MIT 6.875

Foundations of Cryptography
Lecture 22

TODAY:
Secure Two-Party and Multi-Party Computation

Secure Two-Party Computation

Input: x **Input:** y

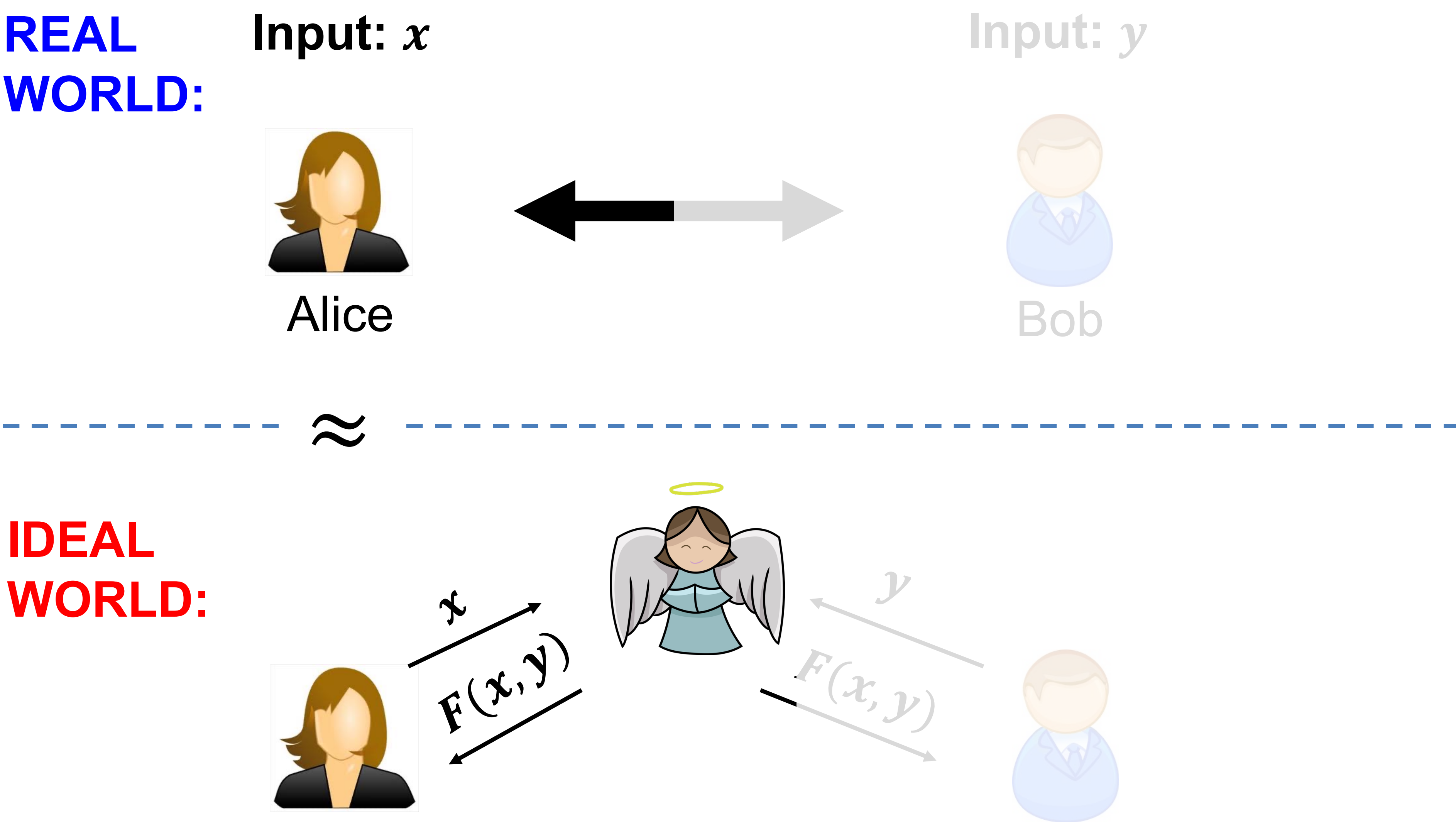
Alice Bob

- Alice and Bob want to compute $F(x, y)$.

