In the first half of the course we covered the basic cryptographic primitives that enable secure communication in insecure environments. These are the foundations on which the security of much modern computing infrastructure including e-mail, messaging, e-commerce, and banking is based.

Sometimes the purpose of communication is not to exchange information but to perform some joint computation on private inputs. For example, several hospitals might want to calculate some joint statistic on their data without revealing information about their individual patients, or a potential buyer may want to convince a real-estate agent that she has sufficient funds to purchase a property without revealing her net worth, or a group of voters want to elect a leader by majority without revealing their individual preferences.

We describe a general framework that in principle solves any question of this type. In this lecture we talk about the two-party setting.

1 Secure two-party computation

A two-party computation is a protocol by which Alice and Bob, which hold private inputs x and y, respectively, jointly compute some function $f(x, y)$. Alice and Bob can certainly do this by sharing their inputs and computing f together, but this would reveal their private inputs to one another, which might be undesirable.

One illustrative example is Yao's millionaire problem: Alice and Bob want to know who is the richer one among them but they do not want to reveal their individual net worths x and y. They could do this by submitting their respective inputs x and y to a trusted referee Charlie who would compare the numbers and declare the winner. The objective of a secure two-party computation is for Alice and Bob to emulate this interaction with a trusted Charlie completely on their own.

It will be somewhat easier to start off with an asymmetric definition of functionality in which Bob learns the value $f(x, y)$ but there is no such requirement for Alice.

Definition 1. An (asymmetric) two-party computation for a function f on two inputs is an interactive protocol in which, given inputs x for Alice and y for Bob, Bob's output equals $f(x, y)$ (with probability one over the randomness of the protocol).

The definition of security will based on the premise that Alice and Bob should not find out any unintended information from the execution of the protocol. Recall that the information learned by a party is determined by its view, which consists of its randomness (if any) and the sequence of messages received from the other party. From Alice's perspective this means she should be able to simulate her own view based on her input only. To get a sense of what kind of information is unintended for Bob let's look at some examples where x, y, and $f(x, y)$ are single bits.

1. $f(x, y) = x \text{ XOR } y$. In this case, functionality forces Bob to learn Alice's input because it is an XOR of his own input and output. From Bob's perspective, there are no secrets. The protocol in which Alice forwards her input x to Bob (and Bob outputs $f(x, y)$) should be considered secure.
2. $f(x, y) = y$. This example is trivial as Bob can compute f without interaction, but the previous protocol is no longer secure because his view should now reveal no information at all about x.

3. $f(x, y) = x$ AND y. Here, $f(x, y)$ completely reveals x as in example 1 when $y = 1$ and it completely hides it as in example 2 when $y = 0$. Suppose Alice and Bob run protocol 1 if $y = 1$ and protocol 2 when $y = 0$. Then Bob doesn't learn any unintended information, but Alice learns y, violating security.

The AND function turns out to be "complete" for secure two-party computation so the last example is far from naive. Say Alice and Bob might want to go on a date but each fears the embarrassment of being rejected by the other. Some dating websites ask both Alice and Bob to give an opinion of the other's suitability (usually expressed by a swipe) and make them aware of their mutual interest only if there is a match. This is nothing more than secure computation of AND by a trusted third party. You may imagine scenarios in which Alice and Bob might prefer for a third party not to get involved.

Before we give a protocol for it we need to define security. What we mean by Bob learning no unintended information is that he learns nothing *except* what is implied by the output $f(x, y)$ of the protocol. The simulation-based security definition requires that he should be able to simulate his view given his input *and his output*.

Definition 2. A two-party computation is (s, ε)-simulatable against honest-but-curious parties if there are simulators S_A and S_B such that for every x and y, the random variables $S_A(x)$ and $S_B(y, f(x, y))$ are (s, ε)-indistinguishable from Alice's and Bob's view, respectively.

By "honest-but-curious parties" we mean that both Alice and Bob are required to follow the instructions of the protocol, but they can examine their views for any slip of unintended information. We later look into more powerful adversaries that can deviate from instructions, or even abort the protocol if the interaction is not to their liking.

