowledge

04 November 2009 Fall 2009

Scribe: Stanislav Levin

1 Overview

This scribe will cover the following topics:

- Interactive proofs
- Zero knowledge proofs
- Proof of knowledge

2 Proofs

2.1 "Classical" Proofs

The standard mathematical notion of a proof is the following: you have axioms and inference rules, and the proof for x is a sequence of logical steps 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 say it is complete for a family L of true statements, if you can prove all statements in L using it. In other word, the traditional notion views a proof as a static string that was written down somewhere and anyone can verify. Furthermore, a valid proof gives absolute certainty that the statement is true.

The class NP represents this intuitive notation. Recall that NP is the class of languages L, such that there is an algorithm V as follows:

1. (Completeness) If $x \in L$, then there exists proof s.t. $V(x, proof) = accept$.
2. (Soundness) If $x \notin L$, then for all $proof^*$, $V(x, proof^*) = reject$.
3. (Efficiency) $V(x, proof)$ runs in time $poly(|x|)$.

2.2 Interactive Proofs

Introduction. The concept of interactive proofs as put forth in a paper by Goldwasser, Micali and Rackoff in 82 ([4]) generalizes the classical 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. Furthermore, the soundness is now probabilistic. That is, the verifier may not get convinced with absolute certainty that the statement is true but "only" with high 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 several times and Alice always answers correctly then Bob can conclude with high probability that she really can tell the difference.

Definition 1. *An interactive proof for the decision problem π, is a the following process:*

1. *There are two participants, a prover and a verifier.*
2. *In the beginning of the proof both participants get the same input (representing the statement to be proven).*
3. *The verifier and the prover exchange messages.*
4. *Both the verifier and the prover can perform some private computation and use local randomness.*
5. *At the end, the verifier states whether he was convinced or not.*

In other word, interactive proofs add *Interaction and Randomization*. Instead of a *static* proof object, we have a dynamic prover who interacts with the verifier. The verifier V is probabilistic and is allowed to make a small error probability.

The formal definition of the intuition described earlier is as follows:

Definition 2. An interactive proof *for a language L with error ν is a two-party protocol (P, V) such that:*

1. *(Completeness) If $x \in L$, then*
$$Pr([P, V](1^n, x; 1^n, x) = (1, 1) > 1 - \nu$$
2. *(Soundness) If $x \notin L$, then for all P^*,*
$$Pr([P^*, V](1^n, x; 1^n, x) = (\perp, 1) < \nu$$
3. *(Efficiency) V is a PPT.*

Unless stated otherwise, ν is $negligible(|x|)$.

Remark: According to this definition for $x \in L$ the verifier V accepts with a high probability, for $x \notin L$ the verifier V may still accept, but with a low (i.e. negligible) probability.

Definition 3. IP *is the class of languages that have interactive proofs.*

Remark: The above definition does not restrict the complexity of the prover P.

The following theorem demonstrates the power of interactive proofs.

Theorem 1. ***[LFKN90, s90]****: $IP = PSPACE$.*

- This means any language in $PSPACE$ has an interactive proof, which ultimately means that given randomness and interaction one can verify membership in very complex languages. The combination of a probabilistic model and interactive proofs form a powerful tool, much stronger than either of them apart (which may be BPP or NP).
- Note the if V is restricted to be deterministic then IP collapses to NP.

Example: $GNI \in IP$

Recall: Two graphs $G_0 = (V_0, E_0), G_1 = (V_1, E_1)$ are isomorphic (denoted $G_0 \approx G_1$) if there exists a permutation $\sigma : V_0 \longrightarrow V_1$ such that $(i, j) \in E_0 \iff (\sigma(i), \sigma(j)) \in E_1$.

- GI (graph isomorphism) is the language of all isomorphic graph pairs. $GI \in NP$, as we can consider a permutation of nodes as a witness, which may serve as a convincing proof given to a verifier.
- GNI (graph non-isomorphism) is the language of all non isomorphic graph pairs (i.e. $\overline{GI}$).
- GNI is not known to be in NP. We can however, provide an interactive proof that may be verified in polynomial time.

