MIT 6.875

Foundations of Cryptography
Lecture 18

New Topic:

Secure Computation

Secure Two-Party Computation

Input: x | **Input:** y

Alice | Bob

Output: $F_A(x, y)$ | **Output**: $F_B(x, y)$

Secure Two-Party Computation

Input: x

Alice

Output: $F_A(x, y)$

Input: y

Bob

Output: $F_B(x, y)$

Security:

Semi-honest Security:

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

Secure Two-Party Computation

Input: x

Input: y

Alice

Bob

Output: $F_A(x, y)$

Output: $F_B(x, y)$

<u>Malicious Security:</u>

- No (PPT) Alice* can learn anything more than x^* and $F_A(x^*, y)$.
- No (PPT) Bob* can learn anything more than y^* and $F_B(x, y^*)$.

Tool 1: Secret Sharing

Secret Sharing

- ❑ Any **"authorized"** subset of players **can recover** b.
- ❑ No other subset of players **has any info** about b.

- ○ Threshold (or t-out-of-n) SS [Shamir'79, Blakley'79]:

  "authorized" subset = has size $\geq$ t.

secret b ∈ Z_p

2-out-of-n Secret Sharing?

Dealer

P_1 P_2 P_3 P_4 … P_n

Here is a solution.

Repeat for every two-person subset $\{P_i, P_j\}$:

- Generate a 2-out-of-2 secret sharing (s_i, s_j) of b.
- Give s_i to P_i and s_j to P_j

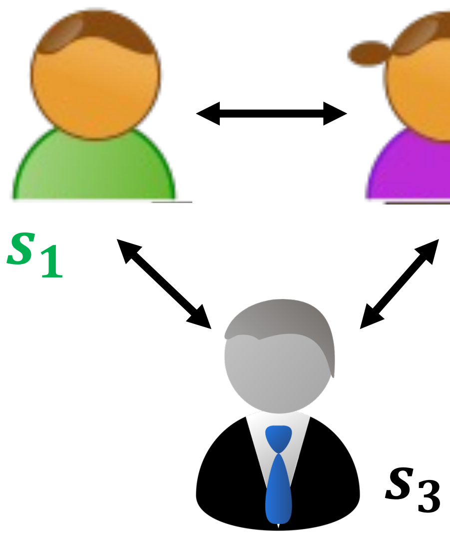
What is the size of shares each party gets?

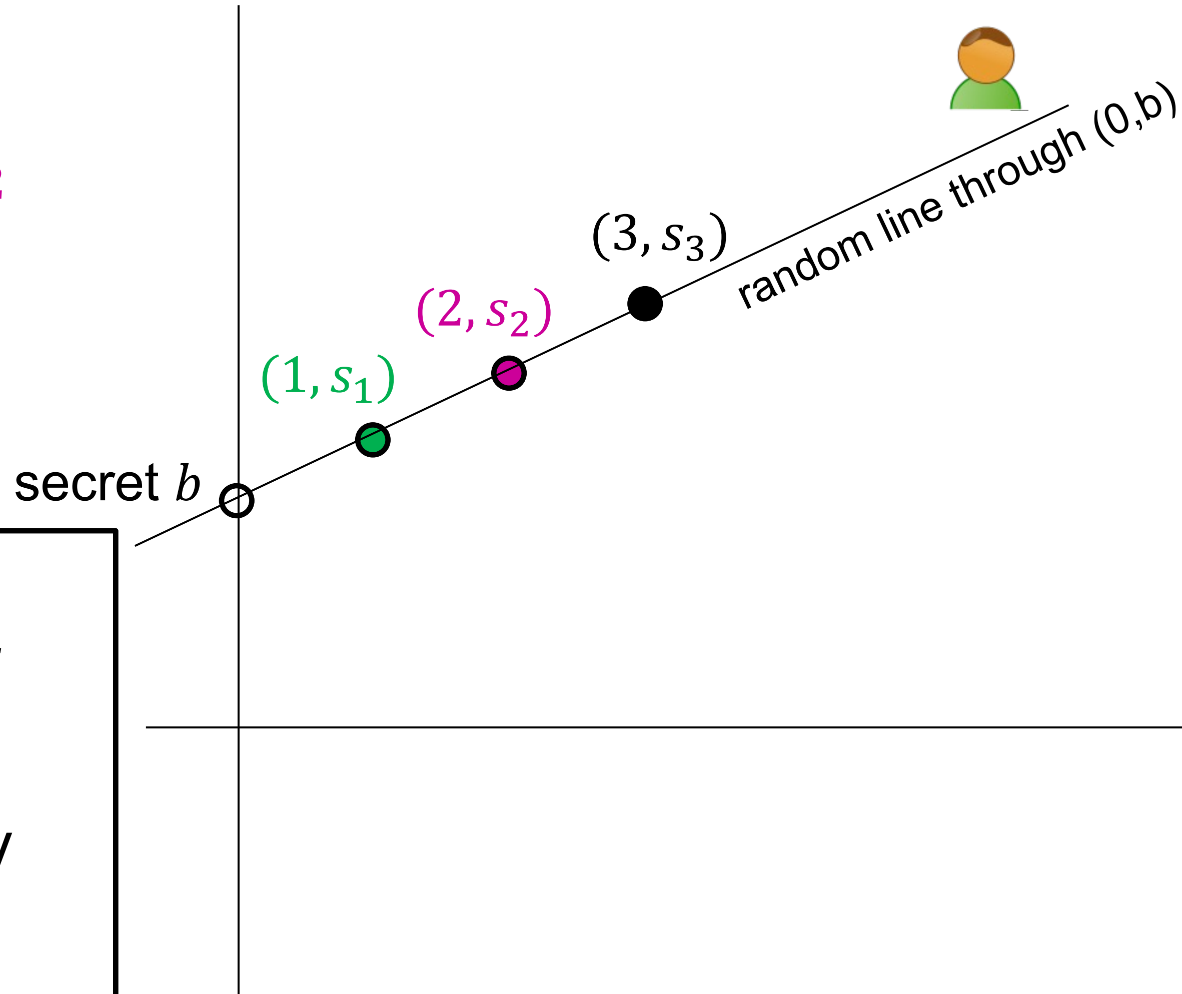
How does this scale to t-out-of-n?

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

Shamir's 2-out-of-n Secret Sharing





Each share s_i is truly random (independent of secret b)

Any two shares uniquely determine b.

Shamir's 2-out-of-n Secret Sharing

1. The dealer picks a uniformly random line **(mod p)** whose constant term is the secret b.

$$f(x) = ax + b \text{ where } a \text{ is uniformly random mod } p$$

2. Compute the shares:

$$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Correctness: can recover secret from any two shares.

Proof: Parties i and j, given shares $s_i = ai + b$ and $s_j = aj + b$ can solve for b $\left(= \frac{js_i - is_j}{j-i}\right)$.

Shamir's 2-out-of-n Secret Sharing

1. The dealer picks a uniformly random line **(mod p)** whose constant term is the secret b.

$$f(x) = ax + b \text{ where } a \text{ is uniformly random mod } p$$

2. Compute the shares:

$$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Security: any single party has no information about the secret.

