

Lecture 4: Zero-Knowledge Proofs II

*Instructor: Susan Hohenberger**Scribe: Lori Kraus*

1 Administrative

The first problem set will be distributed on Monday, February 5. We are encouraged to work together on problem sets as we can probably learn a lot from each other.

We had some good questions yesterday. Feel free to email Susan and to interrupt in class.

2 Zero Knowledge Proofs

The term zero knowledge was first introduced by Goldwasser, Micali, and Rackoff in [GMR85]. The paper additionally gave some examples. In 1986, the idea of zero knowledge was further refined and explored by Goldreich, Micali, and Wigderson in [GMW86]. There is a constantly changing definition of zero knowledge proofs and many papers are still coming out. We will not have *the* definition by the end of today, but a pretty good definition.

General characteristics and working definition of zero knowledge proofs from yesterday:

1. Completeness
2. Soundness
3. Zero-Knowledgeness

$$\forall V_{\text{PPT}}^* \exists S_{\text{PPT}} \forall x \in L \text{VIEW}_{P,V^*}(x) = S(x)$$

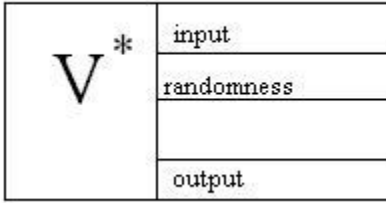
The above statement says that the “VIEW” conversation should look identical to the simulated conversation. A stronger version of the definition is:

$$\exists S_{\text{PPT}} \forall V_{\text{PPT}}^* \forall x \in L \text{VIEW}_{P,V^*}(x) = S^{V^*}(x)$$

In the first definition, there can be a different simulator for each verifier. Most people agree that this is a correct definition, although it is hard to prove anything with respect to this definition. Most people agree that the second definition is easier to work with.

In the second definition, the simulator needs black box access to the verifier. We can think of this as a Turing machine where the verifier has a random tape and an input tape, and the simulator writes on these tapes. The simulator doesn’t get to choose the verifier’s output (e.g., whether the verifier sends back G or H in the case of the graph isomorphism example). If the simulator gets stuck, it can start over and play several games with the verifier, only recording the runs of its choice.

PPT means that the protocol can finish in probabilistic polynomial time.



Let's recall our protocol for proving that two graphs are isomorphic. For k protocols, the expected number of runs is $2k$ in order for the simulator to be able to reproduce the desired conversation.

To imitate the isomorphism conversation, the simulator first guess G or H . Suppose it chooses G . Then it chooses a random permutation π and computes $C = \pi(G)$ to send to the verifier (as in the first step of the real protocol). The simulator will know the correct response for the verifier only if the verifier picks G or H the same as the simulator. If the opposite graph is chosen, that round is disregarded. The simulator starts over with a new round, setting the verifier's random tape as before and running the verifier forward on all good runs out to this point. Only if the simulator is *very* unlucky will it not be able to complete the simulation in PPT. We now modify our definition:

$$\exists S_{Expected\ PPT} \forall V_{PPT}^* \forall x \in L\ VIEW_{P,V^*}(x) = S^{V^*}(x)$$

The big question among researchers was whether "Expected PPT" should be allowed in the definition; that is, whether the simulator must always finish in PPT or whether it was enough if the simulator's expected running time was PPT. (In our above example, the simulator runs in expected PPT, rather than strict PPT.) Since we don't always know how to construct simulators that run in strict PPT, we allow simulators that run in expected PPT (but there are some negative consequences of this that are beyond our scope for today). Note that $VIEW_{P,V^*}(x)$ and $S^{V^*}(x)$ represent distributions and $VIEW_{P,V^*}(x) = \langle \text{Messages from } P, \text{Coins (randomness) of } V^* \rangle$. So, this is basically the elements required to reconstruct the entire conversation including the random coins which determine what the verifier sends. (We will see examples of how it breaks down when coins are not used later.) All of the situations possible for the VIEW are also possible for the simulator (with the same probabilities). This is what we mean by them being the same distribution.

