# Goldreich, Micali, and Wigderson 2PC

Noah Stephens-Davidowitz

June 9, 2023

## 1 Recap

In the previous lectures, we saw the definition of secure two-party computation (2PC, in the honest-but-curious setting), and oblivious transfer (OT). We also constructed OT. In this lecture, we will show how to use OT in order to build 2PC for arbitrary efficient functions. Specifically, we will see a construction due to Goldreich, Micali, and Wigderson [GMW87], which is commonly known as the GMW protocol. In the next lecture, we will see a very different strategy, based on Yao's garbled circuits.

## 2 The GMW protocol

The idea behind the GMW protocol starts with the concept of secret sharing. We will actually only need a two-out-of-two secret-sharing scheme for a one-bit secret $s$. For this, as we saw earlier, there is an extremely simple scheme. Alice has a one-bit *share* $a \in \{0,1\}$ and Bob has a one-bit *share* $b \in \{0,1\}$. The shares are uniformly random subject to the restriction that $s = a \oplus b$. Recall that, from Alice's perspective, $a$ is a uniformly random bit independent of $s$, and similarly from Bob's perspective $b$ is a uniformly random bit. But, clearly the two bits together are enough to reconstruct $s$.

We view the function $f$ that we are trying to compute as a circuit, and we imagine the computation defining a value for each gate in the circuit. Of course, if we want a secure protocol for two-party computation, then in general we will not want either Alice or Bob to learn the value of any individual gate (except the input and output), but if we want the protocol to be correct, they should have enough information between the two of them to determine the value of any gate.

This makes it quite natural to use secret sharing for two-party computation. In particular, for any gate $G : \{0,1\}^2 \to \{0,1\}$, we wish to show how Alice and Bob can engage in a protocol whose input is two bits $a_1, a_2 \in \{0,1\}$ for Alice and two bits $b_1, b_2 \in \{0,1\}$ for Bob and whose output is a single bit $a_3 \in \{0,1\}$ for Alice and one for Bob $b_3 \in \{0,1\}$ with the following property. $a_3$ and $b_3$ should be uniformly random bits satisfying $a_3 \oplus b_3 = G(a_1 \oplus b_1, a_2 \oplus b_2)$. In other words, Alice and Bob start out with a secret sharing $(a_1, b_1)$ of the first input bit $w_1$ and a secret sharing $(a_2, b_2)$ of the second input bit $w_2$, and they end up with a secret sharing $(a_3, b_3)$ of the output bit $w_3 = G(w_1, w_2)$.

If Alice and Bob can compute such a functionality securely, then Alice and Bob can securely compute *any* efficiently computable function $f$. To do so, they first create shares $a_i, b_i$ for each input gate. I.e., for each input gate $w_i$ corresponding to one of Alice's input bits $x_j$, Alice samples $b_i \sim \{0,1\}$ uniformly at random and sends it to Bob, keeping $a_i := x_j \oplus b_i$ for herself. Bob does

the same with his input bits. Then, Alice and Bob can engage in the above protocol to compute shares of the value of each gate in the circuit. In particular, they will eventually compute shares of the output gates. Let's say for simplicity that there is just one output bit $w_m$ and that Alice is meant to learn the output bit, but Bob learns nothing. (I.e., the function that we wish to compute has the form $f(x, y) = (g(x, y), \perp)$, where $g$ has one bit as output. This can be generalized to many output bits for each party by repeating the protocol many times—though of course this is not the most efficient way to accomplish this.) Then, at the end of the protocol, Bob can simply send Alice $b_m$, allowing Alice to compute $w_m = a_m \oplus b_m$. (One can be slightly more clever about the output gate to just directly have $w_m = a_m$.)

### 2.1 Aside: XOR and NOT for free

Before we see how to compute a general gate, let's notice that Alice and Bob can compute some gates *locally*, i.e., without any interaction at all. E.g., suppose that $G(x_1, x_2) = x_1 \oplus x_2$ is the XOR gate. Then, if Alice has $a_1, a_2 \in \{0, 1\}$ and Bob has $b_1, b_2 \in \{0, 1\}$ with $x_1 = a_1 \oplus b_1$ and $x_2 = a_2 \oplus b_2$, they can simply compute $a_3 := a_1 \oplus a_2$ and $b_3 := b_1 \oplus b_2$ to get shares of $x_1 \oplus x_2$. In particular, $a_3 \oplus b_3 = a_1 \oplus a_2 \oplus b_1 \oplus b_2 = x_1 \oplus x_2$, as needed.

