CMSC 858K — Introduction to Secure Computation October 7, 2013

Lecture 14

Lecturer: Jonathan Katz *Scribe(s): Adam Groce*

1 Beaver-Micali-Rogaway Protocol

The protocols we have already discussed for semi-honest MPC have a round complexity that is linear in the depth of the circuit being evaluated. This protocol will instead have constant round complexity. Security is guaranteed for $t < n$. The basic idea is to use a garbled circuit, but instead of a single party computing that locally the parties will run one of the MPC protocols we already know to compute the garbled circuit, then all can evaluate that circuit locally.

Doing the above the most straightforward way will result in round complexity linear in the depth of the circuit that generates garbled circuits. This is constant, but potentially very large. To improve efficiency, we note that creating garbled circuits is a highly parallelizable task. Each gate's garbled table can be computed independently (with the exception of first fixing the keys associated with each wire).

Take as our generic gate one that has wires 1 and 2 as input and wire 3 as output. Wire 1 has two keys associated with it, k_1^0 and k_1^1, associated with values on the wire of 0 and 1, respectively. Similarly, wire 2 has keys k_2^0 and k_2^1 associated with it, and wire 3 has keys k_3^0 and k_3^1. Each gate has a unique gate ID. We will assume this gate has gate ID g. In the two-party setting, the table would consist of four values. For example, the value associated with wires 1 and 2 both having 0 values on them would be $k_3^a \oplus F_{k_1^0}(g|00) \oplus F_{k_2^0}(g|00)$ where a is the value wire 3 should have when both input wires have 0 values. The garbled table would have 4 such values, corresponding to combinations of values for wires 1 and 2, randomly permuted.

In the multiparty setting, we change the keys to now be a combination of subkeys, one controlled by each party. For example, $k_2^0 = (k_{2,1}^0, \ldots, k_{2,n}^0)$, where $k_{2,i}^0$ is held by P_i (and generated randomly by P_i at the start of the protocol). Where we would previously have $F_{k_1^0}(g|00)$ we now have $F_{k_{1,1}^0}(g|00) \oplus \ldots \oplus F_{k_{1,n}^0}(g|00)$. This means that the example table value given before is now

$$k_3^a \oplus F_{k_{1,1}^0}(g|00) \oplus \ldots \oplus F_{k_{1,n}^0}(g|00) \oplus F_{k_{2,1}^0}(g|00) \oplus \ldots \oplus F_{k_{2,n}^0}(g|00).$$

Having the key split up like this between parties means that collectively the players can compute the garbled gates, just as a single player would do in the past, but no player actually knows the full wire keys, which would compromise security.

We now have to figure out how to compute these garbled tables. The simple first idea is to just take each player to have the six relevant inputs, $k_{1,i}^0, k_{1,i}^1, k_{2,i}^0, k_{2,i}^1, k_{3,i}^0$, and $k_{3,i}^1$, and then to use an MPC protocol to compute the garbled table. This works, but the circuit will have depth (and therefore round complexity) linear in the depth of F (and therefore the security parameter).

We can, however, improve the efficiency. For the four input key values, we can let the relevant party compute the pseudorandom function first, and use its output (ex., $F_{k^0_{1,1}}(g|00)$) as the input to the MPC protocol. This means that the MPC protocol is just computing XORs of the inputs and then permuting the four outputs randomly, which is a much simpler circuit. It can be done in reasonable depth, independent of the security parameter.