# CSE 5854: Class 07

Benjamin Fuller

February 8, 2018

## 1 Using Zero-Knowledge

Last reading we saw how to build a zero knowledge protocol with overwhelming soundness and completeness in a constant number of rounds (using parallel repetition). Our motivation for introducing zero knowledge was to enforce honest behavior in a joint computation. We'll now see if we are ready to build a computation from these zero-knowledge proofs.

We'll consider a classic example of rock-paper-scissors. There two players $P_0$ and $P_1$ who think of numbers $n_i\{0,1,2\}$ and $P_i$ wins if $n_i = n_{1-i} + 1 \mod 2$. (The standard formulation of rock, paper, scissors can easily be translated to this setting.) The classic way this game is played is for both players to be in the same play and simultaneously announce their moves. Unfortunately, requiring simultaneous action is difficult. (There is a robot that always wins rock paper scissors by seeing what a human is doing and quickly doing the winning move `https://www.youtube.com/watch?v=3nxjjztQKtY`.) Using the tools we've seen so far a natural solution is to force each player to commit to their action before things are announced. For concreteness we'll assume use of Pedersen commitment.

$P_0$

1. Input $b_0$ and $g, h \in \mathbb{Z}_p$.
2. Sample $r_0 \leftarrow \mathbb{Z}_p^*$. Set $c_0 = g^{r_0} \cdot h^{b_0}$.
3. Send $c_0$ to $P_1$.

7. Receive $c_1$.
8. Open $c_0$ (by sending $b_0, r_0$).

11. Receive $b_1, r_0$ ensure commitment verifies.
12. Compute winner.

$P_1$

1. Input $b_1$ and $g, h \in \mathbb{Z}_p$.

4. Receive $c_0$ from $P_0$.
5. Sample $r_1 \leftarrow \mathbb{Z}_p^*$. Set $c_1 = g^{r_1} \cdot h^{b_1}$.
6. Send $c_1$ to $P_1$.

9. Receive $b_0, r_0$ ensure commitment verifies.
10. Open $c_1$.

12. Compute winner.

It seems like this has fixed the problem as both parties have to compute a commitment before seeing an opening and thus both parties have to be honest. However, this is not the case. Suppose that instead of sampling $r_1$ and computing $c_1$, $P_1$ does the following: set $c_1 = c_0 \cdot h$. This value $c_1$ is a valid commitment and $P_1$ can send it and execute the protocol. $P_1$ however has no idea what value they just committed to. However, on seeing the opening from $P_0$ they can properly produce a decommitment value and they win whenever $b_0 \in \{0, 1\}$.[1] You might say this behavior is easy to catch, however I can fully randomize this behavior by multiplying this value by $g^{r_1}$ for some random power $r_1$. The goal of zero-knowledge was to enforce honest behavior. We might ask each party (or just $P_1$) to compute a zero-knowledge proof that they have produced a "good" commitment. We could easily ask each party to prove that their value is in the output space of the commitment scheme.

This doesn't prevent the behavior that we're concerned about for two reasons: 1) every value in $\mathbb{Z}_p$ is a "good" commitment so there's "nothing" to prove and 2) we're really trying to prevent $c_1$ from depending on $c_0$. We'll try and address problem 2 first. We could require that the two parties "commit" to the randomness that they are going to use first to prevent their commitments from being related. However, we'd then have the same problem on those commitments.

We're going to address both of these problems by strengthening zero-knowledge by requiring more from the prover. We want the prover to "know" the value they are proving. For statements in $\mathcal{NP}$ we can say that we want the prover to know some witness $w$. In the example we saw $P_1$ does not "know" values $b_1$ and $r_1$ that open its malicious commitment (until seeing values from $P_0$).

