University of Illinois, Urbana Champaign
CS 598DK Special Topics in Cryptography

Instructor: Dakshita Khurana
Scribe: Sourav Das and Jong Chan Lee
Date: September 13, 2019

LECTURE
6

Oblivious Transfer, Secure Computation - I

6.1 Introduction

In the last lecture we have discussed what Public Key Encryption, Private Key Encryption are. We have also talked about El-Gamal's public key encryption scheme. In this lecture we will first quickly review construction of El-Gamal's public key encryption scheme. We then introduce Oblivious Transfer and discuss a construction, prove its security with semi-honest (honest but curios) adversarial model. Finally, I will briefly introduce the idea of Simulation Based Security.
NOTE: Unless otherwise stated, from here on we will use pk to denote the public key of a cryptographic system and sk to denote the corresponding secret key. For symmetric key encryption, $pk = sk$. Also we use m denote the message.

6.2 Preliminaries

El-Gamal Public Key Encryption. Given a cyclic group G of size $|G| = q$ with g as one of its generator. El-Gamal's Public Key Encryption can be described using three PPT algorithm KeyGen, Encrypt, and Decrypt with the following descriptions.

- $\mathsf{KeyGen}(1^k, r) \to x$, here $sk = x$ and $pk = (g, g^x = h)$.
- $\mathsf{Encrypt}(pk, m, r') \leftarrow (g^y, mh^y)$ where $y \leftarrow \{0,1\}^k$ and chosen according to the randomness r'.
- $\mathsf{Decrypt}(x, ct = (c_1, c_2)) \to c_2 c_1^{-1}$, where c_1^{-1} is the multiplicative inverse of c_1 in G

Just like any other Public-Key-Encryption (PKE) scheme, we want El-Gamal public key encryption to provide the following guarantees.

- **Correctness:** $\forall k, r, r', m$, $\mathsf{Decrypt}(x, \mathsf{Encrypt}(g^x, m, r')) = m$
- **Security:** (IND-CPA) $\forall m_0, m_1, (g, h, m_0 h^y) \approx_c (g, h, m_1 h^y)$

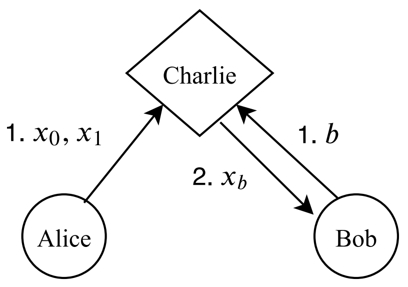

FIGURE 6.1: 1-2 Oblivious Transfer using trusted party

Proof Sketch of El-Gamal scheme's security. Security of El-Gamal builds upon the underlying argument that Decisional Diffie Helman (DDH) is hard i.e no efficient (polynomially bounded) nuPPT adversary can distinguish between (g^x, g^y, g^{xy}) and (g^x, g^y, g^z) where $x, y, z \leftarrow G$ with more than $\frac{1}{2} + \mathsf{negl}(k)$ probability.

6.3 Oblivious Transfer (OT)

DEFINITION 6.1. *1-2 Oblivious Transfer* (Informally) Alice and Bob are interested in executing the following protocol. Alice holds two input $x_0, x_1 \in \{0,1\}^k$. Alice wants to send exactly one of its input $x_b, b \in \{0,1\}$ to Bob where Bob holds the bit b. From protocol we seek first, x_{1-b} remains hidden from Bob and second, b remains hidden from Alice.

Specifically, Alice with input $x \in \{0,1\}^k$ and Bob with input $b \in \{0,1\}$ wants to participate in a protocol that computes the functions $f(x_1, x_2, b) = x_b$ and outputs x_b to Bob and Alice gets no output. We first demonstrate a protocol to achieve this with the presence of a trusted third party Charlie and then later demonstrate a protocol without Charlie. But before that, I would like to ask, why are we studying oblivious transfer?

