In the last lecture we gave a definition of secure two-party computation for honest-but-curious adversaries, and showed a secure protocol for oblivious transfer (assuming the existence of enhanced trapdoor permutations). We now sketch how to implement a protocol for general functionalities.

1 Secure two-party computation

Let us now consider an arbitrary, but not too large functionality (f, g). By "not too large", we mean that Alice's and Bob's inputs come from some set $\{1, \ldots, N\}$, where N is reasonable (so computations in time N or N^2 are tractable for Alice and Bob). For example, we can think of the functionality $g(x, y) = 1$ if $x < y$ and 0 otherwise, but where x and y are known not to be too large, say between 1 and $10,000$.

Without loss of generality, we will only consider functionalities of the form $(\perp, g)$ – namely, Alice obtains no input at the end of the interaction. The reason is that we can obtain an $(s, 2\varepsilon)$ secure protocol for (f, g) by running an (s, ε) secure protocol for $(\perp, g)$ followed by an (s, ε) secure protocol for $(f, \perp)$. (I'll leave the proof as an exercise.)

Consider the following protocol for $(\perp, g)$: Alice and Bob get inputs x and y from $\{1, \ldots, N\}$ and run the N-oblivious transfer protocol on inputs $(g(x, 1), \ldots, g(x, N))$ (for Alice) and y (for Bob).

This protocol is clearly functional. Its security follows directly from the security of N-oblivious transfer: Since Alice's view and Bob's view are identical as in the oblivious transfer protocol, the simulators for oblivious transfer can be used directly. Namely, on input $(x, \perp)$, Alice runs her oblivious transfer simulator on input $(g(x, 1), \ldots, g(x, N), \perp)$, and Bob runs exactly the same simulator as for oblivious transfer.

But what happens if the domain of g is large – say x and y take values in $\{0, 1\}^n$? It turns out there is a protocol whose running time is proportional to the *circuit size* of g.

To explain the protocol we need to recall a bit about circuits. Recall that a circuit $C\colon \{0, 1\}^n \to \{0, 1\}$ of size s consists of n gates consisting of the input gates $x_1, \ldots, x_n$ and $s - n$ internal gates $G_{n+1}, \ldots, G_s$ (some of which are marked as output gates). Each internal gate G_k is labeled by either AND, OR, or NOT and takes two inputs i and j $(i, j < k)$ where i and j are the indices of some previous internal gates. The computation of the circuit proceeds by propagating the values of the gates in increasing order, until the outputs are obtained.

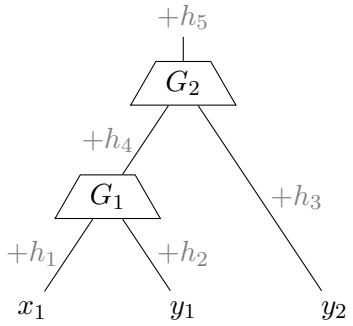
The protocol we describe is asymmetric. Alice will hide the value of each wire in the circuit by masking it with a random bit of her choice. Bob will learn the masked values of the wires in the circuit one by one by "securely evaluating" the circuit starting at the input gates all the way to the output gate. To do so, for each gate G Alice and Bob engage in an oblivious transfer protocol that reveals to Bob the masked value of the output of G given that he knows the masked values of the inputs of G. At the end of the protocol, Alice merely reveals the randomness corresponding to the output gate(s) of C' – which allows Bob to recover the values at this gate without learning anything else.

The two-party protocol On inputs $x \in \{0,1\}^n$ (for Alice) and $y \in \{0,1\}^n$ (for Bob):

- A: For each gate k in the circuit, choose a random bit $h_k \sim \{0,1\}$. Send the values $x_1 + h_1, \ldots x_n + h_n$ (for Alice's input gates) and the values $h_{n+1}, \ldots, h_{2n}$ (for Bob's input gates).
- B: Upon receiving $u_1, \ldots, u_n, h_{n+1}, \ldots, h_{2n}$, set $u_{n+k} = y_k + h_{n+k}$ for k from 1 to n.
- A, B: For k from $2n+1$ to s, run the 4-oblivious transfer protocol on inputs $(w_{00}w_{01}w_{10}w_{11}, u_i u_j)$, where $w_{ab} = G_k(a + h_i, b + h_j) + h_k$, $ab \in \{0,1\}^2$, and i and j are the inputs to gate k. Bob sets u_k to equal the output of the oblivious transfer (namely, $u_k = G_k(u_i + h_i, u_j + h_j) + h_k$).
- A: Send the values h_o for every output gate o.
- B: Output the values $u_o + h_o$.

