Two-Party Computation and Oblivious Transfer

Noah Stephens-Davidowitz

June 9, 2023

1 Secure computation

We now turn to the topic of *secure computation*, which will be the last major topic of this course. The idea is as follows. There are ℓ parties, $P_1, \ldots, P_\ell$, each with its own *private* input $x_1, \ldots, x_\ell$. They wish to compute a function $(z_1, \ldots, z_\ell) = f(x_1, x_2, \ldots, x_\ell)$, in such a way that P_i learns z_i "but nothing more." For example, the parties might be companies and the x_i might be the maximum they're each willing to spend on some good that is up for auction. The z_i could reveal whether or not you have won the auction. I.e., $z_i = 1$ if $x_i > x_j$ for all $j \neq i$ and $x_i = 0$ otherwise.

We will focus mostly on the case $\ell = 2$, i.e., the case in which there are two parties Alice and Bob who wish to jointly and securely compute some function $(z_A, z_B) = f(x_A, x_B)$. Cryptographers often simply call this *two-party computation* (2PC) or *secure two-party computation.* (The special case in which only one party receives any output is often called either *secure function evaluation* (SFE) or *private function evaluation* (PFE), but we will not make this distinction.)

Of course, to make this formal, we need to specify what it means that "Alice learns nothing except z_A and Bob learns nothing except z_B." We formalize this in the same way that we did in the case of zero-knowledge proofs: using simulators. Here is the formal definition.

Definition 1.1. *A protocol $\Pi = (A, B)$ between two efficient parties A, B* computes a functionality $f = (f_A, f_B)$ *if* $\mathsf{out}_A\langle A(1^n, x_A), B(1^n, x_B)\rangle = f_A(x_A, x_B)$ *and similarly* $\mathsf{out}_B\langle A(1^n, x_A), B(1^n, x_B)\rangle = f_B(x_A, x_B)$*.*

Π securely computes a functionality $f = (f_A, f_B)$ against honest-but-curious adversaries *if it computes f* and *there exist PPT simulators S_A and S_B such that for every PPT adversary $\mathcal{E}$ there exists negligible $\varepsilon(n)$ such that for all x_A, x_B,*

$$\Pr[\mathcal{E}(1^n, S_A(1^n, x_A, f_A(x_A, x_B))) = 1] - \Pr[\mathcal{E}(1^n, \mathsf{View}_A\langle A(1^n, x_A), B(1^n, x_B)\rangle) = 1] \leq \varepsilon(n) \ ,$$

and similarly,

$$\Pr[\mathcal{E}(1^n, S_B(1^n, x_B, f_B(x_A, x_B))) = 1] - \Pr[\mathcal{E}(1^n, \mathsf{View}_B\langle A(1^n, x_A), B(1^n, x_B)\rangle) = 1] \leq \varepsilon(n) \ .$$

More succinctly, $(S_A(1^n, x_A, f_A(x_A, x_B))) \approx_c \mathsf{View}_A\langle A(1^n, x_A), B(1^n, x_B)\rangle$, and similarly for B.

The intuition behind this definition is the same as the intuition behind the definition of zero knowledge that we saw when we studied zero-knowledge proofs. It says that "anything that Alice learned from interacting with Bob, she could have generated herself, given only her input x_A and her output $f_A(x_A, x_B)$." (Or, more accurately, she can produce something that is computationally indistinguishable from what she learned from interacting with Bob.) In that sense, "she learns nothing except for $f_A(x_A, x_B)$."

Notice that this security definition depends *crucially* on f. E.g., if f_A is a function such that $f_A(x_A, x_B)$ reveals a lot of information about x_B, then the security notion is quite a bit weaker, in the sense that we will give the simulator $f_A(x_A, x_B)$, which contains a lot of information about x_B. This is of course necessary, since correctness requires that "Alice learns at least as much information about x_B as she learns from $f_A(x_A, x_B)$."

We say that this security only holds against *honest-but-curious* adversaries because we only required the simulator above to output a view that is indistinguishable from the view of an honest Alice A (and likewise for Bob). We will later see a more sophisticated definition that allows for the possibility that a malicious Alice might deviate from the protocol. In fact, if we have time, in one of our last lectures, we will show how to convert any protocol that is secure against honest-but-curious adversaries into one that is secure against malicious adversaries.

We sometimes break the security definition up into "security for Alice" and "security for Bob." Notice that security for Alice means that *Bob's view* can be simulated by a simulator S_B, and security for Alice means the *Alice's view* can be simulated. In other words, Alice is happy if Bob learns nothing that he's not supposed to—i.e., if Bob's view can be simulated—and Bob is happy if Alice learns nothing but what she is supposed to.

1.1 Which functions can we compute?

Of course, it is natural to ask which functions f can be computed securely.

We will see that the answer is any efficiently computable function! Perhaps this actually isn't very surprising given what we've already seen. For example, maybe you can think of a construction of 2PC from FHE? (The most naive construction of 2PC from FHE is actually not secure, though. . .)

However, we won't need anything nearly as fancy as FHE to do this. We will instead use a seemingly much weaker primitive called oblivious transfer. (In fact, for *multiparty computation* with three or more parties, in some sense "we won't need anything." It is possible to compute any function in this setting securely without any computational assumptions (using Shamir's secret-sharing scheme)! But, for two-party computation, computational assumptions are necessary.)

2 Oblivious transfer

An *oblivious transfer* protocol is a special case of two-party computation for a function $f_{OT} = (f_{OT,A}, f_{OT,B})$ that might not seem tremendously useful. Specifically, Bob's input is a single bit $b \in \{0, 1\}$, and Alice's input is a pair of bits $m_0, m_1 \in \{0, 1\}$. At the end of the protocol, Alice learns absolutely nothing. I.e., $f_{OT,A}((m_0, m_1), b) = \perp$. (This $\perp$ symbol is pronounced "bottom" and it is often used to represent the empty output. The LaTeX code for this is `\bot`. The $\perp$ symbol is sometimes also used to represent a party refusing to do something or aborting a protocol—e.g., if a decryption algorithm determines that a ciphertext is malformed, it might output $\perp$.) Bob learns m_b, i.e., $f_{OT,B}((m_0, m_1), b) = m_b$.

