CS 600.442 – Modern Cryptography October 26th

Lecture 13: Secure Computation - III

Instructor: Abhishek Jain *Scribe: Neil Fendley*

1 Secure Computation - Yao's Garbled Circuits

We want to answer the question of how Alice and Bob can securely compute *any* function f over their private inputs x and y. In the previous lecture we used the Goldreich-Micali-Wigderson (GMW) Protocol to securely compute functions. This was a highly interactive solution, which naturally extended to any number of parties. We are now going to look at Yao's Garbled Circuits, another technique for securely computing a function. Yao's technique requires little interaction between Alice and Bob, but only works with two parties. In order to examine Yao's Garbled Circuit technique, we must first define what Garbled circuits are.

Definition 1 *A Garbling Scheme consists of two procedures, Garble and Eval:*

- *Garble(C): Takes a circuit C as input and will output a collection of garbled gates* $\widehat{G}$ *and garbled input wires* $\widehat{In}$ *where*
$$\widehat{G} = \{\widehat{g}_1, \ ... \ , \widehat{g}_{|c|}\}$$
$$\widehat{In} = \{\widehat{in}_1, \ ... \ , \widehat{in}_n\}$$
- *Eval(*$\widehat{G}$*,* $\widehat{In}_x$*): Takes as input a garbled circuit* $\widehat{G}$ *and garbled input wires* $\widehat{In}$ *corresponding to an input* x *and outputs* $z = C(x)$

Now we will outline how Garbling Schemes work.

- Each wire i in the circuit C is associated with two keys (k_0^i, k_1^i) of a secret-key encryption scheme, one corresponding to the wire value being 0 and other for wire value being 1
- For an input x, the evaluator is given the input wire keys $(k_{x_1}^1, \ ... \ , k_{x_n}^n)$ corresponding to x. Also for every gate $g \ \in C$, it is also given an encrypted truth table of g, which is something we will show later.
- We want the evaluator to use the input wire keys and the encrypted truth tables to uncover a single key k_v^i for every internal wire i corresponding to the value v of that wire. However, k_{1-v}^i should remain hidden from the evaluator.

In order to implement this we will have to define a special encryption scheme.

Definition 2 **Special Encryption Scheme** : *We need a secret-key encryption scheme* (Gen, Enc, Dec) *with an extra property: there exists a negligible function* $\nu(\cdot)$ *s.t. for every* n *and every message* $m \ \in \{0,1\}^n$,

$$Pr[k \leftarrow Gen(1^n), k' \leftarrow Gen(1^n), Dec_k(Enc_k(m)) = \perp] \ < \ 1 - \nu(n)$$

Essentially this is saying if a ciphertext is decrypted using a different or "wrong" key, then answer is always $\perp$

Construction : In order to create this special secret encryption simply modify the secret-key encryption scheme discussion in the secret lecture, except instead of encryption m, we encrypt $0^n||m$. Upon decrypting we check if the first n bits of the message are all 0's; if they aren't we output $\perp$

2 Garbled Circuits Construction

We are now going to define Garble and Eval for our Garbled Circuit. Let (Gen, Enc, Dec) be a special encryption scheme (as defined above). Assign an index to each wire in C s.t. the input wires have indices $1, ..., n$.

Garble(C):

- For every non-output wire i in C, sample $k_0^i \leftarrow Gen(1^n)$, $k_1^i \leftarrow Gen(1^n)$. For every output wire i in C, set $k_0^i = 0$, $k_1^i = 1$.
- For every $i \in [n]$, set $\widehat{in}_i = (k_0^i, k_1^i)$. Set $\widehat{In} = (\widehat{in}_1, \ ... \ , \widehat{in}_n)$
- For every gate g in C with input wire (i

First Input	Second Input	Output
k_0^i	k_0^j	$z_1 = Enc_{k_0^i}(Enc_{k_0^j}(k^l_{g(0,0)}))$
k_0^i	k_1^j	$z_2 = Enc_{k_0^i}(Enc_{k_1^j}(k^l_{g(0,1)}))$
k_1^i	k_0^j	$z_3 = Enc_{k_1^i}(Enc_{k_0^j}(k^l_{g(1,0)}))$
k_1^i	k_1^j	$z_4 = Enc_{k_1^i}(Enc_{k_1^j}(k^l_{g(1,1)}))$

Set $\widehat{g}$ = RandomShuffle(z_1, z_2, z_3, z_4). Output $(\widehat{G} = (\widehat{g}_1, \ ... \ , \widehat{g}_{|C|}), \widehat{In})$

Why is the random shuffle necessary? If we do not randomly shuffle the outputs an Adversary would know information about the combination of k_i, k_j used to achieve the output just based on the index of the returned value.

