CS 294-121 – Secure Computation February 9, 2016

Lecture 7: Zero Knowledge II

Instructor: Sanjam Garg *Scribe: James Wei*

1 Overview: Hamiltonian Cycles

Suppose we have some undirected graph $G = (G_V, G_E)$ represented as an adjacency matrix with $|G_V|$ rows and $|G_V|$ columns. G is said to be *Hamiltonian* if there exists a cycle C_H that passes through each vertex in G_V exactly once. This cycle is called a *Hamiltonian cycle*. Finding a Hamiltonian cycle in a graph if one exists is an NP-complete problem. We would like to use a zero-knowledge protocol to demonstrate that a graph is Hamiltonian without revealing the Hamiltonian cycle contained within it.

2 Protocol

Let P be the prover and V be the verifier. In this zero-knowledge exchange, P wishes to prove to V that G is Hamiltonian. Note that both parties begin with a copy of G. The protocol proceeds as follows:

1. P samples some random permutation ϕ over G_V such that $\phi : G_V \to G_V$. P then uses ϕ to construct $\phi(G)$, the permuted version of the adjacency matrix of G isomorphic to the original graph.
   P commits to each edge $e_{i,j} \in \{0, 1\}$ in $\phi(G)$, the permuted adjacency matrix. Denote these commitments $c_{i,j} = \text{Commit}(e_{i,j})$. Additionally, P commits to the permutation ϕ that was used to permute the adjacency matrix; denote this commitment $c_\phi = \text{Commit}(\phi)$. P proceeds to send c_ϕ and each $c_{i,j}$ to V.

2. After receiving P's commitments, V responds with a challenge bit $b \in \{0, 1\}$.

3. If $b = 0$, P sends the keys corresponding to c_ϕ and each $c_{i,j}$ to V. V then uses these keys to unlock all of P's commitments to verify that the permutation ϕ over G_V yields a permuted edge map identical to the one formed by the set of all $e_{i,j}$.

   If $b = 1$, P translates the Hamiltonian cycle C_H onto $\phi(G)$ and sends the keys for only those $c_{i,j}$ whose edges are in the permuted Hamiltonian cycle. V then uses these keys to unlock the value of each $e_{i,j}$ in the cycle and checks that the set of unlocked $e_{i,j}$'s constitutes a Hamiltonian cycle on the permuted graph $\phi(G)$.

3 Completeness

If P knows a Hamiltonian cycle C_H in G, P will successfully respond to V's challenge bit b.

If $b = 0$, P can prove that $\phi(G)$ is isomorphic to G by revealing ϕ and all $e_{i,j}$. Since P correctly committed to ϕ, V can verify that the permutation ϕ over G_V yields a permuted edge map identical to the one formed by the set of all $e_{i,j}$.

If $b = 1$, P can prove that a Hamiltonian cycle exists in $\phi(G)$ by revealing only the $e_{i,j}$ contained in the permuted cycle. P can identify the necessary $e_{i,j}$ by transforming C_H onto $\phi(G)$. V can verify that the unlocked $e_{i,j}$ corresponds to a Hamiltonian cycle.

4 Soundness

If P does not know a Hamiltonian cycle C_H in G, P can cheat by attempting to anticipate the challenge bit b. P can either correctly generate $\phi(G)$ isomorphic to G or construct an arbitrary Hamiltonian cycle on the complete graph on $|G_V|$ vertices. Since P does not know the Hamiltonian cycle on G, P cannot do both. Since the challenge bit b is sampled from $\{0, 1\}$ uniformly at random, the probability that P correctly predicts b in a single round is $\frac{1}{2}$. With a soundness error of $\frac{1}{2}$, if the protocol is run κ times, where κ is some security parameter, then the probability that P convinces V without knowing C_H is $2^{-\kappa}$.

5 Zero Knowledge

The information that P sends to V during each round does not reveal any information about the the Hamiltonian cycle C_H in G. Depending on the challenge bit b, V learns either a graph permutation $\phi(G)$ or a Hamiltonian cycle C_H^{ϕ} on $\phi(G)$. V needs both ϕ and C_H^{ϕ} to recover C_H on G. As long as P generates a distinct ϕ every round, V gains no knowledge about C_H on G.