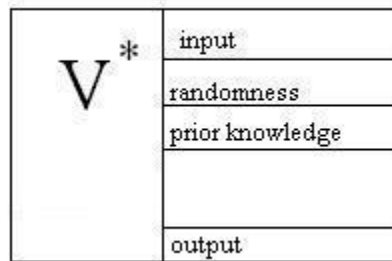
Now that we have yet another working definition, we still have to evaluate whether it covers what our intuition says. I.e., Is all we know is that the statement is true or do we gain any other bits of information?

2.1 Sequential Composition

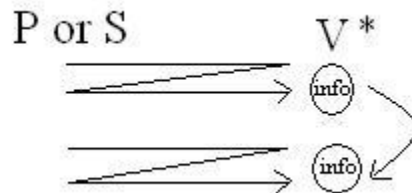
Should the definition require that if I have a zero knowledge protocol I can run it many times and still have it be zero knowledge?

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? A related question is whether the definition should require that for sequential composition the protocol remains the same. 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.

Yesterday, 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. See below.



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 “a”. (Note that in the first round, the advice string is blank.)



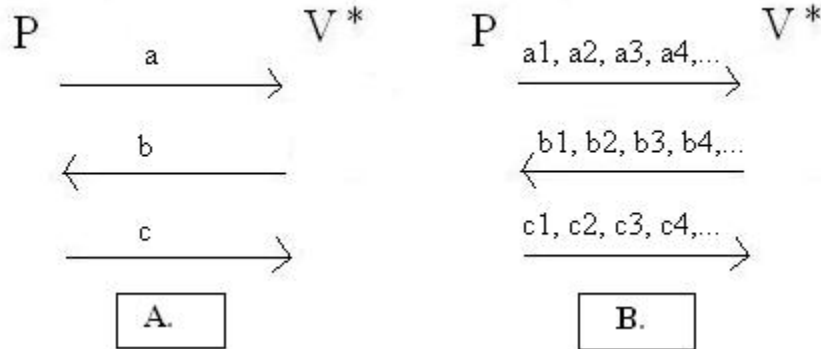
Then we can say that if the VIEW equals the simulation then it is still a zero knowledge proof. We re-write the definition as:

$$\forall V_{\text{PPT}}^* \exists S_{\text{PPT}} \forall x \in L \forall a \in \{0, 1\}^{\text{poly}(|x|)} \text{VIEW}_{P, V^*(a)}(x) = S^{V^*}(a, x)$$

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.

2.2 Parallel Composition

We now ask the question of whether protocols that satisfy the above definition are closed under parallel composition. Is it ok to send a series of information on each line of communication, thereby requiring many fewer rounds and less communication time. For example, if the original protocol was such as described in figure A., is it ok to send messages that would have normally been sequential composition into parallel such as shown in figure B.?



The motivation for parallel composition is clear, but...

1. Does our current definition imply that it should also be zero knowledge?
2. Do we even want to require that this should work?
3. Do our example protocols allow for parallel composition?

It actually turns out that the parallel version of the graph isomorphism example retains its completeness and soundness properties with the probability of the prover being able to cheat still on the order of $(\frac{1}{2})^n$ (where n is the number of equivalent rounds of the protocol).

We would like to know if soundness decreases in general, though, when protocols are parallelized. Pietrzak and Wikström have a paper coming out next month at the TCC 2007 which shows that it is not always the case that parallelization means decreased soundness, but that it is in fact true for most cases [PW07]. (Note, their paper has nothing to do with zero knowledge, just general proof systems.)

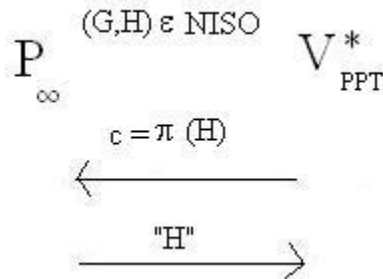
Does parallel composition make it harder for the verifier to cheat since he can't adopt a strategy based on complete previous rounds of the protocol? Consider the cheating verifier that disregards the random tape and instead chooses G or H based on the hash of $a_1, a_2, a_3, \dots, a_n$ (representing graphs in the first step of the ISO protocol). This means that each time the simulator changes his first step of the simulated protocol, the verifier may also change all his guesses in the second step. In the parallel case, using our simulation strategy from before, the simulator needs to guess all n choices of G and H correctly at the same time (otherwise, he cannot answer the verifier in the third step of the protocol). But,

our simulator's chance of correctly guessing all n choices correctly is at most $(\frac{1}{2})^n$ which is very small for even a moderate n . So, *this* simulator would not run in expected PPT, but this does not necessarily mean that *no* simulator would.

