CS 294-121 – Secure Computation February 11, 2016

Lecture 8: Zero Knowledge – Non-Black-Box

Instructor: Sanjam Garg *Scribe: James Wei*

1 Witness Indistinguishability

We begin by defining *witness indistinguishability*, or WI for short. A witness indistinguishable proof or argument system requires that the verifier does not learn anything about x after seeing a proof of $x \in L$; however, WI is a weaker property than zero knowledge. If the prover has both y and y' as witnesses for $x \in L$, then under a WI system, the verifier cannot know whether y or y' was used to prove $x \in L$. Formally, we have the following definition.

Definition 1 *For some NP-relation R, let L_R the language over R and let (P, V) be an interactive argument for $x \in L_R$. (P, V) is witness indistinguishable if for every polynomial-sized cheating verifier V^*, every input $x \in L$, and every distinct witness w_1, w_2, the following two views are computationally indistinguishable:*

$$\text{view}_{V^*}\langle P(x, w_1) \leftrightarrow V^*(x, z)\rangle \cong \text{view}_{V^*}\langle P(x, w_w) \leftrightarrow V^*(x, z)\rangle$$

In the case of the Hamiltonian cycle example discussed previously, if a graph G contains two Hamiltonian cycles w_1 and w_2, then V^* does not know which of the two cycles is being used by the prover to answer the verifier's challenges, nor is V^* even aware of the fact that there is more than one cycle.

Note that the naive parallelized Hamiltonian cycle protocol (the non-preamble version) is not zero knowledge because we were unable to construct a valid simulator; however, we can show that is is witness indistinguishable.

To prove WI for the parallelized Hamiltonian cycle protocol, we can construct a hybrid argument that transitions from the a proof system using w_1 to a proof system using w_2. Specifically, let H_0 denote the proof system that uses w_1 to answer all κ challenges sent to P from V^* and let H_κ denote the proof system that uses w_2 to answer all κ challenges sent to P from V^*. We can construct H_i, which is the proof system in which w_1 is used to answer the first $\kappa - i$ challenges and w_2 is used to answer the last i challenges. Note that the view of H_j and the view of H_{j+1} are computationally indistinguishable. The computational indistinguishability of the transition can be seen by replacing the use of w_1 with some simulator S. Thus, constructing $H_1, \ldots, H_{\kappa-1}$ gives us the hybrid argument we need to prove WI.

2 Proofs of Knowledge

Previously, we constructed proof and argument systems by which a prover can convince a possibly cheating verifier that $x \in L$. Now, we present a construction under which a prover can also convince the verifier that it knows a witness w to the fact that $x \in L$. With such a construction, it is then possible to efficiently *extract* the witness out of a cheating prover (analogous to how a simulator can be used to transparently emulate the interaction with a cheating verifier). This is a strengthening of the soundness requirement for an interactive proof system.

Definition 2 *For an NP-relation R, let L_R be the language over the relation and let (P, V) be an interactive argument. (P, V) is a proof of knowledge with knowledge error $\kappa \in [0, 1]$ if there exists a PPT oracle machine E (called the knowledge extractor) such that for any $x \in L_R$ and for every (possibly unbounded) cheating prover P^* whereas P^* convinces V with probability $p_x^* = \Pr[\text{out}_V\langle P^*(x) \leftrightarrow V(x)\rangle] > \kappa$, we have*

$$\Pr[E^{P^*(x)}(x) \in R(x)] \geq \text{poly}(p_x^* - \kappa)$$

In other words, the probability that the extractor E retrieves a valid witness from P^* is at least polynomially related to the probability that P^* convinces V that $x \in L_R$, minus some knowledge error.