<u>Semi-honest Security:</u>

- Alice should not learn anything more than x and $F(x, y)$.
- Bob should not learn anything more than y and $F(x, y)$.

Secure Two-Party Computation



Secure Two-Party Computation

Input: x **Input:** y

Alice Bob

There exists a PPT simulator SIM_A such that for any x and y:

$$SIM_A(x, F(x, y)) \cong View_A(x, y)$$

Secure Two-Party Computation

Input: x **Input:** y

Alice Bob

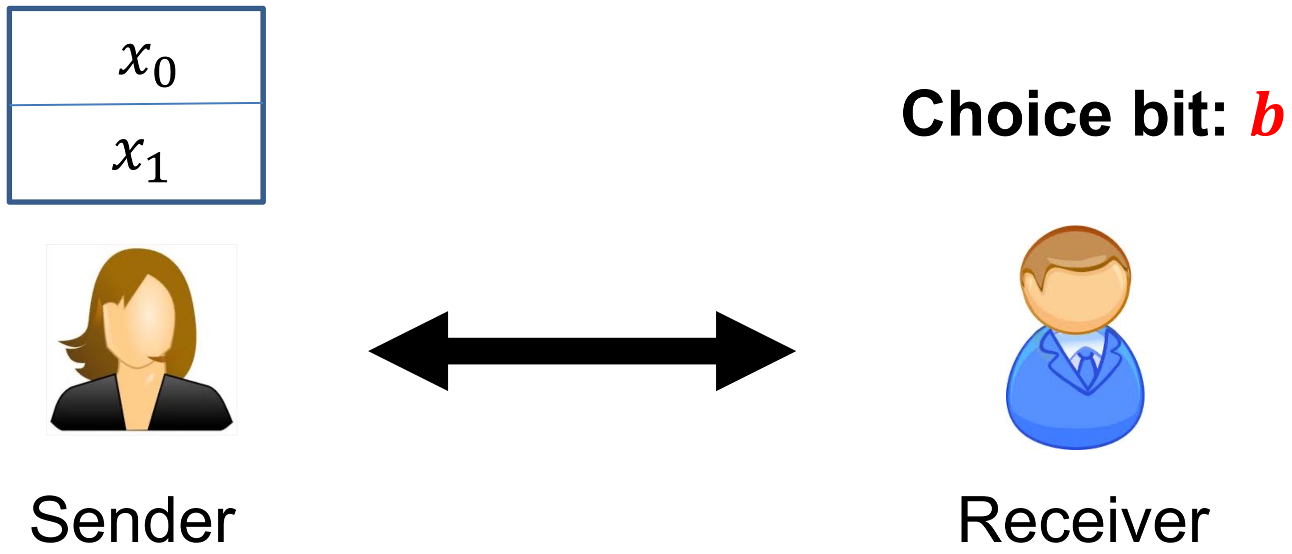
There exists a PPT simulator SIM_B such that for any x and y:

$$SIM_B(y, F(x,y)) \cong View_B(x,y)$$

Secure 2PC from OT

Theorem **[Goldreich-Micali-Wigderson'87]**:
OT can solve ***any*** two-party computation problem.