In other words, in an oblivious transfer protocol, Bob "requests either m_0 or m_1 (but *not* both!), and Alice sends the appropriate message to Bob, while somehow remaining oblivious to whether Bob requested m_0 or m_1." Security for Bob says that "Alice is in fact oblivious"—i.e., there exists a simulator S_A that takes as input $1^n, m_0, m_1$ (but not b!) and produces a view that is computationally indistinguishable from Alice's view in a true run of the protocol with Bob. Security for Alice says that "Bob learns m_b but nothing else (particularly nothing about m_{1-b})"—i.e., there exists an efficient simulator S_B that takes as input 1^n, b, and m_b (but not m_{1-b}!) and produces a view that is computationally indistinguishable from Bob's view in a true run of the protocol with Alice. Though this might not seem useful, oblivious transfer turns out to be very important.

In the next couple of lectures, we will prove the following theorem, which says that "f_{OT} is complete for 2PC." In fact, we will give two very different proofs of this theorem, yielding two very different protocols.

Theorem 2.1. *If there exists a protocol for oblivious transfer (i.e., a protocol that securely computes f_{OT} against honest-but-curious adversaries), then for any efficiently computable function f, there exists a protocol that efficiently computes f (against honest-but-curious adversaries).*

2.1 Constructing OT—a thoroughly unsatisfying protocol

In this lecture, we will simply show how to build OT. The protocol that we come up with will be extremely unsatisfying in the sense that, while it is secure against honest-but-curious adversaries, there will be a really really really obvious way for a very slightly malicious Bob to completely break the security of the scheme. So, we will aggressively exploit the fact that we only want security against honest-but-curious adversaries.

Anyway, here's how the construction goes. We will need to use the following primitive. Intuitively, this is a public-key encryption scheme in which there is a way to "generate a useless public key." I.e., there should be a way to generate a public key without being able to decrypt. (We will not use this primitive for anything else in this course, though it is rather interesting in its own right.) Notice that we insist right in the definition that the underlying scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ should be correct and semantically secure, since we're not really interested otherwise.

Definition 2.2. *A PPT algorithm* OGen *is an* oblivious key-generation algorithm *for a correct and semantically secure public-key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *if it satisfies the following two properties.*

- **(Obliviousness.)** *For any PPT algorithm $\mathcal{A}$ there exists negligible ε such that for any two plaintexts m_0, m_1,*

  $$\Pr_{r \sim \{0,1\}^\ell, b}[pk = \mathsf{OGen}(1^n; r),\ \mathcal{A}(1^n, r, \mathsf{Enc}(pk, m_b)) = b] \leq 1/2 + \varepsilon(n) \ .$$

  In other words "the encryption scheme with a public key generated by OGen *remains semantically secure even if you know the randomness r that was used to generate the key." Including the randomness here is very important.*

- **(Indistinguishability.)** *For any PPT $\mathcal{A}$, there exists negligible ε such that*

  $$\Pr[\mathcal{A}(1^n, \mathsf{OGen}(1^n)) = 1] - \Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^n),\ \mathcal{A}(1^n, pk) = 1] \ .$$

  In other words, a public key generated by OGen *is computationally indistinguishable from a public key generated by* Gen.

Many of the public-key encryption schemes that we have seen have an OGen algorithm associated with it. We will run through them quickly below.

- Remember that the ElGamal encryption scheme has as its public key $(g, h = g^a)$, and its secret key is the discrete logarithm a. To "obliviously" generate a public key, simply sample h uniformly at random, rather than sampling its discrete logarithm a. (E.g., say that the group is $\mathbb{Z}_p^*$ for simplicity. Then, it is easy to sample a random element h in $\mathbb{Z}_p^*$ without the associated discrete logarithm, by simply sampling a uniformly random element from $\{1, \ldots, p-1\}$.) In this case, the obliviously generated key is identically distributed to an honestly generated key, and obliviousness follows from the security of the original scheme.

- The Goldwasser-Micali encryption scheme has as its public key $(N = pq, y)$, where y is a quadratic *non*-residue modulo N (with Jacobi symbol equal to one), and the secret key was p, q. You might try to obliviously sample a public key by sampling N without knowing the factorization, but this is problematic because it is not known how to sample N that has exactly two prime factors (or that it is indistinguishable from an integer with exactly two prime factors) without knowing the factorization. Instead, simply sample N as normal, but take y to be a quadratic residue instead of a non-residue. Under the quadratic residuosity assumption, this is indistinguishable from an honestly generated public key. And, when we proved security of this scheme, we saw that if y were a quadratic residue, then the ciphertext is independent of the plaintext, so that this is certainly oblivious (even if you know the factorization of N).

- The Regev encryption scheme has as its public key $(\boldsymbol{A}, \boldsymbol{b} := \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \bmod q)$. To sample a public key obliviously, simply sample $\boldsymbol{b} \sim \mathbb{Z}_q^m$ instead. The LWE assumption tells us that these two public keys are computationally indistinguishable. And, when we proved security of the Regev encryption scheme, we showed that, when the public key is replaced by uniformly random elements, the ciphertext is (statistically close to) independent of the plaintext.

Now, here's the protocol. We assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{OGen})$ is a semantically secure PKE scheme with an oblivious key-generation algorithm.