Returning to the parallelized Hamiltonian cycle protocol, we can argue that the interactive proof system is a proof of knowledge by constructing an extractor E (with negligible knowledge error $\kappa = 2^{-k}$, where k is the number of parallelized instances) as follows:

1. Receive permutation and edge state commitments $\{c_{\phi_i}\}_{\forall i}$ and $\{c_{i,m,n}\}_{\forall i,m,n}$ from P^*
2. Send some random length-k challenge string $b \in \{0, 1\}^k$ where each bit b_i corresponds to the challenge bit for the i^{th} round of the protocol running in parallel
3. Receive the keys corresponding to each challenge bit b_i from P^*.
4. Rewind P^* to its state immediately before step 2. Send some other random length-k challenge string $b' \in \{0, 1\}^k$ where each bit b'_i corresponds to the challenge bit for the i^{th} round of the protocol running in parallel
5. Receive the keys corresponding to each challenge bit b'_i from P^*. If b and b' differ in at least one position, then the extractor can compute P^*'s witness since E now has access to both the permutation ϕ_i and the edges on $\phi_i(G)$ that constitute a Hamiltonian cycle.

3 Non-Black-Box Zero Knowledge

3.1 Motivation

Previously, when constructing simulators, we always viewed the cheating verifier V^* as a black box where the simulator S only has access to V^*'s random tape. Additionally, the primary advantage of S over V^* lay exclusively in S's ability to rewind the interaction. While this rewinding technique has been shown to be useful, it is not without its consequences. Primarily, it is difficult to use this technique to construct constant-round zero-knowledge proof systems under parallelized compositions; it is also difficult to construct strict polynomial time simulators that use the power of rewinding.

Thus, we construct non-black-box simulators that can leverage the code of a possibly cheating verifier V^*.

3.2 An initial attempt

We begin by presenting an initial protocol for non-black-box zero knowledge where the simulator does not rewind. Note that this construction will have several flaws that we will fix down the line.

First, recall the definition of a non-black-box zero-knowledge proof system: an interactive proof system (P, V) for some NP language L is *non-black-box zero knowledge* if for any PPT cheating verifier V^*, there exists a simulator S_{V^*} for V^* such that $\{\text{View}_{V^*}\langle P(x,w) \leftrightarrow V^*(x,z)\rangle\} \cong \{\text{View}_{V^*}\langle S(x,w) \leftrightarrow V^*(x,z)\rangle\}$ for any $x \in L$, any valid witness w, and any $z \in \{0,1\}^*$.

Assume we have some prover P and some verifier V. Consider the following protocol.

1. P sends V^* the commitment $c = \text{Commit}(z = 0^l)$ where $z = 0^l$ for some predetermined length l. Let $n = |c|$.
2. V^* runs some function that takes in c as its input and outputs a length-$3n$ random bit string $r = \{0,1\}^{3n}$; V^* sends r to P
3. P and V^* engage in some WI, universal proof system where P proves either that $x \in L$ or that z is a machine that outputs r on input c (i.e. $z(c) = r$)

Note that a possibly cheating prover P^* shows that z is a machine that outputs r on input c (i.e. z is the code of the verifier) with probability $< 2^{-n}$ since P^* does not have access to the code or the random coins of V^*. Thus, the prover (honest or cheating) must resort to convincing V^* that $x \in L$. The soundness of this protocol then follows from the soundness of the interaction that shows that $x \in L$.

Now consider the simulator for the above protocol. Assume we have some cheating verifier V^* and some non-black-box simulator S_{V^*} that has access to the code of V^* and any hard-wired randomness used by the verifier. Instead of setting $z = 0^l$, the simulator commits to $z = \Pi$, where Π is the code of V^*. Now consider the following protocol.

1. S_{V^*} sends V^* the commitment $c = \text{Commit}(z = \Pi)$. Let $n = |c|$.
2. V^* runs some function that takes in c as its input and outputs a length-$3n$ random bit string $r = \{0,1\}^{3n}$; V^* sends r to S_{V^*}
3. S_{V^*} and V^* engage in some WI, universal proof system where P proves either that $x \in L$ or that z is a machine that outputs r on input c (i.e. $z(c) = r$)

Now, since S_{V^*} has committed to exactly the code of the cheating verifier Π, it is easy to show that $z(c) = r$ since z is exactly the code that the cheating verifier ran to generate r.

