# CSE 5854: Class 12

Benjamin Fuller

February 27, 2018

## 1 Maliciously Secure OT

In the first part of this class we covered how to construct maliciously secure oblivious transfer directly from the decisional Diffie-Hellman assumption. This material is covered in Class 10 notes.

## 2 Defining Secure Multi-party computation

We're going to start moving from building a secure computation between two parties to multiple parties. Even in the case of two parties we saw our definition of security getting more and more complicated. As an example, for an interactive proof we first consider a malicious prover and sad they couldn't be convincing when $x \notin L$. We then added a zero knowledge property which talked about the actions of a malicious verifier. Finally, we considered knowledge extraction which required us to again strengthen the requirements against a malicious prover.

This seems like it would get combinatorially more complicated as we add multiple parties. Some of the natural questions we should consider:

1. Who are we trying to defend against? Is the adversary a single party or controls multiple parties in the computation?
2. What kind of network resources are available? Can the adversary see every message that is sent? Can they modify it?
3. Are we concerned about adversaries with unbounded computational power?
4. Does the adversary follow the protocol or deviate arbitrarily?
5. Are we able to ensure that the computation represents some well defined input on the part of all parties?
6. Are we able to ensure that the adversary's input(s) are independent of honest parties input?
7. Is the adversary in a set position at the start of the protocol? Are they able to decide which parties to adapt mid way through the protocol? Are they able to move to different positions at different parts of the protocol?

8. What do we consider a breach of security? In the zero-knowledge case this was quite sophisticated, we considered learning anything beyond the validity of the statement even in the presence of auxiliary input.

To deal with all of this complexity we're going to make two conceptual changes:

1. We're going to switch to a positive definition of security. Instead of asking that an adversary can't do things, we will say that the protocol is secure if the adversary can do only a prescribed set of things. We'll use the simulation paradigm (used in zero-knowledge) to make this change.

2. We'll separate out the question of adversary capabilities from the question of what the protocol is supposed to do. So we can say a protocol is "secure" with respect to one class of adversaries but not another class without changing what it means for a protocol to be secure.

## 2.1 Real-Ideal

We note that the concerns above can be effectively split into what is considered "secure" and what an adversary is allowed to do. This new paradigm is called the real-ideal paradigm. We consider two worlds called the *real* and *ideal* world. In the real world we will consider honest parties executing the protocol $\Pi$ in the presence of an attacker or adversary. In the ideal world we will define something called an ideal functionality. Roughly, an ideal functionality $\mathcal{F}$ consists of three stages:

1. Each party gives its input $x_i$ to the functionality $\mathcal{F}$.

2. Upon receiving all inputs $x_i$, $\mathcal{F}$ computes $f(x_1, ..., x_n) = y_1, ..., y_n$.

3. The functionality delivers $y_1, ..., y_n$.

The idea is that for each attacker $\mathcal{A}$ going after the protocol $\Pi$ in the real world we should be able to find an attacker $S$ that does just as well interacting with the ideal world. This is exactly what we required for simulation in zero-knowledge proofs: if $S$ knew that $x \in L$ it was required to produce a transcript that was the same as what would be produced in the actual zero-knowledge protocol. More formally we say the following.

**Definition 1.** *[GMW87] Let $\Pi$ be a protocol between $n$ players. We say that $\Pi$ realizes functionality $\mathcal{F}$ with respect to adversaries $A \in \mathcal{A}$ if for every $A$ there exists a simulator $S$ such that for every $x_1, ..., x_n$*

$$A \leftrightarrow \langle P_1(x_1), ..., P_n(x_n) \approx_c S^{\mathcal{F}}(1^n).$$

**Note:** Recall that the symbol $\approx_c$ means computationally indistinguishable. In the worst case we think of $A$ outputting any inputs, transcripts, and internal randomness. Thus, it suffices to have $S$ output an indistinguishable transcript. Furthermore, we implicitly assume the ideal functionality can only be executed once as honest parties would notice if they were asked to participate twice. Apriori this definition does not allow the simulator to see any inputs of the protocol and its not clear how the simulator interacts with the ideal functionality. Often

we will consider ideal functionalities that allow the simulator to specify that they would like to control this party. This is usually known as "corrupting" a party.

We can extend definition 1 to allow parties in the protocol $\Pi$ to have access to some resource. This might be a public-key infrastructure, a broadcast channel, or private channels between each party. The basic definition does not consider such resources.

# References

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.