CMSC 858T—Secure Distributed Computation February 4, 2021

Lecture 4

Lecturer: Jonathan Katz *Scribe(s): David Wang, Georgios Tsimos*

1 The GMW Multi-Party Protocol

The GMW protocol that we saw in the last lecture can be extended to the multi-party setting.

Theorem 1 *Assuming semi-honest (two-party) OT, for any n-input functionality F there is an n-party protocol that $(n-1)$-securely computes F (for semi-honest adversaries).*

We describe the protocol in the *OT-hybrid* model. Recall we may assume F is deterministic. The main idea is that the parties share the values on every wire of the circuit in an n-out-of-n fashion. The protocol proceeds as follows:

Input-sharing phase: A party with input $x \in \{0,1\}$ chooses uniform bits $x_1, \ldots, x_n$ subject to the constraint that $x = x_1 \oplus \cdots \oplus x_n$. It then sends x_i to party P_i.

Gate evaluation: As in the last lecture, we consider evaluating a NOT gate, an XOR gate, and an AND gate. In each case we assume the invariant holds for the input wire(s) to the gate, and show how the parties can ensure it holds for the output wires.

- Say $y = \neg x$, and the parties hold $x_1, \ldots, x_n$ with $x_1 \oplus \cdots \oplus x_n = x$. They can compute shares of y by having P_1 set $y_1 := \neg x_1$ and having all other parties set $y_i := x_i$.
- Say $z = x \oplus y$, and the parties hold $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ with $x_1 \oplus \cdots \oplus x_n = x$ and $y_1 \oplus \cdots \oplus y_n = y$. They can compute shares of z by having every party set $z_i := x_i \oplus y_i$.
- Say $z = x \wedge y$, and the parties hold $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ with $x_1 \oplus \cdots \oplus x_n = x$ and $y_1 \oplus \cdots \oplus y_n = y$. In contrast to the previous two cases, the parties cannot compute shares of z through local computation alone. Instead, they will rely on oblivious transfer. The parties want to compute shares of

$$z = (x_1 \oplus \cdots \oplus x_n) \cdot (y_1 \oplus \cdots \oplus y_n) = \left(\bigoplus_i x_i \cdot y_i \right) \oplus \left(\bigoplus_{i<j} (x_i \cdot y_j \oplus x_j \cdot y_i) \right).$$

  Each P_i can compute $x_i \cdot y_i$ locally. But P_i and P_j must cooperate to compute shares of $x_i \cdot y_j \oplus x_j \cdot y_i$. They do this using a 1-out-of-4 OT functionality. Specifically, for all $i < j$ parties P_i, P_j do:

  - P_i chooses uniform bit z_{ij}. It then sets $z_{ji}^{ab} := z_{ij} \oplus x_i \cdot b \oplus a \cdot y_i$ for $a, b \in \{0,1\}$, and sends $z_{ji}^{00}, z_{ji}^{01}, z_{ji}^{10}, z_{ji}^{11}$ to the OT functionality (playing the role of the sender).
  - P_j, playing the role of the receiver, sends $x_j y_j$ to the OT functionality, and receives in return $z_{ji} := z_{ji}^{x_j y_j}$. Note that $z_{ij} \oplus z_{ji} = x_i \cdot y_j \oplus x_j \cdot y_i$.

  Finally, each party P_i computes $z_i := x_i \cdot y_i \oplus \left(\bigoplus_{j \neq i} z_{ij} \right)$. These form shares of z.

Output reconstruction: If party P_i is supposed to learn the value of some output wire z, then all parties send their shares of z to P_i (who can then reconstruct z).

It is not hard to prove that the above protocol is perfectly secure in the OT-hybrid model.

Just as in the 2-party case, the round complexity of the GMW protocol is high. All AND gates at the same level can be computed by the parties in parallel, but AND gates at different levels must be computed sequentially. Thus, the round complexity of the protocol is $O(d)$, where d is the depth of the circuit computing F. (Depth is measured in terms of AND gates only.)

The number of OTs is $O(n^2)$ per AND gate, and a natural question is whether this can be reduced. Some results about this are known but we will not cover them here.

