CSE 5854: Class 14

Benjamin Fuller

March 8, 2018

1 A semi-honest multi-party computation protocol

In this class we will define a private protocol for multiple parties. We will consider only *semi-honest* behavior where all parties follow the protocol. We saw how to transform a semi-honest protocol to a maliciously secure protocol for two parties. The transform is similar but more complicated in the multiple party setting. We denote by n the number of parties that are participating in the protocol. We assume that parties are computing a function $f(x_1, ..., x_n) = y$. It is easy to extend to the case where each party receives a different output.

We will assume that parties are able to directly speak to one another. That is, we assume there is a communication channel between each pair of parties i, j. We assume that the adversary is not able to see, insert, remove, or modify messages on any of these channels. Using our previous tools (encryption, signatures) it is possible to implement such channels. We furthermore assume that channels are synchronous, that is, messages are instantly delivered to the other party. Building this functionality in practice is not possible. Often protocols use timeouts but it is possible to manipulate timeouts and drastically delay a protocol. We also place no limit on the number of adversarial parties that exist in the system but we assume the malicious parties are decided before the protocol begins to execute.

We stress that all of the decisions above are important and affect the efficiency and possibility of a protocol existing. With all of these conditions set we are reading to define an ideal functionality for the protocol. Recall the ideal functionality is supposed to take inputs and create the output. In the setting where an arbitrary number of parties can be corrupted we need to reveal the output to the malicious parties first.

Definition 1 (Ideal Model - No fairness)**.** *Let $f : (\{0,1\}^*)^n \to \{0,1\}^*$ be an n-ary functionality. For $I = \{i_1, ..., i_t\} \subset \{1, ..., n\}$, and define $\overline{I} = \{1, ..., n\} \setminus I$ and $(x_1, ..., x_n)_I = (x_{i_1}, ..., x_{i_t})$. A pair (I, S) where $I \subseteq \{1, ..., n\}$ and S is a PPT algorithm represents an adversary in the ideal model. The* joint execution of f under (I, S) in the ideal model *(on input $(x_1, ..., x_n)$ and auxiliary input z), denoted $\texttt{IDEAL}_{f,I,S(z)}(x_1, ..., x_n)$ is defined by selecting a uniformly random tape for S and defining the output of the protocol as follows:*

1. *The simulator defines he malicious inputs $\vec{x}'_I = S((x_1, ..., x_n)_I, I, z, r)$. For all parties $i \in \overline{I}$, $x'_i = x_i$.*

2. *S is given $y = f(x')$. S then either says "output" or "stop". If S says "stop" the value of ideal is:*

$$(\perp^{|\overline{I}|}, S(\vec{x}_I, I, z, r, y)).$$

Otherwise if S says "output" the value of ideal is:

$$(f_{\overline{I}}(\vec{x}'), S(\vec{x}_I, I, z, r, y)).$$

We stress a few things. First the only information that S learns about the input of other parties is contained in the output of f. Furthermore, they don't receive this information until they specify an input for all malicious parties. The ability for S to say output or stop is considered a deficiency of the ideal protocol and corresponds to our inability to provide fairness in the setting where an arbitrary number of parties may be malicious. It is possible to limit this behavior to I that contain certain parties but we ignore this detail.

We also note the important of including the output of the good parties in the random variable defined by ideal. Without this behavior defining a simulator is easy, S just runs the real world adversary and makes up inputs for the honest parties. Including the output of the honest parties forces S to be consistent with the true output of the protocol. This is especially important in the case where the function has different outputs for each party.

For the moment our goal is to define an S for any semi-honest A in the private channels model. (Or rather to define a protocol where we know how to construct such an S.)

2 The GMW protocol

We are now ready to define the GMW multi-party computation [GMW87]. We consider a function where each input is a single bit and the output is a single bit. Again we generalize to more sophisticated functions over more realistic fields. The first thing we do is have each party secret share it input to all other parties. Consider some player i, they generate $r_1, ..., r_{n-1}$ and set $r_n = r_1 \oplus .... \oplus r_{n-1} \oplus x_i$.

We can think of each party having a share of each other parties input. Our goal is to operate on these shares in a way that computes the function. We will show how to perform NOT, XOR, and AND gates directly on shares. We can then think of our function as described by a circuit made out of these three gate types (similar to what we did for Garbled circuits). We will assume that each party i holds a share of each input to a gate G that we are evaluating. Our goal will be to have party i have a share of the output value of the gate when we are done. If we can keep this invariant we can privately compute the whole circuit.

2.1 NOT gates

Suppose our input is b and we wish to compute $c = 1 - b = b + 1 \mod 2$. Each party holds a share of b denoted b_i such that $\sum_{i=1}^{n} b_i = b$. The key observation is that a single party can add 1 to their value to create a sharing of c. Suppose that player 1 (this selection is arbitrary as long as parties agree who player 1 is). Then player 1 computes $c_1 = b_1 + 1 \mod 2$ while other parties compute $c_i = b_i$ for $2 \leq i \leq n$.