Proof: Party i's share $s_i = a * i + b$ is uniformly random, independent of b, as a is random and so is $a * i$.

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

1. The dealer picks a uniformly random degree-(t-1) polynomial **(mod p)** whose constant term is the secret b.

   $$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

   where a_i are uniformly random mod p

2. Compute the shares:

   $$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Correctness: can recover secret from any t shares.

Security: the distribution of $any\ t-1$ shares is independent of the secret.

Note: need p to be larger than the number of parties n.

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Correctness: *via Vandermonde matrices.*

Let's look at shares of parties $P_1, P_2, \dots, P_t$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \dots \\ s_t \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^{t-1} \\ 1 & 3 & 3^2 & \dots & 3^{t-1} \\ 1 & \dots & \dots & \dots & \dots \\ 1 & t & t^2 & \dots & t^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \dots \\ a_{t-1} \end{bmatrix} \pmod{p}$$

t-by-t Vandermonde matrix which is ***invertible***

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Correctness: Alternatively, *Lagrange interpolation* gives an explicit formula that recovers b.

$$b = f(0) = \sum_{i=1}^{t} f(i) \left(\prod_{1 \le j \le t, j \ne i} \frac{-x_j}{x_i - x_j} \right)$$

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Security:

Let's look at shares of parties $P_1, P_2, \ldots, P_{t-1}$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \ldots \\ s_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^{t-1} \\ 1 & 3 & 3^2 & \ldots & 3^{t-1} \\ 1 & \ldots & \ldots & \ldots & \ldots \\ 1 & t-1 & (t-1)^2 & \ldots & (t-1)^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \ldots \\ a_{t-1} \end{bmatrix} \pmod p$$

$(t-1)$-by-t Vandermonde matrix

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Security: For every value of b there is a unique polynomial with constant term b and shares $s_1, s_2, \ldots, s_{t-1}$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \ldots \\ s_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^{t-1} \\ 1 & 3 & 3^2 & \ldots & 3^{t-1} \\ 1 & \ldots & \ldots & \ldots & \ldots \\ 1 & t-1 & (t-1)^2 & \ldots & (t-1)^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \ldots \\ a_{t-1} \end{bmatrix} \pmod{p}$$

$(t-1)$-by-t Vandermonde matrix

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

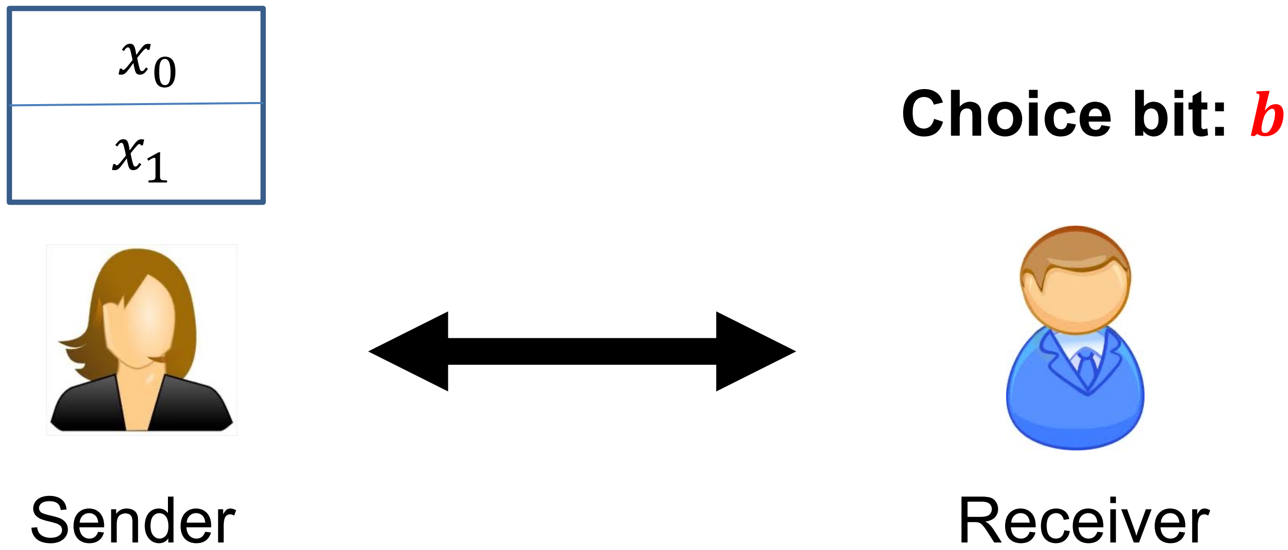
$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Security: For every value of b there is a unique polynomial with constant term b and shares $s_1, s_2, \ldots, s_{t-1}$.

Corollary: for every value of the secret b is equally likely given the shares $s_1, s_2, \ldots, s_{t-1}$. In other words, the secret b is perfectly hidden given $t-1$ shares.

Tool 2: Oblivious Transfer

Oblivious Transfer (OT)



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

  (We will consider **honest-but-curious** adversaries; formal definition in a little bit…)

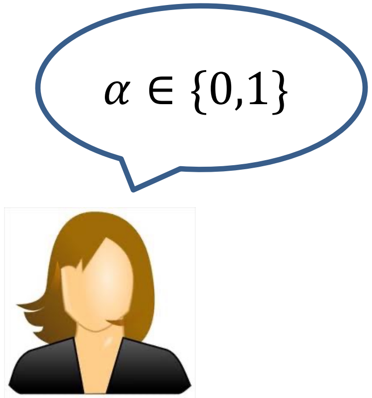
Why OT? The Dating Problem

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

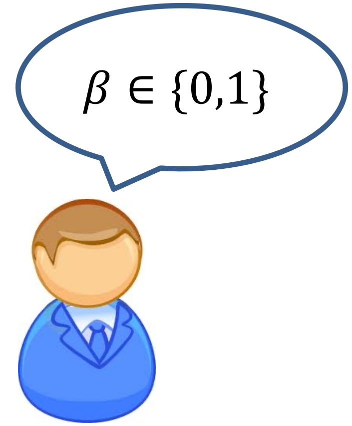
Alice and Bob want to compute the AND $\alpha \wedge \beta$.

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

Why OT? The Dating Problem



Alice and Bob want to compute the AND $\alpha \wedge \beta$.



$x_0 = 0$
$x_1 = \alpha$

Run an OT protocol

Choice bit $b = \beta$

Bob gets α if β=1, and 0 if β=0

Here is a way to write the OT selection function: $\boldsymbol{x_1 b + x_0(1 - b)}$

which, in this case is $= \alpha\beta$.

The Billionaires’ Problem

Who is richer?

The Billionaires’ Problem

$f(X, Y) = 1$
if and only if $X > Y$

X

...	0	1	0	0	...