Consider the following interactive proof (see also Figure 1), which we denote by Π_{GNI}.

Figure 1: An interactive proof for the GNI language.

P (prover)	communication	V (verifier)
		Picks $b \in \{0, 1\}$ randomly.
		Picks a random permutation $\pi \in S_n$ where $n = \|V\|$.
		Computes a new graph $H = \pi(G_b)$.
	$\xleftarrow{H}$	
Computes b' such that $H \approx G_{b'}$.		
	$\xrightarrow{b'}$	
		Accepts if $b = b'$.

Common input: graphs G_0, G_1.

1. V chooses $b \in \{0, 1\}$ randomly.
2. V chooses a random permutation $\pi \in S_n$ where $n = |V|$ and computes a new graph $H = \pi(G_b)$, and sends H to P
3. P computes b' such that H is isomorphic to $G_{b'}$ ($H \approx G_{b'}$) and sends b' to V.
4. V accepts if $b = b'$.

Intuitively, a single run of Π_{GNI} gives a soundness error of at most $\frac{1}{2}$ (that is, V accepts $x \notin L$ with probability at most $\frac{1}{2}$). A formal proof of this intuition will shortly follow.

Let us denote by Π^k_{GNI} the protocol which runs k instances of Π_{GNI} and accepts iff all k instances accept. Then, Π^k_{GNI} has the soundness error of $\frac{1}{2^k}$ (a proof will follow shortly).

Lemma 1. *The $\Pi^{|x|}_{GNI}$ protocol is an IP.*

Proof: Run Π^k_{GNI} with $k = |x|$, accept iff all instances accept.

Completeness If G_0, G_1 are non isomorphic then there is only one b' such that $H \approx G_{b'}$ (i.e. H and $G_{b'}$ are isomorphic). In this case, P always answers b' correctly. Hence, $b = b'$ always, and

$$Pr([P, V](1^n, x; 1^n, x) = (1, 1)) = 1$$

Soundness If $G_0 \approx G_1$ then $\{\pi(G_0)\}_{\pi \in S_n} \equiv \{\pi(G_1)\}_{\pi \in S_n}$ (recall that for sets A,B : $A \equiv B$ if A, B contain the same elements), i.e. the set of graphs resulted by applying all possible permutations of size n on G_0 is identical to the set of graphs resulted by applying all possible permutation of size n on G_1. This is so because $G_1 = \rho(G_0)$ for some permutation ρ, and so $\{\pi(G_1)\}_{\pi \in S_n} \equiv \{\pi(\rho(G_0))\}_{\pi,\rho \in S_n} \equiv \{\pi'(G_0)\}_{\pi' \in S_n}$ for some permutation π'. But then P cannot determine exactly whether b' is 0 or 1, and it needs to guess b which is random (and hence independent), in which case it has a probability of $1/2$ of being correct each round.

Hence, the probability that P guesses b correctly for all $|x|$ rounds is $\frac{1}{2^{|x|}}$ (since the rounds are statistically independent) which is negligible.

$$Pr([P, V](1^n, x; 1^n, x) = (\perp, 1)) = \frac{1}{2^{|x|}} = \frac{1}{2^n}$$

Efficiency The communication and running time of V is $poly(|x|)$.

3 Zero knowledge

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 $113x237 = 26781$.

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 [1].

Notice that the $\Pi_G NI$ suggested earlier (see Figure 1) seemingly has an additional feature: The verifier V gets convinced that the prover P knows how to tell apart G_0 from G_1, but it did not learn *how*. In fact it appears that V learned *nothing* from this interaction, beyond the fact that $G_0.G_1$ are not isomorphic.

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 else as a result of this process.

That is, Bob will gain "zero knowledge". The concept of Zero knowledge proofs was invented by Goldwasser, Micali and Rackoff ([4]). 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 identification schemes to proving that many NP-complete problems are hard even to approximate.

