CMSC 858T—Secure Distributed Computation January 26, 2021

Lecture 2

Lecturer: Jonathan Katz *Scribe(s): Hunter Kippen, Wenqing (William) Xu*

1 Composition

A natural question to ask is what happens when two or more secure protocols are *composed* with each other in various ways.

1.1 Sequential Composition

Assume there exist protocols π_1 and π_2 that securely compute functionalities f_1 and f_2, respectively. Then running π_1 followed by π_2 securely computes f_1 followed by f_2; i.e., it emulates an ideal world in which the parties first use a trusted party to computes f_1 followed by using the trusted party to compute f_2. Thus, we say that security is preserved under *sequential composition.* While clearly desirable, this is non-trivial to prove, and in fact the presence of auxiliary input in the definition of secure computation is necessary in order to prove security of sequential composition.

1.2 Modular Composition

Modular composition refers to invoking one protocol invoking other protocols as sub-routines. Modular composition helps ease the job of a protocol designer, since it is often easier to design a protocol for some complicated functionality F by breaking down the protocol into smaller sub-components that can be handled by their own protocols. It also can help simplify the overall proof of security for a protocol: first prove security of the sub-protocols, and then rely on security of the sub-protocols to prove security of the entire protocol.

Our overall strategy will be the following. We design a protocol Π computing F in a *hybrid world* in which the parties have access to ideal functionalities $f_1, \ldots, f_n$. That is, the parties running Π will exchange messages as usual, but now Π may also instruct the parties to send input to one of the ideal functionalities f_i and then do something with the result they get back. (See Figure 1.) Execution of Π in the presence of an adversary $\mathcal{A}$, where the parties begin holding security parameter κ and inputs $\vec{x} = (x_1, \ldots, x_n)$, and $\mathcal{A}$ also holds auxiliary input z, defines a random variable $\textsc{hybrid}^{f_1,\ldots,f_m}_{\Pi,\mathcal{A}}(\kappa, \vec{x}, z)$ consisting of the outputs of the honest parties and the joint view of the corrupted parties.

Definition 1 A hybrid-world protocol Π computing F is t-secure if for all PPT adversaries $\mathcal{A}$ corrupting at most t parties, there is a PPT simulator $\mathcal{S}$ corrupting the same parties such that: $\{\textsc{hybrid}^{f_1,\ldots,f_m}_{\Pi,\mathcal{A}}(\kappa, \vec{x}, z)\}_{\kappa,\vec{x},z}$ and $\{\textsc{ideal}_{F,\mathcal{S}}(\kappa, \vec{x}, z)\}_{\kappa,\vec{x},z}$ are computationally indistinguishable. $\diamondsuit$

Theorem 1 *If $\pi_1, \ldots, \pi_m$ are secure (real-world) protocols for computing functionalities $f_1, \ldots, f_m$, and if Π is a secure protocol for computing F in the $(f_1, \ldots, f_m)$-hybrid world, then the composed (fully real-world) protocol $\Pi^{\pi_1,\ldots,\pi_m}$ is a secure protocol for computing F.*

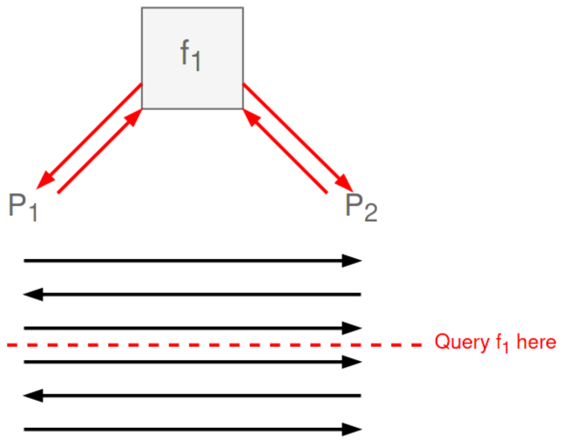

Figure 1: Hybrid-world protocol execution.