Alice		**Bob**
`INPUT:` $1^n, m_0, m_1$		`INPUT:` $1^n, b$
		$(sk_b, pk_b) \leftarrow \mathsf{Gen}(1^n)$, $pk_{1-b} \leftarrow \mathsf{OGen}(1^n)$
	$\xleftarrow{\quad pk_0, pk_1 \quad}$	
	$\xrightarrow{c_0 \leftarrow \mathsf{Enc}(pk_0, m_0), c_1 \leftarrow \mathsf{Enc}(pk_1, m_1)}$	
		`OUTPUT` $\mathsf{Dec}(sk_b, c_b)$

Intuitively, this protocol is secure for Bob because Alice cannot distinguish an obliviously generated key from an honestly generated key. And, the protocol is secure for Alice because, *if Bob behaves honestly*, then pk_{1-b} was sampled obliviously, and it follows that c_{1-b} is indistinguishable from, e.g., an encryption of zero, *even from the perspective of Bob, who of course knows the random coins used to sample* pk_{1-b}.

Of course, this is quite unsatisfying, since we are really really really relying on the assumption that Bob behaves honestly. If he is willing to behave even slightly maliciously, then there is an obvious attack: Bob can just sample both pk_0 and pk_1 from the original Gen algorithm so that he has the secret key for both! Then, he can decrypt both of Alices plaintext m_0, m_1! We will show later how to convert *any* 2PC protocol that is secure in the honest-but-curious model to one that is secure in a fully malicious model. So, we can formally get away with this rather silly construction. We will also see a less silly construction below.

Anyway, let's prove security formally in the honest-but-curious model, even if it's a little unsatisfying.

Theorem 2.3. *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{OGen})$ *is a semantically secure PKE scheme with an oblivious key-generation algorithm, then the above oblivious transfer protocol is secure against honest-but-curious adversaries.*

Proof. We need to show a simulator for Alice S_A and a simulator for Bob S_B.

I'm going to be very pedantic here—more pedantic than I normally am and more pedantic than I expect you to be on the homework—by explicitly describing the views of the two parties and the simulated views. While we normally don't bother to do this, you should always be sure that a proof provides enough information that it should be pretty easy (if tedious) to fill in the remaining detail.

We will do Bob first, since he is more interesting in this case. The view of Bob consists of his input $1^n, b$, the randomness r_b used to run Gen, the randomness r_{1-b} used to run OGen, the resulting keys (sk_b, pk_b) and pk_{1-b} and the ciphertexts c_0, c_1 sent by Alice. The simulator for Bob S_B takes as input 1^n, b, and m_b, samples $(sk_b, pk_b) \leftarrow \mathsf{Gen}(1^n)$ and $pk_{1-b} \leftarrow \mathsf{OGen}(1^n)$. It then sets $c_b \leftarrow \mathsf{Enc}(pk_b, m_b)$ and $c_{1-b} \leftarrow \mathsf{Enc}(pk_{1-b}, 0)$ (there is nothing special about 0 here) and outputs the view $(1^n, b, r_b, r_{1-b}, sk_b, pk_b, pk_{1-b}, c_0, c_1)$.

Clearly S_B is efficient. The fact that the view produced by S_B is indistinguishable from the view of Bob in an honest run of the protocol follows from the obliviousness of OGen. In particular, let $c'_{1-b} \leftarrow \mathsf{Enc}(pk_{1-b}, m_{1-b})$ and $c'_b := c_b$, and suppose that there exists an efficient distinguisher $\mathcal{E}$ such that

$$\varepsilon(n) := \Pr[\mathcal{E}(1^n, b, r_b, r_{1-b}, sk_b, pk_b, pk_{1-b}, c_0, c_1) = 1] - \Pr[\mathcal{E}(1^n, b, r_b, r_{1-b}, sk_b, pk_b, pk_{1-b}, c'_0, c'_1) = 1]$$

is non-negligible. Notice that this is exactly the game of distinguishing the simulated view from a true view. Then, we build an adversary $\mathcal{E}'$ in the obliviousness game that behaves as follows. $\mathcal{E}'$ takes as input randomness r^* (remember that the randomness is included in the obliviousness game!) and a ciphertext $c^* \leftarrow \mathsf{Enc}(pk^*, m^*)$, where $pk^* := \mathsf{OGen}(1^n; r^*)$ and either $m^* = 0$ or $m^* = m_{1-b}$. It sets $r^*_{1-b} := r^*$ and $pk^*_{1-b} := pk^*$. It samples r^*_b itself and sets $(sk^*_b, pk^*_b) := \mathsf{Gen}(1^n; r^*_b)$. It then computes $c^*_b \leftarrow \mathsf{Enc}(pk^*_b, m_b)$ and $c^*_{1-b} := c^*$. Finally, it runs $\mathcal{E}(1^n, b, r^*_b, r^*_{1-b}, sk^*_b, pk^*_b, pk^*_{1-b}, c^*_0, c^*_1)$ and outputs the resulting bit (i.e., it outputs 1 if $\mathcal{E}$ outputs 1 and 0 otherwise).

Clearly, $\mathcal{E}'$ is efficient. Furthermore, when $m^* = 0$, the input to $\mathcal{E}$ is distributed exactly as $(1^n, b, r_b, r_{1-b}, sk_b, pk_b, pk_{1-b}, c_0, c_1)$, and when $m^* = m_{1-b}$, the input to $\mathcal{E}$ is distributed exactly as $(1^n, b, r_b, r_{1-b}, sk_b, pk_b, pk_{1-b}, c'_0, c'_1)$. It follows that the advantage of $\mathcal{E}'$ in the obliviousness game is exactly $\varepsilon(n)$, which contradicts the assumption that OGen was oblivious.

