Session 2: The Yao Construction and its Proof of Security

Yehuda Lindell
Bar-Ilan University
Yao’s Protocol

- Protocol for general secure two-party computation
  - Constant number of rounds
  - Secure for semi-honest adversaries
  - Many applications of the methodology beyond secure computation

- General secure computation
  - Can be used to securely compute any functionality
  - Based on the **Boolean circuit** for computing the function
Outline

- **Garbled circuit**
  - An encrypted circuit together with a pair of keys \((k_0, k_1)\) for every input wire so that given one key on every wire:
    - It is possible to compute the output (based on the input determined by the key provided on every wire)
    - It is not possible to learn anything else

- **Oblivious transfer**
  - Sender has \(x_0, x_1\); receiver has \(b\)
  - Receiver obtains \(x_b\) only
  - Sender learns nothing
Outline

- Yao’s protocol
  - Party P₁ constructs a **garbled circuit**
  - P₁ sends P₂ the keys associated with its input on its own input wires
    - P₁ sends only the keys so P₂ doesn’t know what the actual input is
  - P₁ and P₂ use oblivious transfer so that for every one of P₂’s input wires:
    - P₂ obtains the correct key associated with its input
    - P₁ learns nothing about P₂’s input
  - P₂ computes the circuit and receives the output, and sends it back to P₁
Oblivious Transfer – Background

- Trapdoor permutation (I,D,F,F⁻¹)
  - I: samples a function f and trapdoor t in the family
  - D(f): uniformly samples a value in the domain of f
  - F(f,x): computes f(x)
  - F⁻¹(t,y): computes f⁻¹(y)
  - Hard to invert a random y, given f (but not t)

- Enhanced trapdoor permutations
  - Hard to invert y, even given the random coins used to sample y (using D)
Oblivious Transfer – Background

- **Hard-core predicate B**
  - Given $y=f(x)$, can guess $B(x)$ with probability only negligibly greater than $\frac{1}{2}$
  - Equivalently, given $y=f(x)$, the bit $B(x)$ is pseudorandom
Oblivious Transfer Protocol

- **Sender’s input:** \((z_0, z_1)\); receiver’s input \(b\)
- **Sender’s first message:**
  - Sender chooses \((f, t)\) using sampling algorithm \(I\)
  - Sender sends \(f\) to receiver
- **Receiver’s first message:**
  - Receiver chooses \(x_b\) and computes \(y_b = f(x_b)\)
  - Receiver chooses random \(y_{1-b}\)
  - Receiver sends \((y_0, y_1)\) to sender
- **Sender’s second message:**
  - Sender computes \((x_0, x_1)\) by inverting
  - Sender computes \(a_i = z_i \oplus B(x_i)\)
  - Sender sends \((a_0, a_1)\) to receiver
- **Receiver outputs** \(z_b = a_b \oplus x_b\)
Oblivious Transfer Protocol

\[ S(z_0, z_1) \]

Choose \((f, t)\)

\[ x_0 = f^{-1}(y_0) \]
\[ a_0 = z_0 \oplus B(x_0) \]

\[ x_1 = f^{-1}(y_1) \]
\[ a_1 = z_1 \oplus B(x_1) \]

\[ R(b) \]

Choose \(b\)

Choose \(x_b\), compute \(y_b = f(x_b)\)

Choose \(y_{1-b}\)

Output \(z_b = a_b \oplus B(x_b)\)
Security – P₁ Corrupted

- Simulator is given \((z₀,z₁)\); there is no output
  - SIM generates \((f,t)\)
  - SIM chooses random \(y₀, y₁\) using \(D(f)\)
  - SIM computes \(a₀, a₁\) as in sender’s instructions

- The transcript is exactly like a real protocol execution
  - Choosing \(x_b\) using \(D(f)\) and computing \(y_b = f(x_b)\) is identical to choosing \(y_b\) using \(D(f)\)
Security – P₂ Corrupted

- Simulator is given \((b, z_b)\)
  - SIM generates \((f, t)\)
  - SIM chooses random \(x_b, y_{1-b}\) using \(D(f)\)
  - SIM computes \(y_b = f(x_b)\)
  - SIM computes \(a_b = B(x_b) \oplus z_b\)
  - SIM chooses \(a_{1-b}\) at random

- The transcript is indistinguishable from a real execution
  - By the hard-core property of \(B\) and the enhancement property of TDP, \(B(x_{1-b})\) is indistinguishable from random
A Garbled Circuit

