CSE 5854: Class 08

Benjamin Fuller

February 13, 2018

1 Zero Knowledge Proofs of Knowledge

Last class we introduced the idea of a proof of knowledge which assured that a prover had a "witness in mind" when they were proving something. The goal was to prevent malicious provers from forwarding proofs from other parties or manipulating other individuals proofs (there are other technical reasons as well). However, introducing the zero-knowledge property seemed to be at conflict with zero-knowledge. Setting things up to be easy for the simulator to rewind seemed to make it difficult for an extractor to rewind. The crux of the problem is the simulator wants to get the malicious verifier to fix their actions (often thru commitment) and be able to change. The extractor wants to do the same with the malicious prover. However, there is tension between which party does this first.

To untie this knot we have to actually modify the internals of the protocol (its not enough to add commitments). However, this change works for any zero-knowledge proof that is "public-coins" where all the verifier sends as messages is randomness that isn't determined by the prover.

The main idea is to allow the prover to "contribute" to the coins that will be used by the verifier. However, we are only trying to give power to the simulator and not actually allow a malicious prover to impact anything. To keep our discussion clear we'll remove some of the details of the protocol. Most of the core protocols we saw were of form: prover commits to some hard problem, verifier asks to see some part of solution, prover responds with solution (three messages). So we can talk about the prover's first message p_1, verifiers message v, and provers second message p_2. Our proof of knowledge (which isn't clear how to parallelize is as follows):

P	V
1. Send p_1	
	2. Send v
3. Open p_1 as specified in v as p_2.	
	4. Verify transcript (p_1, v, p_2).

The simple way to modify this protocol for parallel execution is to make v commit to its input before p_1. This results in a five message protocol as follows:

P	V
1. Send commitment key ck to V.	
	2. Commit to v using ck.
3. Send p_1	
	4. Open v.
5. Open p_1 as specified in v as p_2.	
	6. Verify transcript (p_1, v, p_2).

We make two main changes here. First we're going to move the prover's message to the beginning of the protocol. Second we're going to allow the prover to announce part of the random coins that will be used for the verifier's randomness. This announcement needs to be before these coins are revealed. The new protocol looks like this.

P	V
1. Send p_1 and commitment key ck to V.	
	2. Commit to q_1 using ck.
3. Commit to random value q_2.	
	4. Open q_1.
5. Verify opening of q_1, open q_2, open p_1 as specified in $v = q_1 \oplus q_2$.	
	6. Verify opening of q_2, verify transcript $(p_1, v = q_1 \oplus q_2, p_2)$.

Lets look at the basics of this protocol. The first is that a simulator only needs to rewind exactly once. In step 1 they select a random value v that they'll be able to answer questions on. They then get V^* to commit and open its commitment to q_1 before rewinding and committing to $q_2 = v \oplus q_1$. The knowledge extractor works as we saw before we run to get an accepting transcript and then rewind until after step 1 and choose a new q_1. Note its possible that P does not use a random value for q_2 so we have to be careful about ensuring that v is likely to be a new value with high probability.

A natural question to ask is why the prover commits to a random value instead of just sending q_2. As one might suspect this is to improve the quality of simulation. The crux of the issue has to do with aborting V^* that we've talked about in the previous class. Once we know q_1 we only have one possible message that the simulator can respond with. If V^* happens to abort on that message we are out of luck since we have to use that q_2. The use of a commitment allows us to randomize the second message of the prover and improve the probability that we can get V^* to accept. Note that before we were preparing different matrices based on V^* commitment and this allowed a natural place for randomization. Since we've already fixed the graph we're using we need some place to randomize V^*.

2 Collapsing interaction

In all of the zero-knowledge proofs that we've seen so far the only role of the verifier is to flip random coins to determine which part of the hard problem

that P should open. The fact that the verifier flips these coins was crucial for two reasons 1) it ensured that they were uniformly distributed so that a cheating prover couldn't produce openings that seemed random 2) the prover didn't know the coins ahead of time so they had to be "ready" to answer both challenges. We may ask for another source of randomness (that can be verifier by both parties) instead of having the verifier create the challenges. However, then the prover may be to able to prepare transcripts ahead of time it seems hard to argue security in this setting.