Now, let's do Alice. Alice's view consists of the following: her input $1^n, m_0, m_1$, Bob's message pk_0, pk_1, the randomness s_0, s_1 used to run the encryption algorithm twice, and the ciphertexts $c_0 \leftarrow \mathsf{Enc}(pk_0, m_0; s_0), c_1 \leftarrow \mathsf{Enc}(pk_1, m_1; s_1)$. The simulator for Alice S_A takes as input

$1^n, m_0, m_1$. It then samples $pk_0 \leftarrow \mathsf{OGen}(1^n)$ and $pk_1 \leftarrow \mathsf{OGen}(1^n)$ (we could also use the Gen algorithm—the resulting keys will be indistinguishable by definition—but somehow using the OGen algorithm seems nicer to me). It then samples randomness s_0, s_1 for the encryption algorithm and computes $c_0 \leftarrow \mathsf{Enc}(pk_0, m_0; s_0)$ and $c_1 \leftarrow \mathsf{Enc}(pk_1, m_1; s_1)$. Finally, it produces the view $(1^n, m_0, m_1, pk_0, pk_1, s_0, s_1, c_0, c_1)$.

Clearly the simulator S_A is efficient. To show that the view produced by this simulator is indistinguishable from $\mathsf{View}_A\langle A(1^n, m_0, m_1), B(1^n, b)\rangle$ (for any b), we consider S'_A, which behaves like S_A except it replaces pk_b with a public key sampled using the Gen algorithm. (Of course, S_A does not know b. We are only construction S'_A as part of our analysis, so we are allowing its output to depend on b.)

Notice that S'_A produced a view that is identical to the view of Alice in a true run of the protocol with Bob. Suppose there exists an efficient distinguisher $\mathcal{E}$ such that $\varepsilon(n) := \Pr[\mathcal{E}(1^n, S_A(1^n, m_0, m_1) = 1] - \Pr[\mathcal{E}(1^n, S'_A(1^n, m_0, m_1, b) = 1]$ is non-negligible for some b. Then, we construct $\mathcal{E}'$ in the indistinguishability game against OGen. $\mathcal{E}'$ takes as input 1^n and pk^* which is either sampled by Gen or OGen. It sets $pk_b := pk^*$ and $pk_{1-b} \leftarrow \mathsf{OGen}(1^n)$, computes $c_0 \leftarrow \mathsf{Enc}(pk_0, m_0)$ and $c_1 \leftarrow \mathsf{Enc}(pk_1, m_1)$, creates the resulting view for Alice, and runs $\mathcal{E}$ on it. $\mathcal{E}$ outputs a bit, and $\mathcal{E}'$ outputs the same bit.

Notice that, if pk^* is sampled from Gen, then the view produced by $\mathcal{E}'$ is identical to the view produced by S'_A. If pk^* is sampled from OGen, then the view produced is identical to the view produced by S_A. Therefore, the advantage of $\mathcal{E}'$ in distinguishing these two cases is exactly $\varepsilon(n)$, a contradiction. $\square$

3 2PC for a simple class of functions

Given what we did with FHE, it seems natural to try to build 2PC "one gate at a time." Specifically, it seems natural to first build 2PC for the special case of $\oplus$ and for $\times$. Then, if we want to compute an arbitrary circuit f consisting of $\oplus$ and $\times$ gates, Alice and Bob could work gate-by-gate.

This idea *will* be used in one of our constructions. But, one has to be very careful how to implement it. To see why, suppose that Alice and Bob each have two bits as input $x_{A,1}, x_{A,2}, x_{B,1}, x_{B,2}$, and suppose that they wish to compute, e.g., the function $f(x_{A,1}, x_{A,2}, x_{B,1}, x_{B,2}) = ((x_{A,1} \oplus x_{B,1}) \times (x_{A,2} \oplus x_{A,2}), \perp)$. If, in the process of computing this, either Alice or Bob learns $x_{A,1} \oplus x_{B,1}$, then this will *not* be secure!

More generally, Alice and Bob should only learn the output $f(x_A, x_B)$ of the final computation. They should *not* learn the results of intermediate computation.

Below, we will nevertheless show protocols for $\oplus$ and $\times$. This does not immediately give a protocol for all functions, but these protocols will illustrate the definition a bit and show some of the ideas that we will use going forward. The protocol for $\oplus$ is particularly silly!

3.1 A silly secure protocol for $\oplus$

For bits $x_A, x_B \in \{0, 1\}$, let $f(x_A, x_B) = (\perp, x_A \oplus x_B)$. I.e., a protocol to compute f will result in Alice learning nothing, and Bob learning $x_A \oplus x_B$.

How do we compute this securely? Well, here's the protocol.

Alice		**Bob**
`INPUT:` $1^n, x_A$		`INPUT:` $1^n, x_B$
	$\xrightarrow{\quad x_A \quad}$	
		`OUTPUT` $x_A \oplus x_B$

In other words, Alice just sends her bit to Bob in the clear! That's it.

To see why this is secure, notice that if Bob knows x_B and a protocol teaches him $z := x_A \oplus x_B$, then he can compute $x_A = x_B \oplus z$ himself. In other words, the output of $f_\oplus$ reveals Alice's input to him. So, assuming Alice is happy to let Bob learn $x_A \oplus x_B$, she might as well just send him x_A.

Actually, this fact that $\oplus$ is trivially computable will prove very useful for us later.

3.2 A less silly protocol for $\times$ (i.e., AND)—and more generally, any gate

Finally, let's see a protocol for $f_\times(x_A, x_B) := (\perp, x_A x_B)$, i.e., a protocol in which Bob learns x_A AND x_B and Alice learns nothing. We will need to use OT for this. (In fact, a protocol for AND implies a protocol for OT, and vice versa.) More generally, we will see a protocol for *any* function in which $x_A, x_B \in \{0,1\}$ are single bits.