As we saw defining "knowledge" was a bit tricky. To say that the verifier gained no knowledge we showed how to simulate the verifiers view without knowing a witness. Recall that the simulator had extra power to depend on the verifier (and thus reset, snapshot, rewind). We'll follow the same tact here. We'll define another machine that interacts with the prover and should be able to output the witness. We'll call this machine an "extractor" named so because it extracts the witness from a prover.

**Definition 1.** *Let $L \in \mathcal{NP}$ be a language. A proof system $(P, V)$ for $L$ is a proof of knowledge with soundness error $\delta(|x|)$ if $\forall x, \forall P^*$. there exists $E$ with oracle access to $P^*$ such that*

$$\Pr[w \leftarrow E^{P(x)}(x) \wedge (x, w) \in R_L] \geq \Pr[\langle P^*, V\rangle(x) = 1] - \delta(|x|).$$

Here $E$ is given oracle/black-box access to $P^*$ this means that they are allowed to send messages to $P^*$, reset it, and rewind it. However, they are not allowed to look at its internal state. There are simulators that look at the cheating verifiers state though we are not going to discuss them in class. This area is called non-black box zero knowledge and is a very interesting topic for research. The intuition here is that if you can interacting with a prover multiple times zero knowledge becomes complete knowledge. Thus, it seems that zero-knowledge and proving knowledge are at odds, making the extractor's job easier seems to make the simulator's job more difficult and vice versa. However, it is possible to achieve both properties simultaneously. It is important to note that

[1] The mismatch of groups keeps rounding from working if $b_0 = 2$.

this definition is required for all cheating provers that are able to convince the verifier with probability over $\delta$. If a prover convinces $V$ with probability less than $\delta$ then the knowledge extraction is only guaranteed to occur with probability 0. Lastly, knowledge extraction can replace soundness which seems a little odd at first as knowledge extraction doesn't talk about whether $x$ is in $L$. (You'll prove this on the next homework.)

We'll start by giving a simple proof of knowledge for discrete logarithm. Here both parties will input $g, h$ where $g^x = h$ and the prover will show that they "know" $x$.

| $P$ | $V$ |
|---|---|
| 1. Input $g, h$ and $x$ where $g^x = h \mod p$. | 1. Input $g, h \in \mathbb{Z}_p$. |
| 2. Sample $r \leftarrow \mathbb{Z}_p^*$. Send $y = g^r$ to $V$. | 3. Receive $y$ from $P$. |
| | 4. Sample $c \leftarrow \mathbb{Z}_p^*$, send to $P$. |
| 4. Receive $c$. | |
| 5. Compute $z = cx + r$ send to $V$. | |
| | 6. Check if $g^z \stackrel{?}{=} h^c \cdot y$. |

Before showing a knowledge extractor lets note the intuition behind this protocol. First, the value $r$ is used to blind the other responses of $P$. Without this value $P$ would be sending $z = cx$ and $V$ could easily solve for $x$ and learn the discrete log. This would prevent the system from being zero-knowledge. Roughly $r$ serves as a one time pad for $x$ however $V$ is also presented with $g^r$ so one has to check that this actually remains zero-knowledge. The idea behind knowledge extraction is that the prover is able to "solve" random linear equations that involve $x$. The nice thing about linear equations is that without enough instances the solution is completely random.

The knowledge extraction argument is remarkably similar to the proof of binding for Pedersen commitments. Consider a prover $P^*$ that is interacting with $E$. Here $E$ just honestly executes the protocol and gets $z, c$. It then rewinds $P^*$ (to after step 2) and reexecutes the protocol to get $z', c'$. Before trying to compute a witness it checks that both repetitions of the protocol were successful. It then computes

$$\frac{z - z'}{c - c'} = \frac{cx + r - (c'x + r)}{c - c'} = \frac{(c - c')x}{c - c'} = x.$$

The main difficulty in analyzing this protocol is that the cheating prover $P^*$ doesn't have to do the right thing all of time. It could answer half the time with $\perp$ (just aborting the protocol). Since $E$ is running $P^*$ twice it will see a $\perp$ more frequently (with probability 3/4). The natural solution to this is to repeat the experiment more times. However, you need to be careful about remaining polynomial time. Since $E$ does not know apriori how frequently $P^*$ fails to output a good $z$ it is difficult to distinguish between a $P^*$ that almost always outputs $\perp$ and a $P^*$ where you just happened to get unlucky in terms of experiment randomness.