2 A protocol for AND and oblivious transfer

A protocol for AND can be obtained from public-key encryption that has some additional properties. Suppose Alice encrypts her input x under some public key PK. Bob should be able to decrypt if his input is $y = 1$, but not if it is $y = 0$. Thus Bob should know the corresponding secret key SK when $y = 1$, but have no information about it when $y = 0$. In the case $y = 1$, Bob can generate the key pair as usual. In the case $y = 0$, however, he should be able to generate a random variable indistinguishable from a public key without knowing the secret key. This is possible in both El Gamal encryption and the LWE-based encryption scheme that you worked out in Homework 2. Let us describe the El Gamal-based protocol.

AND protocol: Alice's input is $x \in \{0, 1\}$. Bob's input is $y \in \{0, 1\}$.

1. If $y = 0$, Bob samples $SK \sim \mathbb{Z}_q$ and sets $PK = g^{SK}$.
   If $y = 1$, Bob samples PK uniformly at random from $\mathbb{G}$.[1] Bob sends PK to Alice.
2. Alice sends $Enc(PK, x)$ to Bob, where Enc is El Gamal encryption.
3. If $y = 0$ Bob outputs 0. If $y = 1$, upon receiving C he outputs $Dec(SK, C)$.

This protocol is clearly functional: If $y = 0$ Bob outputs 0, and if $y = 1$ he outputs $Dec(SK, Enc(PK, x)) = x$, so his output always equals x AND y.

[1] Recall that elements of $\mathbb{G}$ are represented by the numbers between 1 and q, so this amounts to uniformly sampling a number in this range.

Claim 3. *If El Gamal encryption is (s, ε)-message simulatable in size t then the AND protocol is (s, ε)-simulatable against honest-but-curious parties in size $t + O(op)$.*

Here, *op* stands generically for the size of an efficient operation like sampling a random element, addition in $\mathbb{Z}_q$, multiplication in $\mathbb{G}$, and base-g exponentiation.

Proof. Alice's view consists of a public key, which is a random element of $\mathbb{G}$. She can sample this view efficiently in one operation.

When $y = 1$, Bob's view consists of SK, PK, and $Enc(PK, x)$. He can simulate this view given his output x in size $O(op)$ by generating a key pair (SK, PK) and then encrypting x under PK. When $y = 0$, Bob's view consists of PK and $Enc(PK, x)$ only. By the message simulatability of El Gamal encryption, this view is (s, ε)-simulatable in size t. Altogether, given his input and output, Bob can simulate his view in size $t + O(op)$. □

Somewhat more general than AND is the oblivious transfer function given by

$$OT(x_0x_1, b) = x_b = (x_0 \text{ AND } \bar{b}) \text{ OR } (x_1 \text{ AND } b).$$

A secure two-party protocol for this function is a mechanism for *the chooser* Bob to learn exactly one of the two values x_0, x_1 held by *the provider* Alice. Alice does not find out which of the two Bob learned and Bob does not find out anything about the other value.

AND is a special case of oblivious transfer with x_0 fixed to zero, so a secure protocol for OT can also be used to evaluate AND. In the other direction, consider the following protocol for OT:

OT protocol: Alice's input is $x_0x_1 \in \{0, 1\}^2$. Bob's input is $b \in \{0, 1\}$.

1. Alice and Bob run the AND protocol on input x_0 for Alice and $\bar{b}$ for Bob.
2. Alice and Bob run the AND protocol on input x_1 for Alice and b for Bob.
3. Bob produces the first output if $b = 0$ and the second one if $b = 1$.

The functionality of this protocol follows from the functionality of the AND protocol.

Theorem 4. *If the AND protocol runs in size t and is (s, ε)-message simulatable in size t then the OT protocol is $(s - t, 2\varepsilon)$- simulatable in size $2t + O(1)$ against honest-but-curious parties.*

Proof. Let S_A and S_B be Alice's and Bob's simulators for the AND protocol. Alice's simulator for the OT protocol runs $S_A(x_0)$ followed by $S_A(x_1)$. This is $(s - t, 2\varepsilon)$-indistinguishable from her actual view by the usual hybrid argument (Lemma 5 in Lecture 3).

Bob can also simulate his view given his input b and his output z. If $b = 0$, he runs $S_B(1, z)$ followed by $S_B(0, 0)$. If $b = 1$, he runs $S_B(0, 0)$ followed by $S_B(1, z)$. This is also $(s-t, 2\varepsilon)$-indistinguishable from his actual view. □

This protocol is slightly more general than advertised because x_0 and x_1 can take arbitrary values in $\mathbb{Z}_q$, not only 0 and 1. Still, oblivious transfer is a relatively simple function. What about other, more complex functions?