To compute AND, Bob will simply use his input bit $b := x_B$ as his input bit in an OT protocol. Alice will set $m_0 := 0$ and $m_1 := x_A$. Then, they will run the OT protocol with these inputs.

Notice that Bob's output will be 0 if x_B is 0 and x_A if $x_B = 1$. This is exactly what AND does. In other words, this protocol is correct.

And, security follows immediately from the security of the OT protocol. So, this works.

In fact, there is nothing special about AND here. We can compute any gate, i.e., any function $f = (f_A, f_B)$, where $f_B : \{0,1\}^2 \to \{0,1\}$ and $f_A = \perp$. (This is of course very far from computing arbitrary functions, because these functions only allow Alice and Bob to take single bits as input! We can, however, extend this to allow for $f_A : \{0,1\}^2 \to \{0,1\}$ as well by simply running the protocol twice: once for Bob and once for Alice.) To do so, Alice simply sets $m_0 := f_B(x_A, 0)$ and $m_1 := f_B(x_A, 1)$. In other words, she sets her first plaintext to be the output that Bob should get if his bit x_B is 0 and her second plaintext to be the output that Bob should get if $x_B = 1$. Bob again sets $b := x_B$, and they engage in the OT protocol with these inputs.

Correctness and security are then immediate. In the next two lectures, we will see two very different constructions that can handle arbitrary functions, not just individual gates.

4 A less unsatisfying protocol—Naor-Pinkas

Now that we have seen that oblivious transfer is useful, we will see a different construction that is less unsatisfying (in my opinion).

I will, however, provide essentially no formal justification for preferring this protocol over the ones that we have already seen. I will only show that it satisfies security in the honest-but-curious model, which the above protocols also satisfied. (In fact, I won't even show that since the proof is nearly identical to the above proof.) And, this protocol *does not* satisfy the strongest notion of security against malicious adversaries that we would like, so it's not ideal. Instead, it lies somewhere in between. From my perspective, it is interesting not because of the specific security definition that it achieves, but just because it's not as silly as the oblivious key-generation-based protocol. In particular, there is no obvious way for Bob to cheat. (And, this can be formalized with an explicit security definition showing that in some sense "Bob really cannot cheat in this protocol." But, I will not introduce this definition as it is rather specific and not the "right" definition of security against malicious adversaries.)

Anyway, this protocol is due to Moni Naor and Benny Pinkas [NP01]. I think of it in the following way: consider the oblivious key-generation-based protocol described above instantiated with ElGamal encryption. In that protocol, Bob samples two keys, $A_b = g^a$ and $A_{1-b} \sim G$ and sends A_0, A_1 to Alice. He knows the discrete logarithm of one of these public keys (i.e., $A_b = g^a$), and he super-duper totally promises that he *only* knows the discrete logarithm of one public key and definitely not the other. But, it's a little hard to trust Bob when he claims that he doesn't just know both discrete logarithms.

So, (at least as I think of it), the Naor-Pinkas protocol starts with this idea: "what if we could *force* Bob to choose A_0, A_1 in some way that guarantees that he cannot know both discrete logarithms?"

Actually, there's a relatively easy way to do this. Suppose that we force Bob to choose A_1 such that $A_1 = hA_0$ for some $h \in G$ that *we* choose. Then, "Bob could only know the discrete logarithm of A_0 and A_1 simultaneously if he knew the discrete logarithm of h." In particular, if $A_0 = g^{a_0}$ and $A_1 = g^{a_1}$, then $h = g^{a_1 - a_0}$.

Here is the protocol. It uses as its public parameters some cyclic group G with order p such that DDH is hard over G.

Alice		**Bob**
`INPUT:` $1^n, m_0, m_1 \in G$		`INPUT:` $1^n, d \in \{0,1\}$
$h \sim G$		
		$a \sim \mathbb{Z}_p$
	$\xrightarrow{\quad h \quad}$	
		$A_0 := g^a h^d, A_1 := g^a h^{1-d}$
	$\xleftarrow{\quad A_0, A_1 \quad}$	
Check that $A_1 = hA_0$.		
$b_0, b_1 \sim \mathbb{Z}_p$		
	$\xrightarrow{B_0 := g^{b_0}, C_0 := A_0^{b_0} m_0, B_1 := g^{b_1}, C_1 := A_1^{b_1} m_1}$	
		`OUTPUT` $m_d = B_d^{-a} C_d$

This protocol is secure in the honest-but-curious model for essentially exactly the same reason that the oblivious key-generation-based protocol was secure (using ElGamal as the encryption scheme). Intuitively, Bob chooses A_0 and A_1 so that he knows a such that $g^a = A_d$, but $\log_g A_{1-d} = a \pm \log_g h$, so that "if Bob does not know $\log_g h$, then Bob cannot know $\log_g A_{1-d}$."

A formal proof of Alice's security follows by noting that $(B_{1-d}, C_{1-d}) = (g^{b_{1-d}}, g^{ab_{1-d}} h^{\pm b_{1-d}} m_{1-d})$. Given a and $B_{1-d} = g^{b_{1-d}}$, $D^* := g^{ab_{1-d}}$ is efficiently computable. So, (B_{1-d}, C_{1-d}) is indistinguishable from random if and only if $(B_{1-d}, (D^*)^{-1} C_{1-d}) = (g^{b_{1-d}} h^{\pm b_{1-d}} m_{1-d})$ is indistinguishable from random. But, this is exactly an ElGamal ciphertext with public key h (or h^{-1}, depending on d). So, this is indistinguishable from random. (That was definitely not a formal proof—just a sketch of what one would look like.)

Security for Bob is information-theoretic. Indeed, nothing in Alice's view depends on d at all. In particular, A_0 is a uniformly random group element and $A_1 = hA_0$, regardless of d. So, we can simulate Alice's view perfectly simply by sampling A_0 uniformly at random, setting $A_1 = A_0 h$ (and using Alice's input to compute all other values in the obvious way).