Unit Vector u_X = 1 in the X^{th} location and 0 elsewhere

Y

...	0	1	1	1	1	1	1

Vector v_Y = 1 from the $(Y+1)^{th}$ location onwards

$$\boldsymbol{f(X,Y) = \langle u_X, v_Y \rangle = \sum_{i=1}^{U} u_X[i] \land v_Y[i]}$$

~~**Compute each AND individually and sum it up?**~~

Detour: OT ⇒ Secret-Shared-AND

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

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

Output: γ

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

Output: δ

$x_0 = \gamma$
$x_1 = \alpha \oplus \gamma$

Run an OT protocol

Choice bit $b = \beta$

Alice outputs γ.

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

The Billionaires' Problem

$f(X, Y) = 1$
if and only if $X > Y$

…	0	1	0	0	…

Unit Vector u_X

…	0	1	1	1	1	1	1

Vector v_Y

$$\boldsymbol{f(X,Y) = \langle u_X, v_Y \rangle = \sum_{i=1}^{U} u_X[i] \wedge v_Y[i]}$$

1. Alice and Bob run many OTs to get (γ_i, δ_i) s.t.

$$\gamma_i \oplus \delta_i = \boldsymbol{u_X[i] \wedge v_Y[i]}$$

2. Alice computes $\gamma = \oplus_i \gamma_i$ and Bob computes $\delta = \oplus_i \delta_i$.
3. Alice reveals γ and Bob reveals δ.

Check (correctness): $\gamma \oplus \delta = \boldsymbol{\langle u_X, v_Y \rangle = f(X,Y)}$.

The Billionaires' Problem

$f(X, Y) = 1$
if and only if $X > Y$

...	0	1	0	0	...

Unit Vector u_X

...	0	1	1	1	1	1	1

Vector v_Y

$$\boldsymbol{f(X,Y) = \langle u_X, v_Y \rangle = \sum_{i=1}^{U} u_X[i] \wedge v_Y[i]}$$

1. Alice and Bob run many OTs to get (γ_i, δ_i) s.t.

$$\gamma_i \oplus \delta_i = \boldsymbol{u_X[i] \wedge v_Y[i]}$$

2. Alice computes $\gamma = \oplus_i \gamma_i$ and Bob computes $\delta = \oplus_i \delta_i$.

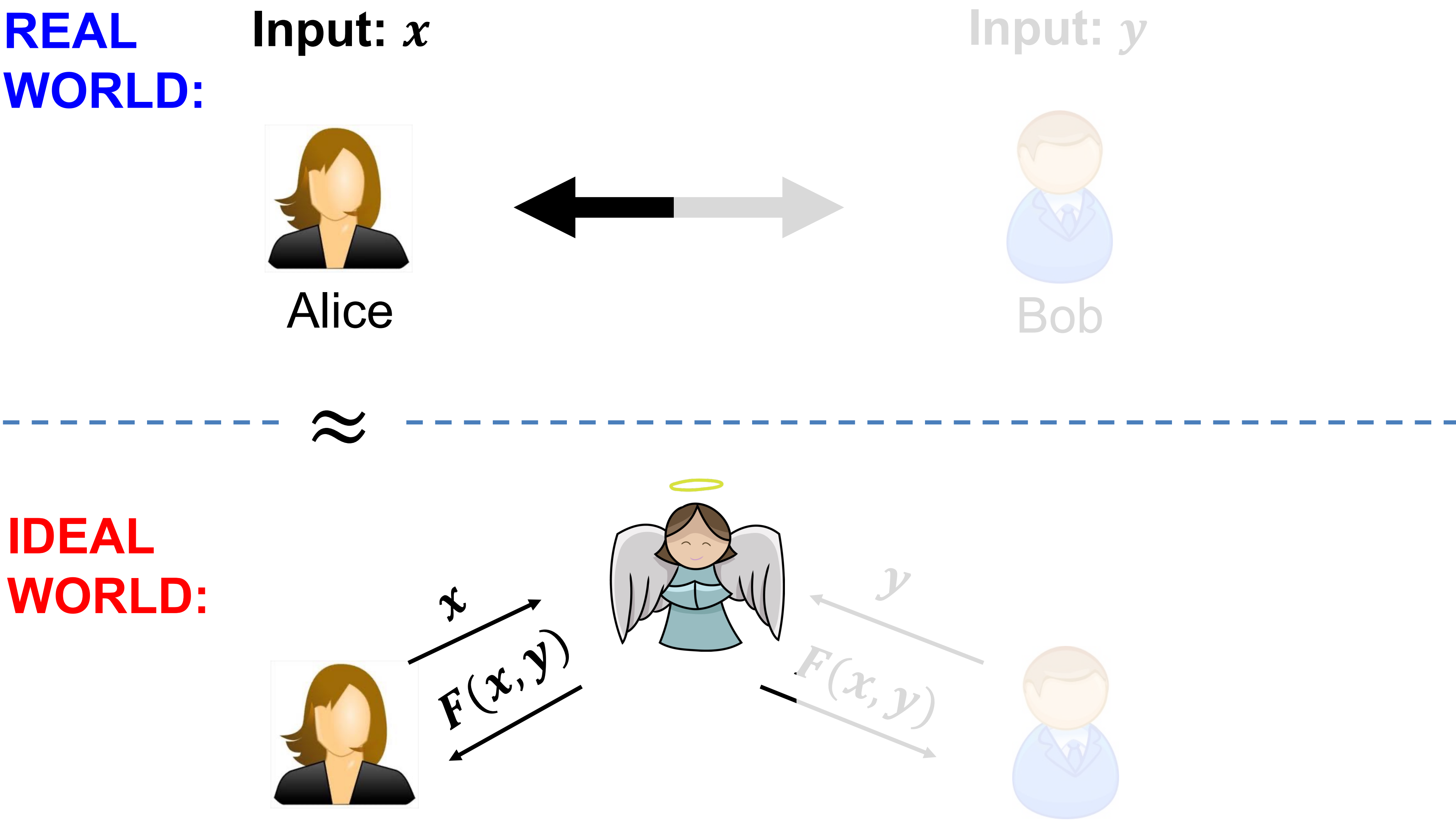
Check (privacy): Alice & Bob get a bunch of random bits.

“OT is Complete”

Theorem** (lec18-19):* OT can solve not just love and money, but ***any two-party (and multi-party) problem efficiently.