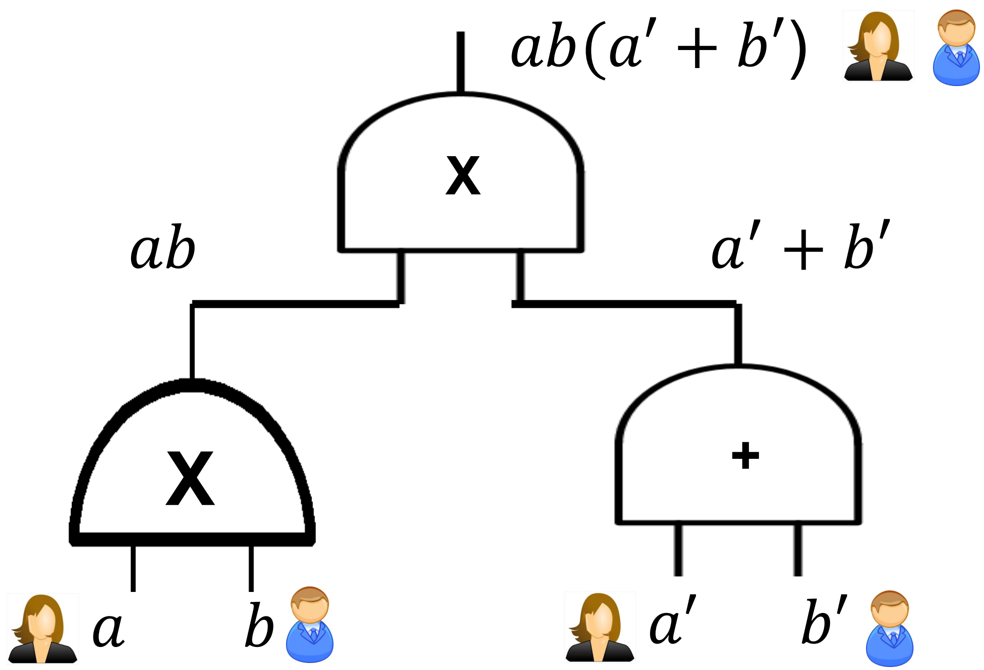
Tool: Oblivious Transfer (OT)



- Sender holds two bits x_0 and x_1.
- Receiver holds a choice bit b.
- Receiver should learn x_b, sender should learn nothing.

How to Compute Arbitrary Functions

For us, programs = functions = Boolean circuits with XOR ($+ \ mod\ 2$) and AND ($\times \ mod\ 2$) gates.



Want: If you can compute XOR and AND *in the appropriate sense*, you can compute everything.

Basic Secret-Sharing

A secret (bit) s is shared between Alice and Bob if Alice holds a bit α and Bob holds a bit β s.t. $\alpha \oplus \beta = s$

α and β are (typically) individually random, so neither Alice nor Bob knows any information about s. Together, however, they can recover s.

Recap: OT ⇒ Secret-Shared-AND

$a \in \{0,1\}$

Alice gets random γ, Bob gets random δ s.t. $\gamma \oplus \delta = ab$.

$b \in \{0,1\}$

Output: γ

Output: δ

$x_0 = \gamma$
$x_1 = a \oplus \gamma$

Run an OT protocol

Choice bit b

Alice outputs γ.

Bob gets $\boldsymbol{x_1 b} + \boldsymbol{x_0(1 \oplus b)} = (\boldsymbol{x_1} \oplus \boldsymbol{x_0})\boldsymbol{b} + \boldsymbol{x_0} = ab \oplus \gamma := \delta$