A priori, Zero knowledge appears to be an elusive concept in the sense that not only is it not clear how to construct such things, it's also not clear even how to define them. However, as we'll see, it can be defined to have a very concrete meaning.

Definition zero knowledge (informal): Consider a Prover who proves he knows that a given theorem is correct and a Verifier who needs to be convinced of the correctness of the theorem. A ZK proof convinces the Verifier of the existence of the proof, but does not reveal any other information (be it related to the proof or not).

If we consider the "view" of a party as the "information" that this party is obtaining from the protocol's execution (i.e. concatenation of all the messages exchanged between the two parties, prefixed with all random coin tosses of verifier). Then zero knowledge may be thought of as having an algorithm called a simulator, that "simulates" the verifier's "view" of the interaction with the prover such that its output distribution is "close" to what the verifier sees when interacting with the actual prover.

Intuitively, this means that the verifier learns nothing except the fact that the statement is true - because it might as well run the simulator (which does not have any details regarding proof) instead of interacting with the actual prover.

Definition 4. ***A party's view*** *of a protocol's execution consists of the party's input, coin-tosses, and received messages. We let* $VIEW_A^{(A,B)}(a, b, 1^n)$ *denote a random variable that is distributed according to* A*'s view when the parties' inputs are* a, b *respectively, the security parameter is* n*, and the coin tosses of both parties are chosen uniformly at random.* B*'s view is defined analogically.*

Definition 5. ***(first attempt at defining Zero Knowledge)*** *A proof system for a language* L *is an interactive proof* (V_L, P_L) *for L, such that there is a probabilistic algorithm* S *(or "Simulator") that runs in expected polynomial time and such that for every string* $x \in L$ *the distribution of outputs of* $S(x)$ *is identical to the distribution of views of* V_L *of the interaction between* P_L *and* V_L *on input* x*.*

The definition captures the intuition that, if a protocol is ZK, then the verifier V_L gains no useful information from the interaction with P_L. In fact, anything that V_L might try to compute about x after interacting with P_L and receiving a proof that $x \in L$, might also be computed without interacting with P_L and using outputs of $S(x)$ instead.

$$\forall x \in L, VIEW_A^{(A,S)}(a, b, 1^n) = VIEW_A^{(A,P_L)}(a, b, 1^n)$$

Theorem 2. *The* Π_{GNI} *protocol (see also Figure 1) satisfies definition 5.*

Proof: The simulator will prepare the following view: (b, σ, H) (where b is V's random bit) such that: $b \xleftarrow{R} \{0, 1\}$, $\sigma \xleftarrow{R} S_n$, and $H = \sigma(G_b)$. This tuple is distributed identically to view of V after the interaction with P.

Remark: This definition for ZK is not strong enough to deal with arbitrary verifiers. Consider the following scenario: V^* has a graph H which he knows to be isomorphic to one of G_0, G_1. V^* wishes to know if H is isomorphic to G_0, or G_1. By sending H to P, he may get this information, which he could not have computed himself. V^* may not learn anything about the problem at hand (i.e. whether G_0, G_1 are not isomorphic), but it *can* learn a piece of information he could not have learned on this own. Thus, this GNI IP is not ZK against this particular verifier V^*.

In other words, definition 5 guarantees security only against verifiers who follow the protocol. The class of such protocols is called *Honest Verifier Zero Knowledge* ($HVZK$).

Definition 6. ***Perfect Zero Knowledge (second attempt at defining Zero Knowledge)*** *A proof system for a language* L *is an interactive proof* (V_L, P_L) *for L, such that for every polynomial time verifier* V^* *there is a probabilistic algorithm S (for Simulator) that runs in average polynomial time and such that for every string* $x \in L$ *the distribution of outputs of* $S(x)$ *is identical to the distribution of views of* V^* *of the interaction between* P_L *and* V^* *on input* x*.*

Remark: Notice that the definition considers V^*, i.e. any arbitrary verifier. Using a specific V may ensure zero knowledge if the verifier follows the protocol. In case the verifier deviates from the protocol some knowledge may "leak", rendering the protocol non zero knowledge.

