CS 600.442 – Modern Cryptography 10/17/2016

Lecture 11: Secure Computation I

Instructor: Abhishek Jain *Scribe: Cheng-Hao Cho*

1 Zero Knowledge

We will first finish off the proof of zero knowledge property of the interactive protocol for graph 3-coloring that we discussed last time.

Intuition. In each iteration of the protocol, V only sees two random colors. The hiding property of Com guarantees that everything else remains hidden from V.

Simulator S(x=G,z):

1. Choose a random edge $(i', j') \xleftarrow{\$} E$ and pick random colors $color'_{i'}, color'_{j'} \xleftarrow{\$} \{1, 2, 3\}$ such that $color'_{i'} \neq color'_{j'}$. For every other $k \in [n] \backslash \{i', j'\}$, set $color'_k = 1$.
2. For every $\ell \in [n]$, computer $C_\ell = Com(color'_\ell)$
3. Emulate execution of $V^*(x, z)$ by feeding it $(C_1, ..., C_n)$. Let (i, j) denote its response.
4. If $(i, j) = (i', j')$, then feed the openings of C_i, C_j to V^* and output its view. Otherwise, restart the above procedure at most $n|E|$ times.
5. If simulation has not succeeded after $n|E|$ attempts, then output fail.

Correctness of Simulation. We will prove this using hybrid experiments.

H_0: Real execution

H_1 : Hybrid simulator S' that acts like the real prover using witness $(color_1, ..., color_n)$, except that it also chooses $(i', j') \xleftarrow{\$} E$ at random and if $(i', j') \neq (i, j)$ in all $n|E|$ trials , then it outputs fail.

H_2 : Simulator S

Now, we want to show that H_0 is indistinguishable from H_2. First, we will show that H_0 and H_1 are indistinguishable. If S' does not output fail, then H_0 and H_1 are identical. Since (i, j) and (i', j') are independently chosen, S' fails with probability at most $(1 - \frac{1}{|E|})^{n|E|} \approx e^{-n}$.

Now, we will show that H_1 and H_2 are indistinguishable. The only difference between H_1 and H_2 is that for all $k \in [n]$ $\{i', j'\}, C_k$ is a commitment to $\pi(color_k)$ in H_1 and a commitment to 1 in H_2. Then, from the multi-value hiding property of Com, it follows that $H_1 \approx H_2$. ■

2 Secure Computation

2.1 Introduction

Consider two billionaires Alice and Bob with net worths x and y, respectively. They want to find out who is richer by computing the following function :

$$f(x,y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{if } Otherwise \end{cases}$$

One potential solution is that Alice sends x to Bob, who sends y to Alice. They each compute f on their own. However, this means that Alice learns Bob's net worth (and vice-versa). There is no privacy here. This presents the question: Can Alice and Bob compute f in a "secure manner" such that they only learn the output of f and nothing more?

To generalize this, consider two parties A and B, with private inputs x and y, respectively. They want to securely compute a function f over their inputs. If both A and B are honest, then they should learn the output $f(x,y)$. Further, even if one party is adversarial, it should not learn anything beyond the output (and its own input). How can we formalize this security requirement? Can we promise that the only thing that the parties learn is the output of the function (and their own respective inputs)?

2.2 Adversary Models

There are two types of adversaries:

Definition 1 (Honest but curious (aka semi-honest)) *Such an adversary follows the instructions of the protocol, but will later analyze the protocol transcript to learn any "extra information" about the input of the other party*

Definition 2 (Malicious) *Such an adversary can deviate from the protocol instructions and follow an arbitrary strategy*

In the context of secure computation, we will only consider semi-honest adversaries. There are generic transformations to amplify security against semi-honest adversaries to security against malicious adversaries.

2.3 Definition

Intuition. We want to formalize the concept that no semi-honest adversary learns anything from the protocol execution beyond its input and the (correct) output. The idea is to use simulation paradigm, as in zero-knowledge proofs. The view of adversary in the protocol execution can be efficiently simulated given only its input and output, and without the input of the honest party.

