0368.4162 Foundations of Cryptography December 21, 2017

Lecture 9: Multi-Party Computation

Lecturer: Nir Bitansky *Scribes: Noam Nissan and Omri Shmueli*

1 Previously on Foundations of Crypto

Having established the most basic crypto objective — secure communication — we now move on to explore more general cryptographic goals. As a preliminary step, we've learned about zero-knowledge (ZK) proofs, and saw that (under minimal computational assumptions) we can prove any NP statement in ZK. While the concept of ZK proofs is well motivated on its own, we will see how ZK proofs can be applied more generally for enforcing (semi) honest behavior.

2 Multi-Party Computation

Today, we will discuss the problem of *multi-party computation* (MPC). Here we would like to design a protocol that will allow a set of parties $1, \ldots, m$ to compute a joint function

$$(y_1, \ldots, y_m) = f(x_1, \ldots, x_m)$$

of their private inputs $x_i \in \{0,1\}^n$, so that each party i will obtain her respective output y_i. However, we'd like the protocol to be secure even if some of the parties may be malicious.

But what does it mean for the protocol to be "secure"?

Naturally, security can reflect various requirements on what the protocol should guarantee in the presence of malicious parties. To get some sense of the requirements we might want to capture, let's consider a specific example of *auctions*. Here there is an auctioneer party, say 1, and each of the parties $2, \ldots, m$ has a bid $x_i \geq 0$. We'd like that at the end of the auction protocol, party 1 will learn the identity of the highest bidder, as well as her bid. The rest of the parties should learn whether they have won or not.

With this example in mind, we can think of several natural requirements:

- **Correctness:** We'd like to ensure that the received outputs indeed correspond to computing the prescribed auction function f, on some set of inputs, and not a different function, say f^* that always announces 2 to be the winner with a bid of 100 Shekel.
- **Input Independence:** Computing the function f on *some* inputs isn't enough, we would also like to ensure that parties cannot choose their inputs based on those of other parties. In particular, 2 shouldn't be able to set his input x_2 to $\max\{x_i \mid i \geq 3\}$.
- **Privacy:** Parties may not want other parties to learn what was their bid.
- **Fairness:** Malicious parties should not be able to prevent the auction from terminating if they don't like the result.

One could think of other security requirements for such a protocol. We will want one definition that can capture all such requirements. We've already faced a similar (but more specific) challenge in the context of ZK, where we tried to capture *all the things that a verifier can learn from a proof.* Our definitional approach will indeed follows a similar approach.

Defining MPC via the Real-Ideal principle. Specifically, we would like our definition to follow the real-ideal principle:

The adversary's affect on the real protocol should not be worse than what it could be in an ideal protocol where the function is computed by a trusted party.

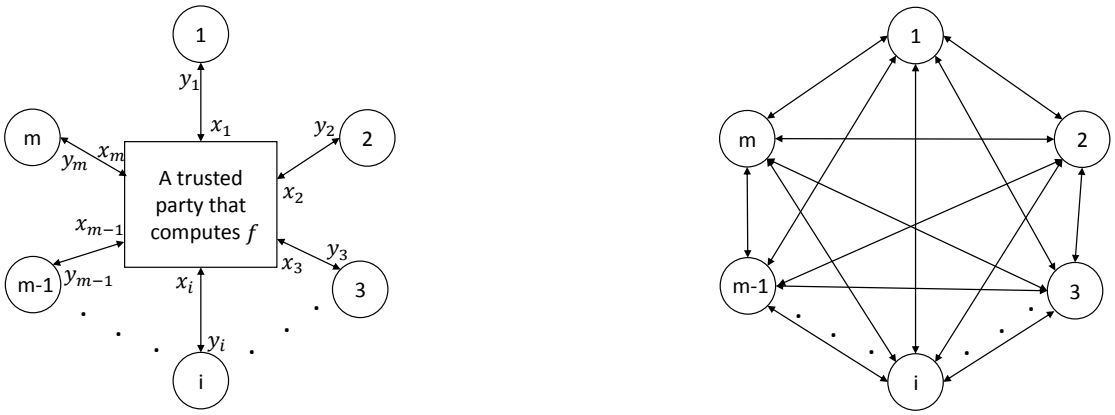

Figure 1: On the left is the ideal world, where a trusted party gets all inputs x_j, computes f, and returns corresponding outputs y_j to each party. On the right we have the real world, where we do not assume the existence of a trusted party, and f is computed through interaction.

