Secure
2-Party Computation

Lecture 14
Yao’s Garbled Circuit
Secure (and correct) if:

\[ \forall F \exists \text{s.t.} \forall \text{output of is distributed identically in REAL and IDEAL} \]
Passive Adversary

- Gets **only read access** to the internal state of the corrupted players (and can use that information in talking to environment)
- Also called “Honest-But-Curious” adversary
- Will require that **simulator also corrupts passively**
- Simplifies several cases
  - e.g. coin-tossing [why?], commitment [coming up]
- Oddly, sometimes security against a passive adversary is more demanding than against an active adversary
  - Active adversary: too pessimistic about what guarantee is available even in the IDEAL world
  - e.g. 2-party SFE for OR, with output going to only one party (trivial against active adversary; impossible without computational assumptions against passive adversary)
Oblivious Transfer

Pick one out of two, without revealing which

Intuitive property: transfer partial information "obliviously"

$x_0, x_1 \rightarrow b \leftarrow x_b$
An OT Protocol (passive corruption)
An OT Protocol (passive corruption)

Using (a special) encryption
An OT Protocol
(passive corruption)

Using (a special) encryption
An OT Protocol
(passive corruption)

Using (a special) encryption

\[(SK_b, PK_b) \leftarrow \text{KeyGen}
\text{Sample PK}_{1:b}\]
An OT Protocol (passive corruption)

Using *(a special)* encryption

PKE in which one can sample a public-key without knowing secret-key
An OT Protocol (passive corruption)

Using (a special) encryption

PKE in which one can sample a public-key without knowing secret-key

\[(SK_b, PK_b) \leftarrow \text{KeyGen}
\]

\[x_0, x_1 \xrightarrow{b} \]

\[x_0, x_1 \xrightarrow{b} \]

\[PK_0, PK_1 \xleftarrow{b} \]
An OT Protocol (passive corruption)

Using *(a special)* encryption

PKE in which one can sample a public-key without knowing secret-key

\[(SK_b, PK_b) \leftarrow \text{KeyGen}
\]

Sample \( PK_{1-b} \)

\[ c_0 = \text{Enc}(x_0, PK_0) \]
\[ c_1 = \text{Enc}(x_1, PK_1) \]

\[ x_0, x_1 \]

\[ x_0, x_1 \]

\[ PK_0, PK_1 \]
An OT Protocol
(passive corruption)

Using (a special) encryption

PKE in which one can sample a public-key without knowing secret-key
Using (a special) encryption PKE in which one can sample a public-key without knowing secret-key

An OT Protocol (passive corruption)

![Diagram of an OT Protocol with symbols and equations:]

\[x \in \{0, 1\}\]

\[b \in \{0, 1\}\]

\[c_0 = \text{Enc}(x_0, PK_0)\]

\[c_1 = \text{Enc}(x_1, PK_1)\]

\[(SK_b, PK_b) \leftarrow \text{KeyGen}\]

\[\text{Sample } PK_{1-b}\]

\[x_b = \text{Dec}(c_b; SK_b)\]
An OT Protocol (passive corruption)

Using *(a special)* encryption

PKE in which one can sample a public-key without knowing secret-key

- $(SK_b, PK_b) \leftarrow \text{KeyGen}$
- Sample $PK_{1:b}$
- $c_0 = \text{Enc}(x_0, PK_0)$
- $c_1 = \text{Enc}(x_1, PK_1)$
- $x_b = \text{Dec}(c_b; SK_b)$
An OT Protocol (passive corruption)

Using (a special) encryption

PKE in which one can sample a public-key without knowing secret-key

$c_{1-b}$ inscrutable to a passive corrupt receiver

$(SK_b, PK_b) \leftarrow$ KeyGen
Sample $PK_{1-b}$

$c_0 = Enc(x_0, PK_0)$
$c_1 = Enc(x_1, PK_1)$

$x_0, x_1 \rightarrow b \rightarrow$ x

$x_0, x_1 \rightarrow b \rightarrow$ x

$(PK_0, PK_1) \leftarrow c_0, c_1$

$x_b = Dec(c_b; SK_b)$
An OT Protocol (passive corruption)