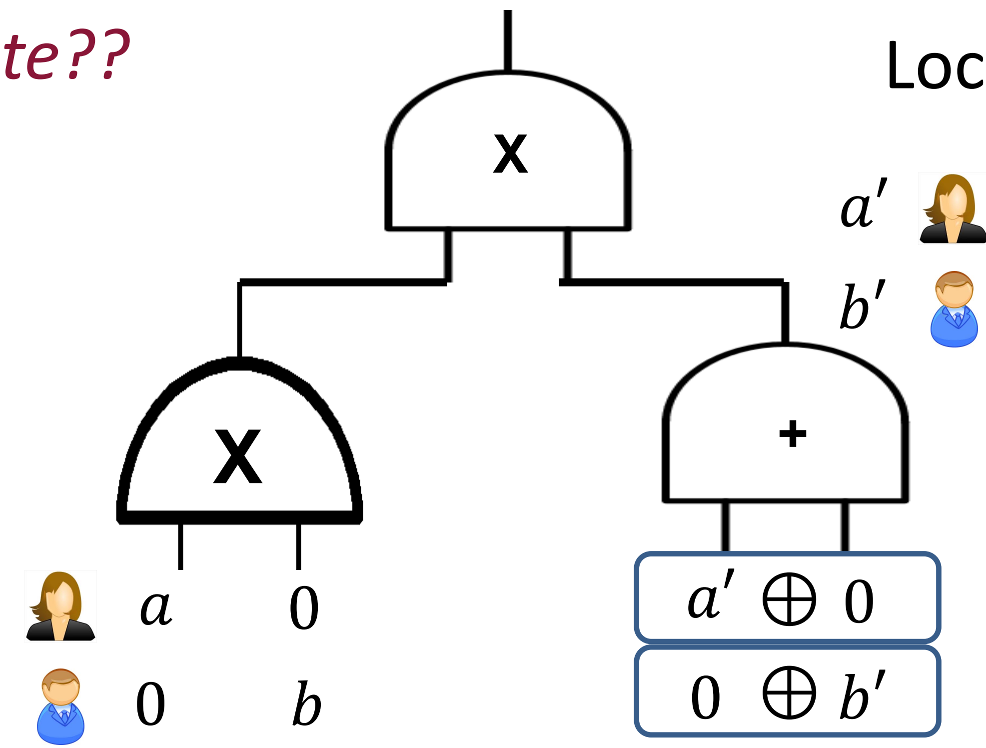
How to Compute Arbitrary Functions

Secret-sharing Invariant: For each wire of the circuit, Alice and Bob each have a bit whose XOR is the value at the wire.

AND gate??

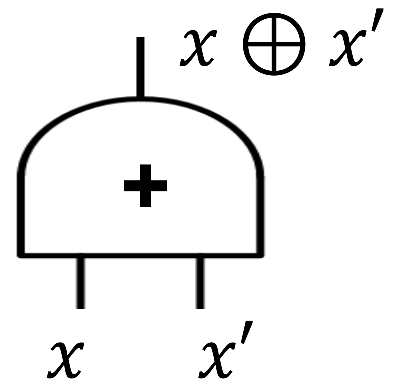
XOR gate:

Locally XOR the shares



Base Case: Input wires

Recap: XOR gate



Alice has α and Bob has β s.t.

$$\alpha \oplus \beta = x$$

Alice has α' and Bob has β' s.t.

$$\alpha' \oplus \beta' = x'$$

Alice computes $\boldsymbol{\alpha \oplus \alpha'}$ and Bob computes $\boldsymbol{\beta \oplus \beta'}$.

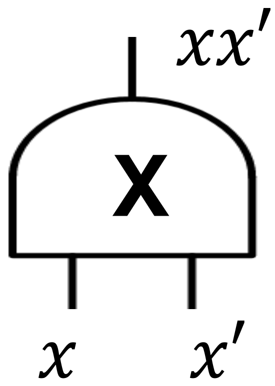
So, we have: $(\alpha \oplus \alpha') \oplus (\beta \oplus \beta')$

$$= (\alpha \oplus \beta) \oplus (\alpha' \oplus \beta') = \mathrm{x} \oplus \mathrm{x}'$$

AND gate

Alice has α and Bob has β s.t.

$$\alpha \oplus \beta = x$$



Alice has α' and Bob has β' s.t.

$$\alpha' \oplus \beta' = x'$$

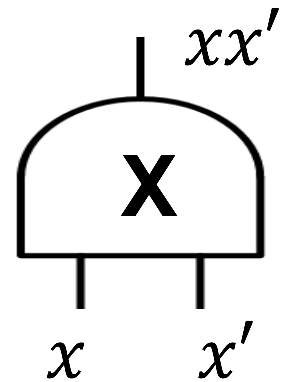
Desired output (to maintain invariant):
Alice wants $\boldsymbol{\alpha''}$ and Bob wants $\boldsymbol{\beta''}$ s.t. $\boldsymbol{\alpha''} \oplus \boldsymbol{\beta''} = xx'$

