MIT 6.875 & Berkeley CS276

Secure two-party computation and Yao Garbled Circuits

Lecture 24

Credit: Some slides adapted from Gil Segev
In this lecture...

Recording...

Secure two-party computation:
- Paradigm
- Security definition for semi-honest adversaries
- Construction via Yao garbled circuits
Alice and Bob want to know who is richer without revealing their inputs to each other. How can they compute $f(x, y)$?
The paradigm of secure computation

Alice and Bob hold inputs $x$ and $y$ and wish to compute $f(x, y)$

Goal: no one learns anything about $x$ or $y$ other than $f(x, y)$

Adversarial models:

- **Semi-honest/honest-but-curious:** Each party follows the protocol, but tries to learn additional information from the transcript
- **Malicious:** Parties can behave arbitrarily, even deviate from the protocol in order to learn additional information
The paradigm of secure computation

Alice and Bob hold inputs $x$ and $y$ and wish to compute $f(x, y)$

Goal: no one learns anything about $x$ or $y$ other than $f(x, y)$

How would you define this?

simulation paradigm
Notation

- \(\langle A(x), B(y)\rangle(1^n)\) is the distribution of the transcript on inputs \(x\) and \(y\)
- \(\text{out}_A[\langle A(x), B(y)\rangle(1^n)]\) is the distribution of \(A\)'s output
- \(\text{out}_B[\langle A(x), B(y)\rangle(1^n)]\) is the distribution of \(B\)'s output
- \(\text{view}_A[\langle A(x), B(y)\rangle(1^n)]\) is the distribution of \(A\)'s view: random tape \(r_A\) and transcript
- \(\text{view}_B[\langle A(x), B(y)\rangle(1^n)]\) is the distribution of \(B\)'s view: random tape \(r_B\) and transcript
Security in the semi-honest model

Definition: An efficient protocol \( \langle A, B \rangle \) securely computes a deterministic function \( f = (f_1, f_2) \) in the semi-honest model if there exist PPT simulators \( S_A \) and \( S_B \) such that for every \( \{x, y\} \in \{0,1\}^* \), the following hold:

Correctness:
\[
\Pr[out_A[\langle A(x), B(y)\rangle(1^n)], out_B[\langle A(x), B(y)\rangle(1^n)]] = f(x, y) = 1
\]

Security against semi-honest Alice:
\[
S_A(x, f_1(x, y)) \approx_c \text{view}_A(\langle A(x), B(y)\rangle)
\]

Security against semi-honest Bob:
Symmetric
Security in the Semi-Honest Model

Theorem (Yao ‘86):
Assuming the existence of a secure Oblivious Transfer protocol in the semi-honest model, any efficiently-computable deterministic two-output function can be securely computed in the semi-honest model.

\[ f(x, y) = (f_1(x, y), f_2(x, y)) \]

- Groundbreaking result initiating research on secure computation
- Inspired fundamental protocols for the multi-party & malicious models
- Various applications beyond secure computation
Tools to recall

- Oblivious Transfer (OT)
- CPA-secure privacy-key encryption scheme
Recall: Oblivious Transfer (OT)

- Sender holds two bits $x_0$ and $x_1$.
- Receiver holds a choice bit $b$.
- Receiver should learn $x_b$, sender should learn nothing.
“Special” CPA Encryption

- We will use a CPA-secure private-key encryption scheme \((G, E, D)\) with two additional properties.
- Notation: \(\text{Range}_n(k) \overset{\text{def}}{=} \{E_k(x) : x \in \{0,1\}^n\}\)

**Property 1: Elusive range**
For every PPT algorithm \(A\) there exists a negligible function \(\nu(\cdot)\) such that
\[
\Pr_{k \leftarrow G(1^n)}[A(1^n) \in \text{Range}_n(k)] \leq \nu(n)
\]

**Property 2: Efficiently verifiable range**
There exists a PPT algorithm \(M\) such that \(M(1^n, k, c) = 1\) if and only if \(c \in \text{Range}_n(k)\)

