COM S 6830 – Cryptography November 22, 2011

Lecture 22: Secure Computation

Instructor: Rafael Pass *Scribe: Sujay Jayakar (dsj36)*

1 Secure Computation

1.1 Introduction

Secure computation, described as "the crown jewel of modern cryptography," is, in some senses, a generalization of a zero knowledge proof. In a zero knowledge proof, a prover, equipped with some statement x and a witness w, proves to a verifier that x is indeed in a language $\mathcal{L}$.

$$P_{x,w} \longleftrightarrow V_x$$

Secure computation has two parties, A and B, who each own a secret input x and y, respectively. Together, they would like to compute some function $f(x, y)$ without revealing their inputs.

$$A_x \longrightarrow f(x, y) \longleftarrow B_y$$

Secure computation is indeed a generalization of zero knowledge proofs, as we may simply set f to check the witness, outputting the result without revealing the witness itself. We may generalize this notion even further to n-party secure computation. Here, n players have their own secret input, and they would like to compute $f(x_1, \ldots, x_n)$ without revealing their input.

A natural extension of this sequence of generalizations would be to consider the secure computation of interactive functions. It turns out that regular secure computation is sufficient to simulate interactive computation. The idea is that we step through the execution of the interactive function one step at a time, each time distributing the state among the parties. Since none of the parties should know the state, we give the i-th player some s_i along σ_i such that $s_1 \oplus \cdots \oplus s_n = \mathsf{state}$ and $\sigma_1 \oplus \cdots \oplus \sigma_n = \sigma$, a signature for the state.

1.2 Definition

Intuitively, the best we can do with secure computation is to have a trusted middleman T_f who takes all of our inputs, returns an output, and then burns all of the evidence. Thus, if the function f we are computing simply reveals our input or if the inputs themselves are suspect, there is not much we can do from a cryptographic standpoint, as the ideal T_f protocol would yield poor results as well. Therefore, a reasonable definition of security is that a protocol is secure if any attack on the protocol can be reduced to an attack on the ideal protocol T_f.

Definition 1 *A protocol Π securely implements f if for all PPT attackers A, there exists a PPT machine $\tilde{A}$ and a negligible function $\varepsilon(\cdot)$ such that for all nuPPT distinguishers $\mathcal{D}$, $n \in \mathbb{N}$, $I \subseteq [n]$, $x_1, \ldots, x_n \in \{0,1\}^n$, and $z \in \{0,1\}^*$, $\mathcal{D}$ distinguishes the following distributions with probability less than or equal to $\varepsilon(n)$.*

$$\{\mathsf{REAL}(\Pi, A, I, 1^n, \vec{x}, z)\}$$
$$\{\mathsf{IDEAL}(F, \tilde{A}, I, 1^n, \vec{x}, z)\}$$

Here, I is a set of players that A controls, REAL *is the output of the malicious and honest players after running the protocol Π, and* IDEAL *is the output of the players after the ideal experiment. There is one caveat: In the real world, if A controls sufficiently many players, he may decide to not send the final result to some of the honest players. Therefore, in the ideal protocol, we allow A to decide who gets the final result. However, the honest parties either get the correct result or no result at all.*

Note that indistinguishability for the result where everyone is honest gives us correctness, and indistinguishability for entirely malicious players gives us security. We will also be interested in a relaxation of this notion where we only consider "honest but curious" attackers.

1.3 Honest but Curious

Definition 2 *An honest but secure (HBC) protocol Π is a secure protocol where attackers A are restricted to following the protocol, but after the experiment is over, they may analyze the data they have received to try to recover other players' inputs. Once they have done so, they may return a result deviating from the regular computation.*

Given an HBC protocol, we may actually construct a secure protocol using commitments. We begin by committing to our inputs and randomness (coin tosses) and then, after each step, prove in zero knowledge that we performed our step correctly. Next, we will discuss a particular instance of secure computation, oblivious transfer.

In an oblivious transfer protocol F_{OT}, party A has two inputs (a_0, a_1), and party B would like to receive one of them: This preference is designated with a bit b. However, B would not like A to know b.

Theorem 1 (Rabin) *The existence of trapdoor permutations on $\{0,1\}^n$ implies the existence of an honest but curious secure implementation of F_{OT}.*

Proof. By construction. Let party A have inputs (a_0, a_1), each one bit, and party B have an input bit b. Party A initializes by sampling $(i, t) \leftarrow \mathsf{Gen}(1^n)$ and sends i to B. Upon receiving i, B samples $x \leftarrow \{0,1\}^n$ and sets $y_b = f_i(x)$ and $y_{1-b} \leftarrow \{0,1\}^n$, sending both y_0 and y_1 to A. Given y_0 and y_1, A computes $z_j = h(f_i^{-1}(y_j)) \oplus a_i$, where h is a hardcore bit for f, and gives both z_0 and z_1 to B. Then B can output $h(x) \oplus z_b = a_i$. Since h is a hardcore bit, B cannot learn anything about the other choice a_{1-b}, as he does not have the inverse $f^{-1}(y_{1-b})$. Also, A cannot learn anything about B's choice as y_b and y_{1-b} are both uniformly random strings. Note that we are crucially using the honest but curious property: If B did not follow the protocol, he could simply set y_{1-b} to be $f(x_{1-b})$ rather than a random string and learn both a_0 and a_1. ■