Using (a special) encryption

PKE in which one can sample a public-key without knowing secret-key

c_{1-b} inscrutable to a passive corrupt receiver

Sender learns nothing about b

\[
\begin{align*}
(SK_b, PK_b) &\leftarrow \text{KeyGen} \\
\text{Sample } PK_{1-b} \\
\end{align*}
\]

\[
\begin{align*}
PK_0, PK_1 &\leftarrow x_0, x_1 \\
c_0, c_1 &\leftarrow Enc(x_0, PK_0, x_1, PK_1) \\
x_b &\leftarrow Dec(c_b; SK_b)
\end{align*}
\]
An OT Protocol
(passive corruption)
An OT Protocol
(passive corruption)

Using a Trapdoor OWP
An OT Protocol
(passive corruption)

Using a Trapdoor OWP
An OT Protocol (passive corruption)

Using a Trapdoor OWP
An OT Protocol (passive corruption)

Using a Trapdoor OWP

Pick \((f, f^{-1})\)

\[ f(x_0, x_1) \]

\[ b \]
An OT Protocol (passive corruption)

Using a Trapdoor OWP

\[ \begin{align*}
&\text{Pick } (f,f^{-1}) \\
&\text{Let } r_b = f(s_b)
\end{align*} \]
Using a Trapdoor OWP

An OT Protocol (passive corruption)

\[ \text{Pick } (f, f^{-1}) \]

\[ \text{pick } s_b, r_{1-b} \]

\[ \text{let } r_b = f(s_b) \]

\[ x_0, x_1 \]

\[ b \]
An OT Protocol (passive corruption)

Using a Trapdoor OWP

\[
\begin{align*}
\text{Pick } (f,f^{-1}) \\
\text{let } s_i = f^{-1}(r_i) \\
z_i = x_i \oplus B(s_i)
\end{align*}
\]
An OT Protocol (passive corruption)

Using a Trapdoor OWP

Pick \((f,f^{-1})\)

Let \(s_i = f^{-1}(r_i)\)

\[ z_i = x_i \oplus B(s_i) \]

Pick \((f,f^{-1})\)

Let \(r_b = f(s_b)\)
An OT Protocol
(passive corruption)

Using a Trapdoor OWP

\[
\begin{align*}
\text{Pick } (f, f^{-1}) \\
\text{let } s_i &= f^{-1}(r_i) \\
z_i &= x_i \oplus B(s_i)
\end{align*}
\]
An OT Protocol
(passive corruption)

Using a Trapdoor OWP

\[
\begin{align*}
\text{Pick} & \quad (f, f^{-1}) \\
\text{let } x_i & = f^{-1}(r_i) \\
z_i & = x_i \oplus B(s_i)
\end{align*}
\]
An OT Protocol (passive corruption)

Using a **Trapdoor OWP**

For passive corrupt receiver: $z_{1-b}$ looks random

**Protocol Details**

**Pick** $(f, f^{-1})$

- $s_i = f^{-1}(r_i)$
- $z_i = x_i \oplus B(s_i)$

**Let** $r_0, r_1$

**Let** $r_b = f(s_b)$

**Pick** $s_b, r_{1-b}$

$f$

$x_0, x_1$

$b$
Using a **Trapdoor OWP**

For passive corrupt receiver: $z_{1-b}$ looks random

Learns nothing about $b$
**2-Party SFE**

Secure Function Evaluation (SFE) IDEAL:

- Trusted party takes \((X;Y)\). Outputs \(g(X;Y)\) to Alice, \(f(X;Y)\) to Bob
2-Party SFE

Secure Function Evaluation (SFE) IDEAL:

- Trusted party takes \((X;Y)\). Outputs \(g(X;Y)\) to Alice, \(f(X;Y)\) to Bob

- Randomized Functions: \(g(X;Y;r)\) and \(f(X;Y;r)\) s.t. neither party knows \(r\) (beyond what is revealed by output)
2-Party SFE

Secure Function Evaluation (SFE) IDEAL:

- Trusted party takes $(X;Y)$. Outputs $g(X;Y)$ to Alice, $f(X;Y)$ to Bob

- Randomized Functions: $g(X;Y;r)$ and $f(X;Y;r)$ s.t. neither party knows $r$ (beyond what is revealed by output)

- OT is an instance of a (deterministic) 2-party SFE
Secure Function Evaluation (SFE) IDEAL:

- Trusted party takes \((X;Y)\). Outputs \(g(X;Y)\) to Alice, \(f(X;Y)\) to Bob

- Randomized Functions: \(g(X;Y;r)\) and \(f(X;Y;r)\) s.t. neither party knows \(r\) (beyond what is revealed by output)

- OT is an instance of a (deterministic) 2-party SFE

\(g(x_0, x_1; b) = \text{none}; f(x_0, x_1; b) = x_b\)
2-Party SFE

Secure Function Evaluation (SFE) IDEAL:

- Trusted party takes \((X;Y)\). Outputs \(g(X;Y)\) to Alice, \(f(X;Y)\) to Bob

Randomized Functions: \(g(X;Y;r)\) and \(f(X;Y;r)\) s.t. neither party knows \(r\) (beyond what is revealed by output)

OT is an instance of a (deterministic) 2-party SFE

- \(g(x_0,x_1;b) = \text{none}; f(x_0,x_1;b) = x_b\)

**Single-Output SFE**: only one party gets any output
2-Party SFE

Can reduce any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = ( g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2) ) \]. Compute \( f'(X, M, r_1; Y, r_2) \) with random \( M, r_1, r_2 \)

Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice
2-Party SFE

Can **reduce** any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = (g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2) ). \]

Compute \( f'(X, M, r_1; Y, r_2) \) with random \( M, r_1, r_2 \)

Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice

Passive secure
2-Party SFE

Can reduce any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = (g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2)) \]. Compute
\[ f'(X, M, r_1; Y, r_2) \] with random \( M, r_1, r_2 \)

Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice

Passive secure

For active security, \( f' \) authenticates (one-time MAC) as well as encrypts \( g(X; Y; r_1 \oplus r_2) \) using keys input by Alice
2-Party SFE

Can reduce any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = (g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2)). \]
Compute
\[ f'(X, M, r_1; Y, r_2) \]
with random \( M, r_1, r_2 \)

Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice

Passive secure

For active security, \( f' \) authenticates (one-time MAC) as well as encrypts \( g(X; Y; r_1 \oplus r_2) \) using keys input by Alice

Generalizes to more than 2 parties
2-Party SFE

Can reduce any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = (g(X, Y; r_1 \oplus r_2) \oplus M, f(X, Y; r_1 \oplus r_2)). \]

Compute \( f'(X, M, r_1; Y, r_2) \) with random \( M, r_1, r_2 \)

Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice

Passive secure

For active security, \( f' \) authenticates (one-time MAC) as well as encrypts \( g(X, Y; r_1 \oplus r_2) \) using keys input by Alice

Generalizes to more than 2 parties

Can reduce any single-output deterministic SFE to OT!
“Completeness” of OT
"Completeness" of OT