Motivation. We demonstrate that, Oblivious Transfer (OT) allows us to compute arbitrary multi-party computation of arbitrary functions.

1-2 OT using trusted third Party. The protocol in with Charlie as a trusted third party is demonstrated in Figure 6.1 which comprises of the following steps:

1. Alice sends its input $x_0, x_1 \in \{0,1\}^k$ to Charlie.
2. Bob sends its input $b \in \{0,1\}$ to Charlie.[1]
3. Charlie sends back Bob x_b and sends nothing to Alice.

Observe that in the presence of a trusted Charlie, the above protocol secure in information theoretic sense. Why?

[1]Step 1 and 2 are interchangeable.

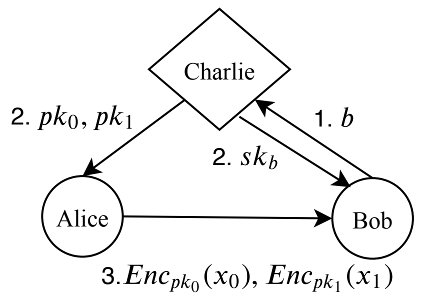

FIGURE 6.2: 1-2 Oblivious Transfer with trusted party during Setup phase.

We seek a seek a cryptographic mechanism to eliminate Charlie. We will design such a protocol in small step. First, we will describe a protocol that uses the trusted third party only during *Setup* phase. Later we will eliminate such requirement as well.

1-2 OT with Trusted third party during Setup. In Figure 6.2 we present a variant of the protocol where Alice and Bob uses the third party only during setup and later run the 1-2 OT by only themselves. Hence we divide the protocol into two phase: *Setup* and *Transfer*. The description of the Setup phase is below.

1. Bob sends its input $b \in \{0, 1\}$ to Charlie.
2. Charlie generates two public-private key pairs say using El-Gamal's public key encryption scheme described above. Let these be (pk_0, sk_0) and (pk_1, sk_1). Charlie then sends sk_b to Bob and pk_0, pk_1 to Alice.

After, Setup phase is complete, Alice and Bob no longer requires Charlie to be part of the Transfer part of the protocol because they can execute the 1-2 OT as follows:

1. Alice sends $(c_1, c_2) = (\mathsf{Encrypt}(pk_0, x_0), \mathsf{Encrypt}(pk_0, x_0))$ to Bob.
2. Bob on receiving (c_1, c_2) tries to decrypt both using sk_b and the keeps the plain text that she can correctly decrypt.

Question: How does Bob identify which decryption was successful as decryption (El-Gamal) of both cipher text will give us a element in the G? Hint: pad extra information to x_1 x_2.

6.4 Oblivious Transfer in the Plain Model

In secure multi-party computation, we generally consider two kinds of adversary (although there are many other variants of it): (i) *Semi-honest* or *Honest but Curious*, and (ii) *Malicious* or *Byzantine* adversary. We define them (informally) as follows:

DEFINITION 6.2. Semi-Honest Adversary (Informal). In a protocol, a semi-honest adversary correctly follows the protocol, i.e sends messages whenever protocol requires it, uses the randomness given by the protocol correctly, but is eager to learn the private input of other participants of the protocol.

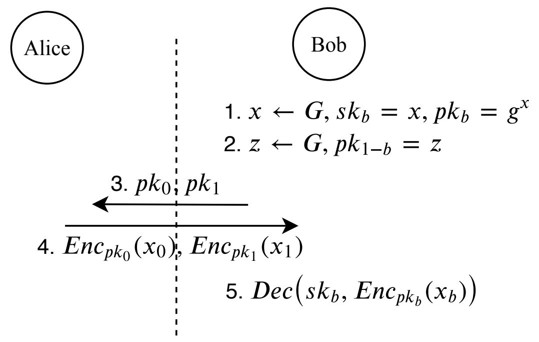
Alice
Bob
1. $x \leftarrow G, sk_b = x, pk_b = g^x$
2. $z \leftarrow G, pk_{1-b} = z$
3. pk_0, pk_1
4. $Enc_{pk_0}(x_0), Enc_{pk_1}(x_1)$
5. $Dec(sk_b, Enc_{pk_b}(x_b))$

FIGURE 6.3: 1-2 Oblivious Transfer without Trusted party.