The most natural way to deal with this is to have the extractor first run $P^*$ multiple times with completely independent transcripts to judge its success probability that is how often it convinces an honest verifier. If this probability is too small (negligible) then we simply give up and don't use $P^*$ Otherwise, we know how frequently $P^*$ will succeed and we can rewind multiple times (proportional to this success probability). This strategy works when $P^*$ succeeds with inverse polynomial probability however, $E$s strategy is still murky when $P^*$ succeeds with negligible probability. $E$ can try and do something more sophisticated in this case like trying to guess $w$ itself. It only has to be successful a negligible fraction of the time so this might be something that is possible in polynomial time. However, one has to be careful on the connection between how frequently $E$ outputs a witness and how frequently $P^*$ succeeds. These are two different negligible functions and the definition requires $E$ to be successful more frequently.

# 2 Proof of Knowledge for $\mathcal{NP}$

It turns out that we have already constructed a proof of knowledge for all of $\mathcal{NP}$ with our zero-knowledge proof of the existence of a Hamiltonian cycle. We review that protocol here for completeness.

Recall that a graph has a Hamiltonian path if there exists a list of edges $(u_1, v_1), ..., (u_\ell, v_\ell)$ that "touches" every vertex of the graph once (all nodes except the source and sink are repeated).

The protocol proceeds as follows:

| $P$ | $V$ |
|---|---|
| 1. Input $G$ and cycle. | 1. Input $G$. |
| 2. Pick random permutation $\pi$. | 2. Pick $b \leftarrow \{0, 1\}$. |
| 3. Commit to adjacency matrix of $\pi(G)$ and $\pi$ and send to $V$. | |
| | 4. Receive commitment to adjacency matrix and commitment to permutation. |
| | 5. Send $b$ to $P$. |
| 6. Receive $b$. | |
| 7. If $b = 0$ open all commitments. | |
| 8. If $b = 1$ open commitments on the cycle. | |
| | 9. Verify commitments. If $b = 0$ check if matrix is $\pi(G)$. If $b = 1$ check that a cycle was revealed. |

Its not hard to see that is a proof of knowledge (with soundness error 1/2). For any $P^*$ it either succeeds with probability $0, 1/2$ or 1 (we can assume that $P^*$ is not random and it uses the "best" randomness to convince the verifier, this lets us consider only the verifier's randomness). We show that we succeed in extracting with $P^*$ works with probability 1 (we are allowed to never succeed when $P^*$ succeeds with probability 1/2 or 0).

We know describe an $E$ for such a setting (recall that $E$ does not need to respect the honest $V$'s behavior). After receiving the commitments we send the challenge $b = 0$ and get the response. We then rewind $P^*$ and send $b = 1$ and get the response. Since the verifier would always accept both of these responses are of the right form. Thus we have been told a permutation $\pi$ of $G$ and a Hamiltonian cycle in the permuted graph $G' = \pi(G)$. We simply compute $\pi^{-1}(G')$ to find a Hamiltonian cycle in the original graph.

This proof is also zero knowledge. Consider the following simulator $S$ for any $V^*$:

1. Repeat some $k$ times.
   (a) Flip $b' \leftarrow \{0, 1\}$.
   (b) If $b' = 0$ commit to a random permutation $\pi$ of $G$ and $\pi$ itself.
   (c) If $b' = 1$ commit to the all 1s adjacency matrix.
   (d) Send values receive $b$.
   (e) $b = b'$ output transcript, otherwise go to step (a).