- For the entire circuit, assign random values/keys to each wire (key $k_0$ for 0, key $k_1$ for 1)
- Encrypt each gate, so that given one key for each input wire, can compute the appropriate key on the output wire
An AND Gate

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
An AND Gate with Garbled Values

\[
\begin{array}{c|c|c}
 u & v & w \\
\hline
 k_0^u & k_0^v & k_0^w \\
 k_1^u & k_1^v & k_0^w \\
 k_1^u & k_0^v & k_0^w \\
 k_1^u & k_1^v & k_1^w \\
\end{array}
\]
A Garbled AND Gate

<table>
<thead>
<tr>
<th>U</th>
<th>V</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k^0_u$</td>
<td>$k^0_v$</td>
<td>$E_{k^0_u} (E_{k^1_v} (k^0_w))$</td>
</tr>
<tr>
<td>$k^0_u$</td>
<td>$k^1_v$</td>
<td>$E_{k^0_u} (E_{k^0_v} (k^0_w))$</td>
</tr>
<tr>
<td>$k^1_u$</td>
<td>$k^0_v$</td>
<td>$E_{k^1_u} (E_{k^0_v} (k^0_w))$</td>
</tr>
<tr>
<td>$k^1_u$</td>
<td>$k^1_v$</td>
<td>$E_{k^1_u} (E_{k^1_v} (k^1_w))$</td>
</tr>
</tbody>
</table>
A Garbled AND Gate

- The actual garbled gate

\[ E_{k_u^1} (E_{k_v^0} (k_w^0)) \]
\[ E_{k_u^0} (E_{k_v^1} (k_w^0)) \]
\[ E_{k_u^1} (E_{k_v^1} (k_w^1)) \]
\[ E_{k_u^0} (E_{k_v^0} (k_w^1)) \]

- Given \( k_u^0 \) and \( k_v^1 \) can obtain \( k_w^0 \) only
- Furthermore, since the table is permuted, the party has no idea if it obtained the 0 or 1 key
Output Translation

- If the gate is an output gate, need to provide the “decryption” of the output wire
- Output translation table

\[
\begin{align*}
& [(0, k_0^w), (1, k_1^w)] \\
& k_0^u \quad k_1^u \quad k_0^v \quad k_1^v \\
& u \quad v
\end{align*}
\]
Constructing a Garbled Circuit

- **Given a Boolean circuit**
  - Assign garbled values to all wires
  - Construct garbled gates using the garbled values

- **Central property:**
  - Given a set of garbled values, one for each input wire, can compute the entire circuit, and obtain garbled values for the output wires
  - Given a translation table for the output wires, can obtain output
  - But, nothing but the output is learned!
An Example Circuit
(input wires $P_1 = d,a$; $P_2 = b,e$)

\[
\left(0, k_f^0 \right), \left(1, k_f^1 \right) \quad \left(0, k_g^0 \right), \left(1, k_g^1 \right)
\]

```
| $E_{k_d^0} (E_{k_c^0} (k_f^0))$ |
| $E_{k_d^0} (E_{k_c^1} (k_f^0))$ |
| $E_{k_d^1} (E_{k_c^0} (k_f^0))$ |
| $E_{k_d^1} (E_{k_c^1} (k_f^0))$ |

AND

```

```
| $E_{k_d^0} (E_{k_c^0} (k_f^0))$ |
| $E_{k_d^0} (E_{k_c^1} (k_f^0))$ |
| $E_{k_d^1} (E_{k_c^0} (k_f^0))$ |
| $E_{k_d^1} (E_{k_c^1} (k_f^0))$ |

AND

```

```
| $E_{k_d^0} (E_{k_c^0} (k_g^0))$ |
| $E_{k_d^0} (E_{k_c^1} (k_g^0))$ |
| $E_{k_d^1} (E_{k_c^0} (k_g^0))$ |
| $E_{k_d^1} (E_{k_c^1} (k_g^0))$ |

OR

```

```
| $E_{k_d^0} (E_{k_c^0} (k_g^0))$ |
| $E_{k_d^0} (E_{k_c^1} (k_g^0))$ |
| $E_{k_d^1} (E_{k_c^0} (k_g^0))$ |
| $E_{k_d^1} (E_{k_c^1} (k_g^0))$ |
```
Computing a Garbled Circuit

- How does the party computing the circuit know which is the “correct” entry
  - It has one key on each wire, but symmetric encryption may decrypt “correctly” even with incorrect keys