Conversely, if P has prior knowledge about the challenge bit b that V will send in the second step of the protocol, then P can architect its commitments in the first step as to fool V into believing that P knows some C_H on G. Specifically, if P knows that V will send $b = 0$, then P can commit to an arbitrary permutation $\phi(G)$ without knowing a Hamiltonian cycle and still pass the challenge. If P knows that V will send $b = 1$, then P can commit to the complete graph on $|G_V|$ vertices that is not a permutation of G; P can then reveal any arbitrary Hamiltonian cycle on the complete graph to V.

As such, it can be shown that the above protocol is black-box zero-knowledge: there exists a PPT simulator S for every PPT cheating verifier V^* such that the output distribution of the interaction between S and each V^* is computationally indistinguishable from the output distribution of the interaction between each V^* and some honest prover P. The construction of such a simulator S follows from the simulator for the zero-knowledge graph coloring problem presented in the previous lecture. To recap, S predicts the challenge bit b' and commits either to a valid graph permutation or the complete graph on $|G_V|$ vertices with a trivial Hamiltonian cycle. If the predicted challenge bit matches the actual challenge bit $b' = b$, then S proceeds by successfully responding to the challenge; otherwise, S aborts, rewinds the transcript, and tries again.

6 Parallelized Protocol

6.1 Constructing a constant-round protocol

Each round of the protocol described above requires three steps; reducing the probability of V accepting a false statement to $2^{-\kappa}$ would require a runtime of 3κ. To save on the number of rounds required while maintaining the security of the protocol, we would like run the rounds in parallel. For some security parameter κ, the parallelized protocol without modification would proceed as follows:

1. P samples κ random permutations $\phi_i, i \in \{1, 2, \ldots, \kappa\}$ and constructs κ permutations of G. P commits to each edge in each permuted graph $c_{i,m,n} = \text{Commit}(e^i_{m,n})$ and to each permutation $c_{\phi_i} = \text{Commit}(\phi_i)$. P sends each c_{ϕ_i} and each $c_{i,m,n}$ to V.

2. After receiving P's commitments, V responds with a length-κ challenge string $b \in \{0,1\}^\kappa$ where each bit b_i corresponds to the challenge bit for the i^{th} round of the protocol running in parallel.

3. For each challenge bit $b_i \in b$, P either sends the keys corresponding to each c_{ϕ_i} and each $c_{i,m,n}$, or just the keys to the $c_{i,m,n}$ whose edges correspond to a Hamiltonian cycle in their respective permutations.

Completeness and soundness for the parallelized protocol follow from the sequential protocol. However, we encounter a problem when trying to extend zero knowledge to the parallelized protocol. Specifically, the $\mathcal{ZK}$ simulator S we used previously would have to correctly guess the length-κ challenge string b in order to avoid rewinding. Since this happens with probability $2^{-\kappa}$, for large κ, this might cause the simulator to run in unbounded time.

6.2 An initial solution for zero knowledge

To solve this problem, an additional message $c_b = \text{Commit}(b)$ is sent from the verifier to the prover before the first step in the original parallelized protocol in which the verifier commits to all κ challenge bits b prior to receiving the prover's commitments. Then, instead of sending b in the clear in step 2, the verifier instead sends the k_{c_b}, the key used to unlock c_b. It is easy to see that the secrecy property of the committed message c_b preserves the soundness of the protocol.

With this new message c_b, one way the $\mathcal{ZK}$ simulator S could operate would be to perform the following:

1. Receive c_b from V^*.

2. Send arbitrary $c_{i,m,n}$ and c_{ϕ_i} to V^*

3. Upon receiving k_{c_b} from V^*, rewind the interaction and send $c_{i,m,n}$ and c_{ϕ_i} to V^* so that S can successfully respond to each challenge bit.

However, since V^* is adversarial, it is free to deviate from the protocol at any time. Specifically, V^* may choose to abort at any time (during either the initial run or the post-rewind run of the protocol). If V^* aborts prior to sending k_{c_b} in the initial run, but does not abort in the rewound run, then S cannot successfully respond to V^* since it was not able to extract the challenge bits in the initial run.