5 A crazy protocol due to Rabin

The original oblivious-transfer paper is due to Michael Rabin [Rab81] (whom we know and love from having studied Rabin's trapdoor function $F_N(x) := x^2 \bmod N$—which uses a similar idea to the construction below). Rabin had an entirely different definition of oblivious transfer, which I personally find to be really rather crazy (in a good way). I therefore want to share it with you :).

In fact, Rabin did not define anything formal at all because this was 1981—i.e., "prehistoric cryptography" before we had formal definitions. Instead, Rabin gave what would now be considered to be an informal description of a primitive as follows: Alice has some plaintext bit $\mu \in \{0, 1\}$. She and Bob then engage in some protocol. At the end of the protocol, with probability 1/2 Bob will output μ, and with probability 1/2 "Bob will not learn anything at all about μ." Furthermore, "Alice does not know whether Bob received μ." This was the primitive that Rabin called "oblivious transfer," and he had a rather elaborate application of such a protocol to a problem that he called "exchange of secrets." To avoid confusion (or, well, to at least lessen confusion), I will refer to Rabin's notion as "blindfolded transfer" and use the phrase "oblivious transfer" to describe the definition that we saw in the previous sections. (Unfortunately, there is *still* some ambiguity about what one means when one says "oblivious transfer." Usually people mean the definition that we have already seen, but sometimes they mean what I am calling blindfolded transfer. Fortunately, one can construct blindfolded transfer from oblivious transfer, and vice versa. So, the ambiguity isn't *too* impactful.)

This definition of blindfolded transfer *can* be made formal. However, that will take us a bit too far afield. (Notice that this definition is *not* captured by our definition of two-party computation. In particular, the functionality in blindfolded transfer is not the functionality $(\perp, \mu)$, nor is it the functionality $(\perp, \perp)$. Instead, it is somehow a "mixture" of both.) Instead, I will simply present a version of Rabin's original protocol without proving anything formal about it at all. I will then show how to convert this protocol into an oblivious transfer protocol (as we defined it above). It should be clear that the conversion process can be made generic—i.e., if we were to formally define blindfolded transfer, then we would be able to convert any blindfolded transfer protocol into an oblivious transfer protocol.

Anyway, here's the protocol. It is crazy! We assume the existence of some semantically secure public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the property that the secret key sk is always a pair of distinct primes $sk := (p, q)$, the public key is always their product $pk := N := pq$, and μ is in the plaintext space. (This can be accomplished, e.g., using Rabin's trapdoor function :)—and particularly the trapdoor permutation variant that we saw for homework. The protocol can also be modified slightly to work with RSA or Goldwasser-Micali.) We also make the very minor assumption that the secret key (q, p) works identically to the secret key (p, q) (i.e., $\mathsf{Dec}((p, q), c) = \mathsf{Dec}((q, p), c)$ for all p, q, c).

Alice		**Bob**
INPUT: $1^n, \mu$		INPUT: 1^n
$(p, q, N) \leftarrow \mathsf{Gen}(1^n)$		
	$\xrightarrow{\quad N \quad}$	
		$x \sim \mathbb{Z}_N^*$, $y := x^2 \bmod N$
	$\xleftarrow{\quad y \quad}$	
Find $x' \in \mathbb{Z}_N^*$ s.t. $(x')^2 = y \bmod N$		
	$\xrightarrow{\quad x' \quad}$	
	$\xrightarrow{c \leftarrow \mathsf{Enc}(N, \mu)}$	
		IF $x' = \pm x \bmod N$, OUTPUT $\perp$ ELSE $\quad p' := \gcd(x - x', N)$ $\quad q' := \gcd(x + x', N)$ $\quad$OUTPUT $\mu := \mathsf{Dec}((p', q'), c)$

Again, we will not prove any formal security properties of this protocol. But, let's get some intuition for what's going on here. First, notice that because Alice knows p, q, Alice can in fact efficiently compute $x' \in \mathbb{Z}_N^*$ such that $(x')^2 = y \bmod N$, using the ideas that we saw when we studied Rabin's trapdoor permutation. So that's good.

Next, there are two cases to consider: either $x' = \pm x \bmod N$ or $x' \neq \pm x \bmod N$. Since there are four square roots of y total, each of these events happens with probability 1/2. (This is true regardless of how Alice chooses to compute x'. From Alice's perspective, since she only knows y, x is a uniformly random element from the set of four square roots of y.) If $x' = \pm x \bmod N$, then Bob is not particularly happy. In particular, Bob already knew that x and $-x$ were square roots of y mod N. So, at least intuitively, in this case Bob "does not learn anything from x' and therefore should not learn anything about μ." On the other hand, if $x' \neq \pm x \bmod N$, then x and x' are a "funny pair of square roots modulo N" like the ones that we studied when we studied Rabin's trapdoor function (see the PKE lecture notes). In particular, we know that $\{\gcd(x - x', N), \gcd(x + x', N)\} = \{p, q\}$. So, in this case, Bob recovers the secret key (p, q) (or, well, either (p, q) or (q, p)), and will of course successfully decrypt $\mu := \mathsf{Enc}(N, c)$ in order to correctly output μ.

On the other hand, "from Alice's perspective these two events look identical." There are exactly four possible square roots of y, and from Alice's perspective, x is equally likely to be any of these square roots. So, "Alice has no idea whether $x' = \pm x \bmod N$."

Again, while the above is certainly not formal, it can be made formal. (E.g., one can use the real/ideal paradigm to define a formal definition that captures this idea and then prove that the above scheme does satisfy this definition, under the assumption that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is semantically secure.)