Defining Security:
The Ideal/Real Paradigm

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

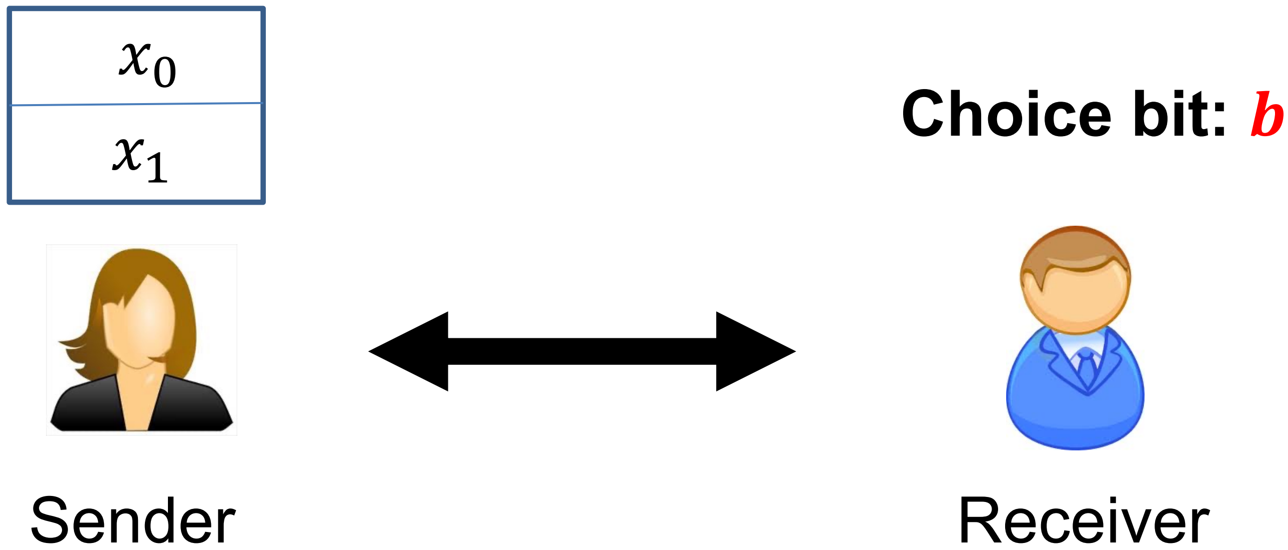
Input: x **Input:** y

Alice Bob

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

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

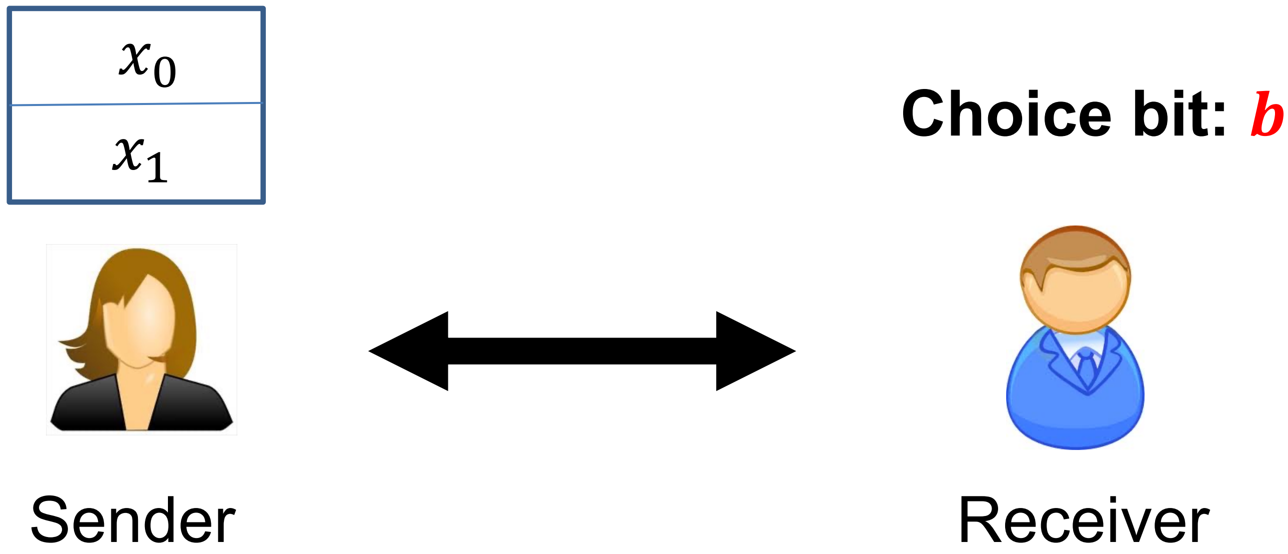
OT Definition



Receiver Security: Sender should not learn b.

Define Sender's view $View_S(x_0, x_1, b)$ = her random coins and the protocol messages.

OT Definition



Receiver Security: Sender should not learn b.

There exists a PPT simulator SIM_S such that for any x_0,x_1 and b:

$$SIM_S(x_0, x_1) \cong Views_S(x_0, x_1, b)$$

OT Definition

x_0
x_1

Choice bit: *b*

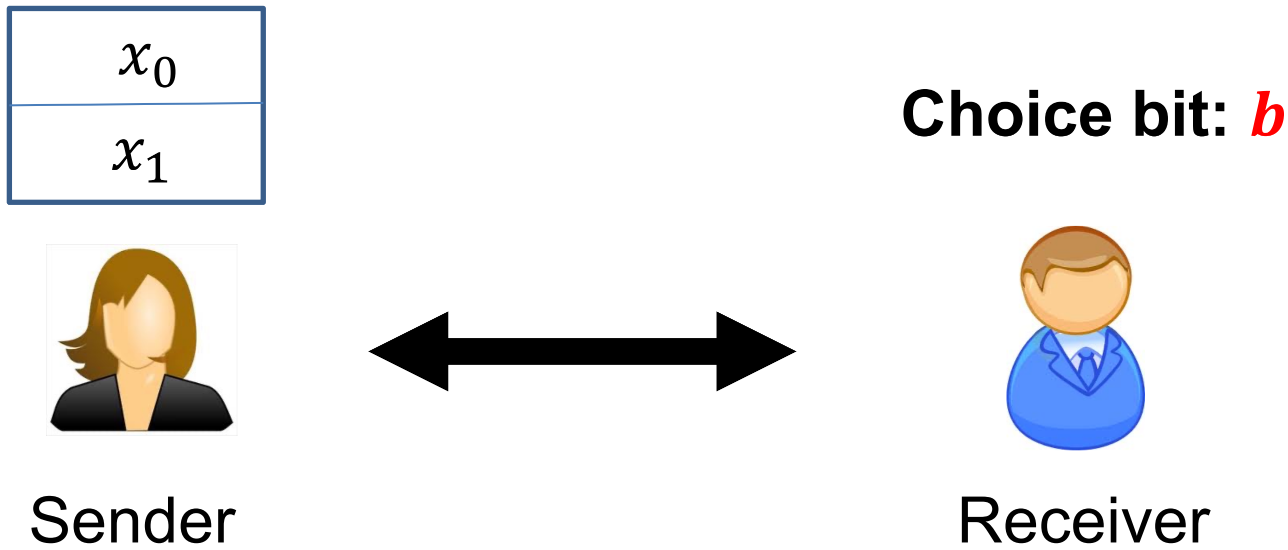
Sender

Receiver

Sender Security: Receiver should not learn x_{1-b}.

Define Receiver's view $View_R(x_0, x_1, b)$ = his random coins and the protocol messages.

OT Definition



Sender Security: Receiver should not learn x_{1-b}.

There exists a PPT simulator SIM_R such that for any x_0,x_1 and b:

$$SIM_R(b, x_b) \cong View_R(x_0, x_1, b)$$

OT Protocols

OT Protocol 1: Trapdoor Permutations

For concreteness, let's use the RSA trapdoor permutation.

Input bits: (x_0, x_1)

Choice bit: b

Pick $N = PQ$ and RSA exponent e.