Similarly, the NOT gate $G(x) = \neg x$ is trivial to compute, since $(\neg a) \oplus b = \neg(a \oplus b)$. So, if Alice and Bob have shares $a, b$ for some bit $x$, they can trivially compute shares $a' := \neg a$ and $b' := b$ of $\neg x$.

These little tricks are not necessary from our perspective. We will show how to handle any gate generically below. But, they are quite important for practical applications because they drastically improve efficiency.

### 2.2 Any gate with one-out-of-four OT

Recall that a one-out-of-two oblivious-transfer protocol (or just an OT protocol) is a secure protocol that behaves as follows. Alice's input is $m_0, m_1 \in \{0, 1\}$, and Bob's input is $b \in \{0, 1\}$. Alice's output is nothing, and Bob's output is $m_b$. Intuitively, "Bob should learn $m_b$ but not $m_{1-b}$, and Alice should not learn Bob's bit $b$."

There is a very natural way to generalize this. In particular, in a $k$-out-of-$\ell$ OT protocol, Alice's input is $n$ bits $m_1, \ldots, m_\ell \in \{0, 1\}$ and Bob's input is $k$ distinct indices $i_1, \ldots, i_k \in [\ell]$. Alice's output is still nothing, and Bob's output is $m_{i_1}, \ldots, m_{i_k}$. In other words, "Bob learns $k$ of the messages but none of the others, and Alice learns nothing about Bob's indices $i_1, \ldots, i_k$."

For our current purposes, we will need one-out-of-four OT. In this case, it is natural to think of Bob's input as two bits $b_1, b_2 \in \{0, 1\}$, and Alice's messages as $m_{0,0}, m_{0,1}, m_{1,0}, m_{1,1} \in \{0, 1\}$.

Let's see how to use one-out-of-four oblivious transfer to allow Alice and Bob to compute shares $a_3, b_3$ of the output of some gate $G$ given shares $a_1, a_2, b_1, b_2$ of the input. To do so, Alice will sample $a_3 \sim \{0, 1\}$ uniformly at random and set $m_{0,0} := a_3 \oplus G(a_1 \oplus 0, a_2 \oplus 0)$, $m_{0,1} := a_3 \oplus G(a_1 \oplus 0, a_2 \oplus 1)$, $m_{1,0} := a_3 \oplus G(a_1 \oplus 1, a_2 \oplus 0)$, and $m_{1,1} := a_3 \oplus G(a_1 \oplus 1, a_2 \oplus 1)$. Bob receives $b_3 := m_{b_1,b_2} = a_3 \oplus G(a_1 \oplus b_1, a_2 \oplus b_2)$. In particular, $a_3 \oplus b_3 = G(a_1 \oplus b_1, a_2 \oplus b_2)$ which is what we want.

Since this is a little abstract, we give a concrete example. Suppose the gate $G$ is an AND gate and Alice's input buts are $a_1 = 0, a_2 = 1$. Suppose she samples $a_3 = 1$. Then, her messages will be $m_{0,0} = 1$, $m_{0,1} = 1$, $m_{1,0} = 0$, and $m_{1,1} = 1$. Notice that $m_{b_1,b_2} \oplus 1 = \mathsf{AND}(b_1, b_2 \oplus 1)$ for all $b_1, b_2$.

## 2.3 Putting everything together and a proof of security

So, the full Goldreich-Micali-Wigderson protocol works as follows. Alice and Bob create shares $a_1, \ldots, a_n, b_1, \ldots, b_n \in \{0,1\}$ for the input gates of the circuit as we described in the beginning. Then, for each gate $G$ with input bits $w_i, w_j$ and output bit $w_k$, they use their shares $a_i, a_j, b_i, b_j \in \{0,1\}$ with $a_i \oplus b_i = w_i$ and $a_j \oplus b_j = w_j$ and the one-out-of-four oblivious transfer protocol to compute shares $a_k, b_k \in \{0,1\}$ with $a_k \oplus b_k = w_k := G(w_i, w_j)$.

Once they have done this for every gate, Alice sends Bob her shares corresponding to Bob's output gates, and Bob sends Alice his shares corresponding to Alice's output gate. The parties can then use these to find their respective outputs.

**Theorem 2.1.** *The GMW protocol described above is a secure protocol for two-party computation against honest-but-curious adversaries, provided that the underlying (one-out-of-four) OT scheme is secure against honest-but-curious adversaries.*

*Proof.* To prove the security of the parties, we need to construct a PPT simulator $S_A$ for Alice and a PPT simulator $S_B$ for Bob. Crucially, we assume that the OT protocol is secure, which means that there are PPT simulators $S_{A,\mathsf{OT}}$ and $S_{B,\mathsf{OT}}$ that simulate the views of Alice and Bob respectively in the OT protocol. Let's build Alice's simulator $S_A$ first.