DEFINITION 6.3. Malicious Adversary (Informal). An adversary is called Byzantine when it can deviate arbitrary from the prescribed protocol. Such deviation include but not limited to aborting during execution of protocol, sending carefully crafted messages to actively extract private input of other parties in the protocol.

Question: Are assumption of Semi-Honest adversary practical?
Answer:

- Any protocol for semi-honest adversary can be converted into a protocol with malicious adversary using Zero-knowledge Proofs [].
- Semi-honest adversary captures a scenario where during the protocol participant was honest but got corrupted (or hacked) by an adversary after the protocol was over. In such an attack, the adversary $\mathcal{A}$ gets access to the entire transcript of the protocol that the victim observes during the protocol.

For today we will only look at Semi-Honest adversary. Fun fact: Oblivious Transfer is a stronger assumption than Public-Key Encryption. Why? We will give a proof sketch of this argument later in the lecture.

1-2 Oblivious Transfer without Trusted Assumption. We give a specific instantiation of the 1-2 OT using El-Gamal's public key encryption. However, El-Gamal's scheme can be replaced by any **known** public key encryption scheme. Figure 6.3 gives a illustration of the protocol described below:

- With (b, k) as its input, Bob generates sk_b, pk_b using $\mathsf{KeyGen}(1^k)$. Additionally Bob uniformly randomly samples a group element $z_{1-b} \leftarrow G$. Note that, in El-Gamal's PKE, $sk_b = x, pk_b = g^x$. Finally, Bob sends $(z_b = pk_b, z_{1-b})$ to Alice.
- With local input x_0, x_1 and receiving the tuple z_0, z_1, Alice computes $c_0 = \mathsf{Encrypt}(z_0, x_0)$ and $c_1 = \mathsf{Encrypt}(\mathsf{z}_1, \mathsf{x}_1)$ and sends the tuple (c_0, c_1) to Bob.
- Bob on receiving (c_0, c_1) decrypts c_b using sk_b and successfully gets back x_b.

Let's consider why the above construction guarantees *Correctness* and *Security* which are defined as below:

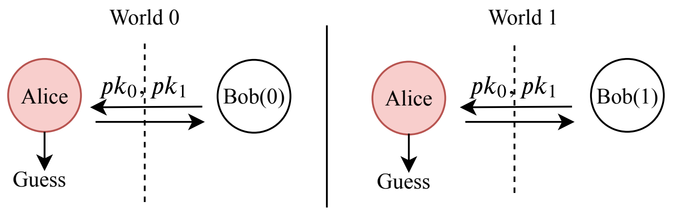

FIGURE 6.4: Receiver's Security in 1-2 Oblivious Transfer

- **Correctness:** Bob successfully recovers x_b
- **Security:** (i) Alice learns nothing about Bob's choice bit b, and (ii) Bob learns no information about x_{1-b} except what is revealed by x_b. If x_b and x_{1-b} are chosen independently Bob learns nothing about x_{1-b}.

It is easy to observe that the correctness of the protocol is obvious so we will focus on the security. We will first present a game based security and then later prove it using simulation based security.

Receiver's Security against unbounded Sender. Consider two different world w_0 and w_1 where Bob's input bit in w_0 (respectively w_1) is 0 (respectively 1). Alice participates in a 1-2 OT transfer in one of the world without knowing which world it is. At the end of the protocol, we seek Alice to correctly identify the world with only $\frac{1}{2} + \mathsf{negl(k)}$ probability on security parameter k.