Can reduce any single-output deterministic SFE to OT!
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
- No computational assumptions needed
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
- No computational assumptions needed
- For passive security
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
- No computational assumptions needed
- For passive security
- Proof of concept for 2 parties: An inefficient reduction
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
- No computational assumptions needed
- For passive security
- Proof of concept for 2 parties: An inefficient reduction
- Yao’s garbled circuit for 2 parties
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
  - No computational assumptions needed
  - For passive security
    - Proof of concept for 2 parties: An inefficient reduction
    - Yao’s garbled circuit for 2 parties
  - “Basic GMW”: Information-theoretic reduction to OT (next time)
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
  - No computational assumptions needed
  - For passive security
    - Proof of concept for 2 parties: An inefficient reduction
    - Yao’s garbled circuit for 2 parties
    - “Basic GMW”: Information-theoretic reduction to OT (next time)
  - Fact: OT is complete even for active security
“Completeness” of OT: Proof of Concept

Single-output 2-party function $f$

- Alice (who knows $x$, but not $y$) prepares a table for $f(x, \cdot)$ with $N = 2^{|y|}$ entries (one for each $y$)
- Bob uses $y$ to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)
“Completeness” of OT: Proof of Concept

- Single-output 2-party function \( f \)

- Alice (who knows \( x \), but not \( y \)) prepares a table for \( f(x, \cdot) \) with \( N = 2^{|y|} \) entries (one for each \( y \))

- Bob uses \( y \) to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)

- Bob learns only \( f(x,y) \) (in addition to \( y \)). Alice learns nothing beyond \( x \).
“Completeness” of OT: Proof of Concept

- Single-output 2-party function $f$

- Alice (who knows $x$, but not $y$) prepares a table for $f(x, \cdot)$ with $N = 2^{|y|}$ entries (one for each $y$)

- Bob uses $y$ to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)

- Bob learns only $f(x,y)$ (in addition to $y$). Alice learns nothing beyond $x$.

- Problem: $N$ is exponentially large in $|y|$
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
Functions as Circuits

- Directed acyclic graph

- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)

- Edges: Boolean valued wires

- Each wire comes out of a unique gate, but a wire might fan-out

- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits
Functions as Circuits

e.g.: OR (single gate, 2 input bits, 1 bit output)
Functions as Circuits

- **e.g.: OR** (single gate, 2 input bits, 1 bit output)

- **e.g.: X > Y** for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  
  \[(x_1 \land \neg y_1) \lor (\neg (x_1 \oplus y_1) \land (x_0 \land \neg y_0))\]
Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: $X > Y$ for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  $$(x_1 \land \neg y_1) \lor (\neg (x_1 \oplus y_1) \land (x_0 \land \neg y_0))$$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size.
Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: $X > Y$ for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  $$(x_1 \land \neg y_1) \lor (\neg (x_1 \oplus y_1) \land (x_0 \land \neg y_0))$$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size
- Can convert any ("efficient") program into a ("small") circuit
Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: \( X > Y \) for two bit inputs \( X=x_1x_0, Y=y_1y_0: \)
  \[
  (x_1 \land \neg y_1) \lor (\neg(x_1 \oplus y_1) \land (x_0 \land \neg y_0))
  \]
- Can directly convert a **truth-table** into a circuit, but circuit size exponential in input size
- Can convert any ("efficient") program into a ("small") circuit
- Interesting problems already given as succinct programs/circuits
2-Party SFE for General Circuits
2-Party SFE for General Circuits

“General”: evaluate any arbitrary circuit
2-Party SFE for General Circuits

“General”: evaluate any arbitrary circuit

One-sided output: both parties give inputs, one party gets outputs
2-Party SFE for General Circuits

“General”: evaluate any arbitrary circuit

One-sided output: both parties give inputs, one party gets outputs

Either party maybe corrupted passively
2-Party SFE for General Circuits

- "General": evaluate any arbitrary circuit
- One-sided output: both parties give inputs, one party gets outputs
- Either party maybe corrupted passively
- Consider evaluating OR (single gate circuit)
2-Party SFE for General Circuits

“General”: evaluate any arbitrary circuit

One-sided output: both parties give inputs, one party gets outputs

Either party maybe corrupted passively

Consider evaluating OR (single gate circuit)

Alice holds $x=a$, Bob has $y=b$; Bob should get $OR(x,y)$
A Physical Protocol
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_{x}$).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box)

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_x$).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box)

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_{x}$).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$

Inside $B_{xy=ab}$ she places the bit OR(a,b) and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box)

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_x$).
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_x$).