There are two problems with the approach of this simulator S_{V^*}. First, the commitment c of Π may not fit inside l bits. If this is the case, then it is impossible for S_{V^*} to commit to Π such that it can still show that $z(c) = r$. Second, if we allow the prover and the simulator to send a commitment of unbounded length in the first step of the protocol, then the communication size of the protocol (specifically, the size of the first message) grows with the size of the verifier.

3.3 An initial constant-size protocol

To make the size of the protocol constant with respect to l and independent of the size of V^*, we make the following modification.

1. V^* sends a hash function $h : \{0,1\}^{\text{poly}(k)} \to \{0,1\}^l$ to P
2. S_{V^*} sends V^* a commitment to the hash of z: $c = \text{Commit}(h(z = \Pi))$. Let $n = |c|$.
3. V^* runs some function that takes in c as its input and outputs a length-$3n$ random bit string $r = \{0,1\}^{3n}$; V^* sends r to S_{V^*}
4. S_{V^*} and V^* engage in some WI, universal proof system where S_{V^*} proves either that $x \in L$ or that the following three conditions hold true:
   (a) $c = \text{Commit}(z)$
   (b) $z = h(M)$, where M is some machine
   (c) $r = M(c)$

By adding the use of a hash to the protocol, we guarantee that the size of the message c does not grow beyond some size n that depends on some predetermined l.

Note that if an honest prover P were to execute the protocol above, instead of hashing $z = \Pi$, P would instead hash $z = 0^{\text{poly}(k)}$. As before, the prover would be forced to show that $x \in L$ in the last step since the prover does not have access to the verifier's code and random coins.

3.4 The Merkle hash

Here we give an example of a hash function that could be used in conjunction with the constant-size protocol presented above. The *Merkle hash* is a tree-structured hashing strategy that allows us to commit to an arbitrarily long string and to later selectively open bits of the string. It gives us the ability to hash a large length-poly(k) data chunk to a smaller length-k hash while maintaining the soundness of a larger hash space.

Denote MH_h as a Merkle hash with key h where h is a collision resistant hash function that maps $h : \{0,1\}^{2k} \to \{0,1\}^k$. The Merkle tree is a binary tree constructed as follows. For input string s of length n:

1. Zero-pad s so that its length is a multiple of k. Split s into k-bit chunks $c_1, c_2, \ldots, c_m$. Let m be the number of these chunks: $m = \lceil \frac{n}{k} \rceil$.
2. For each pair of adjacent chunks (c_i, c_{i+1}) in $\{(c_1, c_2), (c_3, c_4), \ldots, (c_{m-1}, c_m)\}$, do:
   (a) Apply the hash function h to the concatenation of c_i and c_{i+1}. The result is another length-k bit string: $h(c_i || c_{i+1})$
   (b) Set the result $h(c_i || c_{i+1})$ as the parent node of c_i and c_{i+1}
3. Repeat the above process for the newly formed level in the binary tree. Continue to build the binary tree up to the root.

The commitment produced by the Merkle hash tree is the resulting hash value at the root of the constructed binary tree. To show consistency for a particular bit $b_i \in s$, the prover shows sends b_i and opens the nodes in the path from the chunk c_j containing b_i to the root, along with all the siblings of the nodes in the path (i.e. the minimum number of nodes required to reconstruct and verify the resulting hash). The verifier accepts if b_i is equal to the i^{th} bit of s and all of the hashes on the opened nodes are computed correctly.