We conclude that our current definition is not likely to imply parallel composition, and therefore that many protocols proven secure under this definition cannot be used in parallel without further examination.

Question: What happens if some $c_i = c_j$? Does the verifier then know the permutation from c_i to G and c_j to H and therefore create the permutation mapping G to H ? This would definitely not be zero knowledge. **Answer:** The probability of having some $c_i = c_j$ is very small. It is about the probability of the verifier being able to just guess the mapping between G and H . Therefore, it is not something we have to worry about.

Here is another example of a cheating verifier. Consider the following situation where a prover is trying to show that two graphs, G and H , are NOT isomorphic. In this protocol, the honest verifier is supposed to send over a random graph which is isomorphic to either G or H . Then the prover is supposed to determine whether the received graph is isomorphic to either G or H . In more detail, the verifier chooses G or H in addition to a random permutation π . Suppose he chooses H . Then he sends $\pi(H)$ to the prover who in turn sends back a one bit response of either "G" or "H". See next figure. If the two graphs are indeed distinct graphs, then the prover will have no trouble determining which graph was permuted.

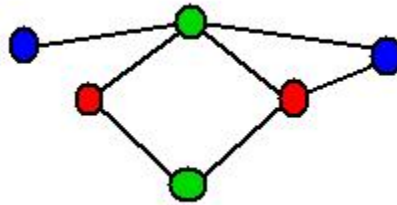


Can the verifier gain any additional information by using this protocol? Is this protocol zero-knowledge? No, it isn't! Suppose that in addition to the graphs G and H , the verifier has two more graphs D and E that he wishes to know how they relate to G and/or H . The verifier can choose a random permutation π and, for example, send $\pi(D)$ to the prover who then sends back G or H if the graph he received is indeed isomorphic to one of them. If the prover is honest and the verifier gets the response "G", then he knows that D is isomorphic to G , quite a bit more information than just knowing that G and H are not isomorphic. (However, if D or E are not isomorphic to G or H , then the prover will see that something fishy is going on and may abort.)

2.3 A Zero Knowledge Proof for 3COL with “hats”

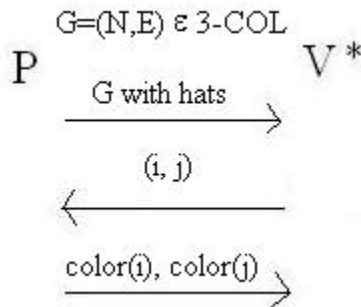
Now, we have seen that the language ISO has a zero-knowledge proof protocol. But, what about other languages? In a major result, Goldreich, Micali, and Wigderson [GMW86] who showed that $NP \subseteq ZK$, that is, all languages in NP have zero-knowledge proofs for membership in the language.

This is an example of a zero knowledge proof for an NP-Complete language, namely all graphs that are 3-colorable. 3_COL is defined as the set of all graphs $G = \langle N, E \rangle$ such that each node $n \in N$ can be colored using at most 3 colors in such a way that for each edge $(i, j) \in E$, we have $color(i) \neq color(j)$ (i.e., there is no edge connecting two nodes with the same color). The graph below is 3 colorable.



Deciding whether a graph is 3 colorable is an NP-hard problem.

We would like a ZK protocol for proving that a graph G is 3 colorable. We can easily prove this to someone based on one known way to 3 color the graph. First have the verifier leave the room. Then choose a random way to flip the colors in our known coloring (that is, suppose we know a coloring for colors 1, 2, and 3; now randomly assign red, blue, and green to the values 1, 2 and 3 and color the graph accordingly). Now, with our “random” (but correct), color the graph, place hats over all the nodes, and invite the verifier back in. Have the verifier point to an edge and remove the hats on the two nodes connected by that edge. The verifier checks that the two nodes are indeed different colors. Repeat this protocol a number of times. Eventually the verifier should be convinced that the graph must indeed be 3 colorable.