Ideas how to construct?
Construction

**Property 1: Elusive range**
For every PPT algorithm $A$ there exists a negligible function $\nu(\cdot)$ such that

$$\Pr_{k \leftarrow G(1^n)} [A(1^n) \in \text{Range}_n(k)] \leq \nu(n)$$

**Property 2: Efficiently verifiable range**
There exists a PPT algorithm $M$ such that $M(1^n, k, c) = 1$ if and only if $c \in \text{Range}_n(k)$

**Construction:**
- Let $F$ be a PRF where $F_k : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ for $k \in \{0,1\}^n$

$$E_k(x; r) = (r, F_k(r) \oplus x0^n)$$

Why does it satisfy the two properties?
Boolean circuits

Gates are Boolean gates (AND, XOR, OR) taking as input two bits and outputting one bit

• How would you express the millionaire’s
  \[ f(x, y) = x > y \]
  as a Boolean circuit \( C \)?
The Millionaires’ Function as a Circuit

\[ f(x, y) = 1 \text{ if and only if } x > y \]

Unit Vector \( u_x = 1 \) in the \( x^{th} \) location and 0 elsewhere

Vector \( v_y = 1 \) from the \( (y + 1)^{th} \) location onwards

\[ f(x, y) = \langle u_x, v_y \rangle = \sum_{i=1}^{U} u_x[i] \land v_y[i] \]

An AND for each \( u_i, v_i \), then OR between all results in a tree-like fashion
Or use comparison circuit

\[ f(x, y) = 1 \]
if and only if \( x > y \)

Magnitude comparator for 2-bit numbers

\( x_1 \) \( x_2 \) \( y_1 \) \( y_2 \)

\( x < y \)
\( x = y \)
\( x > y \)
Garbling Boolean Circuits

- **Input**: Boolean circuit $C: \{0,1\}^n \rightarrow \{0,1\}$
- **Output**: Garbled circuit $G(C)$ and input labels $\{(L_1^0, L_1^1), \ldots, (L_n^0, L_n^1)\}$

**Goal**: Given $G(C)$ and $L_1^{x_1}, \ldots, L_n^{x_n}$

- It is possible to compute $C(x_1 \cdots x_n)$
- It is not possible to learn any additional information other than size of circuit or input

For example, for $x = 010$, labels are $L_1^0, L_2^1, L_3^0$
Using garbled circuits for secure 2-party computation

Input will be \( x, y \)

Common input: \( C: \{0,1\}^{2n} \to \{0,1\} \)

Garbled circuit \( G(C) \)

Input labels \( L_1^{x_1}, \ldots, L_n^{x_n} \) for \( x \)

OT for each \( i \in [n] \) in parallel:
- Alice’s input: \( (L_{n+i}^0, L_{n+i}^1) \)
- Bob’s input: \( y_i \)

Compute \( C(x, y) \) using \( G(C) \) and \( L_1^{x_1}, \ldots, L_n^{x_n}, L_1^{y_1}, \ldots, L_{2n}^{y_n} \)
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
Garbled circuit
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)

- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(g(0,0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>(g(0,1))</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(g(1,0))</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(g(1,1))</td>
</tr>
</tbody>
</table>
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows

<table>
<thead>
<tr>
<th>(L_u^0)</th>
<th>(L_v^0)</th>
<th>(L_w^{g(0,0)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_u^0)</td>
<td>(L_v^1)</td>
<td>(L_w^{g(0,1)})</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^0)</td>
<td>(L_w^{g(1,0)})</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^1)</td>
<td>(L_w^{g(1,1)})</td>
</tr>
</tbody>
</table>
The garbling procedure
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows

<table>
<thead>
<tr>
<th>(L_u^0)</th>
<th>(L_v^0)</th>
<th>(L_w^{g(0,0)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_u^0)</td>
<td>(L_v^1)</td>
<td>(L_w^{g(0,1)})</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^0)</td>
<td>(L_w^{g(1,0)})</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^1)</td>
<td>(L_w^{g(1,1)})</td>
</tr>
</tbody>
</table>