We then have the following:

$$c = \sum_{i=1}^{n} c_i = c_1 + \sum_{i=2}^{n} c_i = b_1 + 1 + \sum_{i=2}^{n} b_i = 1 + \sum_{i=1}^{n} b_i = 1 + b.$$

2.2 XOR gates

We compute the XOR gate directly by operating on players shares. We assume that the two inputs are x, y and we wish to compute $z = x \oplus y$. Each player has inputs x_i, y_i which are shares of x and y respectively. We wish to compute $z = x \oplus y$. The parties just locally add their two shares $z_i = x_i \oplus y_i$. This is in fact a valid sharing of z.

$$z = \sum_{i=1}^{n} z_i = \sum_{i=1}^{n} (x_i \oplus y_i) = \left(\sum_{i=1}^{n} x_i \right) \oplus \left(\sum_{i=1}^{n} y_i \right) = x \oplus y.$$

2.3 AND gates

Computing AND gates is significantly more complicated then either XOR and NOT gates. We again assume that parties have inputs x_i, y_i which are sharings of x and y respectively. Our goal is compute a sharing of $x \cdot y$. First we right the expanded z in terms of shares of x and y. We have that

$$z = xy = \left(\sum_{i=1}^{n} x_i \right) \cdot \left(\sum_{i=1}^{n} y_i \right).$$

Unfortunately, unlike the XOR case this expression has n^2 terms:

$$\begin{aligned} z &= \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i \cdot y_i) \\ &= \sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} \left(\sum_{j=i+1}^{n} x_i y_j + x_j y_i \right). \end{aligned}$$

Furthermore, while n terms are computable by individual parties (namely $x_i \cdot y_i$) most of the terms contain values that are distributed between parties. A natural idea is to try and announce some subset of the values x_i, y_i to other parties to allow them to perform the computation. Unfortunately, if $n - 1$ parties are controlled by the adversary even revealing a single share can completely break privacy.

The idea is rather than directly computing $x_i y_j$ to compute a secret sharing of this value between the two parties i and j. Then since this value is linear it also works as output share. Specifically we will focus on two parties i and j. We want them to hold a sharing of $x_i y_j \oplus x_j y_i$ at the end of the protocol. We will then add this sharing to the values $x_i y_i$ and $x_j y_j$ held by parties i and j respectively. If we do this for all pairs of parties we will have each party holding a value that when collected add to z (we show this describing how to obtain the sharing). Without loss of generality we assume that $i < j$. The idea is for i to simply pick a random bit (which we call $r_{i,j}$). Our goal is to then give party j

the value $r_{i,j} + x_i y_j + x_j y_i$. Then if these values are later added they will yield $x_i y_j + x_j y_i$. We write the value that j should receive as a partial truth table:

x_j	y_j	$r_{i,j} + x_i y_j + x_j y_i$
0	0	$r_{i,j}$
0	1	$r_{i,j} + x_i$
1	0	$r_{i,j} + y_i$
1	1	$r_{i,j} + x_i + y_i$

We have intentionally defined the truth table in terms of j's values as the value that j should have is defined purely in terms of values that i holds. Thus, it is possible to send the right value to j using a 1-out-of-4 oblivious transfer protocol.

Consider some party i after they have executed this protocol with all other j. The party i now holds n values. These are:

Value Number	Value Held
1	$r_{1,i} + x_1 y_i + x_i y_1$
2	$r_{1,i} + x_2 y_i + x_i y_2$
....	....
i	$x_i y_i$
$i+1$	$r_{i,i+1}$
...	..

Each party then adds the n values they hold (which defines the new share z_i):

$$z_i = r_{1,i} + x_1 y_i + x_i y_1 + ... + r_{i-1,i} + x_{i-1} y_i + x_i y_{i-1} + x_i y_i + r_{i,i+1} + ... + r_{i,n}$$

We note when all of the z_i are added the values $r_{i,j}$ occur exactly twice in the summation. That is,

$$z = \sum_{i=1}^{n} z_i = \sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} \left(\sum_{j=i+1}^{n} x_i y_j + x_j y_i \right).$$

Thus, we have retained our invariant of having a good sharing of the output of the gate.

This protocol is repeated for each gate in the circuit. At the end of the circuit all parties announce their shares and the output can be recovered. This protocol is a semi-honest multi-party computation protocol in the private channel model for any function that is computable in polynomial time. We note that NOT and XOR are very efficient while AND requires a $n-1$ calls of 1-out-of-4 OT for each party yielding $n(n-1)$ calls to OT for each multiplication.

References

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.