N, e →

Choose random r_b and set $s_b = r_b^e \bmod N$

← s_0, s_1

Choose random s_{1-b}

Compute r_0, r_1 and one-time pad x_0, x_1 using hardcore bits

$x_0 \oplus HCB(r_0)$ →
$x_1 \oplus HCB(r_1)$

Bob can recover x_b but not x_{1-b}

OT Protocol 1: Trapdoor Permutations

Input bits: (x_0, x_1)

N, e →

← s_0, s_1

$x_0 \oplus HCB(r_0)$ →
$x_1 \oplus HCB(r_1)$

Choice bit: b

How about Bob's security
(a.k.a. Why does Alice not learn Bob's choice bit)?

Alice's view is s_0, s_1 one of which is chosen randomly from Z_N^* and the other by raising a random number to the e-th power. They look exactly the same!

OT Protocol 1: Trapdoor Permutations

Input bits: (x_0, x_1)

N, e →

s_0, s_1 ←

$x_0 \oplus HCB(r_0)$ →
$x_1 \oplus HCB(r_1)$

Choice bit: b

How about Bob's security
(a.k.a. Why does Alice not learn Bob's choice bit)?

Exercise: Show how to construct the simulator.

OT Protocol 1: Trapdoor Permutations

Input bits: (x_0, x_1)

N, e →

← s_0, s_1

$x_0 \oplus HCB(r_0)$ →
$x_1 \oplus HCB(r_1)$

Choice bit: b

How about Alice's security
(a.k.a. Why does Bob not learn both of Alice's bits)?

Assuming Bob is semi-honest, he chose s_{1-b} uniformly at random, so the hardcore bit of $s_{1-b} = r_{1-b}^d$ is computationally hidden from him.

OT Protocol 1: Trapdoor Permutations

Input bits: (x_0, x_1)

N, e →

s_0, s_1 ←

$x_0 \oplus HCB(r_0)$ →
$x_1 \oplus HCB(r_1)$

Choice bit: b

How about Alice's security
(a.k.a. Why does Bob not learn both of Alice's bits)?

Exercise: Show how to construct the simulator.

OT Protocol 2: from Oblivious PKE

A public-key encryption scheme (PKE) where there is an oblivious public-key generation algorithm that outputs a random public key "without knowing" the secret key.

$$pk \leftarrow \mathrm{OblivGen}(1^n; r)$$

Security: IND-CPA holds even given the randomness used by OblivGen.

Example: for El Gamal encryption where the public key is a pair $(g, h = g^x)$ and the private key is x, OblivGen simply outputs two random elements from the group.

OT Protocol 2: from Oblivious PKE

Input bits: (x_0, x_1)

Choice bit: b

Generate random pk_b with sk_b by running Gen. and pk_{1-b} by running OblivGen

pk_0, pk_1 ←

$c_0 \leftarrow Enc(pk_0, x_0)$ →
$c_1 \leftarrow Enc(pk_1, x_1)$

Decrypt c_b using sk_b

OT Protocol 3: Additive HE

Input bits: (x_0, x_1)

Choice bit: b

Encrypt choice bit b

$c \longleftarrow \mathrm{Enc}(sk, b)$

c ⟵

Homomorphically evaluate the selection function

$$\boldsymbol{SEL_{x_0,x_1}(b) = (x_1 \oplus x_0)b \oplus x_0}$$

$c' = \mathrm{Eval}(SEL_{x_0,x_1}(b), c)$ ⟶

Decrypt to get x_b

Bob's security: computational, from CPA-security of Enc.

Alice's security: statistical, from function-privacy of Eval.

Many More Constructions of OT

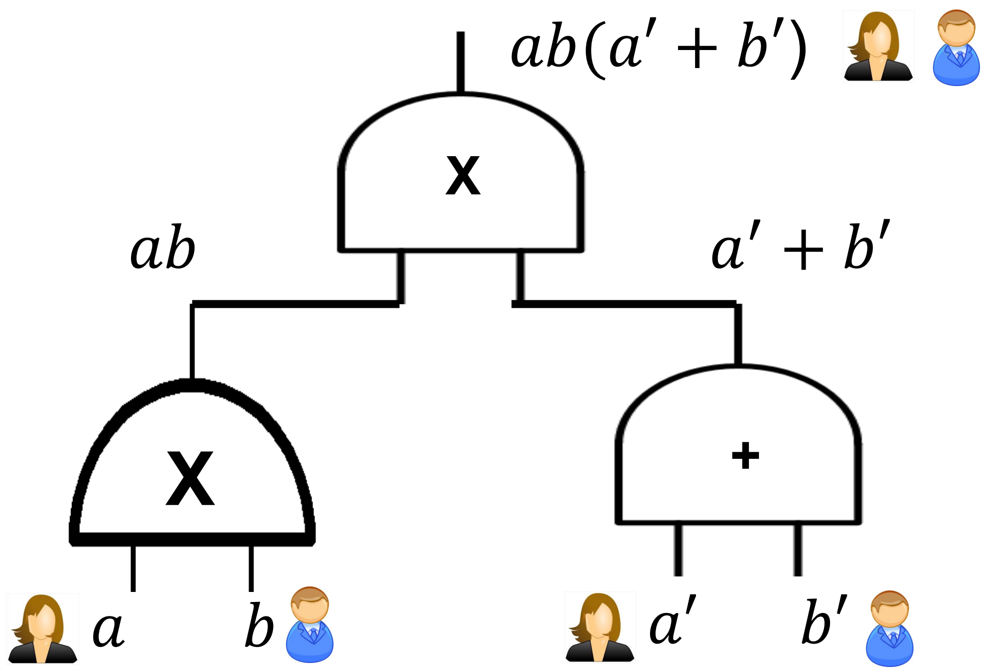
***Theorem**:* OT protocols can be constructed based on the hardness of the Diffie-Hellman problem, factoring, quadratic residuosity, LWE, elliptic curve isogeny problem etc. etc.

Secure 2PC from OT

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

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.

Recap: OT $\Rightarrow$ 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.