3 Garbled gates

Let's take another look at securely computing the AND function. Although we already have a protocol for this because AND is oblivious transfer specialized to $x_0 = 0$, let's now describe a different protocol that will more easily generalize to arbitrary functions. The idea is to encrypt both the inputs x and y and the output $z = x \text{ AND } y$ by "garbled values" X_x, Y_y and Z_z that represent x, y, and z uniquely, but do not reveal information about what these values are.

Specifically, each of the three wires, namely Alice's input wire x, Bob's input wire y, and the output wire z, is associated with a pair of random garbled values (X_0, X_1), (Y_0, Y_1), and (Z_0, Z_1), respectively. The protocol for evaluating $x \text{ AND } y$ has the following form.

0. Bob chooses $X_0, X_1, Y_0, Y_1, Z_0, Z_1$ uniformly at random.
1. Alice and Bob run a protocol in which Alice learns nothing but X_x, Y_y, and $Z_{x \text{ and } y}$.
2. Alice sends Bob the last value Z_z and Bob outputs z.

A protocol of this form should be secure because Alice learns nothing beyond three random values, while Bob learns nothing beyond the output $z = x \text{ AND } y$. All that remains is to implement step 1.

Among the three values that Alice should learn, Y_y is easiest: Bob can reveal it to Alice. In contrast, Bob cannot reveal X_x because he doesn't (and isn't supposed to) know x. But this is precisely the purpose of oblivious transfer: Alice learns X_x while Bob doesn't find out anything about x.

It remains for Alice to learn the value $Z_{x \text{ and } y}$. The idea is for Bob to encrypt the values Z_0, Z_0, Z_0, and Z_1 under the "keys" X_0Y_0, X_0Y_1, X_1Y_0, and X_1Y_1, respectively. Since Alice knows only the key X_xY_y, she will be able to decrypt Z_z for $z = x \text{ AND } y$, but not $Z_{\overline{z}}$.

Two challenges arise in implementing this idea. For the protocol to be functional Alice must not only decrypt Z_z successfully, but she must also know that her decryption of $Z_{\overline{z}}$ failed. One way to address this challenge is to include some information in the plaintext, like a leading string of zeroes, that is unlikely to arise if decryption is performed with the wrong random key.

For the protocol to be secure, Alice must not find out any information about x and y. For example, if Bob gives Alice the four encryptions in the order above and Alice manages to decrypt the second ciphertext but not the other three, she learns $x = 0$ and $y = 1$. This challenge can be addressed by randomly permuting the order of the ciphertexts.

Here is a possible implementation. We will assume that Z_0 and Z_1 are k-bits long, while X_0, X_1, Y_0, Y_1 are $8k$-bits long each. We split X_0 into four $2k$-bit long blocks denoted by $X_0^1, X_0^2, X_0^3, X_0^4$ and similarly for the others. The notation $0^k z$ stands for a string z prefixed by k zeros.

Garbled AND transfer (perfect variant):
Bob's input is $X_0, X_1, Y_0, Y_1 \sim \{0,1\}^{8k}$ and $Z_0, Z_1 \sim \{0,1\}^k$.
Alice's input is X_x and Y_y.

1. Bob chooses a random permutation (a, b, c, d) of $(1, 2, 3, 4)$, creates the four messages

$$X_0^a \oplus Y_0^a \oplus 0^k Z_0, \quad X_0^b \oplus Y_1^b \oplus 0^k Z_0, \quad X_1^c \oplus Y_0^c \oplus 0^k Z_0, \quad X_1^d \oplus Y_1^d \oplus 0^k Z_1,$$

   unpermutes them so that the top indices are in order 1, 2, 3, 4, and sends them to Alice.

2. Upon receiving (C_1, C_2, C_3, C_4), Alice calculates $\hat{Z}_i = C_i \oplus X_x^i \oplus Y_y^i$ for $i \in \{1, 2, 3, 4\}$.
   If exactly one $\hat{Z}_i$ starts with k zeros, Alice outputs the last k bits of $\hat{Z}_i$.