This protocol is complete since if there is a 3 coloring, than the honest prover can always

make the verifier accept. The protocol is sound, after running it for enough times. For each run of the protocol, the worst probability of catching a cheater is $1 - \frac{1}{|E|}$ where $|E|$ is the number of edges in the graph. (Consider that if the graph is *not* 3-colorable, then there is at least one edge connecting two nodes of the same color, and the verifier picks this edge with $1/|E|$ probability.)

For soundness, we want $\forall P^*, \forall x \notin L, Pr[(P^*, V)[x] = \textit{accept}] \leq \frac{1}{2}$.

For one iteration, $Pr[\textit{accept}] \leq 1 - \frac{1}{|E|} = \frac{|E|-1}{|E|}$. For graphs with more than two edges, this is too big.

However, over $|E|^2$ iterations, $Pr[\textit{accept}] \leq (\frac{|E|-1}{|E|})^{|E|^2}$, which easily satisfies our definition. (In the next lecture, we'll analyze this step more precisely.)

Sequential Composition. Since we *must* execute this protocol repeatedly (i.e., sequentially) to achieve soundness, then we must prove that the ZKness property holds after several runs of the protocol. If zero knowledgeness does not remain as we repeat the protocol, then it is not a zero knowledge proof.

Let's argue that this protocol does satisfy our (most current) definition of zero-knowledge. First, we must construct a simulator. Consider a simulator for the 3 coloring example that works as follows. Do not originally color the graph. Allow the verifier to come back into the room and see the graph with hats over every node. (Note: He cannot tell that the graph is not colored.) The verifier chooses an edge and then the simulator rewinds and colors the nodes of the chosen edge with different colors and starts the protocol again with the verifier using the same randomness. Since the verifier is using the same randomness and sees the same "input" (i.e., a graph covered by hats), then the verifier will pick the same edge as before. Thus, the simulator will now be able to remove the two hats and reveal nodes with different colors.

Another technique would be for the simulator to just randomly color the graph at the beginning. Then, it only needs to rewind if the coloring of the nodes on the chosen edge happen to both be the same. This second method may actually result in a faster simulation, since the simulator does not always need to rewind .

In either case, the simulator doesn't need to do anything extra with the auxiliary input. It simply feeds its "a" to V^* , and no matter what strategy V^* uses to pick the edges, the simulator can react as indicated above.

Second, we need to argue that one of these simulators actually produce a VIEW identical to that of the real world. This is easy to see. First, consider that in step one, the view is always of (completely opaque) hats over the nodes. Next, the verifier chooses an edge according to whatever strategy it likes, given a view identically distributed to its view in the real world. That is, the verifier sees the same input (graph covered by hats), is given a random tape, and is given the auxiliary input "a". Finally, the simulator always responds correctly, during any *recorded* rounds, since it can always rewind when it gets into trouble. Using the first simulation strategy, our simulator has to rewind each time and thus our ex-

pected running time is on the order of $|E|$, which is polynomial in the description of graph G .

Great; now we have seen a ZK proof for an NP-complete language using “hats” – a physical device.

Remark: Note, when we say rewind one step, we actually have to go all the way back to the beginning, and repeat everything exactly as it was before up until the last step.

2.4 Going Further

Next class, we will discuss how to get rid of the hats and do everything in the proof digitally. We can replace the hats with cryptography. The problem with cryptography that we did not have with the hats is how to “hide” the colored nodes in such a way that the verifier cannot learn anything about the coloring that helps him guess an edge, but at the same time the verifier will be convinced when the prover “removes” the hats, that the prover didn’t change the colors.

References

- [GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, 1985.
- [GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge, and a methodology of cryptographic protocol design. *LNCS: CRYPTO '86*, 263:171–185, 1986.
- [PW07] Krzysztof Pietrzak and Douglas Wikström. Parallel repetition of computationally sound protocols revisited. *LNCS: Theory of Cryptography - TCC 2007*, 4392:86–102, 2007.