Before continuing one should note if we don't have access to some random source we have no hope for an interactive protocol. This is because a good prover should always be able to convince a verifier on $x \in L$ and no prover should be able convince V when $x \notin L$. We can directly use the message output by the prover as a witness and the verifier's code as the relation R_L. Thus, the prover message can be seen as w and the protocol cannot possibly be zero knowledge as the verifier has now learned a witness. After knowing that some randomness is necessary it still isn't clear what it means to simulate a non-interactive protocol. If a simulator S can produce a good message without having access to the witness it doesn't seem like access to a witness is important and it seems to put the language polynomial time. It doesn't seem like there is a natural way for the simulator to have additional power over a cheating prover.

This is where we connect the randomness. In non-interactive zero-knowledge we allow the simulator to set the randomness used by the parties. This also goes towards showing non-transferability of the proofs. If a verifier has (x, p) executed on some random resource σ they can't convince another player of x's validity unless they can demonstrate the "randomness" of the resource σ. A simulator could give another resource that "looks" the same.

Definition 1. *A pair of probabilistic machines (P, V) is called* non-interactive proof system *for L if V is polynomial time and the following two conditions hold for some random resource Σ:*

1. *Completeness: For every $x \in L$,*

$$\Pr[V(x, \Sigma, P(x, \Sigma)) = 1] \geq 2/3.$$

2. *Soundness: For every $x \notin L$ and every P^*:*

$$\Pr[V(x, \Sigma, P^*(x, \Sigma)) = 1] \leq 1/3.$$

Definition 2. *A non-interactive proof system (P, V) for L is* zero-knowledge *if there exists a polynomial time S such that*

$$(x, \Sigma, P(x, \sigma)) \approx_c S(x).$$

that is the simulator produces the same view with the ability to create a random resource.

The natural way to try and produce a zero knowledge proof is to run an interactive proof but have Σ do the random coins. For concreteness we assume that the shared resource is a random oracle [BR93]. That is, we assume that both parties have to a random function $H : \{0,1\}^* \to \{0,1\}^\ell$ that is selected

apriori. We often think of this function as being implemented by cryptographic hashes but note that the existence of a random oracle is not an assumption on any particular hash function. The idea of using a random oracle to make proofs non-interactive is due to Fiat and Shamir [FS86] and was originally introduced for creating signatures from identification. Naturally this design is called the Fiat-Shamir paradigm. Recall the proof systems we've been using have the form p_1, v, p_2 where v is a random string from the verifier. To make this system non-interactive the prover uses $v = H(p_1)$ as the random coins of the verifier and sends the pair p_1, p_2 at the same time. The verifier then queries $H(p_1)$ to get v and computes acceptance as though the transcript was p_1, v, p_2. Its easy to see that this system is complete as the distribution produced by P is exactly what would be seen in the interactive setting.

Arguing soundness is a more difficult task. As we noted earlier one of the advantages of using the verifier is that the prover doesn't know what the randomness will be when preparing p_1. Here v is randomly distributed for any p_1 however the prover sees v before deciding whether to send p_1. So the prover can prepare a p_1 that they will only be able to answer with probability $1/2$ and if v is wrong value they just prepare a new p_1'. If soundness error of the proof is relatively high say $1/2$ the prover can always find a p_1 that they can answer quickly. However, if the soundness error for the proof system is relatively low say 2^{-k} for $k = p(|x|)$ then we hope that with a bounded number of queries to the random oracle the adversary can't bias the randomness too much and that the proof retains some soundness. Unfortunately, this analysis is not modular and must be redone for every protocol that Fiat-Shamir is used in. There are some applications where this analysis carries through [PS96]. Importantly, use of a random oracle is crucial here, security can often break when using any hash function [GK03]. To get around this difficulty next class we'll build a non-interactive proof from the ground up using a different random resource a long random string that is known by both parties (which can be created using a random oracle).

References

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the fiat-shamir paradigm. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 102–113. IEEE, 2003.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398. Springer, 1996.