This protocol is clearly functional: After running the protocol for gate i, Bob has computed the masked value $u_i = G_i + h_i$ corresponding to the output of gate i. In the last step, Bob outputs $u_o + h_o = G_o$, the value at the output gate of the circuit.

Instead of proving the security of this protocol in full generality, I think it is more instructive to do an example. Suppose Alice and Bob want to jointly compute the following functionality:



Before we argue security, let's introduce some notation. Let $\text{view}_A^{OT_i}, \text{view}_B^{OT_i}$ denote the views of Alice and Bob, respectively, when executing the oblivious transfer protocol for gate G_i, and let $S_A^{OT_i}$ and $S_B^{OT_i}$ denote the output of the simulator in these two protocols.

The views of Alice and Bob in the two-party protocol on inputs x, y are

$$\text{view}_A(x,y) = (h_1, h_2, h_3, h_4, h_5, \text{view}_A^{OT_1}, \text{view}_A^{OT_2})$$
$$\text{view}_B(x,y) = (x_1 + h_1, y_1 + h_2, y_3 + h_3, \text{view}_B^{OT_1}, G_1 + h_4, \text{view}_B^{OT_2}, G_2 + h_5, h_5).$$

(For simplicity we omit the inputs to $\text{view}_A^{OT_i}$ and $\text{view}_B^{OT_i}$.) We now show how to simulate Alice's view. Recall that Alice's input in OT_1 are the four values $w_{1,ab} = G_1(a + h_1, b + h_2)$, and her inputs in OT_2 are the values $w_{2,ab} = G_2(a + h_4, b + h_3)$. So it is reasonable to replace Alice's views in OT_1 and OT_2 with the corresponding simulations:

$$S_A(x, \perp) = (h_1, h_2, h_3, h_4, h_5, S_A^{OT_1}(w_{1,ab}), S_A^{OT_2}(w_{2,ab})).$$

We now sketch the argument that S_A and view_A are computationally indistinguishable. (We will omit Alice's simulator input $(x, \perp)$ for simplicity; besides, it is not used in the simulation.) To do so consider the hybrid distribution

$$H_A = (h_1, h_2, h_3, h_4, h_5, \text{view}_A^{OT_1}, S_A^{OT_2}(w_{2,ab})).$$

If S_A and view_A are (s, ε) computationally distinguishable, then either (S_A, H_A) or (H_A, view_A) are $(s, \varepsilon/2)$ computationally distinguishable. In the first case, we can distinguish between $S_A^{OT_1}(w_{1,ab})$ and $\text{view}_A^{OT_1}$ by a distinguisher $D_{x,y}(T)$ that works like this: First, produce a "distinguishing view" for Alice by running the protocol except that the messages in OT_1 are replaced by T and the messages in OT_2 come from $S_A^{OT_2}(w_{2,ab})$, then invoke the distinguisher for (S_A, H_A). In the second case, we can distinguish between $S_A^{OT_2}(w_{2,ab})$ and $\text{view}_A^{OT_2}$ by a distinguisher $D'_{x,y}(T)$ that produces a "distinguishing view" for Alice by running the protocol, with $\text{view}_A^{OT_1}$ in the first round, and the messages coming from T used in the second round, then invoking the distinguisher for (H_A, view_A).