Why can’t I leave the output labels this way? Because they leak (e.g. type of gate)
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_u^0)</td>
<td>(L_v^0)</td>
<td>(E_{L_u^0} \left( E_{L_v^0} \left( L_w^{g(0,0)} \right) \right))</td>
</tr>
<tr>
<td>(L_u^0)</td>
<td>(L_v^1)</td>
<td>(E_{L_u^0} \left( E_{L_v^1} \left( L_w^{g(0,1)} \right) \right))</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^0)</td>
<td>(E_{L_u^1} \left( E_{L_v^0} \left( L_w^{g(1,0)} \right) \right))</td>
</tr>
<tr>
<td>(L_u^1)</td>
<td>(L_v^1)</td>
<td>(E_{L_u^1} \left( E_{L_v^1} \left( L_w^{g(1,1)} \right) \right))</td>
</tr>
</tbody>
</table>
The garbling procedure

- Assign two random labels \((L_w^0, L_w^1)\) to each wire \(w\)
  - \(L_w^0 \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_w^1 \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows

\((G, E, D)\) has elusive & efficiently verifiable range

Given \(L_u^\alpha\) and \(L_v^\beta\) can identify the row corresponding to inputs \((\alpha, \beta)\) and compute \(L_w^{g(\alpha,\beta)}\)
The garbling procedure

- Assign two random labels \((L^0_w, L^1_w)\) to each wire \(w\)
  - \(L^0_w \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L^1_w \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows
- Construct an output translation table

<table>
<thead>
<tr>
<th>0</th>
<th>(L^0_{\text{out}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(L^1_{\text{out}})</td>
</tr>
</tbody>
</table>

Can handle any number of output wires by constructing a table for each one
The garbling procedure

- Assign two random labels \((L_0^w, L_1^w)\) to each wire \(w\)
  - \(L_0^w \leftarrow G(1^n)\) corresponds to value 0 on wire \(w\)
  - \(L_1^w \leftarrow G(1^n)\) corresponds to value 1 on wire \(w\)
- For each gate \(g\) construct a doubly-encrypted translation table with randomly permuted rows
- Construct an output translation table
- Output all tables
The garbling procedure
Yao’s protocol

Common input: $C: \{0,1\}^{2n} \rightarrow \{0,1\}$

Garbled circuit $G(C)$

Input labels $L_1^{x_1}, \ldots, L_n^{x_n}$ for $x$

Input: $y \in \{0,1\}^n$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_0^{n+i}, L_1^{n+i})$
- Bob’s input: $y_i$

Compute $C(x, y)$ using $G(C)$ and $L_1^{x_1}, \ldots, L_n^{x_n}, L_1^{y_1}, \ldots, L_2^{y_n}$

Input: $x \in \{0,1\}^n$

Compute $G(C)$ and labels $\{(L_i^0, L_i^1)\}_{i \in [2n]}$
Recall: Security in the semi-honest model

**Definition:** An efficient protocol \(\langle A, B \rangle\) securely computes a deterministic function \(f = (f_1, f_2)\) in the semi-honest model if there exist PPT simulators \(S_A\) and \(S_B\) such that for every \(\{x, y\} \in \{0,1\}^*\), the following hold:

**Correctness:**

\[
\Pr[out_A[\langle A(x), B(y)\rangle(1^n)], out_B[\langle A(x), B(y)\rangle(1^n)] = f(x, y)] = 1
\]

**Security against semi-honest Alice:**

\[S_A(x, f_1(x, y)) \approx_c view_A(\langle A(x), B(y)\rangle)\]

**Security against semi-honest Bob:** symmetric
Alice’s simulator

Input: $x \in \{0,1\}^n$

Compute $G(C)$ and labels $\{(L_i^0, L_i^1)\}_{i \in [2n]}$

Garbled circuit $G(C)$

Input labels $L_1^{x_1}, \ldots, L_n^{x_n}$ for $x$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_{n+i}^0, L_{n+i}^1)$
- Bob’s input: $y_i$

$C(x, y)$
Bob’s simulator

Garbled circuit $G(C)$

Input labels $L_{1}^{x_{1}}, \ldots, L_{n}^{x_{n}}$ for $x$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_{n+i}^{0}, L_{n+i}^{1})$
- Bob’s input: $y_i$