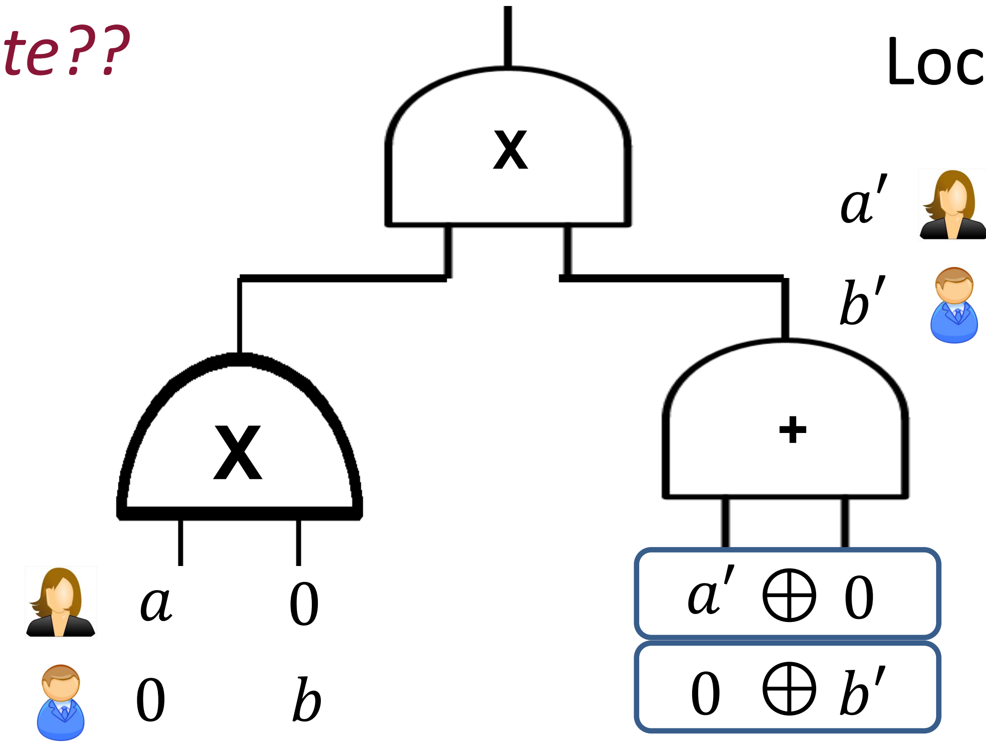
Bob gets $\boldsymbol{x_1 b + x_0(1 \oplus b) = (x_1 \oplus x_0)b + x_0} = ab \oplus \gamma := \delta$

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.

XOR gate:
Locally XOR the shares

AND gate??

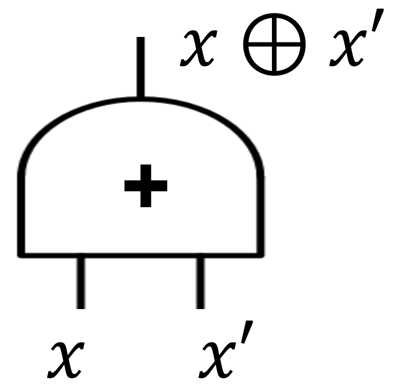


Base Case: Input wires

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 $\boldsymbol{\beta \oplus \beta'}$.

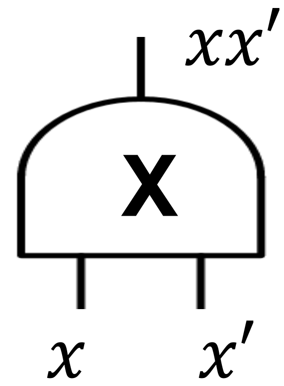
So, we have: $(\alpha \oplus \alpha') \oplus (\beta \oplus \beta')$

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

AND gate

Alice has α and Bob has β s.t.

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



Alice has α' and Bob has β' s.t.

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

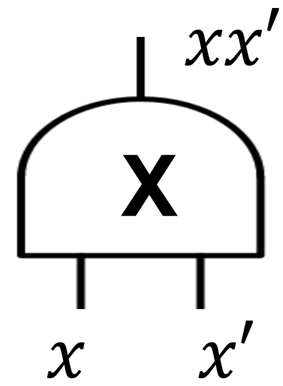
Desired output (to maintain invariant):
Alice wants $\boldsymbol{\alpha''}$ and Bob wants $\boldsymbol{\beta''}$ s.t. $\boldsymbol{\alpha''} \oplus \boldsymbol{\beta''} = xx'$

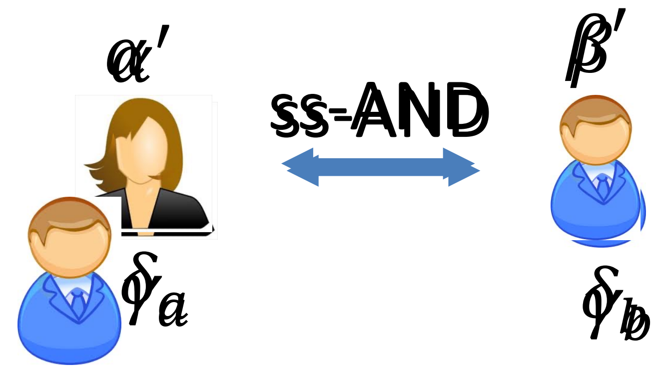
AND gate

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

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


$$\oplus \gamma_b \qquad \oplus \delta_b$$



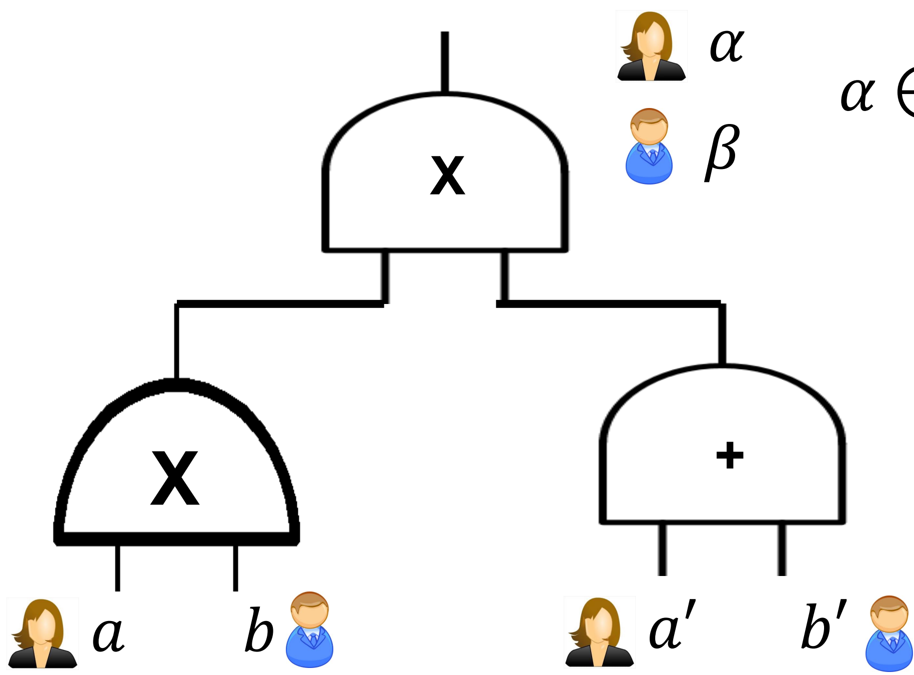


$$\alpha'' = \alpha\alpha' \oplus \gamma_a \oplus \delta_a \qquad \beta'' = \beta\beta' \oplus \gamma_b \oplus \delta_b$$

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.