The class of languages that admit a perfect zero knowledge proof system is denoted by PZK.

This stronger condition implies that the prover can still deliver a convincing proof that $x \in L$ without giving away any information about x, even if the prover does not trust the verifier to follow the protocol of the proof system.

Consider the following protocol for GI (see also Figure 2), which we denote Π_{GI}.

Common input: graphs G_0, G_1.

1. P chooses a random permutation σ.

2. P computes $H = \sigma(G_0)$ and sends H to V.
3. V chooses a random bit b.
4. V sends b to P.
5. P computes $H = \rho(G_i)$ where

$$\rho = \begin{cases} \sigma & \text{if } b = 0 \\ \pi \circ \sigma & \text{if } b = 1 \end{cases}$$

   where the permutation π is the isomorphism.
6. P sends ρ to V.
7. V accepts if $H = \rho(G_b)$.

Figure 2: A zero knowledge proof for the GI language.

P (prover, supposedly knows the isomorphism π)		V (verifier)
Picks $\sigma \overset{R}{\leftarrow} S_n$. Computes $H = \sigma(G_0)$.		
	$\overset{H}{\longrightarrow}$	
		Picks $b \overset{R}{\leftarrow} \{0,1\}$ randomly.
	$\overset{b}{\leftarrow}$	
Compute $H = \rho(G_i)$, $\rho = \begin{cases} \sigma & \text{if } b = 0 \\ \pi \circ \sigma & \text{if } b = 1 \end{cases}$		
	$\overset{\rho}{\longrightarrow}$	
		Accept if $H = \rho(G_b)$.

Lemma 2. *The Π_{GI} protocol (see Figure 2) is PZK.*

First we show that a single run of the Π_{GI} protocol is ZK.

Proof:

Completeness Holds (because if π really is an isomorphism then it will always be the case that $\rho(\pi(G_1)) = \rho(G_0)$).

Soundness The probability of V accepting in case P does not know the isomorphism π is the probability V will choose $b = 0$, which is $\frac{1}{2}$, since this choice is random.

ZK Consider the simulator S described in Figure 3. It has the following properties:

- S's *expected* running time is twice the runtime of V. (Note that S has only expected polynomial runtime. Obtaining fixed polynomial runtime is much more complicated.)
- The output of S is identically distributed to that of $[P, V^*](x, x)$. Thus S can in fact simulate the interaction of V with P, meaning anything V could learn from such an interaction could be learned without actually interacting with P.

Figure 3: A simulator for the GI problem.

V (verifier)		S (simulator)
		Picks $b \in \{0,1\}$ randomly. Picks a random permutation $\pi \in S_n$ where $n = \|V\|$. Computes a new graph $H = \pi(G_b)$.
	$\xleftarrow{\text{graph } H}$	
Picks $b' \in \{0,1\}$ randomly.		
	$\xrightarrow{\text{bit } b'}$	
	$\xleftarrow{\pi, \text{ if } b=b'}$	if $b = b'$, output whatever V outputs, else - repeat with a freshly chosen π.

A single execution has a soundness error of $\frac{1}{2}$. One can reduce the probability of error by repeating the protocol several times. If the protocol is repeated several times, however, it is not clear that the general zero knowledge property is preserved. We will deal with this issue in section 3.1. □

Clearly, every problem in BPP is also in PZK (using a proof system where no message is exchanged, which is easy to simulate!). A few problems that are not believed to be in BPP, are also in PZK, in particular GI.

3.1 Sequential composition of ZK

In the graph isomorphism example, a cheating prover can trick the verifier half of the time. If the protocol is run 30 times, there is roughly a one in a billion chance that the verifier will be able to cheat. This is good for soundness, but is it good for zero knowledge? Intuitively, it might seem odd to be able to get information after more times if none was obtained after the first time, but we should look more closely at this. We saw an informal argument that you could sequentially compose the graph isomorphism proof. But if the verifier is malicious, then it can pass any information it learned during the first round down to the second round.