THEOREM 6.4. *In the 1-2 Oblivious Transfer in the Plain-Model construction given above, no efficient honest-but-curios sender does learn the input bit b of the receiver with probability higher than* $\frac{1}{2}+\mathsf{negl}(k)$ *probability. Note that for this specific construction, this theorem holds even for computationally unbounded adversary.*

Proof. Since both $z_0, z_1 \in G$ and are public key corresponding to some unique secret key, even a computationally unbounded Alice can't distinguish between z_0 and z_0 since $g^x \approx_u z$. Hence the best strategy for Alice is to randomly guess Bob's input and Alice's guess will only be correct with probability $\frac{1}{2}$. ∎

Sender's Security against Computationally bounded Receiver. Just like earlier, consider two world w_0 and w_1 where in world w_0 (respectively w_1) sender's inputs are (x_0, x_1) (respectively (x'_0, x'_1)) where $x_b = x'_b$ and $x_{1-b} \neq x'_{1-b}$. The receiver participates in one of the world without knowing which it is, with its input bit b and thus gets $x_b = x'_b$. Now at the end of the protocol, we seek the computationally bounded receiver to correctly identify the world he did participate only with $\frac{1}{2} + \mathsf{negl}(k)$ for security parameter k.

THEOREM 6.5. *In the 1-2 Oblivious Transfer in the Plain-Model construction given above, no efficient honest-but-curios receiver will be able to distinguish between world* w_0 *and* w_1 *with probability higher than* $\frac{1}{2} + \mathsf{negl}(k)$ *probability.*

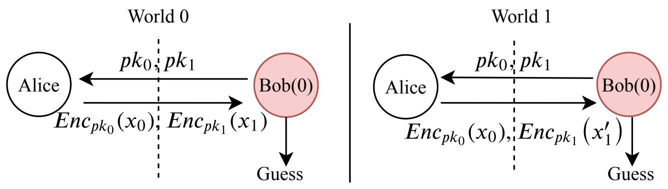

FIGURE 6.5: Sender's security against Semi-honest Receiver

Proof. We will prove this by contradiction. Suppose the receiver could distinguish between the world w_0 and w_1 with probability greater than $\frac{1}{2} + \mathsf{negl}(k)$, then we show that an polynomial time adversary $\mathcal{A}$ could use receiver to break the employed public key encryption scheme as follows:

- $\mathcal{A}$ internally runs OT with the receiver mentioned above to get the public keys pk_0, pk_1.
- To break security of any of the public key say pk_1, $\mathcal{A}$ sends the challenger $\mathcal{C}$ two messages x_1, x'_1 and gets its encryption say under public key pk_1. Let these encrypted messages be c_d.
- $\mathcal{A}$ internally runs a OT protocol with the receiver mentioned where $\mathcal{A}$ sets the input of the sender as $(\mathsf{Encrypt}(pk_0, x_0), c_d)$.
- $\mathcal{A}$ then outputs the decision bit of the receiver of the simulated OT.

$\square$

Question. Earlier we briefly mentioned that Oblivious transfer is a harder assumption than PKE, but notice that we have constructed the above protocol using El-Gamal's PKE. Is it not contradicting our previous argument?
Answer: Notice that in the above construction we were implicitly making an assumption that the receiver of the OT output can sample a public key without actually sampling its secret key. This guarantee is not mandatory for a public-key scheme.

6.5 Two-Party Secure Computation of XOR

At the beginning of the lecture, we mentioned that the existence of an oblivious transfer protocol that allowed two parties (Alice and Bob) to compute the very simple functionality $F((x_0, x_1), b) = x_b$ securely in the semi-honest, or "honest-but-curious" adversary model would be sufficient evidence that we should be able to compute any arbitrary functionality securely under the same semi-honest adversary assumptions. In this section, we examine the almost trivial case of computing $x \oplus y$ given the inputs x and y, and try to make sense of what the definition of security should be for this particular situation.

We note that these semi-honest adversaries can be converted into malicious adversaries in a black-box way, but we do not dive deeper into the subject during this lecture.

Let us define the problem in the form of a question. A diagram representing the situation is given in Figure 6.6.