- Two possibilities (actually many…)
  - Use encryption based on a PRF with redundant zeroes; only correct keys give redundant block
  - Add a bit to signal which ciphertext to decrypt
Computing a Garbled Circuit

- **Option 1:**
  - Encryption: $E_K(m) = [r, F_K(r) \oplus (m||0^n)]$
  - By pseudorandomness of $F$, probability of obtaining $0^n$ with an incorrect $K$ is negligible

- **Option 2:**
  - For every wire, choose a random signal bit together with the keys
Computing a Garbled Circuit

- The actual garbled gate

\[
\begin{align*}
(0,0) & \rightarrow E_{k_u^1} (E_{k_v^0} (k_{w}^0 || 1)) \\
(1,1) & \rightarrow E_{k_u^0} (E_{k_v^1} (k_{w}^0 || 1)) \\
(0,1) & \rightarrow E_{k_u^1} (E_{k_v^1} (k_{w}^1 || 0)) \\
(1,0) & \rightarrow E_{k_u^0} (E_{k_v^0} (k_{w}^0 || 1))
\end{align*}
\]

- Advantage

  ◦ Computing the circuit requires just two decryptions per gate (rather than an average of 5)
Double-Encryption Security

- Need to formally prove that given 4 encryptions of a garbled gate and only 2 keys
  - Nothing is learned beyond one output
- Actually, in order to simulate the protocol, we need something stronger
- Notation:
  - Double encryption: $\overline{E}(k_u, k_v, m) = E_{k_u}(E_{k_v}(m))$
  - Oracle: $\overline{E}(:, k_v, :)$
Double-Encryption Security

\[ \text{Expt}^{\text{double}}_{\mathcal{A}}(n, \sigma) \]

1. The adversary \( \mathcal{A} \) is invoked upon input \( 1^n \) and outputs two keys \( k_0 \) and \( k_1 \) of length \( n \) and two triples of messages \((x_0, y_0, z_0)\) and \((x_1, y_1, z_1)\) where all messages are of the same length.
2. Two keys \( k'_0, k'_1 \leftarrow G(1^n) \) are chosen for the encryption scheme.
3. \( \mathcal{A} \) is given the challenge ciphertext \( \bar{E}(k_0, k'_1, x_\sigma), \bar{E}(k'_0, k_1, y_\sigma), \bar{E}(k'_0, k'_1, z_\sigma) \) as well as oracle access to \( \bar{E}(\cdot, k'_1, \cdot) \) and \( \bar{E}(k'_0, \cdot, \cdot) \)
4. \( \mathcal{A} \) outputs a bit \( b \) and this is taken as the output of the experiment.
Yao’s Protocol

- **Input:** x and y of length n
- P₁ generates a garbled circuit G(C)
  - k₀₀, k₀¹ are the keys on wire w₁
  - Let w₁,…,wₙ be the input wires of P₁ and wₙ₊₁,…,w₂ₙ be the input wires of P₂
- P₁ sends P₂ the strings k₁ₓ₁,..., kₙₓₙ
- P₁ and P₂ run n OTs in parallel
  - P₁ inputs kₙ₊ᵢ₀, kₙ₊ᵢ¹
  - P₂ inputs yᵢ
- Given all keys, P₂ computes G(C) and obtains C(x,y)
  - P₂ sends result to P₁
The Example Circuit
(input wires $P_1 = d, a; P_2 = b, e$)

\[
\left(0, k^0_f \right), \left(1, k^1_f \right) \quad \left(0, k^0_g \right), \left(1, k^1_g \right)
\]

AND

\[
E_{k^0_d} \left( E_{k^0_c} \left( k^0_f \right) \right) \\
E_{k^0_d} \left( E_{k^1_c} \left( k^0_f \right) \right) \\
E_{k^1_d} \left( E_{k^0_c} \left( k^0_f \right) \right) \\
E_{k^1_d} \left( E_{k^1_c} \left( k^1_f \right) \right)
\]

OR

\[
E_{k^0_g} \left( E_{k^0_c} \left( k^0_f \right) \right) \\
E_{k^0_g} \left( E_{k^1_c} \left( k^1_f \right) \right) \\
E_{k^1_g} \left( E_{k^0_c} \left( k^0_f \right) \right) \\
E_{k^1_g} \left( E_{k^1_c} \left( k^1_f \right) \right)
\]