If the simulator cannot also get this prior knowledge, then it may not be able to simulate the same conversations. Therefore, we must allow the simulator to have more power by allowing it too to have the prior knowledge, sometimes referred to as an "advice string" "auxiliary input", or simply z.

We then require that the simulation will be valid even for any value of z.

This definition is closed under sequential composition; that is, it still requires that the verifier does not learn any more information than what he started with even if many copies of the protocol are executed sequentially. For example, if he started with knowledge of half of the permutation that maps G to H, then he should still not know anything more after the protocol is run.

Definition 7. ***An interactive proof (P,V) with auxiliary input is Zero Knowledge*** *if for any polynomial time algorithm V^* there exists a polynomial simulator S such that, for any input $x \in L$ and for any z,*

$$S(x, z) = [P, V^*](x, (x, z))$$

Theorem 3. *Let (P, V) be an auxiliary-input ZK two-party protocol with polytime V, and let (P^k, V^k) be the protocol that runs (P, V) sequentially k times (on the same common input). Then (P^k, V^k) is auxiliary input ZK.*

Remark: this theorem is based on the auxiliary input ZK definition.

Proof sketch: Consider some "cheating verifier" V^{k^*}. To construct a simulator S^* for V^{k^*}, we first define a "cheating verifier V^* for the original protocol (P, V): V^* treats its auxiliary input z as a state for V^{k^*}. It then runs V^{k^*} from state z, for a single interaction with the prover. V^* then outputs the current internal state of the V^{k^*}.

By definition, V^* has a simulator S. We now construct S^k from S: S^k simply runs S, k times, where the first run starts with the auxiliary input of S^k, and for $i > 1$ the $i - th$ run starts with auxiliary input which is the output of the $(i - 1)$ run of S.

3.2 Parallel composition of ZK

We now ask whether protocols that satisfy the above definition are closed under parallel composition. That is, does the ZK property of an IP still holds when we run several parallel instances of it.

The general proof mentioned above doesn't work.

What about a parallel repetition of the GI protocol previously demonstrated (Figure 3) ?

The simple extension fails, since S has to guess k bits at once. This means that the expected runtime of S would be 2^k.

We'll see that this problem in the simulation is quite general:

Definition 8. *(Black-Box ZK): An interactive proof (P, V) is BB (black box) Zero Knowledge if there exists a polytime simulator S such that for any polytime algorithm V^*, for any input $x \in L$ and for any z,*

$$S^{V^*}(x, z) = [P, V^*](x, (x, z))$$

(That is, in $BBZK$, the only way in which S uses V^* is to run it "as an oracle", without making any direct use of the code of V^*.)

Observation: The Π_{GI} protocol (see also Figure 2) is BBZK.

Theorem 4. *If there exists a 3-round $BBZKIP$ for language L then $L \in BPP$.*

Remark: Whether there exist 3-round $ZKIP$s for languages outside BPP is a fascinating open problem. Currently the only protocols we have depend on very strong hardness assumptions on the DL problem (which are not BBZK).

3.3 Statistical and Computational ZK

So far we have considered only the case where the output of the simulator is distributed identically to that of the real interaction. This is called prefect ZK (PZK) in the literature. Two natural relaxations are:

Statistical ZK (SZK) Same as PZK, except that $S^{V^*}(x, z)$ and $[P, V^*](x, (x, z))$ are only required to be statistically indistinguishable.

Computational ZK (CZK) Same as PZK, except that $S^{V^*}(x, z)$and $[P, V^*](x, (x, z))$ are only required to be computationally indistinguishable. Here it is important to stress that the distinguisher may depend on x, z. In particular it may "know" a witness for the fact that $x \in L$.

We note that the sequential composition theorem works also for SZK and CZK. The proofs are somewhat more intricate (esp. CSK).

Similarly, the lower bound on $BBZK$ and the open question regarding 3-round ZK proofs extend even to CZK (and even to the case of only "computationally sound" proofs).

