CMSC 858K — Introduction to Secure Computation	Sept 18, 2013
Lecture 6	
Lecturer: Jonathan Katz	*Scribe(s): Andrew Miller*

1 GMW Protocol, cont'd

Recall that the GMW protocol for securely computing any circuit proceeds in three phases:

1. Input sharing
2. Circuit evaluation (using 1-out-of-4 OT)
3. Output recovery

1.1 Security (in the semi-honest OT-hybrid model)

OT-hybrid model means we have ideal functionalities computing the 1-out-of-4 OT's. Security here is information theoretic. To show security, we construct the following two simulators that simulate respectively each player's view. A simulator of P_1's view is indistinguishable from P_1's view during a correct protocol execution, however it is defined independently of P_2's private inputs, therefore demonstrating that the protocol leaks no information. Likewise for P_2's view.

1.1.1 $Sim_1(x_1, y_1)$:

- 1a) generate random values $r_1, \ldots, r_\ell$ (P_1's randomness)
- 1b) generate random values $s_1, \ldots, s_\ell$ (initial message)
- 2) for each non-input wire of the circuit, generate random r_i
- 3) for each output wire of P_1 let $\hat{r}_1, \ldots, \hat{r}_\ell$ be P_1's shares of the wires
- let $sh_i = \hat{r}_i \oplus y_{1,i}$
- place $sh_1, \ldots, sh_\ell$ in the view

1.1.2 $Sim_2(x_2, y_2)$:

- 1) same as Sim_1
- 2) for each non-input wire of the circuit, choose uniform r_i as the output of the functionality
- 3) same

Note: For Sim_1, x_1 is P_1's input, while y_1 is player P_2's output. However, for Sim_2, x_2 is P_1's output, and y_2 is P_2's input.

1.2 Randomized Computations

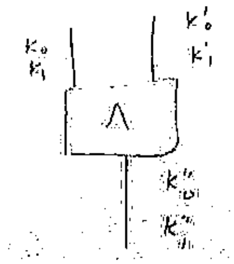
If we can securely compute any deterministic function, then we can securely compute any randomized functionality. Let $g(x, y)$ be a randomized function; we can construct a deterministic equivalent $\hat{g}(x, y, r) = g(x, y)$ where the random coins r are chosen uniformly at random. Then we can securely compute $\overline{g}((x, r_1), (y, r_2)) = \hat{g}(x, y, r_1 \oplus r_2)$ using GMW.

- 1) Each player P_i chooses uniform r_i
- 2) parties compute $\overline{g}$ on inputs (x, r_1) and (y, r_2) respectively

2 Yao's Garbled-Circuits

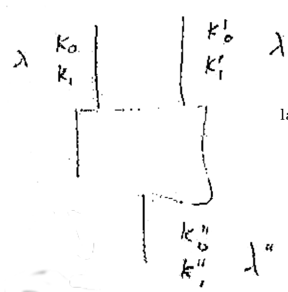
Motivation: GMW protocol has round complexity linear in the depth of the circuit. Yao's garbled-circuit approach has $O(1)$ round complexity, with a pretty small constant.

One party acts as a garbled circuit generator. For each wire, she generates a pair of symmetric encryption keys, corresponding to a possible value (0 or 1). For each gate (assume two input wires, one output wire), she constructs a garbled table representing the truth table for the gate. (Note: the table should be randomly permuted). The following example is for an AND gate:



label$_1$	label$_2$	garbled key
0	0	$Enc_{k_0}(Enc_{k'_0}(k''_0))$
0	1	$Enc_{k_0}(Enc_{k'_1}(k''_0))$
1	0	$Enc_{k_1}(Enc_{k'_0}(k''_0))$
1	1	$Enc_{k_1}(Enc_{k'_1}(k''_1))$

Instead of having to decrypt every row, evaluation can be simplified if the garbled circuit generator also chooses a random bit λ. Thus the label of k_b will be $\lambda \oplus b$. Then the circuit evaluator can use the label to immediately access the correct row of the table.



label$_1$	label$_2$	garbled key
0	0	$Enc_{k_\lambda}(Enc_{k'_{\lambda'}}(k''_{(\lambda\wedge\lambda')\oplus\lambda''}, (\lambda \wedge \lambda') \oplus \lambda''))$
b$_1$	b$_2$	$Enc_{k_{\lambda\oplus b_1}}(Enc_{k'_{\lambda'\oplus b_2}}(k''_{((\lambda\oplus b_1)\wedge(\lambda'\oplus b_2))\oplus\lambda''}, ((\lambda \oplus b_1) \wedge (\lambda' \oplus b_2)) \oplus \lambda'')$

2.1 Garbled-circuit protocol

- 1) Input-preparation phase
- 1a) P_1 sends the input-wire keys corresponding to his inputs
- 1b) P_2 obtains the input-wire keys for its inputs using 1-out-of-2 OT
- 2) Garbled-circuit evaluation
- 3) Output determination:
- - P_1 send λ-values on output wires, P_2 sends output keys
- - P_1 send λ-values on output wires, P_2 sends output keys