5.1 From blindfolded transfer to oblivious transfer (sorry for the terrible names)

Now, let's see how to convert Rabin's protocol for blindfolded transfer into a good old-fashioned oblivious transfer protocol. We will then actually formally prove security of this protocol.

Here is the protocol.

Alice		**Bob**
`INPUT:` $1^n, m_0, m_1$		`INPUT:` $1^n, b$
`FOR` $i = 1\ldots,n$ $\quad(p_i, q_i, N_i) \leftarrow \mathsf{Gen}(1^n)$		
	$\xrightarrow{N_1,\ldots,N_n}$	
		$x_1 \sim \mathbb{Z}^*_{N_1},\ldots,x_n \sim \mathbb{Z}^*_{N_n}$ $y_i := x_i^2 \bmod N_i$
	$\xleftarrow{y_1,\ldots,y_n}$	
$x'_1 \in \mathbb{Z}^*_{N_1},\ldots,x'_n \in \mathbb{Z}^*_{N_n}$ $\quad$s.t. $(x'_i)^2 = y_i \bmod N_i$		
	$\xrightarrow{x'_1,\ldots,x'_n}$	
		Sample i_0, i_1 randomly $\quad$with $x'_{i_b} \neq \pm x_{i_b} \bmod N_{i_b}$ $\quad$and $x'_{i_{1-b}} = \pm x_{i_{1-b}} \bmod N_{i_{1-b}}$ (If no such i_0, i_1 exist, start over.)
	$\xleftarrow{i_0, i_1}$	
$\mu_0, \mu_1 \sim \{0,1\}$		
	$\xrightarrow{c_0 \leftarrow \mathsf{Enc}(N_{i_0}, \mu_0), c_1 \leftarrow \mathsf{Enc}(N_{i_1}, \mu_1)}$	
	$\xrightarrow{d_0 := \mu_0 \oplus m_0, d_1 := \mu_1 \oplus m_1}$	
		$p' := \gcd(x_{i_b} - x'_{i_b}, N_{i_b})$ $q' := \gcd(x_{i_b} + x'_{i_b}, N_{i_b})$, $\mu_b := \mathsf{Dec}((p', q'), c_b)$ `OUTPUT` $d_b \oplus \mu_b$

In other words, Alice and Bob first run n of these blindfolded transfer protocols in parallel, so that at the end of these many protocols, Bob presumably knows the factorization of many of the integer N_i but does not know the factorization of others. (With probability 2^{1-n}, he will either know all factorizations or no factorizations, in which case Bob tells Alice that he would like to start over. This happens very rarely, so the protocol is efficient even though there is some small chance that Alice and Bob must start over.) Bob will then ask Alice to encrypt m_0 and m_1 under two public keys N_{i_0} and N_{i_1}, with the property that Bob knows the factorization of N_{i_b} but not $N_{i_{1-b}}$.

Anyway, let's prove security.

Theorem 5.1. *Assuming that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure, the above protocol is secure (against honest-but-curious adversaries).*

Proof. We need to construct a simulator S_A for Alice and a simulator S_B for Bob. For simplicity, here we will ignore unlikely event that the protocol must start over.

The simulator for Alice is relatively straightforward S_A. It takes as input $1^n, m_0, m_1$, and samples $(p_i, q_i, N_i) \leftarrow \mathsf{Gen}(1^n)$ for $i = 1,\ldots,N$. It then samples $x'_1 \sim \mathbb{Z}^*_{N_1},\ldots,x'_n \sim \mathbb{Z}^*_{N_n}$ and sets $y_i := (x'_i)^2 \bmod N$. Then, it samples two uniformly random distinct indices i_0, i_1, samples $\mu_0, \mu_1 \sim \{0,1\}$, and sets $c_0 \leftarrow \mathsf{Enc}(N_{i_0}, \mu_0), c_1 \leftarrow \mathsf{Enc}(N_{i_1}, \mu_1)$ and $d_0 := \mu_0 \oplus m_0, d_1 := \mu_1 \oplus m_1$. Finally, it outputs the resulting view.

The output of S_A is actually distributed identically to the view that Alice sees in a true run of the protocol. In particular, the y_i, x'_i are distributed identically in both cases. And, in both cases, i_0, i_1 are independent of the y_i and the x'_i.

The simulator for Bob S_B behaves as follows. It takes as input $1^n, b, m_b$, and samples $(p_i, q_i, N_i) \leftarrow \mathsf{Gen}(1^n)$ for $i = 1, \ldots, n$. (Of course, the p_i, q_i will not be part of the view output by S_B.) It samples $x_1 \sim \mathbb{Z}^*_{N_1}, \ldots, x_n \sim \mathbb{Z}^*_{N_n}$, sets $y_i := x_i^2 \bmod N_i$, and computes square roots $x'_1 \in \mathbb{Z}^*_{N_1}, \ldots, x'_n \in \mathbb{Z}^*_{N_n}$ with $(x_i)'^2 = y_i \bmod N_i$, using whatever process Alice uses to do this.

It then samples i_0 and i_1 uniformly at random subject to the constraint that $x_{i_b} \neq \pm x'_{i_b} \bmod N_{i_b}$ and $x'_{i_{1-b}} = \pm x'_{i_{1-b}} \bmod N_{i_{1-b}}$.

Finally, it samples $\mu_b \sim \{0,1\}$, sets $c_b \leftarrow \mathsf{Enc}(N_{i_0}, \mu_b)$ and $c_{1-b} \leftarrow \mathsf{Enc}(N_{i_1}, \mu_{1-1})$, $d_b := \mu_b \oplus m_b$ and $d_{1-b} \sim \{0,1\}$. Finally, it computes p', q' like Bob and outputs the view consisting of

$$(b, N_1, \ldots, N_n, x_1, \ldots, x_n, y_1, \ldots, y_n, x'_1, \ldots, x'_n, i_0, i_1, c_0, c_1, d_0, d_1, p', q', \mu_b, m_b) \ .$$

Intuitively, the output of S_B is indistinguishable from a real view because it only differs from the real view in c_{1-b}. So, the assumed semantic security of the encryption scheme should imply that the two views are computationally indistinguishable. We now prove this formally via reduction.