Input: $y \in \{0,1\}^n$

Compute $C(x, y)$ using $G(C)$ and $L_{1}^{x_{1}}, \ldots, L_{n}^{x_{n}}, L_{n+1}^{y_{1}}, \ldots, L_{2n}^{y_{n}}$

$C(x, y)$
Bob’s simulator: step 1

Replace Bob’s view in the OTs with the assumed OT simulator $S^\text{OT}_B$

Indistinguishable from Bob’s original view by the security of the OT (standard hybrid argument over the $n$ OTs)

From this point on, $S_B$ needs to know $L^y_{n+i}$ but does not use $L^{1-y}_{n+i}$

Garbled circuit $G(C)$

Input labels $L^x_1, ..., L^x_n$ for $x$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L^0_{n+i}, L^1_{n+i})$
- Bob’s input: $y_i$

Compute $C(x, y)$ using $G(C)$ and $L^x_1, ..., L^x_n, L^y_{n+1}, ..., L^y_{2n}$
Bob’s simulator: step 2

Replace $G(C)$ with an indistinguishable $\tilde{G}(C)$ that evaluates to $C(x, y)$ on all input labels.

Intuition: Bob should not notice that $\tilde{G}(C)$ computes a constant function since he knows only one of $(L_{n+i}^0, L_{n+i}^1)$ by the security of the OT.

Garbled circuit $G(C)$

Input labels $L_1^{x_1}, ..., L_n^{x_n}$ for $x$

$C(x, y)$

Input: $y \in \{0,1\}^n$

Compute $C(x, y)$ using $G(C)$ and $L_1^{x_1}, ..., L_n^{x_n}, L_{n+1}^{y_1}, ..., L_{2n}^{y_n}$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_{n+i}^0, L_{n+i}^1)$
- Bob’s input: $y_i$
Bob’s simulator: step 1

Replace Bob’s view in the OTs with the assumed OT simulator $S_B^{OT}$

Can now replace $L_1^{x_1}, ..., L_n^{x_n}$ with $L_1^0, ..., L_n^0$

This view can be generated given $y$ and $C(x, y)$, and without knowing $x$

Garbled circuit $G(C)$

Input labels $L_1^{x_1}, ..., L_n^{x_n}$ for $x$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_0^{0}, L_1^{i})$
- Bob’s input: $y_i$

Input: $y \in \{0, 1\}^n$

Compute $C(x, y)$ using $G(C)$ and $L_1^{x_1}, ..., L_n^{x_n}, L_{n+1}^{y_1}, ..., L_{2n}^{y_n}$
The fake $\tilde{G}(C)$

- Assign two random labels $(L_w^0, L_w^1)$ to each wire $w$
- For each gate $g$ construct a randomly permuted translation table doubly-encrypting the zero label

<table>
<thead>
<tr>
<th>Real table</th>
<th>Fake table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{L_u^1} \left( E_{L_v^1} \left( L_w^{g(1,1)} \right) \right)$</td>
<td>$E_{L_u^1} \left( E_{L_v^1} \left( L_w^0 \right) \right)$</td>
</tr>
<tr>
<td>$E_{L_u^0} \left( E_{L_v^0} \left( L_w^{g(0,0)} \right) \right)$</td>
<td>$E_{L_u^0} \left( E_{L_v^0} \left( L_w^0 \right) \right)$</td>
</tr>
<tr>
<td>$E_{L_u^0} \left( E_{L_v^1} \left( L_w^{g(0,1)} \right) \right)$</td>
<td>$E_{L_u^0} \left( E_{L_v^1} \left( L_w^0 \right) \right)$</td>
</tr>
<tr>
<td>$E_{L_u^1} \left( E_{L_v^0} \left( L_w^{g(1,0)} \right) \right)$</td>
<td>$E_{L_u^1} \left( E_{L_v^0} \left( L_w^0 \right) \right)$</td>
</tr>
</tbody>
</table>
Leverage CPA security of $\mathbf{(G, E, D)}$

Real and fake tables are indistinguishable because only one label is known from each pair $(L^0_u, L^1_u)$ and $(L^0_v, L^1_v)$

(Subtle hybrid argument due to dependencies between tables corresponding to different gates)