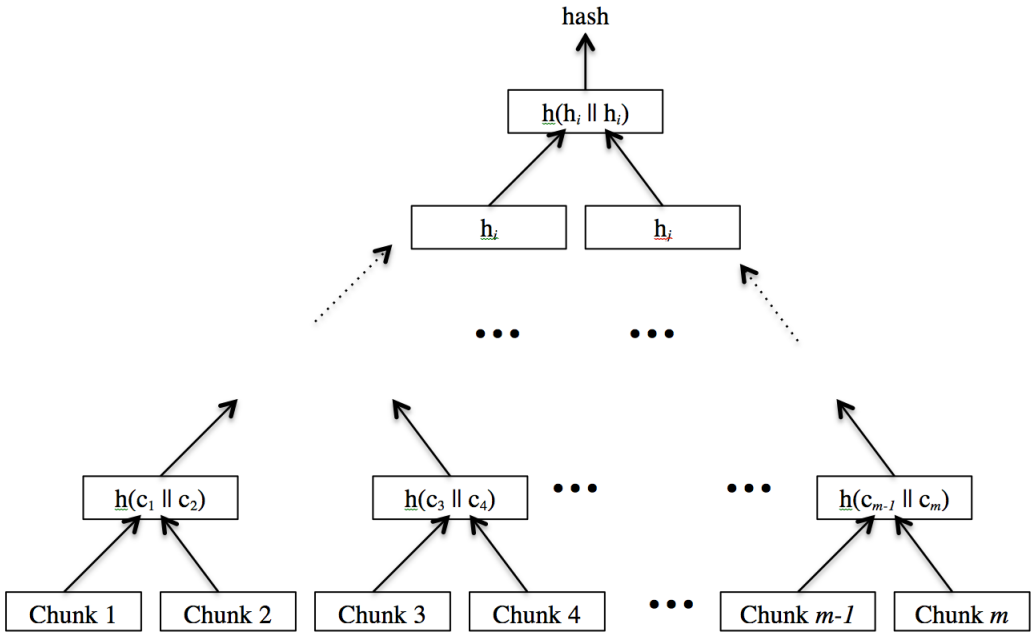

Figure 1: Merkle hash tree construction

3.5 A verifier-independent, constant-size protocol

Using the Merkle hash presented above, we can construct a protocol whose communication cost is independent of the size of the verifier.

1. V^* sends P a Merkle hash key $h : \{0,1\}^{2k} \to \{0,1\}^k$
2. P sends V^* a value $\alpha = MH_h(\Pi_x)$, the result of a Merkle hash with key h on input Π_x. For an honest prover, $\Pi_x = \text{PCP}(x, w)$, a probabilistically checkable proof of the statement $x \in L$ with w as the witness for which the verifier can be convinced by opening some number of bits. For a simulator, $\Pi_x = \text{PCP}(\Pi)$, a probabilistically checkable proof that shows that the simulator has a copy of the cheating verifier's code.
3. V^* sends P a value β, which are the bits of Π_x that the verifier wants to read. The number of bits the verifier asks to open is $k \cdot \text{polylog}(|\Pi_x|) < k^2$.
4. P sends γ to V^*, where γ contains the openings for each of the queried bits in β. Each opening consists of the values of nodes in the Merkle hash tree required to verify the hash α.

3.6 Soundess

The soundness of this protocol can be argued using the collision resistance of the hash function.

Suppose a cheating prover p^* is able to convince a verifier V that $x \in L$ with probability greater than ϵ. We argue that either we break the hash function, or the statement $x \in L$ is indeed true.

Let Q be the set of all possible distinct pairs of first two messages (h_i, α_j). For each pair $q_k \in Q$, let x_k be a random variable denoting the probability that p^* convinces V when q_k is the pair of first two messages. Thus, $\frac{1}{|Q|}\sum_k x_k > \epsilon$.

First, we claim that there exists a large number of first two messages $q_k = (h_i, \alpha_j)$ for which the verifier completes the next two messages (β, γ). In other words, a large number of q_k has corresponding $x_k > 0$.