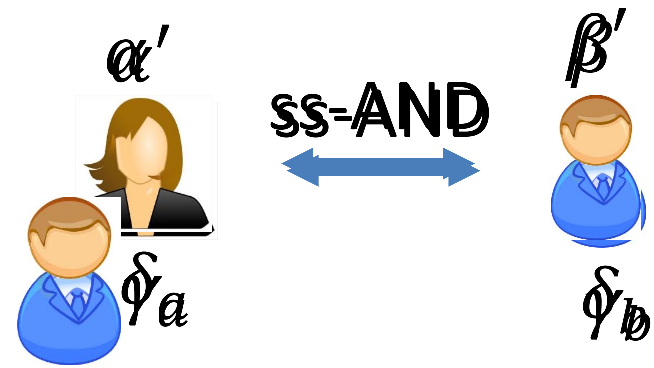
AND gate

$$xx' = (\alpha \oplus \beta)(\alpha' \oplus \beta')$$

$$= \alpha\alpha' \oplus \gamma_a \oplus \delta_a \oplus \beta\beta'$$

$\oplus$ γ_b $\oplus$ δ_b



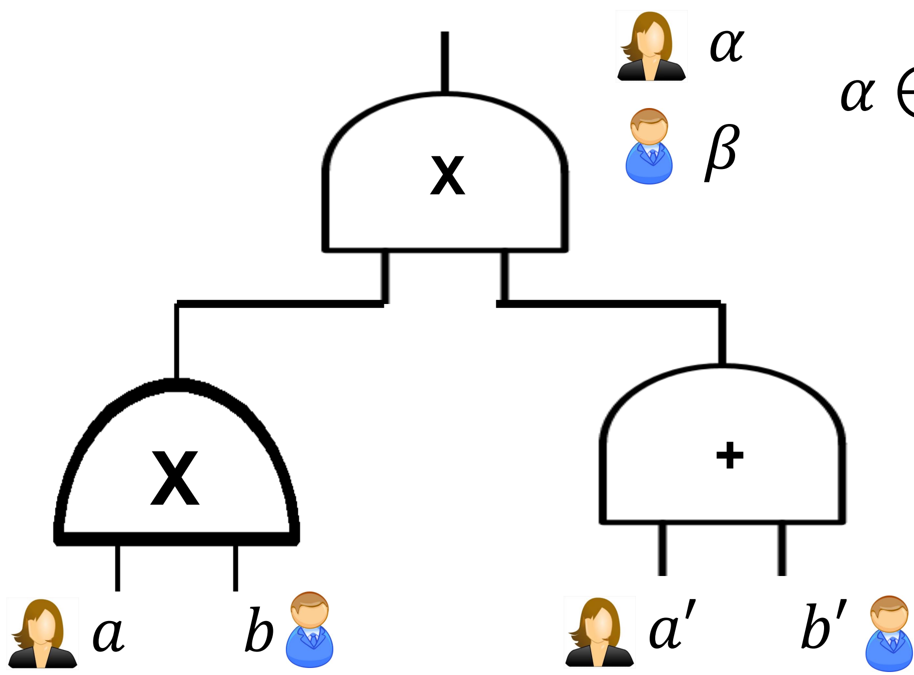


$$\alpha'' = \alpha\alpha' \oplus \gamma_a \oplus \delta_a \qquad \beta'' = \beta\beta' \oplus \gamma_b \oplus \delta_b$$

How to Compute Arbitrary Functions

Secret-sharing Invariant: For each wire of the circuit, Alice and Bob each have a bit whose XOR is the value at the wire.