Alice's simulator $S_A$ takes as input Alice's own input $x$ and her output $f_A(x,y)$, and for every input pair $x, y$, the simulator's output $S_A(x, f_A(x,y))$ must be computationally indistinguishable from Alice's view in a real run of the protocol with Bob. Alice's view consists of four things: her bits $a_i$, the bits $b_i$ from Bob that she gets to see when $w_i$ is either one of Alice's input gates or one of Alice's output gates, and for each gate, the messages $m^{(G)}_{0,0}, m^{(G)}_{0,1}, m^{(G)}_{1,0}, m^{(G)}_{1,1} \in \{0,1\}$ that Alice produces for the OT protocol, and Alice's view in the OT protocol corresponding to the gate.

Alice's simulator first samples uniformly random bits $a_1, \ldots, a_\ell \sim \{0,1\}$, one for each gate in the circuit. And, for each input gate $w_i$ corresponding to input bit $x_j$, the simulator computes $b_i := a_i \oplus x_j$. Similarly, for each output gate $w_i$ corresponding to output bit $z_j$, the simulator computes $b_i := a_i \oplus z_j$.

Then, for each gate $G$ in the circuit, the simulator computes $m^{(G)}_{0,0}, m^{(G)}_{0,1}, m^{(G)}_{1,0}, m^{(G)}_{1,1} \in \{0,1\}$ just like Alice would, i.e., $m^{(G)}_{b'_1,b'_2} = a_{k_G} \oplus G(a_{i_G} \oplus b'_1, a_{j_G} \oplus b'_2)$, where $i_G, j_G$ are the indices of the parent gates of $G$ and $k_G$ is the index of the gate itself. The simulator $S_A$ then runs the $S_{A,\mathsf{OT}}$ on input $m^{(G)}_{0,0}, m^{(G)}_{0,1}, m^{(G)}_{1,0}, m^{(G)}_{1,1} \in \{0,1\}$, receiving a simulated view for the run of the OT protocol corresponding to gate $G$. (Remember that $S_{A,\mathsf{OT}}$ takes as input $m_{0,0}, m_{0,1}, m_{1,0}, m_{11} \in \{0,1\}$ and outputs a view that is indistinguishable from Alice's view in an honest run of the protocol. If Alice had any output in the OT protocol, then the simulator would need output as well.) $S_A$ outputs Alice's input bits, her output bits, the $a_i$, the $b_i$, the plaintexts $m^{(G)}_{b'_1,b'_2}$, and the simulated views (in the appropriate order).

Notice that the $a_i$ are distributed identically to those produced in the honest protocol, since in both cases, they are simply uniformly random bits. The messages $m^{(G)}_{b'_1,b'_2}$ are deterministic functions of the $a_i$, so they must be distributed correctly as well. Similarly, the $b_i$ corresponding to the input and output gates are deterministic functions of the $a_i$ and Alice's input and output, so they are also distributed correctly. Finally, by the security of the OT protocol, the views simulated by $S_{A,\mathsf{OT}}$ are indistinguishable from the views produced in an honest interaction with Bob. (Formally, we should do a hybrid argument here, with one appeal to OT security for each gate.)

Now, for Bob's simulator, $S_B$. $S_B$ basically does what $S_A$ did. Specifically, it needs to compute three things: Bob's bits $b_1, \ldots, b_\ell$ (most of which are output from the OT protocol), the $a_i$ corresponding to Bob's input and output, and Bob's views in the many OT protocols. I.e., it samples uniformly random bits $b_1, \ldots, b_\ell \sim \{0,1\}$, for each input gate $w_i$ corresponding to one of Bob's input bits $y_j$, it computes $a_i := b_i \oplus y_j$, and similarly for Bob's output. Then, for each OT gate with input bits $w_i$, $w_j$ and output bit $w_k$, it runs $S_{B,\mathsf{OT}}$ on input $(b_i, b_j)$ and $m_{b_i,b_j} := b_k$, and receives as output some view for Bob. (Recall that $S_{B,\mathsf{OT}}$ takes as input two input bits $(b_1, b_2)$ and one output plaintext $m_{b_1,b_2}$.) Finally, $S_B$ outputs Bob's input and output bits, the $b_i$, the $a_i$, and all of the simulated OT views (in the appropriate order).