If S simply outputs the transcript in which V^* aborts prior to sending k_{c_b} in either of the two runs, then the resulting distribution would be skewed towards aborted runs since V^* has an increased number of opportunities to abort with the addition of the rewound thread.

6.3 A zero-knowledge parallelized protocol

Here, we construct a six round protocol as a proof system to demonstrate zero knowledge. Let k be some additional security parameter.

The protocol begins with a three step "preamble" before proceeding to the parallelized challenges:

1. V sends P the following:
   (a) $c = \text{Commit}(\sigma)$, where σ is a random κ-bit string
   (b) $\{c_i^b = \text{Commit}(\sigma_i^b)\}, \forall i \in \{1, \ldots, k\}, b \in \{0, 1\}$ such that $\sigma_i^0 \oplus \sigma_i^1 = \sigma, \forall i \in \{1, \ldots, k\}$
2. P sends V $r = r_1 \ldots r_k, r_i \in \{0, 1\}$, where r is a random k-bit string
3. V sends the keys for $\{c_i^{r_i}\}, \forall i \in \{1, \ldots, k\}$ to P
4. Run the parallelized protocol.
   (a) P samples κ random permutations $\phi_i, i \in \{1, 2, \ldots, \kappa\}$ and constructs κ permutations of G.
   P commits to each edge in each permuted graph $c_{i,m,n} = \text{Commit}(e_{m,n}^i)$ and to each permutation $c_{\phi_i} = \text{Commit}(\phi_i)$. P sends each c_{ϕ_i} and each $c_{i,m,n}$ to V.
   (b) After receiving P's commitments, V responds with the keys for c and each $\{c_i^{1-r_i}\}, \forall i \in \{1, \ldots, k\}$ to P. P can now verify that each $\sigma_i^0 \oplus \sigma_i^1 = \sigma$. Once the consistency of σ is verified across all pairs of c_i^b, P treats each bit in σ as the challenge bit for the i^{th} round of the parallelized protocol.
   (c) For each challenge bit in σ, P either sends the keys corresponding to each c_{ϕ_i} and each $c_{i,m,n}$, or just the keys to the $c_{i,m,n}$ whose edges correspond to a Hamiltonian cycle in their respective permutations.

It can be shown that the above protocol is black-box zero-knowledge: there exists a PPT simulator S for every PPT cheating verifier V^* such that the output distribution of the interaction between S and each V^* is computationally indistinguishable from the output distribution of the interaction between each V^* and some honest prover P. For this version of the protocol, the $\mathcal{ZK}$ proof relies on the ability of S to robustly extract the challenge bit string σ through repeated rewound calls to the preamble.

Specifically, consider the following sequence of events in the interaction between S and V^*:

1. V^* sends S the following:

(a) $c = \text{Commit}(\sigma)$, where σ is a random κ-bit string

(b) $\{c_i^b = \text{Commit}(\sigma_i^b)\}, \forall i \in \{1, \ldots, k\}, b \in \{0, 1\}$ such that $\sigma_i^0 \oplus \sigma_i^1 = \sigma_i, \forall i \in \{1, \ldots, k\}$

2. S sends V^* some random $r \in \{0, 1\}^k$

3. V^* sends the keys for $\{c_i^{r_i}\}, \forall i \in \{1, \ldots, k\}$ to S

4. `while true`:

   (a) S rewinds

   (b) S sends V^* some random $r' \in \{0, 1\}^k$

   (c) If V^* sends `ABORT`, then `continue`; otherwise, if V^* sends the keys for $\{c_i^{r'_i}\}, \forall i \in \{1, \ldots, k\}$ to S, then `break`

5. Proceed with the parallelized portion of the protocol

If r' differs from r in at least one location, then S can compute σ by unlocking some commitment pair c_i^0 and c_i^1 and computing $\sigma = \sigma_i^0 \oplus \sigma_i^1$, thereby figuring out what challenges V^* plans on issuing in each thread of the parallelized protocol. The probability that r is exactly equal to r' is 2^{-k}, which grows negligible as k grows large. We can also claim that the σ extracted from the rewound thread differs from the σ unlocked in the main thread (during the parallelized protocol) with negligible probability; this claim follows from the computational binding property of the commitment scheme. With this information, S can reliably simulate an honest prover during the parallelized protocol that follows.