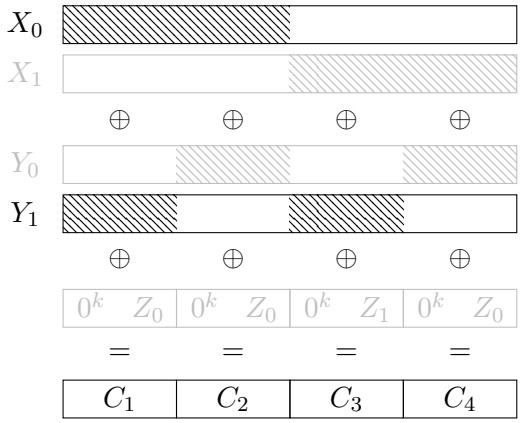

Figure 1: The garbled transfer protocol with $x = 0$, $y = 1$, and $(a, b, c, d) = (2, 1, 4, 3)$. The hatched (random) values are used by Bob for masking. Alice knows the values in the dark blocks X_x and Y_y. She does not know the ordering of the hatched boxes or the lightly shaded values. This allows her to recover $Z_0 = Z_{x \text{ and } y}$ with high probability (from C_1 in this example), but obtain no information about Z_1.

Bob's message will certainly include $X_x^i \oplus Y_y^i \oplus 0^k Z_{x \text{ and } y}$ for some i, so $\hat{Z}_i = 0^k Z_{x \text{ and } y}$ and $Z_{x \text{ and } y}$ figures as a possible output for Alice. However, this is not sufficient to conclude functionality as some other $\hat{Z}_j$ might also start with k zeros.

For example, suppose Alice's input is X_0, Y_1 and $(a, b, c, d) = (2, 1, 4, 3)$ as in Figure 1. Then $C_1 \oplus X_0^1 \oplus Y_1^1$ starts with k zeros as desired. But it could also happen that $\hat{Z}_3 = C_3 \oplus X_0^3 \oplus Y_1^3$ starts with k zeros. However, this is quite unlikely to happen: In our example C_3 equals $X_1^3 \oplus Y_1^3 \oplus 0^k Z_1$, so $\hat{Z}_3$ can start with k zeros only if $X_0^3 \oplus X_1^3$ does. These two values are random and independent, so the probability is only 2^{-k}. We can summarize this reasoning in the following claim.[2]

Claim 5. *Alice outputs $Z_{x \text{ and } y}$ in the garbled AND transfer with probability at least $1 - 3 \cdot 2^{-k}$.*

Regarding security, Bob does not receive any information from Alice so his view is trivially simulatable. We now prove that Alice's view is also simulatable.

Claim 6. *Alice's view in the garbled AND protocol is perfectly simulatable from her input and output by size $O(k)$.*

Proof. Alice's views consists of her (random) inputs X_x and Y_y and the message C_1, C_2, C_3, C_4 she receives from Bob. These are equal to

$$X_0^a \oplus Y_0^a \oplus 0^k Z_0, \quad X_0^b \oplus Y_1^b \oplus 0^k Z_0, \quad X_1^c \oplus Y_0^c \oplus 0^k Z_0, \quad X_1^d \oplus Y_1^d \oplus 0^k Z_1,$$

in some order. Apart from the one that contains parts of both X_x and Y_y, the rest all have some component that depends on $X_{\overline{x}}$ or $Y_{\overline{y}}$. This makes them random, mutually independent, and also independent of X_x and Y_y.

For example, if $x = 0$, $y = 1$, $(a, b, c, d) = (2, 1, 4, 3)$ (see Figure 1), then Alice's view will be

$$(X_0, Y_1, X_0^1 \oplus Y_1^1 \oplus 0^k Z_0, X_0^2 \oplus Y_0^2 \oplus 0^k Z_0, X_1^3 \oplus Y_1^3 \oplus 0^k Z_1, X_1^4 \oplus Y_0^4 \oplus 0^k Z_0).$$

This is identically distributed to

$$(X_0, Y_1, X_0^1 \oplus Y_1^1 \oplus 0^k Z_0, R_2, R_3, R_4)$$

[2]The protocol can be made fully functional at the cost of a small drop in security. In the unlikely case that Alice detects a functionality error, she and Bob can run an insecure protocol.

because each of the last three blocks contains randomness Y_0^2, X_1^3, and X_1^4, respectively, that does not appear in any of the other blocks.

Since (a, b, c, d) is a random permutation, the "non-random block" is equally likely to arise in any of the four possible positions. Therefore, given inputs X, Y, and output Z, Alice's view can be simulated by $(X, Y, C'_1, C'_2, C'_3, C'_4)$, where C'_i equals $X \oplus Y \oplus 0^k Z$ for a random index i and the remaining C' are random and independent of X, Y, and C'_i. □

One annoyance in the garbled AND protocol is that the garbled inputs are eight times as long as the garbled outputs. It will be convenient to have a length-preserving variant. This can be accomplished using a pseudorandom generator $G\colon \{0,1\}^k \to \{0,1\}^{8k}$.