Now let's argue that Bob's view can also be simulated. Recall that Bob's input in OT_1 are the values $u_1 = x_1 + h_1, u_2 = y_1 + h_2$ and his output is $u_4 = G_1 + h_4$. Bob's input in OT_2 are the values $u_4 = G_1 + h_4, u_3 = y_2 + h_3$ and his output is $u_5 = G_2 + h_5$. So it makes sense for Bob to try and carry out the following simulation:

$$(x_1 + h_1, y_1 + h_2, y_3 + h_3, S_B^{OT_1}(x_1 + h_1, y_1 + h_2), G_1 + h_4, S_B^{OT_2}(G_1 + h_4, y_2 + h_3), G_2 + h_5, h_5).$$

The problem is that Bob does not know some of the values here, like x_1 and G_1. However, since these values are masked by independent random bits, Bob can pretend that the unknown values are random without affecting the distribution of the simulated view. This suggests the following simulation:

$$S_B(y, G_2(x, y)) = (u_1, u_2, u_3, S_B^{OT_1}(u_1, u_2), u_4, S_B^{OT_2}(u_4, u_3), u_5, u_5 + G_2)$$

These two distributions are identical, but now Bob does not need to know the values at the various gates to carry out the simulation. The only issue is the last value h_5, which is correlated with $G_2 + h_5$. Bob can simulate this value since he knows the output value G_2 of the circuit.

We will omit the argument that $S_B(y, G_2(x, y))$ and $\text{view}_B(x, y)$ are computationally indistinguishable. It is essentially identical to the argument we used for Alice.

I hope it is plausible that this argument can be generalized to arbitrary circuits. However, the proof appears to be quite tedious. See Goldreich (Section 7.3 of Volume 2) for the general proof (of a slightly different protocol).

2 Malicious adversaries: A preview

As we noticed last time, one deficiency of the secure two-party computation protocols we described (specifically, the oblivious transfer protocol) is that they only work if the adversary behaves as prescribed. If Bob deviates from his instructions in the protocol, then he can learn Alice's input.

To achieve security beyond honest-but-curious adversaries, we need to design our protocols to be resilient against scenarios where Alice and Bob behave in arbitrary malicious ways. This seems like a hopeless task, as there are infinitely many "bad" ways in which the parties in a protocol can behave. How should we go about handling all of them?

Perhaps a natural way to start would be to revisit the oblivious transfer protocol and try to fix it so it becomes resilient even under malicious attacks. However, it turns out that the problem is easier to solve if we aim higher. Instead of improving a specific protocol like oblivious transfer, we will describe a general methodology that takes *any* two-party computation protocol secure against honest-but-curious adversaries and "compiles" it into a protocol that is secure against malicious adversaries. The ingenious idea of Goldreich, Micali, and Wigderson is to take the protocol for honest-but-curious parties and have arbitrary (malicious) parties act out the protocol, while proving to each other that they are behaving honestly!

This sounds paradoxical: We usually think of honesty as a virtue that has to be taken on faith and cannot be proven. For example, when I issue a take-home exam and ask you not to look up the solutions, you have no way of proving to me that you behaved honestly. (In contrast, you can easily prove to me that you were *dishonest* by showing me where you copied your solutions from.) If we cannot prove honesty in real life, how can we expect to do so in two-party protocols?

An important observation is that once Alice's and Bob's inputs and their internal randomness in the protocols is fixed, the protocol is deterministic. If we forgot about privacy for a moment, an easy way to ensure the parties behave honestly is to run the protocol together: Before starting the protocol, the parties share their inputs and their internal randomness, and each party can simulate the other's view to check that it has behaved honestly.

However, revealing any information about inputs and internal randomness is antithetical to privacy. So instead of revealing the inputs and their randomness to one another as a plaintext, they will give each other the required information in encoded form. The encoding must not reveal anything about the original information, but should be sufficient to verify that the parties behave honestly. This encoding will be obtained using a cryptographic primitive called a *commitment scheme*. This is the digital analogue of sending an item in a locked box: The receiver has no idea what the item is, but should the need arise to reveal the item, the sender can provide a key and the receiver can unlock the box and retrieve the item.

