Secure Computation - II

CS 601.642/442 Modern Cryptography

Fall 2019
Securely Computing *any* Function

Main question: How can Alice and Bob securely compute *any* function $f$ over their private inputs $x$ and $y$?

Solution: Using Yao’s *garbled circuits* with OT
Garbled Circuits

A Garbling Scheme consists of two procedures (\texttt{Garble}, \texttt{Eval}):

- \texttt{Garble}(\mathcal{C})$: Takes as input a circuit $\mathcal{C}$ and outputs a collection of garbled gates $\hat{\mathcal{G}}$ and garbled input wires $\hat{\ln}$ where

$$\hat{\mathcal{G}} = \{ \hat{g}_1, \ldots, \hat{g}_{|\mathcal{C}|} \},$$

$$\hat{\ln} = \{ \hat{\ln}_1, \ldots, \hat{\ln}_n \}.$$

- \texttt{Eval}(\hat{\mathcal{G}}, \hat{\ln}_x)$: Takes as input a garbled circuit $\hat{\mathcal{G}}$ and garbled input wires $\hat{\ln}_x$ corresponding to an input $x$ and outputs $z = \mathcal{C}(x)$
Garbled Circuits: Ideas

- 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 $(k_{x_1}^1, \ldots, k_{x_n}^n)$ corresponding to $x$. Furthermore, for every gate $g$ in $C$, it is also given an “encrypted” truth table of $g$.

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

\[
\Pr[k \leftarrow \text{Gen}(1^n), k' \leftarrow \text{Gen}(1^n), \text{Dec}_{k'}(\text{Enc}_k(m)) = \bot] > 1 - \nu(n)
\]

That is, if a ciphertext is decrypted using the “wrong” key, then the answer is always \(\bot\).

Construction: Modify the secret-key encryption scheme discussed earlier in the class s.t. instead of encrypting \(m\), we encrypt \(0^n\|m\). Upon decrypting, check if the first \(n\) bits of the message are all 0’s; if not, then output \(\bot\).
Garbled Circuits: Construction

Let \((\text{Gen, Enc, Dec})\) be a special encryption scheme. Assign an index to each wire in \(C\) s.t. the input wires have indices 1, \ldots, \(n\).

\textbf{Garble}(\(C\)):

- For every non-output wire \(i\) in \(C\), sample \(k_0^i \leftarrow \text{Gen}(1^n), k_1^i \leftarrow \text{Gen}(1^n)\). For every output wire \(i\) in \(C\), set \(k_0^i = 0, k_1^i = 1\).
- For every \(i \in [n]\), set \(\text{in}_i = (k_0^i, k_1^i)\). Set \(\text{ln} = (\text{in}_1, \ldots, \text{in}_n)\).
- For every gate \(g\) in \(C\) with input wires \((i, j)\), output wire \(\ell\):

<table>
<thead>
<tr>
<th>First Input</th>
<th>Second Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_0^i)</td>
<td>(k_0^j)</td>
<td>(z_1 = \text{Enc}<em>{k_0^i}(\text{Enc}</em>{k_0^j}(k_0^\ell g(0,0))))</td>
</tr>
<tr>
<td>(k_0^i)</td>
<td>(k_1^j)</td>
<td>(z_2 = \text{Enc}<em>{k_0^i}(\text{Enc}</em>{k_1^j}(k_0^\ell g(0,1))))</td>
</tr>
<tr>
<td>(k_1^i)</td>
<td>(k_0^j)</td>
<td>(z_3 = \text{Enc}<em>{k_1^i}(\text{Enc}</em>{k_0^j}(k_1^\ell g(1,0))))</td>
</tr>
<tr>
<td>(k_1^i)</td>
<td>(k_1^j)</td>
<td>(z_4 = \text{Enc}<em>{k_1^i}(\text{Enc}</em>{k_1^j}(k_1^\ell g(1,1))))</td>
</tr>
</tbody>
</table>

Set \(\hat{g} = \text{RandomShuffle}(z_1, z_2, z_3, z_4)\). Output \((\hat{G} = (\hat{g}_1, \ldots, \hat{g}_{|C|}), \hat{\text{ln}})\).
Think: Why is RandomShuffle necessary?

Eval(\(\hat{G}, \hat{n}_x\)):

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

\[
\alpha_p = \text{Dec}_{k^i}(\text{Dec}_{k^j}(\hat{g}^p_i))
\]

If \(\exists \alpha_p \neq \bot\), set \(k^\ell = \alpha_p\)

- Let \(\text{out}_i\) be the value obtained for each output wire \(i\). Output \(\text{out} = (\text{out}_1, \ldots, \text{out}_n)\)
Secure Computation from Garbled Circuits

A plausible strategy for 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 $C$
- $A$ sends the garbled wire keys corresponding to its input $x$ along with the garbled circuit to $B$
- However, in order to evaluate the garbled circuit on $(x, y)$, $B$ also needs the garbled wire keys corresponding to its input $y$
- **Possible Solution:** $A$ sends all the wire keys corresponding to the second input of $C$ to $B$
- **Problem:** In this case, $B$ can not only compute $C(x, y)$ but also $C(x, y')$ for any $y'$ of its choice!
- **Solution:** $A$ will transmit the garbled wire keys corresponding to $B$’s input using Oblivious Transfer!
Secure Computation from Garbled Circuits: Details

**Ingredients:** Garbling scheme (Garble, Eval), 1-out-of-2 OT scheme OT = (S, R)

**Common Input:** Circuit C for f(·, ·)

**A’s input:** x = x₁, …, xₙ, **B’s input:** y = y₁, …, yₙ

**Protocol Π = (A, B):**

**A → B:** A computes (G hat, n hat) ← Garble(C). Parse n hat = (i n₁, …, i n₂ᴺ) where i nᵢ = (kᵢ⁰, kᵢ¹). Set n hatₓ = (kᵢ¹ₓ₁, …, kᵢ¹ₓₙ). Send (G hat, n hatₓ) to B.

**A ↔ B:** For every i ∈ [N], A and B run OT = (S, R) where A plays sender S with input (kᵢⁿ+i⁰, kᵢⁿ+i¹) and B plays receiver R with input yᵢ. Let n hatᵧ = (kᵢⁿ⁺¹ᵧ₁, …, kᵢⁿ⁺¹ᵧₙ) be the outputs of the N OT executions received by B.

**B:** B outputs Eval(G hat, n hatₓ, n hatᵧ)
Intuition for Security

**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).

Overall, $B$ only learns the output and nothing else. $A$ does not learn anything (in particular, $B$’s input remains hidden from $A$ due to security of OT)

**Additional Reading:** Read security proof from [Lindell-Pinkas’04]