Garbled AND tranfer: Same as before, except that $X_0, X_1, Y_0, Y_1 \sim \{0,1\}^k$ (instead of $\{0,1\}^{8k}$, $X_0^1 X_0^2 X_0^3 X_0^4$ is now $G(X_0)$, and $Y_0^1 Y_0^2 Y_0^3 Y_0^4$ is now $G(Y_0)$.

By the usual type of security argument that relates the computational and perfect settings we can conclude that the garbled AND protocol is both functional (with high probability) and secure.

Theorem 7. *If G is (s, ε)-pseudorandom, the garbled AND transfer produces an output with probability at least $1 - 3(\varepsilon + 2^{-k})$ and is $(s, 4\varepsilon)$-simulatable by size $O(t)$, where t is the size of G.*

4 Yao's protocol

We now have all the elements in place to describe Yao's protocol for securely computing an arbitrary function $f(x, y)$ represented by a circuit C. We may and will assume that C is implemented using only binary gates $g\colon \{0,1\}^2 \to \{0,1\}$ like AND, OR, XOR, $NAND$. The protocol we described in the previous section generalizes easily to these other types of gates. The only change is that in Bob's message Z should in general be indexed by $g(x, y)$.

The idea of Yao's protocol is to iteratively perform garbled transfers on the gates of the circuit until Alice learns the garbled value of the output. At this point Alice forwards this value to Bob, who ungarbles it to uncover the actual output $f(x, y)$. For simplicity we will assume that f takes two n-bit inputs x and y and produces a single bit as output, but it is straightforward to generalize to asymmetric inputs and longer outputs.

Yao's protocol: Alice's and Bob's inputs are $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, respectively.

1. For every wire w of C, Bob chooses a random pair of garbled values $W_0^w, W_1^w \sim \{0,1\}^k$.
2. For each of Alice's n input wires $w \in \{1, \ldots, n\}$, Alice and Bob execute the oblivious transfer protocol $OT(W_0^w W_1^w, x_i)$ with Alice playing the part of the chooser. For each of Bob's n input wires $w \in \{n+1, \ldots, 2n\}$, Bob sends the garbled value $W_{y_i}^{n+i}$ to Alice.

   At this point, Alice knows the garbled values $W^1 = W_{x_1}^1, W^2 = W_{x_2}^2, \ldots, W^{2n} = W_{y_n}^{2n}$ associated to the $2n$ input wires of the circuit.
3. Starting from the input wires of the circuit, Alice computes a garbled value W^w associated to every wire w in the circuit by engaging in garbled g-transfer with Bob for every gate g in the order of circuit evaluation. At the time gate g is processed, Alice knows the garbled values W^{g_1} and W^{g_2} associated with the input wires of g and learns the value $W^{g_{out}}$ associated with the output wire of g.
4. When Alice learns the garbled value W^{out} associated to the output gate she sends it to Bob. Bob outputs the value z for which $W^{out} = W_z^{out}$ (if it is unique).

This protocol is functional as long as the oblivious and garbled transfer protocols are and the random values W_0^{out} and W_1^{out} are distinct (which fails with probability 2^{-k}). We outline the proof of security. Alice's view in Yao's protocol consists of a collection of independent random values W^w associated to all the wires in the circuit as well as the views of her executions of the oblivious and garbled transfer protocols. By the security of these protocols she can simulate these views based on the inputs and outputs W^w and her own inputs $x_1, \ldots, x_n$.

On the other hand, Bob's views in the oblivious and garbled transfer executions can be simulated only from his randomness W_0^w, W_1^w for all wires w as these play the role of inputs and the protocols generate no outputs on Bob's side. This accounts for all messages that Bob receives in steps 2 and 3. In step 4, assuming the protocol is functional, Bob receives the message $W^{out} = W_{f(x,y)}^{out}$ which he can simulate from his randomness W_0^{out}, W_1^{out} and his output $f(x, y)$.

The oblivious transfers in step 2 of Yao's protocol can be implemented in parallel, and so can the garbled transfers in step 3. This results in a three message protocol,[3] even if the circuit C describes a highly sequential computation.

[3]It can be optimized down to two messages.