Definition 3 (Semi-Honest Secure Computation) *A protocol $\pi = (A,B)$ securely computes a function f in the semi-honest model if there exists a pair of non-uniform PPT simulator algorithms $\mathbb{S}_A, \mathbb{S}_B$ such that for every security parameter n, and all inputs $x, y \in \{0,1\}^n$, it holds that:*

$$\{\mathbb{S}_A(x, f(x,y)), f(x,y)\} \approx \{e \leftarrow [A(x) \leftrightarrow B(y)] : View_A(e), OUT_B(e)\}$$

$$\{\mathbb{S}_B(y, f(x,y)), f(x,y)\} \approx \{e \leftarrow [A(x) \leftrightarrow B(y)] : View_B(e), OUT_A(e)\}$$

We talk about two simulators because we want privacy for both sides. In the case of zero knowledge, we didn't need any privacy against a cheating prover since the verifier has no private input.

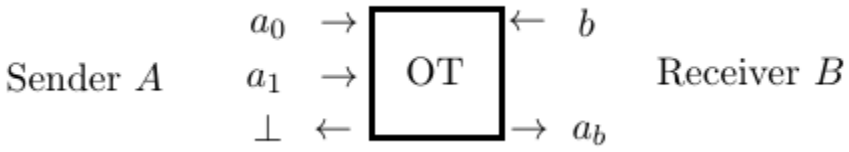
Remarks on Definition. Recall that in zero-knowledge, we only require indistinguishability of simulated view and adversary's view in the real execution. Here, indistinguishability is with respect to the joint distribution over the adversary's view and the honest party's output.

This is necessary for correctness. It implies that the output of the honest party in the protocol execution must be indistinguishable from the correct output $f(x, y)$. This guarantees that when the honest party talks to a corrupted party, the output is still always correct. If we remove this requirement, then a clearly wrong protocol where parties are instructed to output y would be trivially secure! We not only want security, we also want correctness of the output of the honest party.

3 Oblivious Transfer

Consider the following functionality, called, 1-out-of-two oblivious transfer (OT). There are two parties, a sender A and receiver B. A's input is a pair of bits (a_0, a_1), and B's input is a bit b. B's output is a_b and A receives no output.

This is bit OT because the inputs of sender are bits, but it can be easily generalized to string OT.



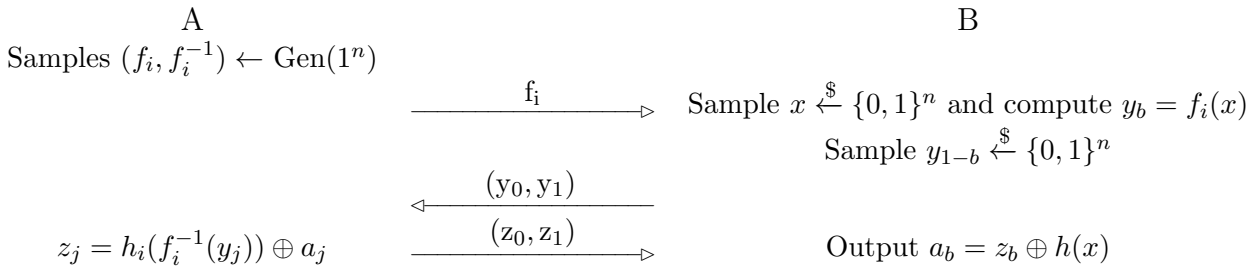
We define security for OT using Definition 3. Note that the definition promises that in a secure OT protocol, A does not learn b and B does not learn a_{1-b}.

3.1 Importance of Oblivious Transfer

OT can be realized from physical channels, as shown by Wiener and Rabin. In particular, a noisy channel can be used to implement OT. Furthermore, OT is *complete* for secure computation. This means that given a secure protocol for OT, any function can be securely computed. Finally, OT is also *necessary* for secure computation.

3.2 Construction

Let $\{f_i\}_{i \in I}$ be a family of trapdoor permutations with sampling algorithm Gen. Let h be a hardcore predicate for f_i.



3.3 Proof of Security

Intuition. Let's first consider security against adversarial A. Note that both y_0 and y_1 are uniformly distributed and therefore independent of b. Thus, b is hidden from A.

Next, let's consider security against adversarial B. We want to argue that a_{1-b} is hidden. We will use the fact that B does not know the pre-image of y_{1-b}. By the security of hardcore predicates, we know that $h_i(f_i^{-1}(y_{1-b})$ will look random to B. If B could learn a_{1-b}, then it would be able to predict the hardcore predicate. In particular, $h_i(f_i^{-1}(y_{1-b}))$ appears to be random for B.