Once Alice has an encoding of Bob's input and randomness, how can she verify that Bob is behaving honestly? To do so, Bob must convince Alice that he is following his prescribed instructions using his input and his internal randomness, which Alice has an encoding of. However, in order to preserve privacy, his "argument" to alice must not reveal anything about his input and randomness. In other words, Bob has to convince Alice that his actions are consistent with Alice's encoding of Bob's private information, without revealing to her what that private information is! The techonlogy we will use to achieve this seemingly paradoxical requirement is called a *zero-knowledge proof*. A zero-knowledge proof allows us to verify a statement (in this case, the statement "my actions in the protocol are consistent with the private information whose encoding you have") without revealing any information beyond the validity of this statement!

Even after committing to his private information and applying zero-knowledge to certify he is behaving honestly, there is one more way in which Bob can cheat in his execution of the protocol. This has to do with his choice of internal randomness. There is no guarantee that the randomness used by Bob comes from a truly random source and that it is independent of other information like his input. How can Alice ensure that the randomness used by Bob indeed consists of truly random independent bits? One possibility is for Alice to supply Bob with his random bits; however, Bob's bits must also remain private, so this is inadequate. The solution is for Alice and Bob to engage in a *(random) coin tossing* protocol. This is a protocol for computing the two-party functionality $(\perp, r)$, where $r \in \{0,1\}^n$ is a uniformly random string.

To summarize, the transformation from security against honest-but-curious adversaries to security against malicious (active) adversaries works by combining three building blocks: commitment schemes, zero-knowledge proofs, and a coin tossing protocol. These building blocks must themselves ensure security even against malicious adversaries. We now describe the security requirements and give candidate implementations of these protocols.

3 Commitment schemes

A *commitment scheme* is the digital analogue of the following two-phase process. Initially, Bob holds an item b that Alice does not know. In the first phase, Bob puts b in a box, locks the box, and sends the locked box to Alice. The box is opaque so Alice cannot see what is inside. However, once Alice receives the box, Bob cannot change its contents. This is called the *commitment phase*.

In the second phase, Bob sends the key of the lock to Alice. Alice opens the box and inspects its contents. This is called the *revealment phase*.

Informally, we expect a commitment scheme to have the following two properties:

- **Hiding:** The commitments of different items appear indistinguishable to Alice.
- **Binding:** The contents of the locked box cannot be changed. In particular, Bob cannot change what he put in the box by sending a different key to Alice in the revealment phase.

For simplicity we will start with one-bit commitment schemes. Here, Bob's input is a single bit $b \in \{0, 1\}$. We will formalize the hiding property by an indistinguishability requirement: The commitments of bit 0 and bit 1 appear identical to Alice.

One thing to remember is that commitment is not an end in itself – at the end, both Alice and Bob get Bob's private input – but a tool for enforcing honest behavior on Bob's part. In our application to secure two-party computation, the revealment phase will never be carried out in the protocol; we will merely use the *existence* of revealments to ensure that Bob behaves consistently throughout the protocol.

Definition 1. An (s, ε) secure *one-bit commitment scheme* with key length k is a pair of algorithms (Com, Rev) with the following properties:

- **Functionality:** For every K and b, $Rev(K, Com(K, b)) = b$.
- **Security - hiding:** The distributions $Com(K, 0)$ and $Com(K, 1)$ are (s, ε) computationally indistinguishable, where K is chosen at random from $\{0, 1\}^k$.
- **Security - binding:** For all K, K', and b, $Rev(K', Com(K, b)) \in \{b, \texttt{error}\}$.

The hiding requirement says that commitments of different bits look indistinguishable to Alice. The binding requirement says that a given commitment cannot be opened in more than one way.[1]

[1] There is an alternative definition that allows for a commitment to be opened in two ways, but makes it computationally infeasible to produce a commitment with two keys that give different openings.