Eval($\widehat{G}, \widehat{In}_x$):

- Parse $\widehat{G} = (\widehat{g}_1, \ ... \ , \widehat{g}_{|C|})$. $\widehat{In}_x = (k^1, \ ... \ , k^n)$
- Parse $\widehat{g}_i = (\widehat{g}_1, \ ... \ , \widehat{g}_4)$
- Decrypt each garbled gate $\widehat{g}_i$ one-by-one in canonical order:
  - Let k^i and k^j be the input wire keys for gate g.
  - Repeat the following for every $p \in [4]$:

$$\alpha_p = Dec_{k^i}(Dec_{k^j}(\widehat{g}_i^p))$$

    if $\exists \alpha_p \neq \perp$, set $k^l = \alpha_p$

- Let out_i be the value obtained for each output wire i. Output $out = (out_1, \ ... \ , out_n)$

3 Secure Computation from Garbled Circuits

Let us discuss a plausible approach for securely computing $C(x, y)$ using Garbled Circuits.

A generates a garbled circuit for $C(\cdot, \cdot)$ along with garbled wire keys for first and second input to C. It then sends the garbled wire keys corresponding to its input x along with the garbled circuit to B. Note, however, that in order to evaluate the garbled circuit on (x, y),B also needs the garbled wire keys corresponding to its input y.

A possible solution is for A to send all the wire keys corresponding to the second input of C to B. At first, this may seem to be a good idea. However this would mean B can not only compute $C(x, y)$ but also $C(x, y')$ for any y' of its choice. This is clearly an insecure solution!

To solve this problem A will transmit the garbled wire keys corresponding to B's input by using oblivious transfer. Below, we describe the solution in detail.

Ingredients: Garbling Scheme (Garble, Eval), 1-out-of-2 OT scheme OT = (S, R) as defined in previous lecture on secure computation.

Common Input: Circuit C for $f(\cdot, \cdot)$

A**'s input:** $x = x_1, \ ... \ , x_n$

B**'s input:** $y = y_1, \ ... \ , y_n$

Protocol $Pi = (A, B)$**:**

$A \to B$	A computes $(\widehat{G}, \widehat{In})$ Parse $\widehat{In} = (\widehat{in}_1, \ ... \ , \widehat{in}_{2n})$ where $\widehat{in}_i = (k_0^i, k_1^i)$. Set $\widehat{In}_x = (k_{x_1}, \ ... \ , k_{x_n}^n)$. Send $(\widehat{G}, \widehat{In}_x)$ to B
$A \leftrightarrow B$	For every $i \in [n]$, A and B run OT $= (S, R)$ where A plays sender S with input (k_0^{n+i}, k_1^{n+i}) and B plays reciever R with in put y_i. Let $\widehat{In}_y = (k_{y_1}^{n+1}, \ ... \ , k_{y_n}^{2n})$ be the outputs of the n OT executions received by B.
B	B outputs Eval$(\widehat{G}, \widehat{In}_x, \widehat{In}_y)$

In order to argue the security of the construction, we use two properties.

Property 1: For every wire i, B only learns one of the two wire keys:

- Input Wires: For input wires corresponding to $A's$ input, it follows from protocol description. For input wires corresponding to B's input it follows from security of OT
- Internal Wires: Follows from the security of the encryption scheme

Property 2: B does not know whether the key corresponds to wire value being 0 or 1 (except the keys corresponding to its own input wires).

From this we can notice that B only learns the output and nothing else. A does not learn anything (in particular, B's input remains hidden from A due to the security of OT). The full proof of security can be found in [1].

References

[1] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. *IACR Cryptology ePrint Archive*, 2004:175, 2004.