Remark 1 (Malicious Receiver) *This protocol is not secure against a malicious receiver. Indeed, a malicious B can easily learn a_{1-b} by deviating from the protocol strategy. B can choose y_{1-b} by first choosing a pre-image x' and then computing $h(x')$.*

To formally prove semi-honest security, we will construct simulators for both cases.

Simulator $\mathbb{S}_A((a_0, a_1), \perp)$:

1. Fix a random tape r_A for A. Run honest emulation of A using (a_0, a_1) and r_A to obtain the first message f_i.
2. Choose two random strings $y_0, y_1 \in \{0,1\}^n$ as B's message
3. Run honest emulation of A using (y_0, y_1) to obtain the third message (z_0, z_1).
4. Stop and output $\perp$

Lemma 1 *The following two distributions are identical:*

$$\{\mathbb{S}_A((a_0, a_1), \perp), a_b\}, \text{ and}$$

$$\{e \leftarrow [A(a_0, a_1) \leftrightarrow B(b)] : View_A(e), Out_B(e)\}.$$

Proof. The only difference between $\mathbb{S}_A$ and real execution is in step 2. However, since f is a permutation, y_0, y_1 are identically distributed in both cases. In the real case, y_b is computed by first sampling a random preimage and then computing f_i over it whereas in the simulation, it is sampled at random. Both of these have the same distribution. ■

Simulator $\mathbb{S}_B(b, a_b)$:

1. Sample f_i, f_i^{-1}.
2. Choose random tape r_B for B. Run honest emulation of B using b, r_B, f_i) to produce (x, y_0, y_1) s.t. $y_b = f_i(x)$ and $y_{1-b} \leftarrow \{0,1\}^n$.
3. Compute $z_b = h(x) \oplus a_b$ and $z_{1-b} \leftarrow \{0,1\}$
4. Output (z_0, z_1) as third message and stop

Lemma 2 *The following two distributions are indistinguishable:*

$$\{\mathbb{S}_B(b, a_b), \perp\}, \text{ and}$$
$$\{e \leftarrow [A(a_0, a_1) \longleftrightarrow B(b)] : View_B(e), OUT_A(e)\}.$$

Proof. The only difference is in step 3, where $\mathbb{S}_B$ computes z_{1-b} as a random bit. However, since $h(f_i^{-1}(y_{1-b}))$ is indistinguishable from random (even given y_{1-b}), this change is indistinguishable ■

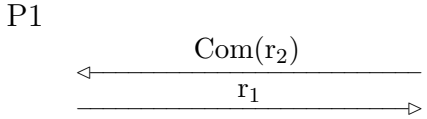
3.4 Remarks

Extension to 1-out-of-k OT. The previous protocol can be easily generalized to construct 1-out-of-k OT for $k > 2$.

Security against Malicious Adversaries. In reality, adversary may be malicious and not semi-honest. Goldreich-Micali-Widgerson [GMW] gave a compiler to transform any protocol that is secure against semi-honest adversaries into one secure against malicious adversaries. This is a general compiler and can be applied to any protocol. Below, we briefly discuss the main ideas in their compiler.

Once we have security against semi-honest adversaries, there are two main additional things that we have to worry about against a malicious adversary.

- First, a malicious adversary may choose a random tape so that the tape is biased. In this case, the protocol may not remain secure. The transformation of GMW uses coin-flipping to make sure that adversary's random tape is truly random. A simple coin-flipping protocol is described below:



  The random tape of P_2 is $r_1 \oplus r_2$, which is indistinguishable from random, and also hidden from P_1. Similarly, we can do coin-tossing to fix the random tape of P_1.

- A malicious adversary may also deviate from the protocol. In this case, all bets are off. For example, in the OT protocol, if B computed y_{1-b} from a pre-image, then it could learn a_{1-b}.

  One way to ensure that the parties do not deviate from the protocol instructions is to require them to attach a proof with every message to establish that it is following the protocol instructions. This way, each party can verify that the other party was acting semi-honestly. Furthermore, this proof must be zero knowledge or else it might reveal the inputs.