We now show a construction of a commitment scheme based on a one-way permutation. Let $f\colon \{0,1\}^k \to \{0,1\}^k$ be a permutation and $h\colon \{0,1\}^k \to \{0,1\}$ be a hardcore bit for f. Consider the following scheme:

$$Com(K,b) = (f(K), h(K)+b) \qquad Rev(K,C,L) = \begin{cases} L + h(K), & \text{if } C = f(K) \\ \texttt{error}, & \text{otherwise.} \end{cases}$$

Theorem 2. *If h is an (s,ε) hardcore bit for f, then (Com, Rev) is an $(s - O(1), 2\varepsilon)$ secure one-bit commitment scheme.*

Proof. The functionality is straightforward. The binding property follows from the fact that f is a permutation. The hiding property follows from the usual argument: If $(f(K), h(K)+0)$ and $(f(K), h(K)+1)$ are $(s, 2\varepsilon)$ distinguishable, then at least one of them is (s,ε) distinguishable from $(f(K), B)$, where B is a random bit, and so h is not an $(s - O(1), \varepsilon)$ hardcore bit for f. □

To obtain a commitment scheme for more than one-bit messages, one solution is to commit to each bit in the message separately (using independent keys). There are also more efficient schemes that use shorter keys.

4 Coin flipping

We now describe a protocol for Alice and Bob to agree on a common random bit. In the honest-but-curious adversary model, this problem is trivial: Alice can choose a random bit and send it to Bob. When Alice and Bob are malicious, this does not work: Instead of selecting a random bit, Alice can choose a bit that is favorable to her and affect the bias of the output.

To ensure that the output is random, it seems that we would need to make the outcome dependent on both Alice and Bob. Here is one idea: Alice chooses a random bit r_A, Bob chooses a random bit r_B, and they output $r_A + r_B$. Then even if one of the parties tries to cheat, as long as the other party plays fairly the outcome will be truly random.

The problem is that this protocol cannot be realized in our communication model. When Alice and Bob communicate, one of them has to go first. But if Alice sends her bit r_A first, then Bob can make his choice r_B dependent of r_A – say if he wants the outcome to be zero, he can set $r_B = r_A$.

It turns out that a simultaneous exchange of messages can essentially be simulated in our usual (asynchronous) communication model with the help of commitments. Before showing how commitments help, let us give a definition of the task we want to achieve: coin flipping with malicious parties.

The functionality requirement is straightforward. To formulate it, we need to extend our definition of "two-party computation" from last lecture to allow for computing randomized functionalities. A randomized functionality is a pair of functions $f(r,x,y)$ and $g(r,x,y)$, when in addition to the inputs x and y provided to Alice and Bob respectively, f and g also take in a random string r. We say that an interactive protocol (A,B) is a *two-party computation* of (f,g) if for every pair of strings a and b,

$$\Pr[A(x) = s \text{ and } B(y) = b] = \Pr[f(r,x,y) = a \text{ and } g(r,x,y) = b].$$

Under this definition, a *coin flipping protocol* is simply a two-party computation for the functionality $f(r, x, y) = g(r, x, y) = r$, where $r \sim \{0, 1\}$ is a random bit.

Definition of security We now want to define security. For now let's not worry about secrecy. However, we want to say that even if one of the parties does not play by the rules of the protocol, he or she cannot affect the randomness of the outcome. To define this, we need to consider what happens in the protocol when the honest parties A and B are replaced with malicious parties A^* and B^* that may deviate from the instructions in the protocol.

Definition 3. Let (A, B) be a coin-flipping protocol in which Alice and Bob exchange k messages. We say that (A, B) is (s, ε) *secure against malicious adversaries* if

- For every interactive algorithm A^* of circuit size s that participates in k rounds of interaction, in the interactive protocol (A^*, B):
$$\big|\Pr[B = 1] - \Pr[R = 1]\big| \leq \varepsilon$$
- For every interactive algorithm B^* of circuit size s that participates in k rounds of interaction, in the interactive protocol (A, B^*):
$$\big|\Pr[A = 1] - \Pr[R = 1]\big| \leq \varepsilon$$

where $R \sim \{0, 1\}$ is a uniformly random bit.

The first condition says that even if Alice tries to deviate from the protocol, she can only affect the bias of the outcome by at most ε. The second condition imposes the same requirement on Bob. So as long as one of the parties behaves honestly, if ε is small, the other one does not have to gain much by deviating from the protocol.

