CS 294 – Secure Computation March 10, 2016

Lecture 16: Concurrent Zero Knowledge

Instructor: Sanjam Garg *Scribe: Akshayaram Srinivasan*

1 Introduction

Zero knowledge proofs are remarkable tools in cryptography as they enable a party (called as the prover) to prove to another party (called as the verifier) the truth of a statement without revealing anything else but the assertion. In the traditional zero knowledge setting, we consider a single interaction between a prover and a verifier where we provably say that the verifier (running in polynomial time) cannot learn anything apart from the truth of a statement. A more realistic and practical scenario (in the age of the internet) is to consider a setting where multiple verifiers (which may be malicious) interact with independent prover executions and mount a joint coordinated attack to try learn non-trivial information from the prover. The coordinated attack could include scheduling (interleaving) of messages across different executions. This scenario is known as the *concurrent setting*. Proving zero knowledgeness in the concurrent setting is a much harder task than proving zero knowledge in the single interactive setting since the verifier has the power to schedule messages across different executions. To see this, let us consider the example of four round Blum's hamiltonicity zero knowledge protocol with negligible soundness error. Recall in that protocol, the verifier commits to the query vector in the first round. The prover commits to the every edge in the random permutation of the graph in the second round. The verifier then opens the query vector in the third round and prover responds according to the query vector in the final round. [1] The proof strategy for the simulator is to commit to some garbage value in the second message. Depending on the verifier's query vector, it rewinds the verifier and commits to appropriate messages. When the same protocol is run in the concurrent setting the simulation strategy fails. To illustrate this, consider n concurrent sessions where the scheduling of messages is such that for every i from $1, \cdots, n$ (in the same order) the first two messages of the protocol are executed first. Then we execute the last two messages of the protocol in the reverse order that is from $n, \cdots, 1$. Furthermore, the random query vector in sessions $i+1, \cdots, n$ is an output of the a pseudorandom function on the prover's message in the second round of session i. To see that the straightforward extension of simulation strategy fails since to simulate session i it becomes necessary to simulate all of the sessions $i+1, \cdots, n$ (since their commitment has changed). Hence, the session n needs to be simulated an exponential number of times.

2 Definition

We now give the definition of black box concurrent zero knowledge.
Let $\mathsf{View}_{V^*}(P(x,w))$ denote the random coins and set of messages received by V^* in its interaction with $P(x,w)$.

Definition 1 *Let L be a language in NP and let R_L be its corresponding witness relation. An interactive proof system (P,V) for L is black box concurrent zero knowledge proof system if there*

[1] In class we actually did a variant of this protocol which runs in six rounds but has a simpler security analysis.

exists a simulator S such that for all polynomials $q(\cdot)$ and for all V^* that runs at most $q(|x|)$ sessions and runs in time $\mathsf{poly}(q(|x|))$ and for all $(x, w) \in R_L$,

$$\mathsf{View}_{V^*}(P(x, w)) \stackrel{c}{\approx} S(q, x)$$

3 Protocol

We will now describe the concurrent zero knowledge protocol. Let Com be a commitment scheme.

- $V \to P$: Sample a random $\sigma \in \{0,1\}^k$. Send $\mathsf{Com}(\sigma), \mathsf{Com}(\sigma^0_{i,j}), \mathsf{Com}(\sigma^1_{i,j})$ where for all $i, j \in [k]$, $\sigma^0_{i,j} \oplus \sigma^1_{i,j} = \sigma$.
- For every $j \in [k]$ (called as a *slot*)
  - $P \to V$: Send $r_{1,j} \cdots r_{k,j}$ where each $r_{i,j} \in \{0,1\}$.
  - $V \to P$: Decommit to $\sigma^{r_{i,j}}_{i,j}$ for all $i \in [k]$.
- Hamiltonicity protocol:
  - $P \to V$: Send the first message of the protocol.
  - $V \to P$: Send decommitments to $\sigma^{1-r_{i,j}}_{i,j}$ for all $i, j \in [k]$.
  - $P \to V$: Respond according to σ.