3.4 CZK proofs for NP

ZK proofs were proposed by Goldwasser, Micali and Rackoff, 83 (paper published in 85). They were considered more of a curiosity, until their general applicability was demonstrated in 86 by Goldreich, Micali and Wigderson.

Any language in NP has a CZK IP, i.e. $NP \subseteqq CZK$.

It suffices to show a CZK IP for an NPC language L. (This is so since, for any language L' in NP, (P, V) will first translate the input instance x' to an instance x of L and run the protocol on x. It is guaranteed that $x \in L$ iff $x' \in L'$.)

The NPC language in [7] is Graph 3-colorability. We'll see a slightly simpler protocol (by Blum) for Graph Hamiltonicity (GH).

A graph is Hamiltonian if there is a simple cycle that connects all nodes.

Input: A graph $G = ([n], E)$.

Prover has a Hamiltonian cycle $H \subsetneq E$.

The following protocol, denoted Π_{GH} (see also Figure 4) uses commitment schemes discussed in the previous lecture (statistically binding and computationally secret).

Protocol description:

1. P generates a permutation π of V.
2. P generates the adjacency matrix of $A = \pi(G)$.
3. P commits to π and sends the commitment to V.
4. P commits to each $\pi(G)_{ij}, i, j \in \{1...n\}$ separately and sends the commitments to V.
5. V randomly chooses $b \xleftarrow{R} \{0, 1\}$ and sends it to P.
6. If $b = 0$, P decommit all adjacency commitments and π. If $b = 1$, P decommit all (i, j) participating in the Hamiltonian cycle, i.e. (i, j) such that $(p(i), p(j)) \in \pi(G)$. P then sends the decommitments to V.
7. V verifies all commitments were opened correctly.
8. If $b = 0$, V accepts if the graph received from P is indeed $\pi(G)$. If $b = 1$, V accepts if the opened path is hamiltonian in $\pi(G)$.

Theorem 5. *Π_{GH} is a CZKIP for GH with soundness error $\frac{1}{2}$.*

Proof:

Completeness If there's a Hamiltonian cycle in the original graph then its permutation is still a Hamiltonian path in the permuted graph. And so it will be opened and verified by V at the final step of the protocol.

Soundness If G is not Hamiltonian then V accepts with probability of at most $\frac{1}{2}$.

ZK To prove the zero-knowledge property, consider the following simulator:

1. S chooses a random bit c'.
2. If $c' = 0$, S commits to (a random permutation of) the adjacency matrix for G.

Figure 4: The Blum protocol for a Hamiltonian cycle problem.

P (Prover)		V (verifier)
Picks a random permutation $\pi \in S_n$ where $n = \|V\|$. Prepares adjacency matrix for $A = \pi(G)_{ij}$. $A_{ij} = 1 \Leftrightarrow (\pi(i), \pi(j)) \in E$	$\xrightarrow{\text{commitment to } A_{ij}, \text{ for all } i,j}$	
	$\xrightarrow{\text{commitment to } \pi}$	
		$b \xleftarrow{R} \{0,1\}$
	$\xleftarrow{\text{a bit } b}$	
if $b = 0$ decommit all adjacency commitments and π if $b = 1$ decommit all (i,j) participating in the Hamiltonian cycle, i.e. (i,j) such that$(p(i), p(j)) \in \pi(G)$		
	$\xrightarrow{\text{decommitments}}$	
		V accepts if: { all commitments were opened correctly } and { [$b = 0$ and $A_{ij} = \pi(G)_{ij}$] or [$b = 1$ and the opened (i,j) form a Hamiltonian path in $\pi(G)$] }

3. If $c' = 1$, S commits to a (randomly permuted) cycle graph, and sends the resulting commitment to the verifier who responds with a bit c.
4. If $c' = c$ then the prover responds in the natural way and S is done. Otherwise S rewinds and tries again.

Since $c' = c$ with probability $\frac{1}{2}$, if the simulator rewinds k times it will succeed at least once with probability $1 - \frac{1}{2^k}$.