Additionally, we say that there are at least $|Q| \cdot \frac{\epsilon}{2}$ such q_k for which $x_k \geq \frac{\epsilon}{2}$. This can be seen from the following counting argument. Suppose, for the sake of contradiction, that there are fewer than $|Q| \cdot \frac{\epsilon}{2}$ such q_k for which $x_k \geq \frac{\epsilon}{2}$. This implies that the number of q_k for which $x_k < \frac{\epsilon}{2}$ is at least $|Q| \cdot (1 - \frac{\epsilon}{2})$. The probability that p^* convinces V is the average of all x_k, which must be strictly less than $\frac{1}{|Q|} \cdot ((|Q|)(1 - \frac{\epsilon}{2})(\frac{\epsilon}{2}) + (|Q|)(\frac{\epsilon}{2})(1)) = \epsilon - \frac{\epsilon^2}{4} < \epsilon$, which is a contradiction.

Let some pair $q_k = (h_i, \alpha_j)$ be good if the prover succeeds in convincing the verifier with probability greater than or equal to $\frac{\epsilon}{2}$ (i.e. $x_k \geq \frac{\epsilon}{2}$).

Fix any good pair $q_k = (h_i, \alpha_j)$. For multiple overlapping queries $\beta, \beta', \ldots$, let $p_i(0)$ and $p_i(1)$ be the probability that the i^{th} bit is correctly revealed to be 0 and 1, respectively, given that the first two messages is $q_k = (h_i, \alpha_j)$. Correctly revealing a bit means that the prover can produce valid hash tree entries that demonstrate consistency.

Let pair q_k be problematic if there exists some bit i for which $p_i(0) > \frac{\epsilon}{100T(n)}$ and $p_i(1) > \frac{\epsilon}{100T(n)}$, where $T(n)$ is the running time. If the prover opens a particular bit of the PCP to 0 and 1, both with substantial probability, this suggests that the hash function used is not collision resistant, thus contradicting our original assumption. Otherwise, if each bit is opened to either 0 or 1 (exclusive) with substantial probability, then we would like to say that p^* has a valid witness w for the statement $x \in L$.

Next, we claim that both $p_i(0)$ and $p_i(1)$ cannot be less than $\frac{\epsilon}{100T(n)}$. If both $p_i(0)$ and $p_i(1)$ are less than $\frac{\epsilon}{100T(n)}$, then that means that p^* is not correctly opening the bits of the PCP, which is a contradiction to our assumption that q_k is good (i.e. the corresponding $x_k > \frac{\epsilon}{2}$).

For the good q_k that we picked earlier, we can construct $\hat{\Pi}_x$ whose i^{th} bit $\hat{\Pi}_{x_i}$ is 1 if $p_i(1) \geq \frac{\epsilon}{100T(n)}$, and 0 otherwise. The probability that the PCP Π_x used by p^* during the protocol differs from $\hat{\Pi}_x$ in at least one location can be bounded by the union bound:

$$\Pr[\Pi_x \neq \hat{\Pi}_x] \leq \sum_i \Pr[\Pi_{x_i} \neq \hat{\Pi}_{x_i}] = \sum_i p_i(1 - \hat{\Pi}_{x_i}) \leq \sum_i \frac{\epsilon}{100T(n)} = T(n) \cdot \frac{\epsilon}{100T(n)} = \frac{\epsilon}{100}$$

The last step follows from the fact that the size of the PCP is proportional to the running time $T(n)$.

Therefore, given that the first two messages is a good pair $q_k = (h_i, \alpha_j)$, p^* uses PCP $\hat{\Pi}_x$ with probability at least $\frac{\epsilon}{2} - \frac{\epsilon}{100}$.

This means that we can construct an extractor that can retrieve Π_x, and subsequently the witness w from p^*, with probability polynomially related to the probability that p^* convinces V, making this interaction a proof of knowledge.

3.7 Adding witness indistinguishability

As a final note, to transform the protocol presented above into a WI protocol, we make changes to steps 2 and 4. Instead of sending α and γ in the clear, change the protocol so that Commit(α) and Commit(γ) are sent instead. The prover (or the simulator) can then prove that its commitments of α and γ are correct using a universal-argument system.