CS 276 – Cryptography Oct 29, 2014

Lecture 17: Yao's Garbled Circuit

Instructor: Sanjam Garg *Scribe: Michael McCoyd*

1 Setup

Yao's Garbled Circuits is presented as a solution to Yao's Millionaires' problem, which asks whether two millionaires can compete for bragging rights of which is richer without revealing their wealth to each other. It started the area of secure computation. We will present a solution for the two party problem; it can be extended to a polynomial number of parties, using the techniques from last lecture.

The solution we saw previously needed an interaction for each AND gate. Yao's solution requires only one message, so it provides a constant size of interaction. We present a solution only for semi honest security. This can be amplified to malicious security, but there are more efficient ways of amplifying this than what we saw last lecture.

1.1 Secure Computation

Recall our definition of secure computation. We define ideal and real worlds. Security is defined to hold if anything an attacker can achieve in the real world can also be achieved by an ideal attacker in the ideal world. We define the ideal world to have the properties that we desire. For security to hold these properties must also hold in the real world.

1.2 $(Garble, Eval)$

We will provide a definition, similar to how we define encryption, that allows us avoid dealing with simulators all the time.

Yao's Garbled Circuit is defined as two efficient algorithms $(Garble, Eval)$. Let the circuit C have n input wires. $Garble$ produces the garbled circuit and two labels for each input wire. The labels are for each of 0 and 1 on that wire and are like encryption keys.

$$(\tilde{C}, \{\ell_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow Garble(1^k, C)$$

To evaluate the circuit on a single input we must choose a value for each of the n input wires. Given n of 2n input keys, $Eval$ can evaluate the circuit on those keys and get the circuit result.

$$C(x) \leftarrow Eval(\tilde{C}, \{\ell_{i,x_i}\}_{i\in[n]})$$

Correctness Correctness is as usual, if you garble honestly, evaluation should produce the correct result.

$$\begin{aligned}&\forall C, x\\ &\quad Pr[C(x) = Eval(\tilde{C}, \{l_{i,x_i}\}),\\ &\qquad (\tilde{C}, \{\ell_{i,b}\}) = Garble(1^k, C)] = 1 - neg(k)\end{aligned}$$

Security For security we require that a party receiving a garbled circuit and n inputs labels can not computationally distinguish the joint distribution of the circuits and labels from the distribution produced by a simulator with access to the circuit and its evaluation on the input that the labels represent. The simulator does not have access to the actual inputs. If this holds, the party receiving the garbled circuit and n labels can not determine the inputs.

$$\exists Sim : \forall C, x$$
$$(\tilde{C}, \{\ell_{i,x_i}\}_{i\in[n]}) \simeq Sim(1^k, C, C(x)) \text{ where}$$
$$(\tilde{C}, \{\ell_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow Garble(1^k, C)$$

For simplicity we pass the circuit to the simulator. You could also use universal circuits and pass in with the inputs the specific circuit that the universal circuit should realize.

2 Use for Semi-honest two party secure communication

Alice, with input x^1, and Bob, with input x^2, have a circuit, C, that they want to evaluate securely. The size of their combined inputs is n, so $|x^1| = n_1, |x^2| = n - n_1, |x^1| + |x^2| = n$. They can do this by Alice garbling a circuit and sending input wire labels to Bob, as in Figure **??**.
Alice garbles the circuit and passes it to Bob, $\tilde{C}$. Alice passes the labels for her input directly to Bob, $\{\ell_{i,x_i^1}\}_{i\in[n]/[n_2]}$. Alice passes all the labels for Bob's input wires into oblivious transfer, $\{\ell_{i,b_i}\}_{i\in[n]/[n_1],b\in\{0,1\}}$, from which Bob can retrieve the labels for his actual inputs, $\{\ell_{i,x_i^2}\}_{i\in[n]/[n_1]}$. Bob now has the garbled circuit and one label for each input wire. He evaluates the garbled circuit on those garbled inputs and learns $C(x^1||x^2)$. Bob does not learn anything besides the result as he has only the garbled circuit and n garbled inputs. Alice does not learn anything as she uses oblivious transfer to give Bob his input labels and receives nothing in reply.