A question may arise as to why this protocol is not PZK altogether, why do we limit it to be CZK?

The reason is that we did not consider the security of commitments in this specific setting where multiple commitments are sent in the protocol.

The reason that the protocol is only CZK is that we're using commitments. The commitments have to be binding even against computationally unbounded committers, or else we wont get soundness. This means that the commitments can be only computationally hiding, which results in CZK.

When one uses commitments in larger protocols (as we did here) one needs to be careful, since the expected properties of commitment may not always hold. It turns out that in this particular case one can indeed prove that the overall protocol is CZK, but this is intricate and requires work.

Remark: In order to get a ZKIP with negligible soundness error one can sequentially repeat this protocol a polynomial number of times retaining the ZK property.

3.5 Interactive proofs with an efficient prover

So far we concentrated on the case where the prover is unbounded. In cryptography we will be mostly interested in the case where all parties involved are computationally bounded. In the context of interactive proofs, this puts forth several issues.

- First we observe that now the scope is limited to languages that can be verified in polytime.
- We would like proofs where the prover algorithm is efficient, say given a witness to the fact that the statement is true (eg, to the fact that $x \in L$).

  For this purpose, we need to extend the formalism of IP to allow the prover to have some auxiliary input that's not available to the verifier. In this case it will be more convenient to consider IPs for relations rather than for languages. That is, say that a binary relation $R()$ is polytime computable if it can be evaluated in time polynomial in the length of the first input.

  Definition 9. ***(IP, efficient prover)****: An interactive proof for some polytime computable relation $R(\cdot,\cdot)$ is a two-party polytime protocol (P, V) with the following properties:*

  Completeness For any x, w s.t. $R(x, w)$ holds,

  $$Pr([P, V]((x, w), x) = (-, 1)) > 1 - \nu(n)$$

  Soundness For any x for which there is no w s.t. $R(x, w)$ holds, and any A,

  $$Pr([A, V]((x, w), x) = (-, 1)) < \nu(n)$$

  (note that if A is taken to be non-uniform then there is no need to give it x, w.)

  for some negligible function $\nu(n)$, where $n = |x|$.
- *Remark:* This is a stronger definiton since P is required to run in polynomial time. We can relax the soundness condition to hold only with respect to polytime "cheating provers". Such proofs are called "computationally sound" or "arguments" in the literature.

3.6 Proofs of knowledge

The above notion of soundness is aimed at the question of whether $x \in L$, or in other words whether there exists a witness proving x is in L. But often in cryptography we are interested in another question: does the prover "know" an appropriate witness w? In fact, it may be the case that a witness always exists, and we're only interested to find out whether a given party "knows" it.

For instance, consider the following natural application of ZK protocols to identification:

Let G be a group of prime order p and let g be a generator of G. Each party P chooses a random $w \leftarrow [p]$, and publishes a public key $x = g^w$.

Then, in order to identify P, run a ZK protocol for the NP relation: $R(x, w) = x, w : x = g^w$.

Intuitively, this should be ok, since

- by ZK the verifier learns nothing about w
- by soundness V gets a convinced only if P has w.

But the second property does not follow: Soundness only gives a guarantee when there is no w s.t. $R(x, w)$ holds, and here there certainly exists one. The only question is if P "knows" this w.

This brings forth an interesting definitional/philosophical question: What does it mean for a computational entity (or, for an algorithm) to "know" something?

Suggestion 1 A machine M "knows" a string w if during its computation the string w appears explicitly, say, on a special tape.

This is too restrictive, since an "adversarial" M may not be nice enough to write w explicitly for us, but may still use it quite explicitly. (e.g., what if w is written in reverse order?)

Suggestion 2 A machine M "knows" a string w if there exists an algorithm that uniquely determines w given M's input and communication.

This is too permissive, since w can be determined and still M has no clue what it might be (eg, the above identification example).

Suggestion 3 machine M "knows" a string w if there exists an EFFICIENT algorithm M' that outputs w given M's input and communication.