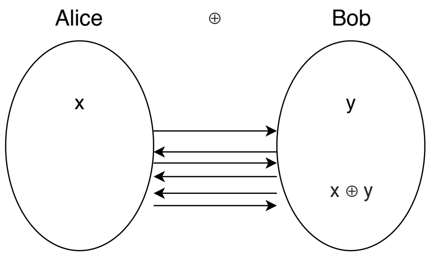

FIGURE 6.6: Two-Party Computation of the XOR functionality

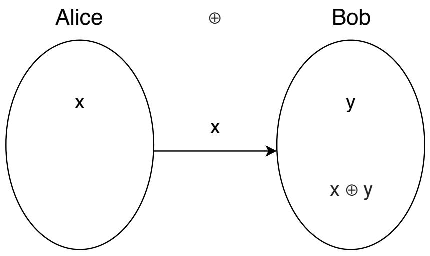

FIGURE 6.7: Trivial Solution Protocol for Two-Party Computation of XOR

PROBLEM 6.6. *Two-Party Secure Computation of XOR.* Let us consider two separate parties Alice($\mathcal{A}$) and Bob($\mathcal{B}$), whom to each other are potentially semi-honest adversaries.
Alice is given as input a single bit $x \in \{0,1\}$, while Bob is also given a single-bit input $y \in \{0,1\}$.
Question. Is there a secure protocol through which Bob can obtain the output $x \oplus y$ (Alice does not have any output), while Alice and Bob learn as close to nothing about each other's input values as possible?

A simple, obvious protocol comes to mind that solves this problem. We can simply have Alice communicate her input x to Bob and have Bob run the computation $(x, y) \rightarrow x \oplus y$ on his own. We illustrate this in Figure 6.7.

At first glance, this protocol may seem insecure since it would lead to Bob learning Alice's input x. It is difficult to come up with a security argument for this scheme under our previous notion of game-based security. However, Bob learning x is an inevitable consequence of the problem setting that arises regardless of what protocol we use to solve this problem. Once Bob obtains the output value $x \oplus y$, he can simply combine that with his own input y to compute $x = (x \oplus y) \oplus y$. Thus, Bob being able to figure out x is not a sign that our trivial protocol is insecure; rather the definition of security that applies to such protocols should be flexible enough to tolerate our simple solution.

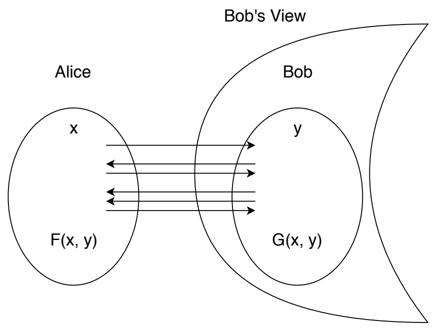

FIGURE 6.8: Real World Protocol and Bob's View

6.6 Real-Ideal Model

To reach a notion of security that will help us with the dilemma of the previous section, we first introduce the idea of a comparison between the "real world" and the "ideal world". First let us define a general version of the two-party secure computation protocol that we actually want to make secure, i.e. the "real world" protocol. For now we will leave out the 'secure' part.

DEFINITION 6.7. *Real World Protocol.* Let us consider two separate parties Alice($\mathcal{A}$) and Bob($\mathcal{B}$), whom to each other are potentially semi-honest adversaries. A *real world two-party (secure) computation protocol* in which Alice and Bob are the participating parties can be summarized as the following:

1. Alice and Bob each receive as input some values x and y, respectively, so that neither party knows the other's input.
2. Alice and Bob send each other messages that help with the computation of functionalities $F(x, y)$ (Alice) and $G(x, y)$ (Bob).
3. Eventually, Alice outputs $F(x, y)$ and Bob outputs $G(x, y)$, either or both of which can be null if we want Alice or Bob (or both) to not have any output value.

Figure 6.8 is a depiction of how the real world protocol might look.

