CMSC 858K — Introduction to Secure Computation	October 4, 2013
Lecture 13	
Lecturer: Jonathan Katz	*Scribe(s): Adam Groce*

1 Secure Multiparty Computation

The (semi-honest) MPC security definition is very similar to the 2-party case. There are now n parties, up to t of which may be corrupted. The attacker's view is the union of the views of all the corrupted parties. We then require, as in the 2-party case, that a simulator with black box access to the ideal functionality can simulate the view of any attacker. It must be specified whether the attacker can eavesdrop on communication between two honest parties. If so, some communication may need to be encrypted. (This is also relevant in 2-party computation when neither party is corrupted.)

1.1 GMW Protocol

The multiparty GMW protocol is similar to the 2-party case. The invariant to be maintained is that for each wire value b, each party P_i holds b_i such that $b_1 \oplus \ldots \oplus b_n = b$. The protocol guarantees security for $t \leq n$ (though the $t = n$ case is trivial). The protocol consists of several pieces:

1. Input sharing
   - P_i has input x_i.
   - P_i chooses $x_{i,1}, \ldots, x_{i,n}$ uniformly at random subject to $x_{i,1} \oplus \ldots \oplus x_{i,n} = x_i$.
   - P_i sends $x_{i,j}$ to P_j.
2. Evaluating XOR gates
   - Assume gate is computing $a \oplus b$, where a and b are wire values.
   - P_i locally computes $c_i = a_i \oplus b_i$.
   - Commutative property of XOR guarantees $c = a \oplus b$.
3. Evaluating multiplication gates
   - Need $c = ab = (a_1 \oplus \ldots \oplus a_n)(b_1 \oplus \ldots \oplus b_n) = (\bigoplus_i a_i b_i) \oplus (\bigoplus_{i<j} a_i b_j \oplus a_j b_i)$.
   - $a_i b_i$ can be computed locally.
   - For each value $a_i b_j \oplus a_j b_i$ a protocol between each pair of parties. P_i picks $c^i_{i,j}$ at random. For each possible set of values (a_j, b_j), P_i knows what value of $c^j_{i,j}$ will cause $c^i_{i,j} \oplus c^j_{i,j} = a_i b_j \oplus a_j b_i$. The two parties then use 1-out-of-4 OT to let P_j select the correct $c^j_{i,j}$ value.
   - Each P_i now sets $c_i = a_i b_i \oplus \bigoplus_{j \neq i} c^i_{i,j}$.

- Correctness is now guaranteed: $\bigoplus_i c_i = \bigoplus_i \left[a_i b_i \oplus \bigoplus_{i<j}(c^i_{i,j} \oplus c^j_{i,j})\right] = (\bigoplus_i a_i b_i) \oplus (\bigoplus_{i<j} a_i b_j \oplus a_j b_i)$.

4. Output revelation

   - Assume no output wire is used as an input wire. (Easy to make the circuit this way.)
   - If wire value b is part of the output for P_i, all parties send their shares of b to P_i.

We skip the formal proof of security, though an intuitive understanding of why the protocol is secure is very straightforward. Each value that each player has is one of n shares of a meaningful value, and an adversary that controls less than n parties can learn nothing from that. The exception is the intermediate values $c^i_{i,j}$ in the multiplication step, which are one of only two shares. In these cases the adversary can learn the $a_i b_j \oplus a_j b_i)$ value, but this was already independently computable by the adversary, who has access to a_i, a_j, b_i, and b_j.

1.2 Ben-Or, Goldwasser, Wigderson (BGW) Protocol

This protocol gives information-theoretic security. It assumes that the adversary cannot eavesdrop on communication between honest parties and that $t < n/2$.

Computation is done over a field $\mathbb{F}$. We assume that $|\mathbb{F}| > n$ and that $1, \ldots, n$ are elements of F or refer to such elements through some public mapping. Again, each party will hold secret shares of the true value on each wire. Formally, the invariant being maintained is that for any wire with value b, each party P_i holds b_i such that there exists a polynomial f of degree t with $f(i) = b_i$ (for all i) and $f(0) = b$.

1. Input sharing

   - P_i has input x_i.
   - P_i chooses a random degree t polynomial f such that $f(0) = x_i$.
   - P_i sends $f(j)$ to P_j.

2. Evaluating addition gates gates

   - Assume gate is computing $a + b$, where a and b are wire values.
   - P_i locally computes $c_i = a_i + b_i$.
   - The pointwise sum of two degree t polynomials is another degree t polynomial, so correctness holds.
   - Note that this can also be used to multiply by a constant.

3. Evaluating multiplication gates

   - P_i locally multiplies to get $c_i = a_i b_i$. These c_i values now are points on the product polynomial c, but this polynomial is degree $2t$ and no longer random.

- Each P_i then shares its c_i value. As noted above, these shares of shares can be used to locally compute shares of any linear function of the shares c_i.
- Lagrange interpolation gives a formula for computing the intercept of the polynomial c given only the points $\{c_i\}_i$. This formula is a linear function with known constants. Therefore the parties can compute this linear function using the shares of each c_i. The result are shares of the intercept value. Unlike the initial c_i values, these are points on a degree t polynomial. (The randomness of this polynomial is more subtle, but also holds.)

4. Output revelation

   - Assume no output wire is used as an input wire. (Easy to make the circuit this way.)
   - If wire value b is part of the output for P_i, all parties send their shares of b to P_i.