1.1 What's Next?

We have shown broad feasibility results for secure computation in the semi-honest setting. In the following lectures we will consider the following questions:

- Can we improve the round complexity of the GMW protocol? In particular, is a constant-round protocol possible?
- Is it possible to achieve unconditional security?
- How can we achieve security against malicious adversaries?

Summarizing known feasibility results addressing the last two questions (not all of which we will cover this semester):

Semi-honest setting: The GMW protocol above achieves $(n-1)$-security based on (semi-honest) oblivious transfer. The cryptographic assumption here is optimal, since general secure computation for $t < n$ obviously implies the ability to compute OT.

The BGW protocol has perfect security, but only tolerates $t < n/2$ corrupted parties. This is known to be optimal, in the sense that there are functions that cannot be computed with unconditional security (even in the semi-honest setting, and even with statistical security) when $t \geq n/2$.

Malicious setting: Without a broadcast channel and with no prior setup, an extension of the BGW protocol achieves perfect security for $t < n/3$ malicious parties. This is optimal, in the sense that there are functions that cannot be computed even with computational security (without broadcast or prior setup) when $t \geq n/3$. There are also functions that cannot be computed with perfect security when $t \geq n/3$ (even given broadcast).

With a broadcast channel (or prior setup that allows broadcast to be implemented), protocols by Beaver or Rabin and Ben Or give statistical security for $t < n/2$. (This is optimal by what we have already said above in the semi-honest setting.)

The protocols above all achieve *full* security, in a sense we will define in a later lecture. When $t < n$ one can consider a relaxed notion called *security-with-abort*. The GMW protocol can be extended to the malicious setting by additionally relying on zero-knowledge proofs, where it achieves security-with-abort for $t < n$ corrupted parties. (In fact, it is possible to achieve security for $t < n$ parties based on OT alone.)

2 Garbled Circuits

We give here an introduction to Yao's *garbled circuits*. A garbled circuit can informally be viewed as an "encrypted" version of a boolean circuit for some function f. The basic idea is that one party (the "garbler") will act as a garbled-circuit generator, generated a garbled circuit GC_f corresponding to f, and the other (the "evaluator") will evaluate GC_f to get the result. We begin by describing the idea behind a garbled gate. The garbler will associate each wire with two cryptographic keys, one for each possible value the wire can take. The evaluator will learn one key per wire. The garbler ensures that the evaluator can only learn the "correct" key on the output wire, i.e., the key that corresponds to correct output based on the keys it knows on the input wires.

For example, consider an AND gate with input wires a, b and output wire c. The garbler will choose keys $a_0, a_1, b_0, b_1, c_0, c_1$ and make sure that if the evaluator knows keys a_x and b_y then it can (only) learn key $c_{x \wedge y}$. This can be done using encryption, by constructing the following table of ciphertexts:

$$\mathsf{Enc}_{a_0}(\mathsf{Enc}_{b_0}(c_0)), \quad \mathsf{Enc}_{a_0}(\mathsf{Enc}_{b_1}(c_0)), \quad \mathsf{Enc}_{a_1}(\mathsf{Enc}_{b_0}(c_0)), \quad \mathsf{Enc}_{a_1}(\mathsf{Enc}_{b_1}(c_1)).$$

(We assume for simplicity that the encryption scheme has the property that decryption with the wrong key yields an error. This is easy to ensure.) A garbled AND gate can be interpreted as the encryption of $c_{x \wedge y}$ using a_x and b_y. The evaluator can obtain $c_{x \wedge y}$ by trying to decrypt each of the above ciphertexts using b_y followed by a_x.

As described, the circuit evaluator learns which entry of the table decrypts correctly and thus learns the bits that the different keys correspond to. We can prevent this by having the circuit evaluator randomly permute the four ciphertexts above before sending them to the evaluator.

Although the above only deals with a single gate, it should be clear that these gates can be connected to give a garbled circuit. The garbled circuit for a function $f : \{0,1\}^n \to \{0,1\}^n$ has the following property: if the evaluator knows the input-wire keys corresponding to $x_1, ..., x_n \in \{0,1\}$, then it can evaluate the garbled circuit to obtain the output-wire keys corresponding to the values $y_1, \ldots, y_n = f(x)$. If the garbler tells the evaluator which keys correspond to which values for the output wires, then the evaluator can obtain the output y itself.

We will see next time how to extend this to a secure two-party protocol for computing f.