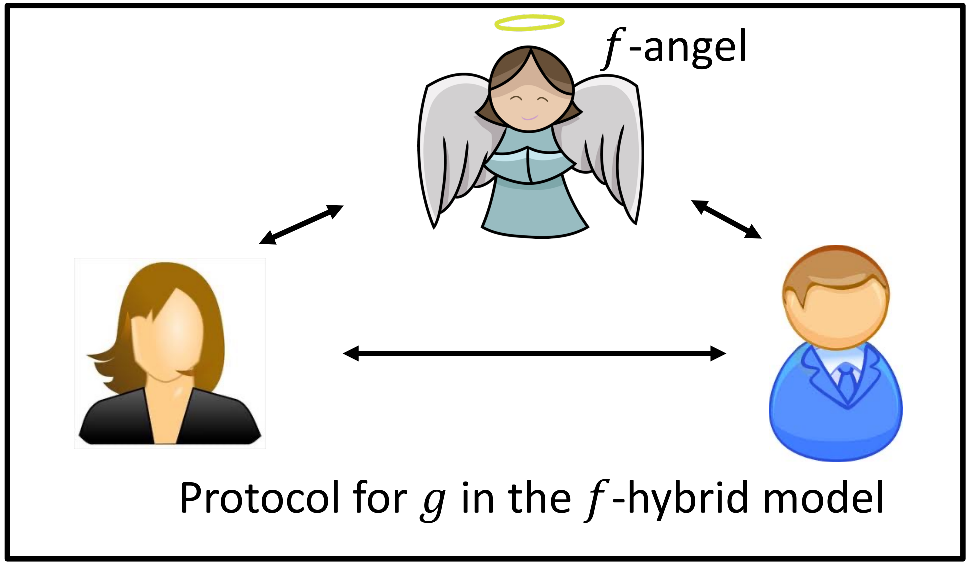
Finally, Alice and Bob exchange the shares at the output wire, and XOR the shares together to obtain the output.



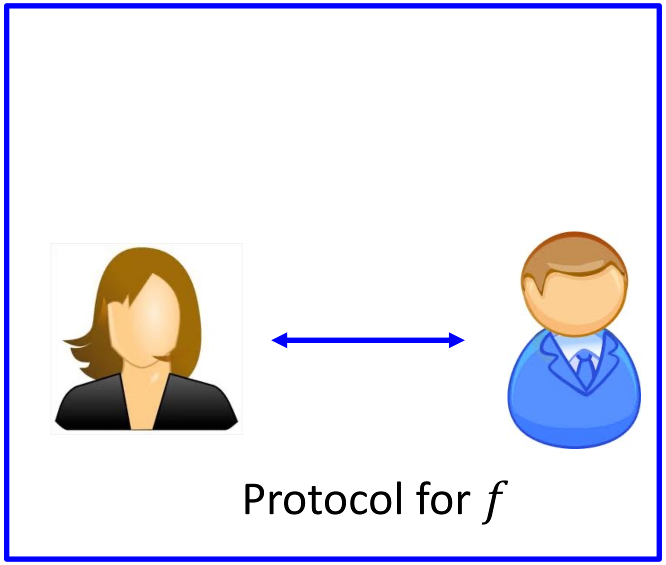
$$\alpha \oplus \beta = ab(a' \oplus b')$$

Security by Composition

Theorem:
If protocol Π securely realizes a function g in the "f-hybrid model" and protocol Π' securely realizes f, then $\Pi \circ \Pi'$ securely realizes g.

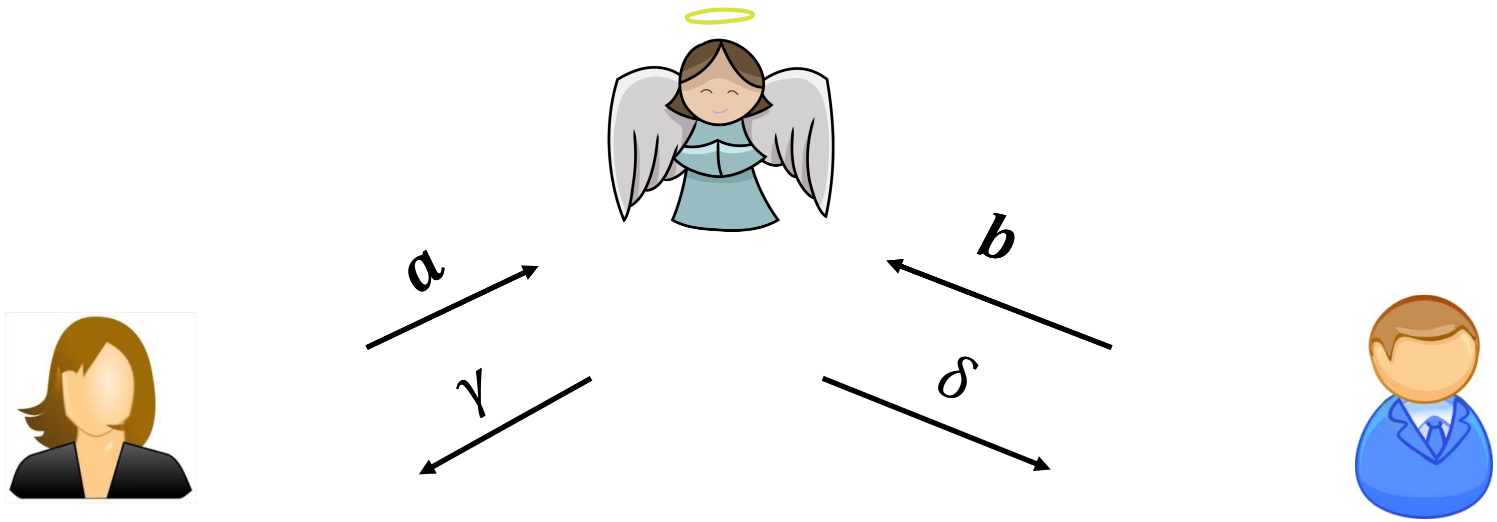


+



Security: Intuition (ss-AND hybrid model)

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



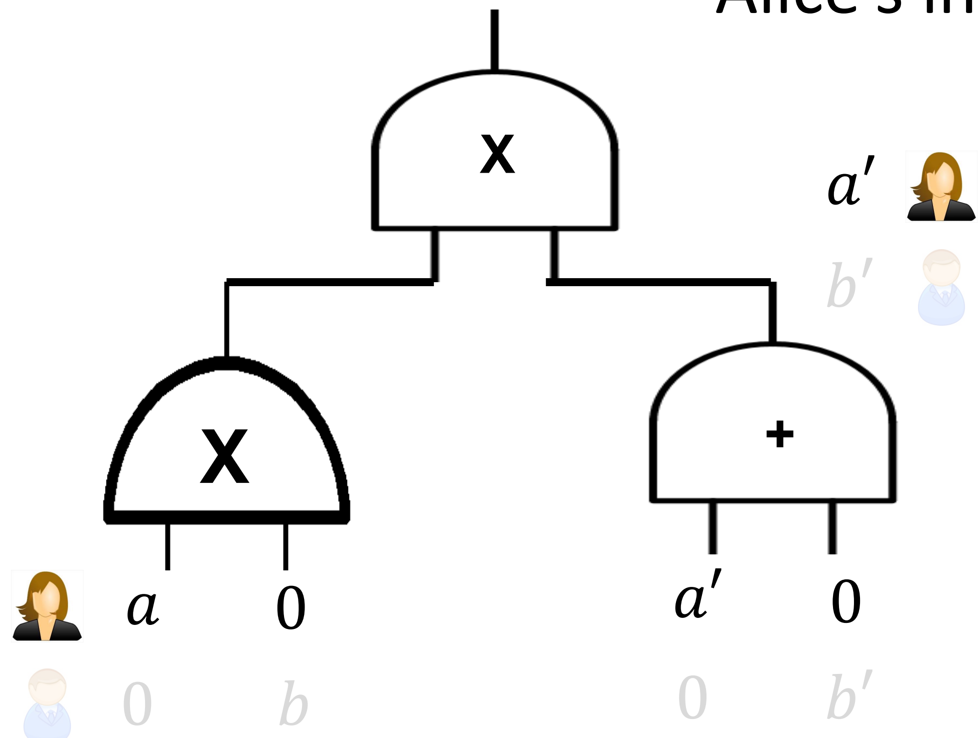
$$\gamma \oplus \delta = \mathrm{ab}$$

Security: Intuition (ss-AND hybrid model)