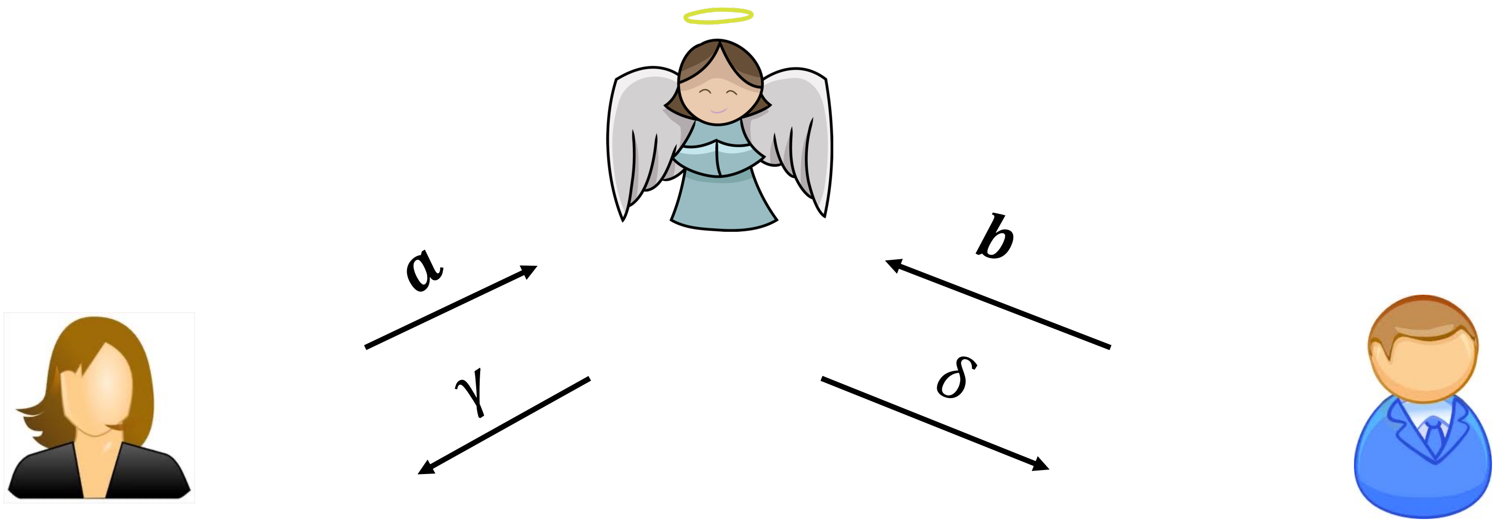
Finally, Alice and Bob exchange the shares at the output wire, and XOR the shares together to obtain the output.



$$\alpha \oplus \beta = ab(a' \oplus b')$$

Security: Intuition

Imagine that the parties have access to an ss-AND angel.



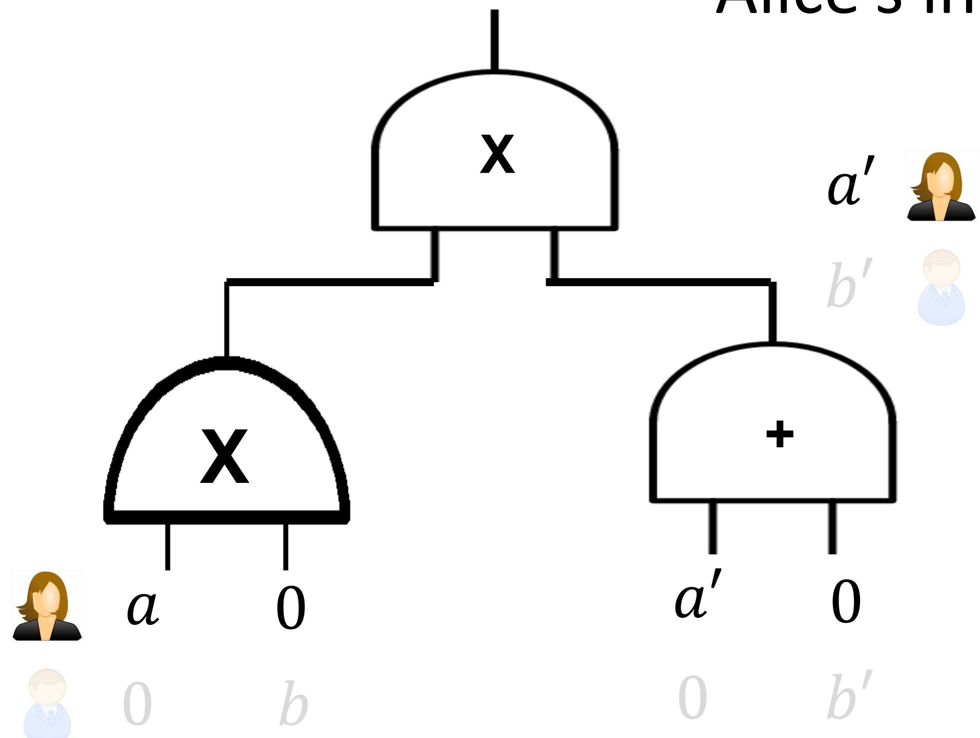
$$\gamma \oplus \delta = \mathrm{ab}$$

Security: Intuition

Imagine that the parties have access to an ss-AND angel.