The ideal world protocol for two party secure computation is a protocol that has the same end goal as the real world protocol, but one that is made much easier thanks to the existence of an ideal trusted third party (TTP; Charlie in our previous discussion in Section 6.3). An ideal world protocol would look something like Figure 6.9 and can be defined as in the following.

DEFINITION 6.8. *Ideal World Protocol.* Let us consider two separate parties Alice($\mathcal{A}$) and Bob($\mathcal{B}$), whom to each other are potentially semi-honest adversaries. In addition let us say that there is a party Charlie($\mathcal{C}$) that both Alice and Bob can communicate securely with. We assume that Charlie is an ideal trusted third party whom both Alice and Bob can trust their inputs with and who will provide only completely reliable output values to Alice and

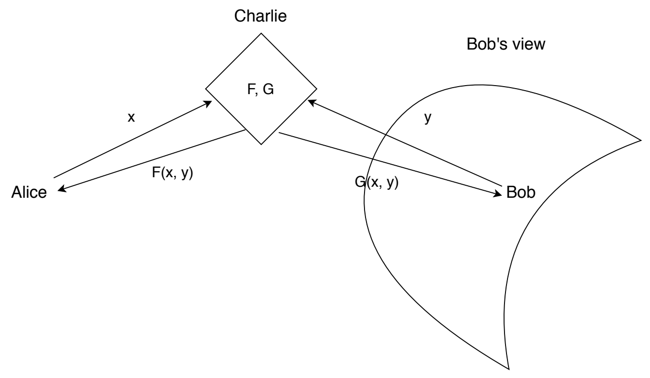

FIGURE 6.9: Ideal World Protocol and Bob's View

Bob. An *ideal world secure two-party computation protocol* in which Alice and Bob are the participating parties can be summarized as the following:

1. Alice and Bob each receive as input some values x and y, respectively, so that neither party knows the other's input.
2. Alice and Bob each send Charlie their respective inputs x and y.
3. Charlie computes the functionalities $F(x, y)$ and $G(x, y)$.
4. Charlie sends $F(x, y)$ to Alice and $G(x, y)$ to Bob.
5. Alice outputs $F(x, y)$ and Bob outputs $G(x, y)$

It is easy to see how the ideal world protocol is secure by pretty much any definition of security as long as we can find an ideal trusted third party Charlie. Then one way to define security in the real world would be to use the above ideal world protocol as a standard. What we want from a secure real world protocol would be that:

1. Alice learns just as much information about y as she would were Alice and Bob participating in the corresponding ideal world protocol, and
2. Bob similarly learns just as much information about x as he would were Alice and Bob following the corresponding ideal world protocol.

Applying this new notion of security, we would be able to claim that the trivial protocol that we proposed for computing $x \oplus y$ is secure. Bob would be able to learn $x = (x \oplus y) \oplus y$ even if Alice and Bob were to adhere to what would be the ideal world protocol of the XOR computation problem, so Bob being able to learn x via our trivial protocol is no longer a violation of security.

A way to formalize this notion of security would be to define a real world protocol to be secure if and only if it were indistinguishable from a corresponding ideal world protocol. Take, for example, the case in which Bob is potentially a semi-honest adversary. Let us collectively refer to

- all of the interactions Bob has with other parties,

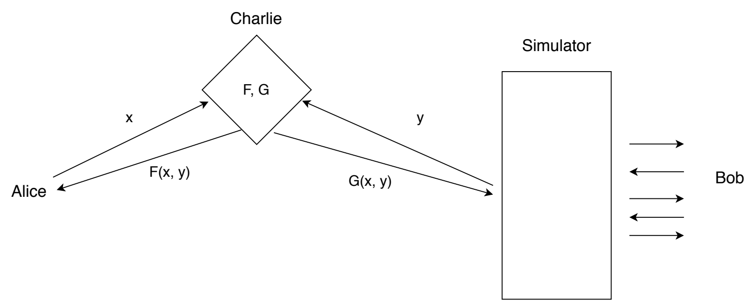

FIGURE 6.10: Ideal World Protocol with Simulator

- his input and output values,
- and the (secret) states he holds during the protocol

as Bob's *view* of the protocol. We can define the security of the protocol with regard to Bob as the computational indistinguishability of Bob's view of the real world protocol from Bob's view of the ideal world protocol.