The definition gives no guarantee on the outcome of the protocol if both parties are malicious. It is easy to see that in such a case no guarantee can be given: If both parties behave deterministically then the outcome can never be a random coin flip.

(One issue I have with this definition is that from a game-theoretic point of view, it does not really provide an incentive to either party to behave honestly. If Alice can gain an advantage of ε by behaving maliciously, then she should change her behavior in order to gain that ε, and the same goes for Bob. But in that case neither of the parties ends up playing by the rules and there is no guarantee on the outcome. It would be nice if one could also provide some sort of game-theoretic guarantee that says Alice could not gain more than ε by switching strategies, regardless of Bob's strategy, and vice-versa. However I don't know if such a definition is achievable.)

One type of attack that is not captured by this definition is the following: If after seeing the first few messages from Alice, Bob does not his odds, one thing he could do in real life is stop playing (participating in the protocol) altogether. In that case no output is produced. One possibility is to run the protocol again and again until an output is obtained, but in that case the output may be biased towards one of the parties. It is known that for two-party computations, this bias cannot be avoided, but it can be made smaller by increasing the number of rounds of interaction. Another possibility is to penalize the party that aborts the protocol.

In our definition, we avoid the abortion problem altogether by requiring that the malicious parties A^* and B^* participate in at least k rounds of interaction.

The coin flipping protocol We now describe the coin flipping protocol. It makes use of a commitment scheme (Com, Rev).

A: Choose a random bit $r_A \in \{0,1\}$, a random $K \in \{0,1\}^k$ and send $Com(K, r_A)$.

B: Upon receiving a commitment C, choose and send a random bit $r_B \in \{0,1\}$.

A: Upon receiving the bit b, send the key K and output $r_A + b$.

B: Upon receiving key K', output $Rev(K', C) + r_B$, if $Rev(K', C) \neq \texttt{error}$, and r_B otherwise.

If both Alice and Bob behave honestly, then the outcome of this protocol for both of them is the sum $r_A + r_B$, which is a uniformly random bit. We now argue that the protocol is secure against malicious adversaries even if the parties are not honest. Let's start with the easier case when Alice is dishonest.

Claim 4. *For every A^* that participates in three rounds of interaction,* $\Pr[B = 1] = 1/2$.

Proof. In the last step of the protocol, Bob outputs either r_B if $Rev(K', C) = \texttt{error}$, and $r_B + r_A$ otherwise. Since r_A is independent of r_B, in either case the output of Bob is a uniformly random bit. □

Claim 5. *Assume (Com, Rev) is an $(s, 2\varepsilon)$ secure commitment scheme. Then for every B^* of size at most s that participates in three rounds of interaction,* $|\Pr[A = 1] - \Pr[R = 1]| \leq \varepsilon$.

Proof. We argue by contradiction: We will show that if $|\Pr[A = 1] - \Pr[R = 1]| > \varepsilon$, then (Com, Rev) is not $(s, 2\varepsilon)$ secure. Specifically, we will show that $Com(K, 0)$ and $Com(K, 1)$ can be distinguished within ε by a circuit of size s.

In the first round of the interaction, the malicious B^* takes as input $Com(K, r_A)$ and produces as output a (possibly randomized) bit $b(Com(K, r_A))$. This b is computed by a circuit of size at most s. By our assumption,

$$|\Pr[r_A + b(Com(K, r_A)) = 1] - \Pr[R = 1]| > \varepsilon.$$

Conditioning on $r_A = 0$ and $r_A = 1$, we obtain

$$|\tfrac{1}{2}\Pr[b(Com(K, 0)) = 1] + \tfrac{1}{2}\Pr[b(Com(K, 1)) = 0] - \Pr[R = 1]| > \varepsilon.$$

Since $\Pr[b(Com(K, 1)) = 0] = 1 - \Pr[b(Com(K, 1)) = 1]$ and $\Pr[R = 1] = 1/2$, we obtain

$$|\Pr[b(Com(K, 0)) = 1] - \Pr[b(Com(K, 1)) = 1]| > 2\varepsilon$$

contradicting the assumption that (Com, Rec) is (s, ε) secure. □