Simulator for Alice's view:

XOR gate: simulate given Alice's input shares



Input wires: can be simulated given Alice's input

Security: Intuition

Simulator for Alice's view:

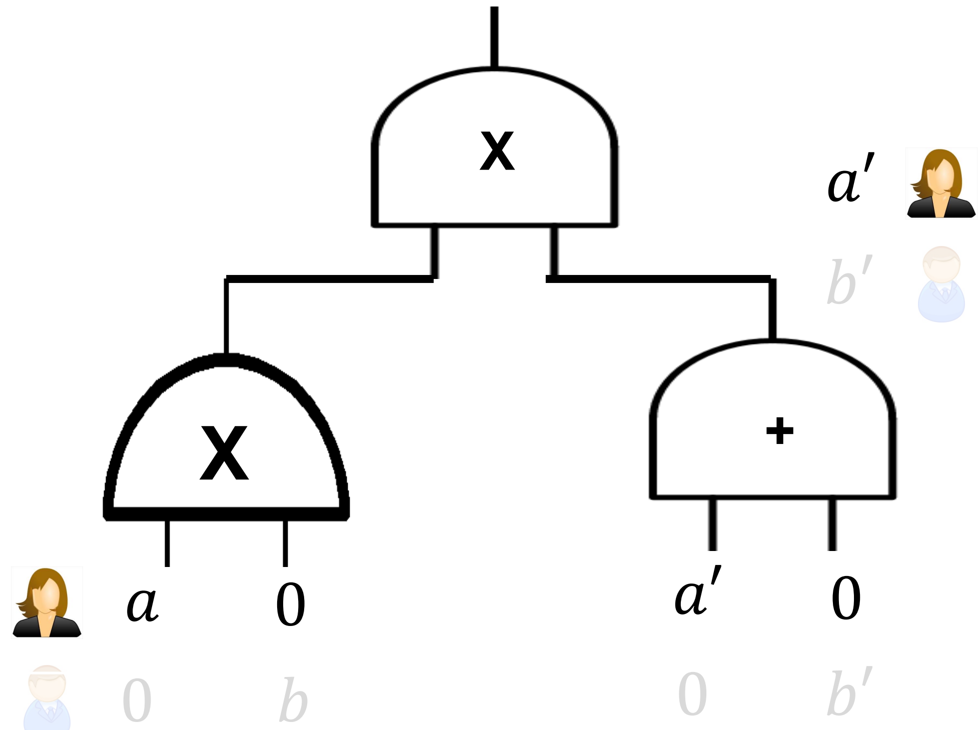
AND gate: simulate given Alice's input shares & outputs from the ss-AND angel.