So far Bob gets no information.
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit $\text{OR}(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_x$).

So far Bob gets no information.

Bob “obliviously picks up” $K_{y=b}$, and tries the two keys $K_x,K_y$ on the four boxes. For one box both locks open and he gets the output.
A Physical Protocol

Alice prepares 4 boxes $B_{xy}$ corresponding to 4 possible input scenarios, and 4 padlocks/keys $K_{x=0}$, $K_{x=1}$, $K_{y=0}$ and $K_{y=1}$.

Inside $B_{xy=ab}$ she places the bit OR$(a,b)$ and locks it with two padlocks $K_{x=a}$ and $K_{y=b}$ (need to open both to open the box).

She un-labels the four boxes and sends them in random order to Bob. Also sends the key $K_{x=a}$ (labeled only as $K_x$).

So far Bob gets no information.

Bob “obliviously picks up” $K_{y=b}$, and tries the two keys $K_x, K_y$ on the four boxes. For one box both locks open and he gets the output.
A Physical Protocol
A Physical Protocol

Secure?
A Physical Protocol

Secure?

For curious Alice: only influence from Bob is when he picks up his key $K_y = b$
A Physical Protocol

Secure?

For curious Alice: only influence from Bob is when he picks up his key $K_y=b$

But this is done “obliviously”, so she learns nothing.
A Physical Protocol

Secure?

For curious Alice: only influence from Bob is when he picks up his key $K_y=b$

But this is done "obliviously", so she learns nothing

For curious Bob: What he sees is predictable (i.e., simulatable), given the final outcome
A Physical Protocol

Secure?

For curious Alice: only influence from Bob is when he picks up his key \( K_y=b \)

But this is done “obliviously”, so she learns nothing

For curious Bob: What he sees is predictable (i.e., simulatable), given the final outcome

What Bob sees: His key opens \( K_y \) in two boxes, Alice’s opens \( K_x \) in two boxes; only one random box fully opens. It has the outcome.
A Physical Protocol

Secure?

For curious Alice: only influence from Bob is when he picks up his key $K_y = b$

But this is done “obliviously”, so she learns nothing

For curious Bob: What he sees is predictable (i.e., simulatable), given the final outcome

What Bob sees: His key opens $K_y$ in two boxes, Alice’s opens $K_x$ in two boxes; only one random box fully opens. It has the outcome.

Note when $y=1$, cases $x=0$ and $x=1$ appear same
Larger Circuits
Larger Circuits
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.
Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_w=0$ and $K_w=1$.

For each gate $G$ with input wires $(u,v)$ and output wire $w$, prepare 4 boxes $B_{uv}$ and place $K_w=G(a,b)$ inside box $B_{uv}=ab$. Lock $B_{uv}=ab$ with keys $K_u=a$ and $K_v=b$. 
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$

For each gate $G$ with input wires $(u,v)$ and output wire $w$, prepare 4 boxes $B_{uv}$ and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$

Give to Bob: Boxes for each gate, one key for each of Alice’s input wires
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.

For each gate $G$ with input wires $(u,v)$ and output wire $w$, prepare 4 boxes $B_{uv}$ and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$.

Give to Bob: Boxes for each gate, one key for each of Alice’s input wires.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.

For each gate G with input wires (u,v) and output wire w, prepare 4 boxes $B_{uv}$ and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$.

Give to Bob: Boxes for each gate, one key for each of Alice's input wires.

Obliviously: one key for each of Bob's input wires.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire $w$ in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.

For each gate $G$ with input wires $(u,v)$ and output wire $w$, prepare 4 boxes $B_{uv}$ and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$.

