CSE 5854: Class 06

Benjamin Fuller

February 6, 2018

1 Constant Round Zero Knowledge

In the previous section we saw how to construct a zero knowledge protocol for all of NP with completeness 1 and soundness $1 - 1/|E|$ (for the graph we use for 3-colorability). In addition, we saw that we cannot automatically parallelize zero-knowledge. However, it does not mean that concurrent parallel application does not work for this particular zero-knowledge protocol. (The protocol we discuss today was first put forth by Goldreich and Kahan [GK96].)

We consider two toy examples of a "parallel" 3-colorability proof. First consider the following:

Recall that P has auxiliary input π that is a 3-coloring of the graph G.

1. The prover inputs G, π.
2. The prover samples $ck \leftarrow \mathsf{Setup}(1^{|x|})$. The prover selects a random $\psi : \{0,1,2\} \rightarrow \{0,1,2\}$ and computes $\phi = \psi \circ \pi$.
3. For each vertex $u \in V$, P creates $(c_u, d_u) \leftarrow \mathsf{Commit}(\phi(u))$ and sends c_u to the verifier (as well as ck).
4. V randomly selects k random edges u, v and sends these edges to P.
5. The prover receives some k edges u, v and sends d_u and d_v to V.
6. The verifier computes $u_c \leftarrow \mathsf{Open}(c_u, d_u), v_c \leftarrow \mathsf{Open}(c_v, d_v)$ and accepts iff $u_c \neq v_c$ for all received edges.

Note that this solution only requires 3 rounds of communication and is fairly efficient as well with total communication roughly $|V| \cdot k + k \cdot k$ if we assume that commitments and openings are also of size of the security parameter k. However, its fairly easy to see that this protocol is not zero-knowledge. If $k \geq |E|$ a cheating verifier can ask for each vertex to be included in some requested edge and learn a 3-coloring of the graph from the prover's response. Its not difficult to see that this interaction cannot be simulated. We found a simulator for the one edge setting by the fact that a random coloring worked for a single edge with high probability (but it was not consistent). Here the simulator somehow needs to be consistent in its coloring which it can't do. Note that even when $k < |E|$ it isn't hard to see that there exists an auxiliary input of V^* that allows for complete witness recovery after k interactions that can't be simulated (a partial coloring of the graph).

A natural way to overcome this problem is to disconnect the edges that the simulator has to try and color. In essence we have the graph no longer be shared. The protocol looks as follows:

1. The prover inputs G, π.
2. The prover samples $ck \leftarrow \mathsf{Setup}(1^{|x|})$.
3. The prover selects k random $\psi_k : \{0,1,2\} \to \{0,1,2\}$ and computes $\phi_i = \psi_i \circ \pi$ for $1 \leq i \leq k$.
4. Perform the following for $1 \leq i \leq k$:

   For each vertex $u \in V$, P creates $(c_{i,u}, d_{i,u}) \leftarrow \mathsf{Commit}(\phi_i(u))$ and sends $c_{i,u}$ to the verifier (as well as ck).
5. V randomly selects k random edges u, v and sends these edges to P.
6. The prover receives some k edges (u_i, v_i) and sends $d_{i,u}$ and $d_{i,v}$ to V.
7. The verifier computes $u_{i,c} \leftarrow \mathsf{Open}(c_{i,u}, d_{i,u}), v_{i,c} \leftarrow \mathsf{Open}(c_{i,v}, d_{i,v})$ and accepts iff $u_{i,c} \neq v_{i,c}$ for all received edges.

Note that this protocol still executes in three rounds. However, the overall communication has grown by a multiplicative factor of k as we have the vertices separately committed k times. Its not clear this protocol has a trivial attack by a malicious verifier V^* as learning many local colorings of edges does not lead to a global covering. However, its also not clear that we can simulate this protocol for a malicious verifier V^*.

Assume we use the same simulator as the basic protocol that commits to a random coloring. Here (assuming for a moment that the commitment scheme is perfectly secure) we expect to answer all of the edges requested by the verifier with probability of $(2/3)^k$. We can reinitialize the verifier V^* as many times as we like and we still aren't likely to get a set of good edges. With overwhelming probability in every execution the verifier will ask for some edge that wasn't consistent.

So far all of the simulators we've seen have done the same thing, they have tried to prepare a set of messages for the verifier and if things go poorly they restart the verifier and try again until they are successful. It turns out a simulator can do more with a verifier V^* that they have running. They can actually snapshot a verifier V^* at any particular point and restore to that state. There isn't any special about restoring to the initial state. The simulator just needs to remember the state, tape head positions, and tape contents. However, it isn't clear that when V^* is reset to this state we can provide new randomness to the verifier. We might be able to detect how much of their randomness string they've read and replace randomness after that. However, V^* may read all of its randomness as its first action. In general, we shouldn't count on being able to randomize V^* as a success strategy. We need to randomize "ourself" as the simulator.

To fix the protocol above consider that answering V^*'s queries is easy once they are known. If we could have V^* tell us the edges before we commitment then things would be easy. Reordering these two messages is however a bad idea

because then a malicious prover could prove false statements breaking soundness.

What we'd like is a way for the simulator to know what edges will be opened by the simulator without a prover knowing what messages are coming. This seems impossible as a prover has unlimited resources and a simulator must run in polynomial time. We need to rely on the ability of the simulator to "rewind" the verifier which the prover can't do.

The intuition behind the protocol is as follows. We ignore the parallel repetition for the moment that is only important when analyzing success probability.

1. The prover inputs G, π.
2. The prover samples $ck \leftarrow \mathsf{Setup}(1^{|x|})$.
3. The verifier samples $ck' \leftarrow \mathsf{Setup}(1^{|x|})$.
4. The verifier selects edge u, v and computes $(c, d) \leftarrow \mathsf{Commit}(u, v)$ and sends c to the prover as well as ck'.
5. The prover selects random $\psi : \{0, 1, 2\} \rightarrow \{0, 1, 2\}$ and computes $\phi = \psi \circ \pi$.
6. For each vertex $u \in V$, P creates $(c_u, d_u) \leftarrow \mathsf{Commit}(\phi_i(u))$ and sends c_u to the verifier (as well as ck).
7. V sends d to P (opening the edge u, v).
8. P checks the validity of the commitment and if valid sends d_u and d_v to V.
9. The verifier computes $u_c \leftarrow \mathsf{Open}(c_u, d_u), v_c \leftarrow \mathsf{Open}(c_v, d_v)$ and accepts iff $u_c \neq v_c$ for all received edges.

Note that as written this is a 4-round protocol: 1) commitment from verifier 2) commitment from prover 3) open from verifier and 4) open from prover. The idea here is that the prover shouldn't know what is contained in the commitment of the prover and thus the protocol doesn't not change for the prover.

However, the simulator now has the following strategy:

1. they get V^* to commit to the edges to be opened,
2. they commit to a random coloring (or whatever),
3. get V^* to open the committed edges.
4. Rewind V^* to after its commitments have been sent.
5. Produce a random coloring subject to the edge that V^* will request being correct.
6. Run V^* to completion.

Discussion Question 10: In the above protocol both parties use the same commitment scheme that is perfectly binding and computationally hiding. Why is this a problem?

Discussion Question 11: Right now both parties generate the key for "their" commitment. Using the Pederson based and encryption based commitments we introduced, what is a better strategy?

Discussion Question 12: The intuition behind this simulation is that V^* should only be able to produce one opening to their commitment and thus S can properly prepare. However, V^* could simply abort the protocol. What should S do in this case?

References

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9(3):167–189, 1996.