As in the case of ZK, we will formalize this principle using the simulation paradigm. There are actually various ways to do this, with different nuances. We will focus on a rather simple definition that already captures the essence of MPC. In particular, we shall assume that the honest parties communicate through ideally secure channels; that is, the adversary only controls malicious parties, and does not control the communication between honest parties. This simplifying assumption can in principle be removed using tools like encryption and digital signatures (albeit there are some subtleties to be dealt with, such as synchrony).

In what follows, $m = m(n)$ is a polynomially bounded function, and $f : (\{0,1\}^n)^m \to (\{0,1\}^*)^m$ is a poly-time computable, possibly randomized function. An m-party interactive protocol for f is given by a PPT algorithm Π, which at each round, can be used by each party i, to compute its next message in the protocol (given the previous messages it received, and the randomness it had used).

Definition 2.1 (MPC (A Slightly Too Strong Definition))**.** *An m-party protocol Π for f is secure if for any n.u. PPT $A = \{A_n\}$, there exists a n.u. PPT $S = \{S_n\}$ such that*

$$\{Real_A(n, C, \vec{x})\}_{n,C,\vec{x}} \approx_c \{Ideal_S(n, C, \vec{x})\}_{n,C,\vec{x}} \ ,$$

where $n \in \mathbb{N}$, $C \subseteq [m]$ (is the subset of corrupted parties), $\vec{x} = x_1, \ldots, x_m \in \{0,1\}^n$ (are the inputs), and $Real, Ideal$ are distributions on outputs $\vec{y} = y_1, \ldots, y_m$ defined as follows:

- *$Real_A(n, C, \vec{x})$: Each honest party $i \in [m] \setminus C$, follows the protocol Π with input x_i, to obtain an output y_i. For each corrupted $j \in C$, the adversary A_n, obtains x_j, and throughout the protocol chooses the messages for the party as well as their output y_j.*
- *$Ideal_S(n, C, \vec{x})$: For each corrupted $j \in C$, the simulator S_n obtains x_j, chooses a possibly new input x'_j, and submits it to the trusted party. Each honest party $i \in [m] \setminus C$, submits their input $x'_i = x_i$ to the trusted party, which computes $(y'_1, \ldots, y'_m) \leftarrow f(x'_1, \ldots, x'_m)$. The parties each obtain y'_j in their turn. For $j \in C$, S_n may arbitrarily choose the final output y_j. For the honest parties $y_i = y'_i$.*

Remarks:

- **Corrupted Parties' Outputs.** The corrupted parties can produce an output that arbitrarily (but efficiently) depends on the adversary's view in the protocol's execution. An equivalent definition that is perhaps more intuitive (but involves more notation) is the real and ideal views include the outputs of the honest parties, as well as the adversary's view in the protocol, including all the messages that she sees during the protocol and her randomness.
- **Universal Simulation.** As was the case for ZK, a common stronger definition requires that there is one *universal PPT simulator* S that can simulate any adversary A, given the code of A as auxiliary input. This definition is very useful when composing different protocols together. For now, we will keep ignoring this difference.

It is instructive to try and convince yourself that the above definition indeed captures the specific properties mentioned above in the context of auction protocols. For instance, convince yourself that for any bid x_2 for party 2 and any two bids $x_3 < x_3' < x_2$, the auctioneer party 1, cannot distinguish an execution where party 3 uses x_3 from one where she uses x_3'. See some more specific examples that could help gain intuition about the definition in Barak's notes.

Why is the Definition Too Strong? One attack that we cannot prevent is *a denial of service attack*; namely, a malicious party can choose to abort the protocol at any point. To understand intuitively why this is a problem, think of the simple two party case. Then in an interactive protocol, the parties send a message, each in their turn, and thus one party learns the output first and may choose to abort if they don't like it. (This is a bit oversimplified since the identity of the party that learns the output first, or rather the round where the protocol terminates, can be randomized. However, there is a formal statement in this spirit that is true [Cle86].)

There are two common ways to remedy this problem. The first is to require an honest majority of parties, i.e. $|C| < m/2$. This, however, doesn't make sense in some situations, for instance in a two-party protocol. A different approach is to allow aborts also in the ideal world. That is, at any point after $j < m$ parties have received the output, the simulator can choose to abort (in which case the output of the rest of the parties is set to $\perp$). The augmented definition is below; most parts do not change and appear below in gray.