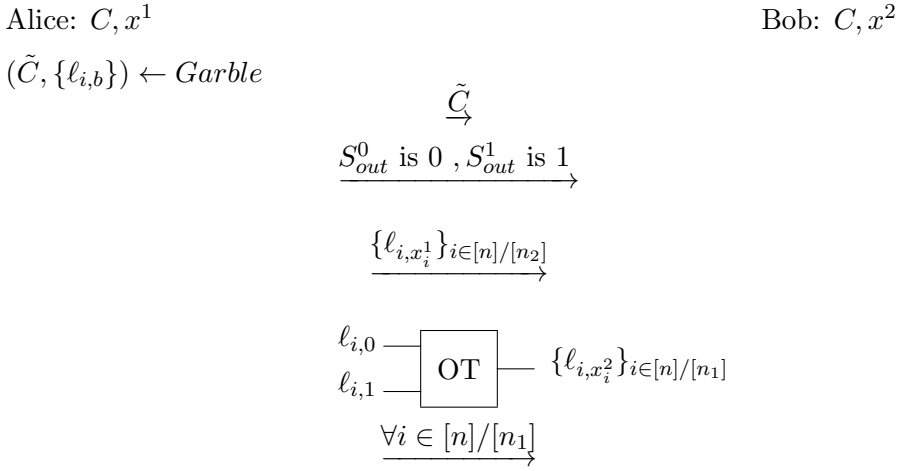

Figure 1: Messages in Yao's Garbeled Circuit

2.1 Construction of Garbled Circuits

We would like to garble a circuit such that there are two keys for each input wire. Correctness should be that given one of the two keys for each wire we can compute the output for the inputs those keys correspond to. Security should be that given one key for each wire you can only learn the output, not the actual inputs.
We build the circuit as a bunch of NAND gates that outputs one bit. If more bits are required, this can be done multiple times. NAND gates can create any logic needed. We define the following sets:

$$W = \text{the set of wires in the circuit}$$
$$G = \text{the set of gates in the circuit.}$$

For each wire in the circuit, sample two keys to label the possible inputs 0 and 1 to the wire

$$\forall w \in W \quad S_w^0, S_w^1 \leftarrow \{0,1\}^k.$$

We can think of these as the secret keys to an encryption scheme (Gen, Enc, Dec). For such a scheme we can always replace the secret key with the random bits fed into Gen.

Wires For each wire in the circuit we will have an invariant that the evaluator can only get one of the wires two encrypted values. Consider an internal wire fed by the evaluation of a gate. The gate receives two encrypted values as inputs and produces one encrypted output. The output will be one of the two labels for that wire and the evaluator will have no way of obtaining the other label for that wire. For example on wire w_i, the evaluator will only learn the value for 1, $S_{w_i}^1$. We ensure this for the input wires by giving the evaluator only one of the two encrypted values for the wire.

Gates For every gate in the circuit we create four cipher texts. For each choice of inputs we encrypt the output key under each of the input keys. Let gate g have inputs w_1, w_2 and output w_3,

$$e_g^{00} = Enc_{S_{w_1}^0}(Enc_{S_{w_2}^0}(S_{w_3}^1, 0^k))$$
$$e_g^{01} = Enc_{S_{w_1}^0}(Enc_{S_{w_2}^1}(S_{w_3}^1, 0^k))$$
$$e_g^{10} = Enc_{S_{w_1}^1}(Enc_{S_{w_2}^0}(S_{w_3}^1, 0^k))$$
$$e_g^{11} = Enc_{S_{w_1}^1}(Enc_{S_{w_2}^1}(S_{w_3}^0, 0^k)).$$

We add k zeros at the end.

Final Output For the final output wire, S_{out}, we can just give out their values,

$$S_{out}^0 \text{ corresponds to } 0$$
$$S_{out}^1 \text{ corresponds to } 1.$$

$\tilde{\mathbf{C}}$ For each gate, Alice sends Bob a random permutation of the set of four encrypted output values.

$$\{e_g^{C_1,C_2}\} \quad \forall g \in G \quad C_1, C_2 \in \{0,1\}.$$

For each gate, Alice sends Bob a random permutation of the set of four encrypted output values

Evaluation With an encrypted gate g, input keys S_{w_1} S_{w_2} for the input wires, and four randomly permuted encryptions of the output keys, $e_g^a, e_g^b, e_g^c, e_g^d$, Bob can evaluate the gate to find the corresponding key S_{w3} for the output wire. Bob can decrypt each of the encrypted output keys until he finds one that decrypts to a string ending in the proper number of 0's, which is very likely to contain the proper output key. We can increase the probability of the correct key by increasing the number of 0's.

$$\exists i \in \{a, b, c, d\} : Dec_{S_{w_2}}(Dec_{S_{w_1}}(e_g^i)) = S_{w_3}, 0^k$$

Given input wire labels $\{\ell_{i,x_i}\}_{i\in[n]}$ the complete encrypted circuit $\tilde{C}$ is evaluated by working up from the input gates.

The evaluator should not be able to infer anything except what they could infer in the ideal world. As a simple example, if the evaluator supplies one input to a circuit of just one NAND gate, they would be able to infer the input of the other party. However, this is true is the ideal world as well.

3 Proof Intuition

What intuition can we offer that the distribution of $\tilde{C}$ with one label per input wire is indistinguishable from what which a simulator could produce with access to the output? For each input wire we are only given one key. As we are doing double encryption, for each input gate we only have the keys needed to decrypt one of the four possible outputs. The other three are protected by semantic security. So from each input gate we learn only one key compounding to its output wire. As the output labels were randomized, we also do not know if that key corresponds to a 0 or a 1. For the next level of gates we again have only one key per input wire, and our argument continues. So for each wire of the circuit we can only know one key corresponding to an output value for the wire. Everything else is random garbage. As we control the mapping from output keys to output values, we can set this to whatever is needed to match the expected output.

Security only holds for evaluation of the circuit with one set of input values and we assume that the circuit is combinatorial and thus acyclic.