Suppose there exists some adversary $\mathcal{E}$ and bits b, m_0, m_1 such that

$$\Pr[\mathcal{E}(1^n, S_B(1^n, b, m_b)) = 1] - \Pr[\mathcal{E}(1^n, \mathsf{View}_B\langle A(1^n, m_0, m_1), B(1^n, b)\rangle) = 1] = \varepsilon(n)$$

for non-negligible $\varepsilon(n)$. We construct an adversary $\mathcal{E}'$ in the semantic security game against $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows. $\mathcal{E}'$ takes as input a public key N^*. It *first* samples uniformly random *distinct* indices i_0, i_1. It sets $N_{i_{1-b}} := N^*$, and for all $j \neq i_{1-b}$ (including $j = i_b$), it samples $(p_j, q_j, N_j) \leftarrow \mathsf{Gen}(1^n)$.

For $j \notin \{i_0, i_1\}$, it samples $x_j \sim \mathbb{Z}^*_{N_j}$, sets $y_j := x_j^2 \bmod N_j$, and (using p_j, q_j) computes $x'_j \in \mathbb{Z}^*_{N_j}$ such that $(x'_j)^2 = y_j \bmod N_j$. It also samples $x_{i_b} \sim \mathbb{Z}^*_{N_{i_b}}$, sets $y_{i_b} := x_{i_b}^2 \bmod N_{i_b}$, and samples random $x'_{i_b} \in \mathbb{Z}^*_N$ such that $(x'_{i_b})^2 = y_{i_b} \bmod N_{i_b}$ *with* $x'_{i_b} \neq \pm x_{i_b} \bmod N_{i_b}$ (using p_{i_b}, q_{i_b}). Then, it samples $x_{i_{1-b}} \sim \mathbb{Z}^*_{N_{i_{1-b}}}$ and $x'_{i_{1-b}} \sim \{\pm x_{i_{1-b}}$. (Notice that this last step can be done using just the public key $N_{i_{1-b}} = N^*$. This captures the intuitive idea that "Bob does not the factorization of $N_{i_{1-b}}$.")

$\mathcal{E}'$ then samples $\mu_b \sim \{0,1\}$, sets $c_b \leftarrow \mathsf{Enc}(N_{i_b}, \mu_b)$, and $d_b := \mu_b \oplus m_b$. It samples $d_{1-b} \sim \{0,1\}$, sets $m_0^* := d_{1-b} \oplus m_{1-b}$ and $m_1^* := d_{1-b}$, and sends (m_0^*, m_1^*) to its challenger, receiving in response $c^* \leftarrow \mathsf{Enc}(N^*, m^*_{b^*})$ for $b^* \sim \{0,1\}$.

Finally, $\mathcal{E}'$ sets $c_{1-b} := c^*$, calls $\mathcal{E}$ on the resulting view of Bob, and outputs whatever bit b' that $\mathcal{E}$ outputs.

Clearly $\mathcal{E}'$ is efficient (though it does do a lot of pretty tedious stuff!). Furthermore, when $b^* = 0$, the input to $\mathcal{E}$ is distributed identically to $\mathsf{View}_B\langle A(1^n, m_0, m_1), B(1^n, b)\rangle$. On the other hand, when $b^* = 1$, this is distributed identically to $S_B(1^n, b, m_b)$. It follows that

$$\begin{aligned}\Pr[b' = b^*] &= \Pr[b' = 1 \mid b^* = 1]/2 + \Pr[b' = 0 \mid b^* = 0]/2 \\ &= \Pr[\mathcal{E}(1^n, \mathsf{View}_B\langle A(1^n, m_0, m_1), B(1^n, b)\rangle) = 1]/2 + 1/2 - \Pr[\mathcal{E}(1^n, S_B(1^n, b, m_b)) = 1]/2 \\ &= 1/2 + \varepsilon(n)/2 \ .\end{aligned}$$

This is a contradiction, so the two views must be indistinguishable, as needed. □

Of course, the above protocol has the same "unsatisfying problem" that our original protocol based on oblivious key generation had: there is an obvious way for Bob to cheat. In particular, instead of choosing i_{1-b} such that $x_{i_{1-b}} = \pm x_{i'_{1-b}} \bmod N$, a malicious actor could just choose a different i_{1-b}, allowing it to decrypt both c_1 and c_2.

This can be largely mitigated by, e.g., having Bob choose two disjoint lists of indices $i_{0,1}, \ldots, i_{0,n/3}$ and $i_{1,1}, \ldots, i_{1,n/3}$. Alice can then sample pads for each index, $\mu_{0,1}, \ldots, \mu_{0,n/3} \sim \{0,1\}$ and $\mu_{1,1}, \ldots, \mu_{1,n/3} \sim \{0,1\}$, encrypt each pad $\mu_{c,j}$ under the key $N_{i_{c,j}}$, and set $d_0 := \mu_{0,1} \oplus \cdots \oplus \mu_{0,n/3} \oplus m_0$ and $d_1 := \mu_{1,1} \oplus \cdots \oplus \mu_{1,n/3} \oplus m_1$.

Then, intuitively, "for Bob to learn both m_0 and m_1, he would need to know $2n/3$ distinct secret keys, which happens with probability less than $2^{-n/100}$." (Again, we do not bother to formalize this because this modified protocol will still not satisfy the "correct" definition of security against malicious adversaries.)

References

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA*, 2001. 8

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical report, Harvard University, 1981. https://eprint.iacr.org/2005/187. 9