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

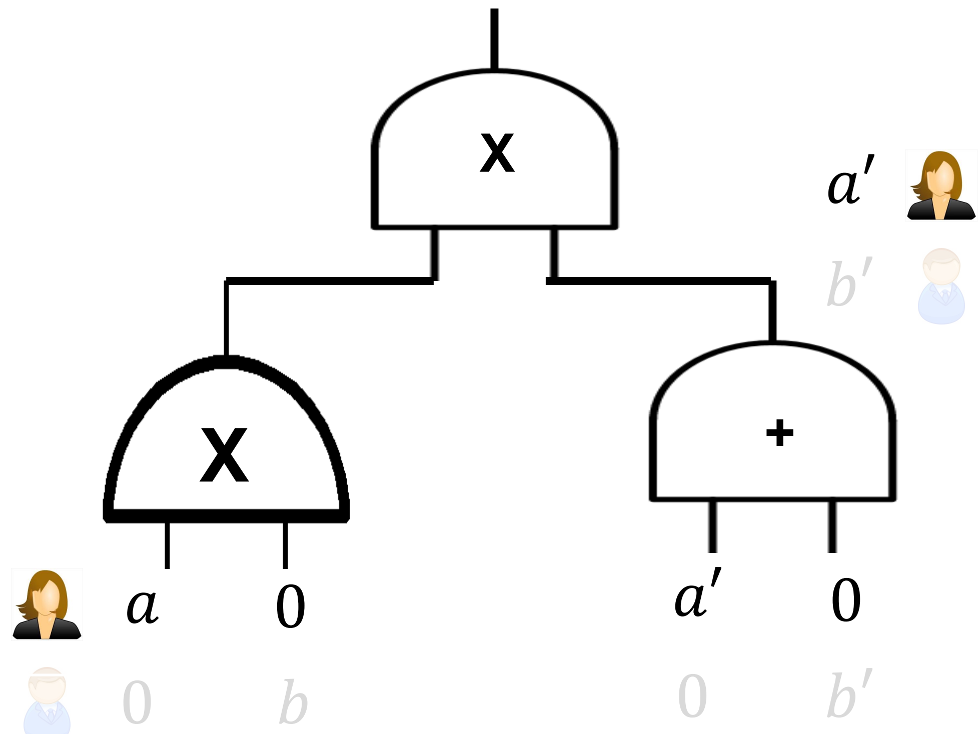
Security: Intuition (ss-AND hybrid model)

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

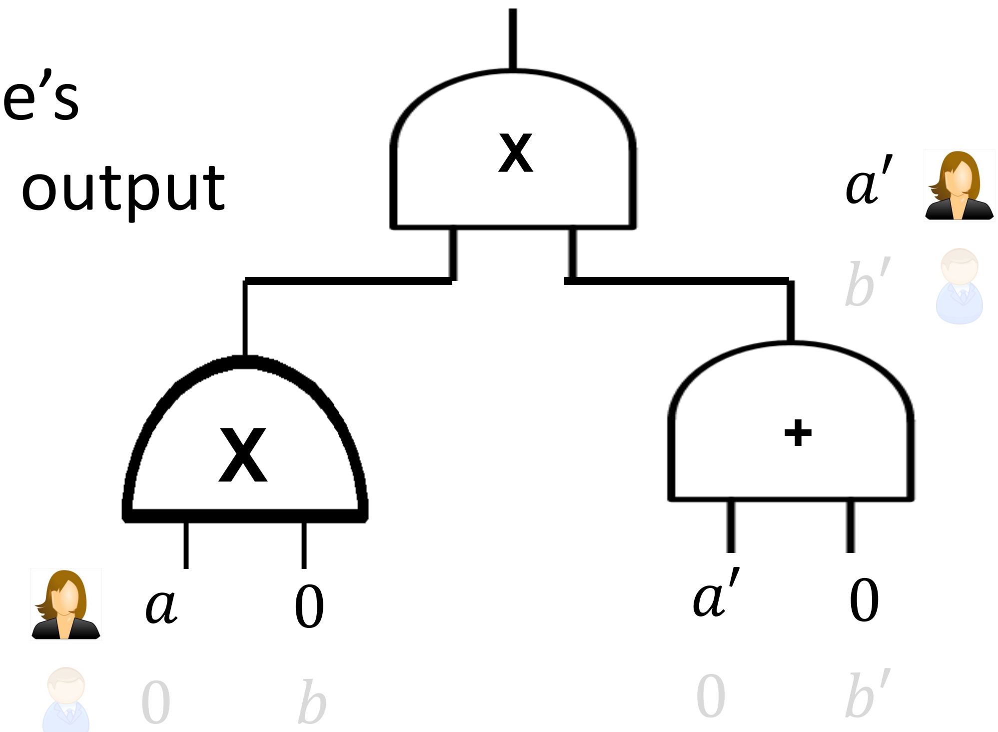
Security: Intuition (ss-AND hybrid model)

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 ⊕ function output

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



Secret-Shared AND protocol

Using the RSA trapdoor permutation.

Input bit: a

Input bit: b

Pick $N = PQ$ and RSA exponent e.

N, e →

Choose random r_b and set $s_b = r_b^e \bmod N$

Choose random s_{1-b}

← s_0, s_1

Let x_0 be random and $x_1 = x_0 \oplus \mathrm{a}$.

Compute r_0, r_1 and one-time pad x_0, x_1 using hardcore bits

$x_0 \oplus HCB(r_0)$ →

$x_1 \oplus HCB(r_1)$

Alice outputs x_0

Bob outputs x_b

Secret-Shared AND protocol

Using the RSA trapdoor permutation.

Input bit: a

Input bit: b

Exercise: Construct simulators for Alice and Bob.

In summary: Secure 2PC from OT

Theorem* [Goldreich-Micali-Wigderson’87]**: Assuming OT exists, there is a protocol that solves ***any two-party computation problem against semi-honest adversaries.

In fact, GMW does more:

Theorem **[Goldreich-Micali-Wigderson'87]**: Assuming OT exists, there is a protocol that solves any ***multi-party*** computation problem against semi-honest adversaries.

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

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

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

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

Exercise!