1.3 Parallel and Concurrent Composition?

Say π_1, π_2 are secure protocols computing f_1, f_2 respectively. What can we say if we have parties executing π_1 and π_2 in parallel (i.e., where the rounds of the protocols are run in lockstep) or concurrently (where the protocol messages may be arbitrarily interleaved)? In the semi-honest setting, one can show that security is preserved by using the fact that a semi-honest adversary executes the two protocols independently. In the malicious setting, however, we cannot in general claim anything about security under parallel or concurrent composition. Later in the course we will see a stronger definition of security that is preserved under concurrent composition.

2 Oblivious Transfer (OT)

A particularly important functionality is oblivious transfer (OT). While several variants can be considered, the most widely used is 1-out-of-N OT. Here there one party (the *sender*) has a vector of strings $x_0, \ldots, x_{N-1}$ as input and the other party (the *receiver*) has an index $i \in \{0, \ldots, N-1\}$; the receiver should learn x_i and nothing else (and the sender should learn nothing). See Figure 2.

2.1 Semi-Honest OT

We show a construction of an OT protocol with semi-honest security. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure public-key encryption scheme that allows for *oblivious key sampling*. Informally, this last property means that it is possible to sample a public key of the scheme without sampling the corresponding secret key. Formally, it means that there are efficient algorithms $\mathsf{SampKey}, \mathsf{SampRand}$ such that the distributions $\{r \leftarrow \{0,1\}^*; pk \leftarrow \mathsf{SampKey}(1^\kappa; r) : (r, pk)\}_\kappa$ and $\{(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa); r \leftarrow \mathsf{SampRand}(pk) : (r, pk)\}_\kappa$ are computationally indistinguishable. Not all encryption schemes have this property, but some (e.g., El Gamal encryption in certain groups) do.

We describe the OT protocol for the case of $N = 2$ for simplicity. Here, we represent the index of the receiver by a bit b. The protocol proceeds as follows:

1. The receiver chooses randomness r_0, r_1 and generates $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa)$ followed by

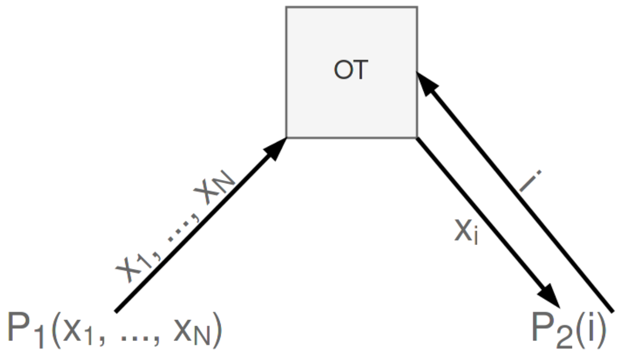

Figure 2: 1-out-of-N OT functionality.

$pk_{1-b} \leftarrow \mathsf{SampKey}(1^\kappa; r_{1-b})$. That is, they generate one public key in the regular way (along with an associated secret key) and another public key obliviously. They send (pk_0, pk_1) to the sender.

2. The sender computes $c_0 \leftarrow \mathsf{Enc}_{pk_0}(x_0)$ and $c_1 \leftarrow \mathsf{Enc}_{pk_1}(x_1)$, and sends c_0, c_1 to the receiver.

3. The receiver computes $x_b := \mathsf{Dec}_{sk_b}(c_b)$.

Theorem 2 *If the encryption scheme used is CPA-secure and has oblivious key sampling, then the above protocol securely computes the OT functionality for semi-honest adversaries.*

Proof First consider the case where the sender is corrupted. Note that the sender's view contains only its own randomness and the two public keys pk_0, pk_1 the receiver sends. The oblivious key sampling property implies that public keys generated using Gen and public keys generated using $\mathsf{SampKey}$ are indistinguishable. We leave a description of a simulator and a full proof of security in this case as an exercise.

We deal with the case where the receiver is corrupted in the next lecture. ■