Definition 2.2 (MPC with Aborts)**.** *An m-party protocol Π for f is secure if for any n.u. PPT $A = \{A_n\}$, there exists a n.u. PPT $S = \{S_n\}$ such that*

$$\{Real_A(n, C, \vec{x})\}_{n,C,\vec{x}} \approx_c \{Ideal_S(n, C, \vec{x})\}_{n,C,\vec{x}} \ ,$$

where $n \in \mathbb{N}$, $C \subseteq [m]$ (is the subset of corrupted parties), $\vec{x} = x_1, \ldots, x_m \in \{0,1\}^n$ (are the inputs), and $Real, Ideal$ are distributions on outputs $\vec{y} = y_1, \ldots, y_m$ defined as follows:

- *$Real_A(n, C, \vec{x})$: Each honest party $i \in [m] \setminus C$, follows the protocol Π with input x_i, to obtain an output y_i. For each corrupted $j \in C$, the adversary A_n, obtains x_j, and throughout the protocol chooses the messages for the party as well as their output y_j.*
- *$Ideal_S(n, C, \vec{x})$: For each corrupted $j \in C$, the simulator S_n obtains x_j, chooses a possibly new input an x_j', and submits it to the trusted party. Each honest party $i \in [m] \setminus C$, submits their input $x_i' = x_i$ to the trusted party, which computes $(y_1', \ldots, y_m') \leftarrow f(x_1', \ldots, x_m')$. The parties each obtain y_j' in their turn. For any $j \in [m]$, S_n can instruct the function to abort, in which case $y_j', \ldots, y_m'$ are set to $\perp$. For $j \in C$, S_n may arbitrarily choose the final output y_j. For the honest parties $y_i = y_i'$.*

Quite remarkably, it turns out that any function can be securely computed under reasonable assumptions, reasonable being roughly assumptions that suffice for public-key encryption (and not one-way functions). More formally, such protocol can be reduced to a primitive known as *oblivious transfer*, which we will define later on.

Theorem 2.3 ([GMW87])**.** *Assuming the existence of oblivious transfer, any m-party function has a secure protocol.*[1]

MPC is Expressive but Costly. MPC is an expressive object that can, in fact, capture all the objectives we discussed so far in the course (and more to come). So what's the catch behind the GMW theorem? why are people still doing research in crypto? The main (but not the only) answer to this question is that general MPC protocols, like the GMW protocol, are simply to costly. They require that all parties communicate with each other in several rounds of computation and possibly perform heavy computation. In some (perhaps in most) cases, heavy communication and computation are unrealistic.

3 The GMW Approach

We will not fully prove the GMW theorem, but we will give an overview of the proof, and elaborate on certain parts.

At high level, the GMW approach consists of two main steps:

1. Construct protocols against a weak type of adversaries that are *semi honest*.
2. Compile any protocol against semi-honest adversaries to a protocol against general adversaries.

In this lecture, we will define what are semi-honest adversaries and give an overview of Step 2. In the next lecture, we will go into more details on constructing protocols with semi-honest security.

Semi-Honest Adversaries. Semi-honest adversaries are basically ones who follow the prescribed protocol Π, but are nosy and aim to learn information from the transcript on the private inputs of honest parties.

Definition 3.1 (Semi-Honest MPC))**.** *An m-party protocol Π for f is semi-honestly secure if for any n.u. PPT $A = \{A_n\}$, there exists a n.u. PPT $S = \{S_n\}$ such that*

$$\left\{Real_A^{sh}(n, C, \vec{x})\right\}_{n,C,\vec{x}} \approx_c \left\{Ideal_S^{sh}(n, C, \vec{x})\right\}_{n,C,\vec{x}} \ ,$$

where $n \in \mathbb{N}$, $C \subseteq [m]$ (is the subset of corrupted parties), $\vec{x} = x_1, \ldots, x_m \in \{0,1\}^n$ (are the inputs), and $Real, Ideal$ are distributions on outputs $\vec{y} = y_1, \ldots, y_m$ defined as follows:

- *$Real_A^{sh}(n, C, \vec{x})$: Each honest party $i \in [m] \setminus C$, follows the protocol Π with input x_i, to obtain an output y_i. For each corrupted $j \in C$, the adversary A_n, obtains the view of j in the protocol, including its input x_j, randomness r_j, and received messages $\vec{m}_j$, and can change the output y_j to any y'_j.*
- *$Ideal_S^{sh}(n, C, \vec{x})$: Every party $i \in [m]$, submits their input x_i to the trusted party, which computes $(y_1, \ldots, y_m) \leftarrow f(x_1, \ldots, x_m)$. For $j \in C$, S_n obtains y_j and may then change the final output from y_j to any y'_j. (For the honest parties the output y_i remains the same).*

Compiling Semi-Honestly-Secure Protocols to Maliciously-Secure Ones. We now outline the transformation behind the second step that takes a semi-honestly-secure protocol Π and turns it into a maliciously-secure protocol Π'. We will focus on the simple case of two-party protocols. The general m-party case relies on similar ideas, with some adjustments. *Furthermore, we will describe a slightly defective transformation and then explain how to fix it.*

The basic idea is to force the parties $\{1, 2\}$ running Π' to behave according to the protocol Π. To do this, whenever a party, say 1, in Π' sends a message it will prove to the other party, say 2, that this message was computed by applying Π to the view of 1 so far in protocol, including her input, randomness, and

[1] More formally, to satisfy Definition 3.1, the corrupted set of parties must be a minority. For the case of a corrupted majority, a relaxed definition is achieved with simulator aborts.

received messages. To ensure privacy, the proofs will be zero-knowledge. To ensure consistency with an actual execution of Π, we will use commitments to the output and randomness. We proceed to describe this in further detail.