This construction solves the issue of V^* arbitrarily aborting that we previously discussed. Consider each place in the interaction with S during which V^* can chose to abort. If V^* choses to abort during step 1, step 3, or any time during the parallelized portion of the protocol, then S can simply output the aborted transcript since the output would be indistinguishable from the output of an honest prover. In the case that V^* aborts during one of the rewound sequences in step 4, S should ignore the abort and continue iterating and rewinding. This yields an expected polynomial running time for the simulator.

6.4 A strict polynomial time simulator

Previously, we presented a solution for a zero-knowledge simulator S that runs in expected polynomial time. We can make a few modifications to the protocol to ensure that the simulator runs in strict polynomial time.

1. Instead of committing to just k pairs of c_i^b, tell V^* to commit to k^2 pairs of c_i^b

2. After receiving the step 1 commitments from V^*, construct $m = k$ slots. Slots should be run sequentially. During the i^{th} slot, the main thread of S will send a k-bit string to V^*, denoted $r^{(i)}$. V^* will then unlock k commitments for share $k(i-1)+1$ through share ki, depending on the value of $r^{(i)}$. If V^* aborts during the main thread in the slot, S should output the aborted transcript.

3. Upon reaching the end of the i^{th} slot, S rewinds exactly $k-1$ times, each time sending a different $r^{(i)'}$ to V^*. If V^* aborts during any rewound thread, ignore the abort and continue. If V^* responds with the appropriate keys in the rewound thread and $r^{(i)} \neq r^{(i)'}$, then by the computational binding property, S will have successfully extracted σ.

Thus, we have the following modified protocol.

1. V^* sends S the following:
   (a) $c = \text{Commit}(\sigma)$, where σ is a random κ-bit string
   (b) $\{c_i^b = \text{Commit}(\sigma_i^b)\}, \forall i \in \{1, \ldots, k^2\}, b \in \{0,1\}$ such that $\sigma_i^0 \oplus \sigma_i^1 = \sigma_i, \forall i \in \{1, \ldots, k^2\}$
2. Run each slot. For slot number $i \in 1, \ldots, m = k$, do
   (a) Main thread: S sends V^* a random bit string $r^{(i)}$
   (b) Main thread: V^* sends the keys for $\{c_{k(i-1)+j}^{r_j^{(i)}}\}, \forall j \in \{1, \ldots, k\}$ to S; if `ABORT` then halt and output aborted transcript
   (c) Rewind. For each rewind $n \in 1, \ldots, k-1$, do
      i. Rewound thread: S sends V^* a random bit string $r^{(i)'}$
      ii. Rewound thread: V^* sends the keys for $\{c_{k(i-1)+j}^{r_j^{(i)'}}\}, \forall j \in \{1, \ldots, k\}$ to S; if `ABORT` then continue; else extract σ
3. Proceed with the parallelized portion of the protocol

Consider a single slot. The probability that V^* does not abort in the main thread, but aborts in each of the $k-1$ rewindings is $< \frac{1}{k}$. This follows from a symmetric swapping argument: suppose all of the random bit strings to be sent in the main and rewound threads $r_j^{(i)}$ and $r_j^{(i)'}$ were determined before the start of the slot. The set of random k-bit strings sent during the main thread could have just as likely been sent in any of the $k-1$ rewound threads. Given that we have a single non-aborting set of random strings and $k-1$ aborting sets of random strings, the likelihood that we select the non-aborting set of random strings to be sent during the main thread is $\frac{1}{k}$.

Thus, the probability that the simulator fails to extract σ in a single slot is $\frac{1}{k}$. Since there are $m = k$ slots, the probability that the simulator fails to extract σ across all slots is $\frac{1}{k^k}$, which quickly grows negligible as we increase k.

Note that to allow for a zero-knowledge simulator S that runs in expected polynomial time, the protocol no longer operates in a constant number of rounds; the number of rounds in this modified depends on the security parameter k, which determines the number of slots.