There is a problem to this real/ideal-world-view-based notion of security, however. The ideal world protocol is obviously distinguishable from the real world protocol by the fact that Bob interacts with Charlie and Charlie only, instead of interacting with Alice. Therefore, Bob's "interface" or Bob's view of the ideal world protocol and that of the real world protocol can never actually be computationally indistinguishable.

Our solution to this is to alter the ideal world protocol so that we can force Bob's view of it to be indistinguishable from that of the real world protocol. This is achieved via the introduction of the concept of simulation.

6.7 Simulation-Based Security

Let us continue to refer to the situation in which Bob is potentially a semi-honest adversary.

A *simulator* is an algorithm that inserts itself between Bob and the rest of the ideal world protocol and pretends to be real world Alice, interacting with Bob in the exact same way Alice would in the real world protocol. It simulates Alice using only the information that ideal world Bob would know. As a result, given that the real world protocol and the (simulated) ideal world protocol are indistinguishable, Bob should not be able to figure out x in the real world. More specifically, $Simulator(1^k, r)$ is an algorithm that takes as input a size k string of 1's (where k is the security parameter) and some randomness r, and simulates real world Alice. A diagram of how this would look is given in Figure 6.10.

One behavior of simulators that differentiates them from a "real world Alice" is that they extract all inputs from the semi-honest adversaries they are dealing with. In our example case, the simulator extracts y from Bob. This is so that the simulator can play the input-output game with Charlie, i.e. give Charlie y and learn $G(x, y)$ in return.

It is helpful to think of the simulator as a *public* algorithm, which Bob should be able to interact with in a manner that is indistinguishable from how he would interact with Alice.

The simulator in this case should also be a probabilistic polynomial-time (PPT) Turing machine, and is thus at most as powerful as Bob is. It is customary to think of the simulator as computationally bounded, even when the semi-honest adversary (Bob) is assumed to be computationally unbounded. There are some exceptions, however, in which the adversary is computationally unbounded, where it may make sense to consider the simulator to be unbounded as well.

It is easy to conjure an analogous definition of security for Alice.

To sum up the results of what we have discussed so far, we can finally define two-party secure computation protocols using the newly introduced notion of simulation-based security.

DEFINITION 6.9. *Two-Party Secure Computation Protocol.* Given two potentially semi-honest, or "honest-but-curious" parties Alice(1^{st}) and Bob(2^{nd}), each with inputs x and y, respectively, that are unknown to the other party, a protocol for the two-party computation of a probabilistic polynomial-time functionality $F(x, y)$ is *secure* if there exists for each i^{th} party ($i \in \{1, 2\}$) a probabilistic polynomial-time algorithm $Simulator_i(1^k, r)$ such that the simulated ideal world two-party computation protocol and the original real world protocol are computationally indistinguishable in terms of the view of the i^{th} party.

Note that we used the terms i^{th} party in the later parts of the definition instead of Alice and Bob. One can infer that this definition can be straightforwardly extended to cases in which 3 or more parties are involved, i.e. it can be generalized to a simulation-based security definition for multi-party computation (MPC).

Simulation-based security will be dealt with in more depth in the next lecture.

6.8 Epilogue

Once again recall our claim that a secure oblivious transfer protocol that computes the functionality $F((x_0, x_1), b) = x_b$ in the "honest-but-curious" adversary model implies our capability to securely compute any arbitrary functionality under the same semi-honest adversary assumptions.

Question for next class. Why is the existence of a secure oblivious transfer protocol sufficient for us to successfully implement a secure two-party protocol for multiplication (AND)? In other words, can we construct a protocol where Alice, given input x, and Bob, given input y, can interact with each other that Bob can output xy (and Alice outputs nothing) in a way that is secure?fa

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.