AND

\[
E_{k^0_a} \left( E_{k^0_b} \left( k^0_c \right) \right) \\
E_{k^0_a} \left( E_{k^1_b} \left( k^0_c \right) \right) \\
E_{k^1_a} \left( E_{k^0_b} \left( k^0_c \right) \right) \\
E_{k^1_a} \left( E_{k^1_b} \left( k^1_c \right) \right)
\]

OT
Proof of Security – $P_1$ Corrupted

- Party $P_1$’s view consists only of the messages it receives in the oblivious transfers.
- In the OT-hybrid model, $P_1$ receives no messages in the oblivious transfers.
- Simulation:
  - Generate an empty transcript.
More difficult case

- Need to construct a fake garbled circuit $G(C')$ that looks indistinguishable to $G(C)$
- Simulated view contains keys to input wires and $G(C')$
- $G(C')$ together with the keys computes to $f(x,y)$
- Simulator doesn’t know $x$, so cannot generate a real garbled circuit
Proof of Security – P2 Corrupted

- Simulator
  - Given y and z = f(x,y), construct a fake garbled circuit G'(C) that always outputs z
    - Do this by choosing wire keys as usual, but encrypting the same output key in all ciphertexts
      \[ \text{E}_{k_u^1}(\text{E}_{k_v^0}(k_w^0)) \quad \text{E}_{k_u^1}(\text{E}_{k_v^1}(k_w^0)) \]
      \[ \text{E}_{k_u^0}(\text{E}_{k_v^1}(k_w^0)) \quad \text{E}_{k_u^0}(\text{E}_{k_v^0}(k_w^0)) \]
    - This ensures that no matter the input, the same known garbled values on the output wires are received
Simulator (continued)

- Simulation of output translation tables
  - Let $k,k'$ be the keys on the $i^{th}$ output wire; let $k$ be the key encrypted in the preceding gate
  - If $z_i = 0$, write $[(0,k),(1,k')]$
  - If $z_i = 1$, write $[(0,k'),(1,k)]$

- Simulation of input keys phase
  - Input wires associated with $P_1$'s input: send any one of the two keys on the wire
  - Input wires associated with $P_2$'s input: simulate output of OT to be any one of the two keys on the wire
Proof of Security – $P_2$ Corrupted

- Need to prove that the simulation is indistinguishable from the real

- First step – modify simulator as follows
  - Given $x$ and $y$ (just for the sake of the proof), label all keys on the wires as **active** or **inactive**
    - **active**: key is obtained on this wire upon inputs $(x, y)$
    - **inactive**: key is **not** obtained on wire upon inputs $(x, y)$
  - The single key to be encrypted in each gate is the **active** one

- This simulation is identical
Proof of Security – P_2 Corrupted

- Proven by a hybrid argument
  - Consider a garbled circuit G_L(C) for which:
    - The first L gates are generated as in the (alternative) simulation
    - The rest of the gates are generated honestly
- Claim: G_{L-1}(C) is indistinguishable from G_L(C)
- Proof:
  - Difference is in L^{th} gate
  - Intuition: use indistinguishability of encryptions to say that cannot distinguish real garbled gate from one where same key is encrypted
Proof of Security – $P_2$ Corrupted

- **Observation – $L^{th}$ gate**
  - The encryption under both active keys is identical in both cases
  - The difference is what the inactive keys encrypt (only the next active key, or also the inactive)
    - The triple in the experiment are all encryptions under inactive keys

- **The problem**
  - The inactive keys in this gate may appear in other gates as well
    - Use oracles to generate rest...
The Example Circuit

(input wires $P_1 = d, a; P_2 = b, e$)

\[
[(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)]
\]

\[
E_{k_d^0}(E_{k_c^0}(k_{f}^0)) \quad E_{k_d^0}(E_{k_c^0}(k_{f}^1))
\]

\[
E_{k_d^1}(E_{k_c^0}(k_{f}^0)) \quad E_{k_d^1}(E_{k_c^0}(k_{f}^1))
\]

\[
E_{k_c^0}(E_{k_c^0}(k_{f}^0)) \quad E_{k_c^0}(E_{k_c^0}(k_{f}^1))
\]

\[
E_{k_c^1}(E_{k_c^0}(k_{f}^0)) \quad E_{k_c^1}(E_{k_c^0}(k_{f}^1))
\]

\[
E_{k_c^0}(E_{k_c^0}(k_{g}^0)) \quad E_{k_c^0}(E_{k_c^0}(k_{g}^1))
\]

\[
E_{k_c^1}(E_{k_c^0}(k_{g}^0)) \quad E_{k_c^1}(E_{k_c^0}(k_{g}^1))
\]
Simulator’s Circuit (Output 01)

\[
[(0, k^0_f), (1, k^1_f)] \quad [(0, k^0_g), (1, k^1_g)]
\]
Inactive Keys
Input (da=01, be=10), Output (fg=01)

\[
\left[\left(0, k_0^\ell\right), \left(1, k_1^\ell\right)\right] \quad \left[\left(0, k_0^g\right), \left(1, k_1^g\right)\right]
\]
Inactive Keys