Give to Bob: Boxes for each gate, one key for each of Alice’s input wires.

Obliviously: one key for each of Bob’s input wires.
Larger Circuits

Idea: For each gate in the circuit Alice will prepare locked boxes, but will use it to keep keys for the next gate.

For each wire w in the circuit (i.e., input wires, or output of a gate) pick 2 keys $K_{w=0}$ and $K_{w=1}$.

For each gate G with input wires (u,v) and output wire w, prepare 4 boxes $B_{uv}$ and place $K_{w=G(a,b)}$ inside box $B_{uv=ab}$. Lock $B_{uv=ab}$ with keys $K_{u=a}$ and $K_{v=b}$.

Give to Bob: Boxes for each gate, one key for each of Alice’s input wires.

Obliviously: one key for each of Bob’s input wires.

Boxes for output gates have values instead of keys.
Larger Circuits
Larger Circuits

Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds
Larger Circuits

- Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds.

- Gets output from a box for the output gate.
Larger Circuits

Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds

- Gets output from a box for the output gate

- Security similar to before
Larger Circuits

Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds.

- Gets output from a box for the output gate
- Security similar to before
- Curious Alice sees nothing
Larger Circuits

- Evaluation: Bob gets one key for each input wire of a gate, opens one box for the gate, gets one key for the output wire, and proceeds.
- Gets output from a box for the output gate.
- Security similar to before.
- Curious Alice sees nothing.
- Bob can simulate his view given final output: Bob could prepare boxes and keys (stuffing unopenable boxes arbitrarily); for an output gate, place the output bit in the box that opens.
Garbled Circuit
Garbled Circuit

That was too physical!
Garbled Circuit

- That was too physical!
- Yao’s Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (i.e., a PRF/PRG)
Garbled Circuit

- That was too physical!

- Yao’s Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (i.e., a PRF/PRG)

- Double lock: $\text{Enc}_{K_x}(\text{Enc}_{K_y}(m))$
Garbled Circuit

That was too physical!

Yao’s Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (i.e., a PRF/PRG)

Double lock: $\text{Enc}_{K_x}(\text{Enc}_{K_y}(m))$

Oblivious Transfer for strings: Just repeat bit-OT several times to transfer longer keys
Garbled Circuit

- That was too physical!
- Yao's Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (i.e., a PRF/PRG)
  - Double lock: $\text{Enc}_{K_x}(\text{Enc}_{K_y}(m))$
- Oblivious Transfer for strings: Just repeat bit-OT several times to transfer longer keys
- OK for passive security
Garbled Circuit

- That was too physical!
- Yao’s Garbled circuit: boxes/keys replaced by Symmetric Key Encryption (i.e., a PRF/PRG)
  - Double lock: $\text{Enc}_{K_x}(\text{Enc}_{K_y}(m))$
- Oblivious Transfer for strings: Just repeat bit-OT several times to transfer longer keys
  - OK for passive security
- Much more efficient than the proof of concept protocol, but relies on one-way functions (PRG) in addition to OT
Today
Today

- 2-Party SFE secure against passive adversaries
Today

- 2-Party SFE secure against passive adversaries
- Yao's Garbled Circuit
Today

- 2-Party SFE secure against passive adversaries
- Yao’s Garbled Circuit
- Using OT and IND-CPA encryption
Today

- 2-Party SFE secure against passive adversaries
- Yao's Garbled Circuit
- Using OT and IND-CPA encryption
- OT using TOWP
Today

- 2-Party SFE secure against passive adversaries
  - Yao’s Garbled Circuit
  - Using OT and IND-CPA encryption
    - OT using TOWP
  - Composition (implicitly)
Today

- 2-Party SFE secure against passive adversaries
- Yao's Garbled Circuit
- Using OT and IND-CPA encryption
  - OT using TOWP
- Composition (implicitly)