To see that the simulated view is indistinguishable from an honest view, simply note that the $b_i$ are in fact uniformly random and independent of everything else in the honest protocol, so that they are distributed identically in the simulated view and in the honest view. (In particular, the $b_i$ corresponding to input bits are sampled uniformly at random from Bob. All other $b_i$ have the form $b_i := a_i \oplus w_i$, where $a_i$ is a fresh uniformly random bit and $w_i$ is fixed given the inputs.) The $a_i$ corresponding to input and output bits are deterministic functions of the $b_i$, $y$, and $f_B(x, y)$. So, they are distributed correctly as well. Finally, the OT views are indistinguishable from honest views by the security of the OT protocol. □

# 3 One-out-of-four OT

The GMW protocol requires a one-out-of-four OT protocol as a subprocedure. So, to actually instantiate it, we still need to show how to build one-out-of-four OT.

We can actually build it directly in the same way that we built one-out-of-two OT directly from an encryption scheme with an oblivious key-generation algorithm. Specifically, in such a protocol, Bob simply samples a public-key secret-key pair $sk_{b_1,b_2}, pk_{b_1,b_2}$, and then for all $b'_1, b'_2$ with $(b'_1, b'_2) \neq (b_1, b_2)$, he samples a public key $pk_{b'_1,b'_2}$ obliviously. He then sends $pk_{0,0}, pk_{0,1}, pk_{1,0}, pk_{1,1}$ to Alice. Alice encrypts each $m_{b'_1,b'_2}$ under the public key $pk_{b'_1,b'_2}$, and sends the resulting ciphertexts to Bob. Bob then decrypts the one ciphertext encrypted using $pk_{b_1,b_2}$ and outputs the result.

The proof of security for the above scheme in the honest-but-curious model is essentially identical to the proof for the similar one-out-of-two OT protocol that we saw earlier. However, as I said repeatedly in the previous lecture, this is a bit of a disappointing protocol since there is such an obvious way for Bob to cheat, which is not captured by the honest-but-curious model. (He can just sample four public-key secret-key pairs!) So, we will show a way to directly convert any one-out-of-two OT protocol into one-out-of-four OT. In particular, this construction will be secure against malicious adversaries if the original one-out-of-two OT protocol is.

The idea behind the construction is quite simple, though some thought is necessary to understand the details. The high-level idea is simply that Alice should sample uniformly random pad bits $k_{b_1,b_2} \sim \{0,1\}$ and send Bob $c_{b_1,b_2} := m_{b_1,b_2} \oplus k_{b_1,b_2}$. Alice and Bob then engage in a few one-out-of-two OT protocols in such a way that at the end, Bob learns the one-time pad $k_{b_1,b_2}$ but all other one-time pads are still uniformly random from his perspective.

Here's the actual protocol. Alice receives as input $m_{0,0}, m_{0,1}, m_{1,0}, m_{1,1}$ and samples six bits

uniformly at random $p_0^{(1)}, p_1^{(1)}, p_0^{(2)}, p_1^{(2)}, p_0^{(3)}, p_1^{(3)} \sim \{0,1\}$. We then define

$$\begin{aligned} c_{0,0} &:= m_{0,0} \oplus p_0^{(1)} \oplus p_0^{(2)} \\ c_{0,1} &:= m_{0,1} \oplus p_0^{(1)} \oplus p_1^{(2)} \\ c_{1,0} &:= m_{1,0} \oplus p_1^{(1)} \oplus p_0^{(3)} \\ c_{1,1} &:= m_{1,1} \oplus p_1^{(1)} \oplus p_1^{(3)} \ . \end{aligned}$$

Notice that, if Alice reveals exactly one of $p_0^{(i)}, p_1^{(i)}$ for each $i = 1, 2, 3$, then Bob will be able to "decrypt" exactly one "ciphertext" $c_{b_1,b_2}$, and all other $c_{b_1',b_2'}$ will be independent uniformly random bits from Bob's perspective.

So, Alice and Bob engage in three one-out-of-two OT protocols. In the first, Alice takes as her input $p_0^{(1)}$ and $p_1^{(1)}$, and Bob takes as his input his first input bit $b_1$, so that he receives $p_{b_1}^{(1)}$. In the second, Alice uses $p_0^{(2)}$ and $p_1^{(2)}$ and Bob uses his second input bit $b_2$, so that he receives $p_{b_2}^{(2)}$. In the third, Alice uses $p_0^{(3)}$ and $p_1^{(3)}$ and Bob again uses his second input bit $b_2$, so that he receives $p_{b_2}^{(3)}$. Finally, Alice sends the ciphertexts $c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1}$ to Bob, and Bob computes $m_{b_1,b_2} = c_{b_1,b_2} \oplus p_{b_1}^{(1)} \oplus p_{b_2}^{(2+b_1)}$.

(It is a nice exercise to try to determine whether this is the most efficient construction. Can you get one-out-of-four OT from just two runs of an OT protocol? Can you get a secure protocol that samples fewer than six random bits?)

## References

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *STOC*, 1987. 1