Input (da=01, be=10), Output (fg=01)

\[
\left[ (0, k^0_f), (1, k^1_f) \right] \quad \left[ (0, k^0_g), (1, k^1_g) \right]
\]

Diagram:

- AND gates with inputs and outputs labeled with keys.
- Example: \( E_{k^0_d} \) and \( E_{k^1_c} \) for different keys.

Secrecy and security implications discussed.
Alternative Simulator
(Encrypt Active Keys Only)

\[
\left[(0, k_f^0), (1, k_f^1)\right], \quad \left[(0, k_g^0), (1, k_g^1)\right]
\]

Note change in encrypted key
Hybrid on OR Gate – Simulated OR

\[ [(0, k^0_f), (1, k^1_f)] \]
\[ [(0, k^0_g), (1, k^1_g)] \]
Hybrid on OR Gate – Real OR

\[
\begin{align*}
\text{REAL:} & \quad \left(0, k_f^0 \right), \left(1, k_f^1 \right) \\
\text{SIM:} & \quad \left(0, k_g^0 \right), \left(1, k_g^1 \right)
\end{align*}
\]
What’s the Difference

- In the simulated OR case, the inactive key $k_c^0$ encrypts the key $k_g^1$
- In the real OR case, the inactive key $k_c^0$ encrypts the key $k_g^0$
- Indistinguishability follows from the indistinguishability of encryptions under the inactive key $k_c^0$
othing Indistinguishability

Follows from the indistinguishability of encryptions under the inactive key $k_c^0$

The good news
- Key $k_c^0$ is not encrypted anywhere (as data) because prior gates are simulated

The bad news
- The key $k_c^0$ needs to be used to construct the real AND gate for the hybrid

The solution
- The special double-encryption CPA game
**Double-Encryption Security**

The adversary $A$ is invoked upon input $1^n$ and outputs two keys $k_0$ and $k_1$ of length $n$ and two triples of messages $(x_0, y_0, z_0)$ and $(x_1, y_1, z_1)$ where all messages are of the same length.

Two keys $k'_0, k'_1 \leftarrow G(1^n)$ are chosen for the encryption scheme.

$A$ is given the challenge ciphertext $\{E(k_0, k'_1, x_\sigma), E(k'_0, k_1, y_\sigma), E(k'_0, k'_1, z_\sigma)\}$ as well as oracle access to $E(\cdot, k'_1, \cdot)$ and $E(k'_0, \cdot, \cdot)$.

$A$ outputs a bit $b$ and this is taken as the output of the experiment.

- $k_0, k_1$ ($k_c^1, k_e^0$) are active keys
- $k'_0, k'_1$ ($k_c^0, k_e^1$) are inactive keys
- Can use oracle to generate the REAL AND gate
Proof of Security – \( P_2 \) Corrupted

Since each gate-replacement is indistinguishable, using a hybrid argument we have that the distributions are indistinguishable.

QED
2–4 rounds (depending on OT and if both or one party receives output)  
8|C| oblivious transfers  
8|C| symmetric encryptions to generate circuit and 2|C| to compute it (using the signal bit)  
For circuit of 33,000 gates:  
- Between 7 and 14 seconds  
- Between 503 and 3162 Kbytes  
  (depends on encryption used)
Assume that the OT is secure for malicious adv:

- A corrupted $P_1$ cannot learn **anything** (it receives no messages in the protocol, in the hybrid–OT model)
  - Thus, we have **privacy**
- We can prove **full security** for the case of a corrupted $P_2$

This can be useful, but…

- Be warned that this doesn’t compose with anything
- E.g., consider $P_1$ that builds circuit so that if $P_2$’s first bit is 0, the circuit doesn’t decrypt
  - If $P_1$ can detect this in the real world, privacy is lost
Summary

Can compute any functionality securely in presence of semi-honest adversaries

Protocol is efficient enough for use, for circuits that are not too large

Recommendation: read full proof