1. **Commitmnet Phase:** Each party $i \in \{1,2\}$ sends a commitments $Com(x_i), Com(r_i)$ to their input and randomness.
2. **Emulation Phase:** The parties proceed according to the protocol Π. After party i sends the t-th message $m_i^{(t)}$, it provides a zero-knowledge proof that there exists a consistent opening of its commitments to x_i and r_i, such that $m_i^{(t)}$ is obtained by applying Π with the transcript given by x_i, r_i and the messages $m_j^{(<t)}$ previously sent by $j = \{1,2\} \setminus \{i\}$.

The are two problems with the above transformation:

1. The randomness r_i chosen by a malicious party may be chosen maliciously and not uniformly as in the semi-honest case. We shall see (next week) that this can be devastating for some protocols.
2. The above protocol may not necessarily guarantee input-independence. The fact that there exists a commitment to an input x_2 such that 2's messages are consistent with x_2, does not mean that 2 "knows" x_2. Instead, it could have somehow *mauled* the commitment of party 1, so that x_2 will now depend on x_1 (without necessarily learning x_1). Indeed, to successfully simulate we need to ensure that the simulator can somehow extract from 2 the value behind the commitment and submit it to the trusted party.

We informally explain the solution to both of these problems.

Coin-Tossing into the Well. Our goal is to design a sub-protocol that will ensure that on one hand 2 uses in the semi-honest emulation a truly random string r_2, but on the other hand will not reveal r_2 to 1. The solution is that after party 1 receives the commitment, it will send 2 a random share r_2', Then throughout the protocol, 2 will have to prove consistency with $r_2 \oplus r_2'$. The same is done symmetrically for the other party. This is a specific instance of the coin-tossing problem where parties would like to jointly toss a fair coin on the telephone. In this variant only one party explicitly learns the randomness, and for the other party it is completely hidden due to the hiding of the commitments. This is known as coin-tossing into the well.

Proof-of-Knowledge. To deal with the input independence problem and more generally allow extraction of corrupted parties' inputs, each party will give a ZK proof-of-knowledge (ZKPOK) that it knows the value underlying the commitment. POK is a strengthening of the usual soundness requirement of ZK protocols. It says that given any sufficiently convincing prover, we can extract from its code a witness for the corresponding NP statement.[2]

[2]For instance, in the 3COL protocol we've seen last week we can extract a coloring as follows. Imagine for simplicity that the prover is *very* convincing, in the sense that it answers the verifier with valid decommitments with very high probability $1 - 1/|E|$, then by rewinding this prover we can reveal an entire coloring. Obtaining a prover that answers with such high probability is achieved using parallel repetition.

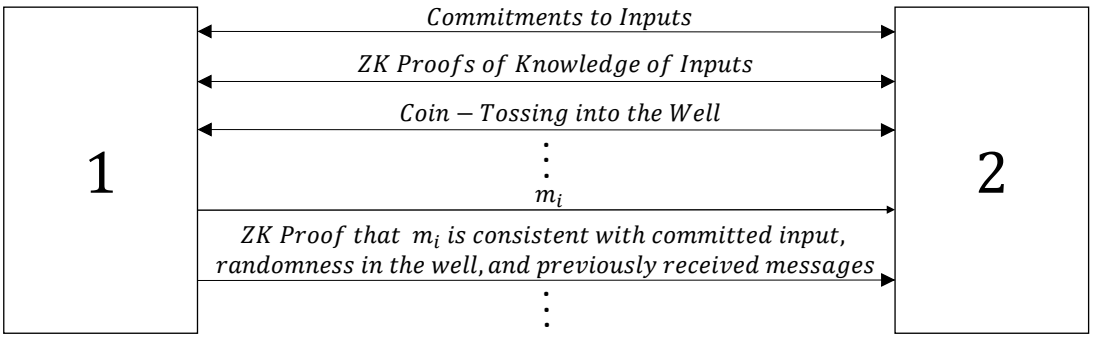

Figure 2: Maliciously-secure protocol for two parties π', which is constructed out of π, a semi-honestly-secure protocol.

References

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.