Alice's share

$= a \cdot 0$ ✓

$+ \gamma_{alice}$ ✓

$+ \delta_{alice}$ ✓



γ_{alice} and δ_{alice} are random, independent of b

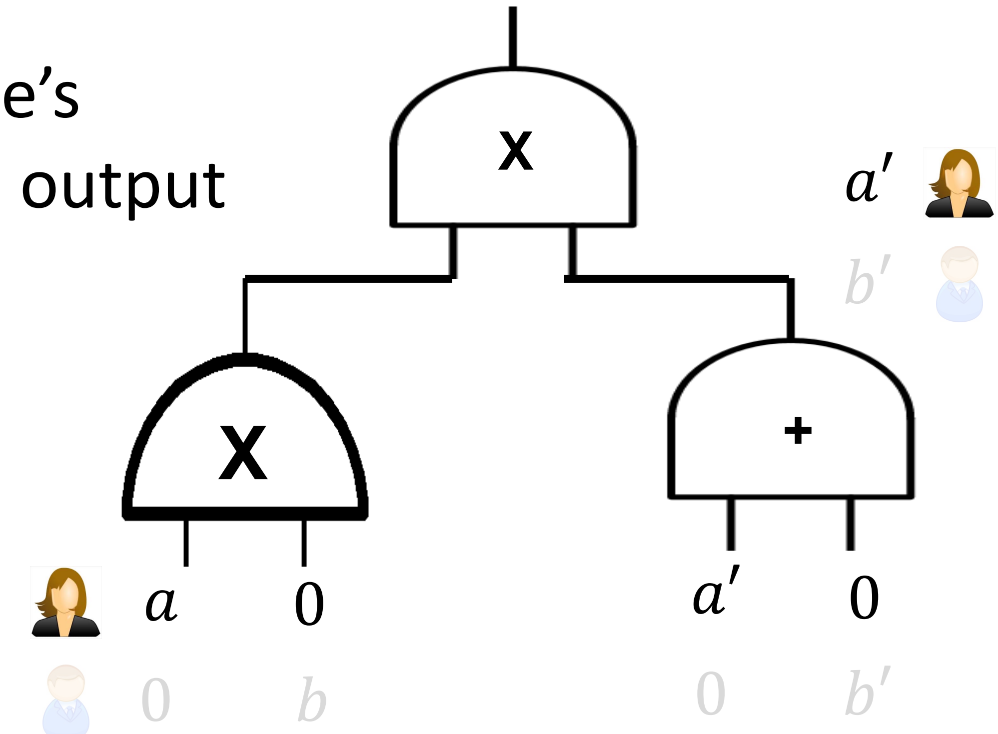
Security: Intuition

Simulator for Alice's view:

Output wire: need to know both Alice and Bob's output shares.

Bob's output share = Alice's output share $\oplus$ function output

Simulator knows the function output, and can compute Bob's output share given Alice's output share.



We showed: Secure 2PC from OT

Theorem **[Goldreich-Micali-Wigderson'87]**:
OT can solve ***any*** two-party computation problem.

In fact, GMW does more:

Theorem **[Goldreich-Micali-Wigderson'87]**:
OT can solve ***any*** multi-party computation problem.

MPC Outline

Secret-sharing Invariant: For each wire of the circuit, **the n parties have a bit each**, whose XOR is the value at the wire.

Base case: input wires.

XOR gate: given input shares $(\alpha_1, \ldots, \alpha_n)$ s.t. $\bigoplus_{i=1}^{n} \alpha_i = a$ and $(\beta_1, \ldots, \beta_n)$ s.t. $\bigoplus_{i=1}^{n} \beta_i = b$, compute the shares of the output of the XOR gate:

$$(\alpha_1 + \beta_1, \ldots, \alpha_n + \beta_n)$$

AND gate: given input shares as above, compute the shares of the output of the XOR gate:

$$(o_1, \ldots, o_n) \text{ s.t } \bigoplus_{i=1}^{n} o_i = ab$$ **Exercise!**

Optimization 1: Preprocessing OTs

Random OT tuple (or AND tuple, or Beaver tuple after D. Beaver): Alice has (α, γ_a) and Bob has (β, γ_b) which are random s.t. $\boldsymbol{\gamma_a \oplus \gamma_b = \alpha\beta}$.

Theorem: Given O(1) many *random* OT tuples, we can do OT with information-theoretic security, exchanging O(1) bits.

Optimization 2: OT Extension

Theorem
[Beaver’96, Ishai-Kushilevitz-Nissim-Pinkas’03]:

Given O(λ) many *random* OT tuples, we can generate n OT tuples exchanging O(n) bits --- as opposed to the trivial O($n\lambda$) bits --- and using only symmetric-key crypto.

Complexity of the 2-party solution

Number of OT protocol invocations $= 2 * \#AND$ gates

Can be made into O(#inputs · λ): Yao's garbled circuits

Number of rounds = AND-depth of the circuit

Can be made into O(1) rounds: Yao's garbled circuits

Communication in bits =

$$O(\#AND \cdot \lambda + \#outputs)$$

Can be made into O(λ #inputs) using FHE: but FHE is computationally more expensive concretely.

Next class:
Secret-Sharing and Information-Theoretically Secure MPC