We will use Suggestion 3.

We'll actually make the stronger requirement that M' behaves "just like M", and in addition outputs w on an additional tape. The algorithm M' is called a "knowledge extractor".

There are a number of formalizations of this definitional approach, all with different properties. We'll present one such formulation, in the context of IP.

Definition 10. *A two-party protocol (P, V) is called a Proof of Knowledge (PoK) for relation $R(x, w)$ if for any polytime (cheating) prover P^* there exists a simulator S such that for any x, and any z we have:*

1. $S(z)_1 \approx_c [P^*, V](z, x)$
2. $Pr(w = S(z)_2, R(x, w) = 0)$ *and* $[P^*, V](z, x) = 1) < \nu(n)$, *for a negligible function* $\nu(n)$.

That is, S generates two outputs: the first one is a simulated interaction between P^* and V, that's indistinguishable from an actual interaction. The second output is a value w such that whenever V accepts the interaction in the first output of S, $R(x, w)$ holds (except for negligible probability.)

Theorem 6. *The (multi-iteration) Blum protocol is a $ZKPoK$ for $R_{GH} = \{G, H : H$ is a Ham. cycle in $G\}$.*

Proof sketch: Let P^* be a polytime malicious prover. S operates as follows:

For each iteration of the protocol:

- S receives the first message from P^* (recall that this message is supposed to contain a committed graph)
  1. S sends a random challenge $b \leftarrow \{0, 1\}$
  2. S records P^*'s response and internal state.
- rewinds P^* to before step *, and reruns P^* on challenge $1 - b$. If P^* opens the commitments correctly and the opening reveals a Ham cycle in G then S outputs G in its second output.
- S returns P^* to the state in step 2.

The first output of S is the final output of P^*.

To show the knowledge extraction property, assume we have a prover who gives a correct proof with probability better than $\frac{1}{2}$. This implies that the prover responds correctly for both possible values of b. So we simply rewind the prover, send him both possible challenges, and use the two (correct) responses to compute a Hamiltonian cycle in G.

Note that this is only a sketch. The actual proof gets more complicated in dealing with an adversarial prover that correctly opens its commitments only part of the time.

- In the end P^* sees an execution that's distributed identically as a real execution with V (since only the non-rewinded versions got into the main execution of P^*). Thus its output is distributed identically to that of the simulator.
- The probability that V accepts all k interactions with P^* and yet S did not manage to find a witness w is negligible.

4 Applications

One motivation to Zero Knowledge is philosophical: the notion of a proof is basic to mathematics and to people in general. It is a very interesting question whether a proof inherently carries with it some knowledge or not.

Another motivation is practical, zero knowledge proofs have found many applications. For instance, protocol design, as we'll see in the next lecture. A protocol is an algorithm for interactive parties to achieve some goal. 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, it is possible to do this in a very general way, applying essentially to all cryptographic protocols. Thus, a general technique 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.

References

[1] Boaz Barak, *Zero knowledge*, Princeton, November 2005.

[2] Eli Biham, *Zero knowledge protocols*, Technion, October 2009.

[3] Ran Canetti, *Introduction to cryptography*, Tel Aviv University, Fall 2008.

[4] S Goldwasser, S Micali, and C Rackoff, *The knowledge complexity of interactive proof-systems*, Proceedings of the seventeenth annual ACM symposium on Theory of computing, ACM Press, 1985, pp. 291–304.

[5] Susan Hohenberger, *Special topics in theoretical cryptography*, The Johns Hopkins University, January 2007.

[6] Jonathan Katz, *Special topics in theoretical cryptography*, University of Maryland, April 2004.

[7] S. Micali O. Goldreich and A. Widgerson, *Proofs that yield nothing but their validity and a methodology of cryptographic protocol design*, Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 174–187.

[8] Rafael Pass, *Cryptography*, Cornell, October 2009.

[9] Salil Vadhan and Alon Rosen, *Introduction to cryptography*, Harvard, December 2006.