4 Proof Sketch

We will now describe the simulation strategy. Let us assume for now that q (which is the maximum number of concurrent sessions that an adversary can handle) is known beforehand. We will later see how to get rid of this assumption. Let S_q be the simulator that handles at most q sessions.
A high level proof strategy is to extract σ by rewinding one of the slots in each session. If we are able to extract σ then we are in good shape to simulate the hamiltonicity protocol accordingly.
We will call the first message of the hamiltonicity protocol to be the *problematic message*. Because in this message the simulator has to commit to either a complete graph or to the current graph. Till that message, simulation strategy is exactly same as the original proof strategy. It just needs to send random strings. If we are able to extract σ in each session before the problematic message is to be sent for that session then the simulation strategy wins.
We first identify a session s where the first problematic message is to be sent. We first give the strategy to simulate this session and we simulate the other sessions in a similar manner.
The adversary could start several sessions within each the slots of session s. We identify $k/2$ slots that have at most $2q/k$ sessions that start within each of the slots. Since the adversary can start at most q sessions, we can find $k/2$ such slots by pigeon hole argument. Looking ahead we will rewind each of the slots once and try and extract σ from at least one of them.
We now give the description of the simulator.

Simulator Description. Simulator identifies the session s where the first problematic message is to be sent by running executing the adversary by following the honest prover strategy until the point where the problematic message is to be sent. It then identifies $k/2$ slots that have at most $2q/k$ sessions that start within the slot. [2] We will be rewinding the adversary in these $k/2$ slots. We call the simulator's rewinding thread to be the lookahead thread. [3] We now describe simulator's behavior in the lookahead thread of a single slot.

- If the adversary starts more than $2q/m$ sessions in the lookahead thread, we abort the rewinding for that slot. [4] We now calculate the probability that the adversary starts at most $2q/m$ sessions in the main thread but starts over $2q/m$ sessions in the lookahead thread. This is at most $1/2$ since the simulator behaves exactly same in the lookahead and the main thread. We would call the sessions that start within rewinding the slot as *new* sessions and sessions that start before the rewinding to be *old*. We need to take care of responding to the old sessions as well as the new sessions in the lookahead thread.

- For those sessions that start in the rewinding thread (which are at most $2q/m$) i.e the new sessions, we recursively call $S_{2q/k}$ to take care of responding to those sessions. We assume $k = \Omega(|x|)$ and hence the depth of the recursion tree is constant and hence in time polynomial in $|x|$ we can take care of new sessions that start within rewinding a slot.

- For old sessions that have already been completed before rewinding this slot in the main thread, the simulator responds exactly the same in the lookahead thread as in the main thread.

- For old sessions that have not been completed before rewinding this slot but are also not complete after rewinding is done, the simulator just responds with random strings in the lookahead thread.

- For old sessions that have not been completed before rewinding this slot but where the problematic message is to be sent in the lookahead thread, the simulator does the exact same rewinding strategy as in main thread. One crucial observation that helps us bound the running time of the simulator is that once a session's σ is extracted either in the main thread or in the lookahead thread then we can simulate the session perfectly i.e we need not again rewind the adversary.

Analysis. We now estimate the running time of the simulator. For every session, the simulator has to run in time proportional to the number of sessions that start before the problematic message for this session is to be sent. This is at most q. Hence, for each session the simulator runs in polynomial time and since there are polynomially many sessions the simulator's running time is bounded by a polynomial.
We now estimate the probability that the simulator's rewinding strategy fails to extract σ in one of the slots. Let p be the probability that the adversary responds to simulator's challenge in the

[2] By within the slot we mean before the adversary responds by opening the commitments to simulator's challenge random string.

[3] In the lookahead thread the simulator rewinds the adversary and gives a new challenge bits $r_{i,j} \in \{0,1\}$ for all $i \in [k]$

[4] We continue the rewinding strategy for the other slots.

main thread. Since the simulator's challenge in the lookahead thread is identically distributed to its challenge in the main thread, the adversary responds to the simulator's challenge in the lookahead thread with probability p. Hence, for every slot the simulator's rewinding strategy fails is bounded by $p(1-p) < 1/2$. Hence, after rewinding $k/2$ slots the probability that the simulator strategy fails is at most $(1/2)^{k/2}$ which negligible.

We have assumed that q is known beforehand. For unknown q, the simulator uses a binary search like strategy. That is, it starts with S_2. If it fails, then it starts with S_4 and so on until the simulation strategy succeeds. This takes an additional, $\log |x|$ time.