<table>
<thead>
<tr>
<th>Real table</th>
<th>Fake table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{L^1_u} \left( E_{L^1_v} \left( L^g_{w}(1,1) \right) \right)$</td>
<td>$E_{L^1_u} \left( E_{L^1_v} \left( L^0_{w} \right) \right)$</td>
</tr>
<tr>
<td>$E_{L^0_u} \left( E_{L^0_v} \left( L^g_{w}(0,0) \right) \right)$</td>
<td>$E_{L^0_u} \left( E_{L^0_v} \left( L^0_{w} \right) \right)$</td>
</tr>
<tr>
<td>$E_{L^0_u} \left( E_{L^1_v} \left( L^g_{w}(0,1) \right) \right)$</td>
<td>$E_{L^0_u} \left( E_{L^1_v} \left( L^0_{w} \right) \right)$</td>
</tr>
<tr>
<td>$E_{L^1_u} \left( E_{L^0_v} \left( L^g_{w}(1,0) \right) \right)$</td>
<td>$E_{L^1_u} \left( E_{L^0_v} \left( L^0_{w} \right) \right)$</td>
</tr>
</tbody>
</table>
The fake $\tilde{G}(C)$

- Assign two random labels $(L_w^0, L_w^1)$ to each wire $w$
- For each gate $g$ construct a randomly permuted translation table doubly-encrypting the zero label
- Construct an output translation table where $L_{out}^0$ is translated to $C(x, y)$

<table>
<thead>
<tr>
<th></th>
<th>$L_{out}^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$C(x, y)$</td>
</tr>
<tr>
<td>1</td>
<td>$1 - C(x, y)$</td>
</tr>
</tbody>
</table>
The fake $\tilde{G}(C)$

- Assign two random labels $(L^0_w, L^1_w)$ to each wire $w$
- For each gate $g$ construct a randomly permuted translation table doubly-encrypting the zero label
- Construct an output translation table where $L^0_{out}$ is translated to $C(x, y)$
- Output all tables
Bob’s simulator

Garbled circuit $\tilde{G}(C)$

Input labels $L_1^0, \ldots, L_n^0$ for $x$

For each $i \in [n]$ invoke $S_B^{OT}(y_i, L_{n+i}^{y_i})$

$C(x, y)$

Input: $y \in \{0,1\}^n$

Compute $C(x, y)$ using $\tilde{G}(C)$ and $L_1^0, \ldots, L_n^0, L_{n+1}^{y_1}, \ldots, L_{2n}^{y_n}$
Yao’s protocol

Common input: $C: \{0,1\}^{2n} \rightarrow \{0,1\}$

Garbled circuit $G(C)$

Input labels $L_1^{x_1}, \ldots, L_n^{x_n}$ for $x$

OT for each $i \in [n]$ in parallel:
- Alice’s input: $(L_{n+i}^0, L_{n+i}^1)$
- Bob’s input: $y_i$

Compute $C(x, y)$

Input: $x \in \{0,1\}^n$
Compute $G(C)$ and labels $\{(L_i^0, L_i^1)\}_{i \in [2n]}$

Input: $y \in \{0,1\}^n$
Compute $C(x, y)$ using $G(C)$ and $L_1^{x_1}, \ldots, L_n^{x_n}, L_{n+1}^{y_1}, \ldots, L_{2n}^{y_n}$

Theorem:
Yao’s protocol securely computes any $C: \{0,1\}^{2n} \rightarrow \{0,1\}$ in the semi-honest model
Efficiency

- Garbling and evaluation tend to be very efficient because it can be implemented via AES, which is in hardware
- Creating a circuit from a program often results in a big circuit
Questions

Q: Say Alice and Bob want to compare their European and US funds. Can they reuse the garbled circuit?

A: No! Yao garbled circuits are one-time. Insecure with multiple input encodings.

Your three instructors had a paper (STOC’11) on how to reuse garbled circuits. Great proof of concept but a very inefficient scheme with nesting of heavy schemes like FHE or ABE.

Q: What are two inputs that reveal all values of \( f(x, y) \)?

A: 00000... and 11111.. because Bob receives all possible labels.
Summary

We learned about secure two-party computation
- definition for semi-honest adversaries, and
- a construction via Yao garbled circuits and OT