Secure Computation II

601.442/642 Modern Cryptography

22nd April 2026

Recap: Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

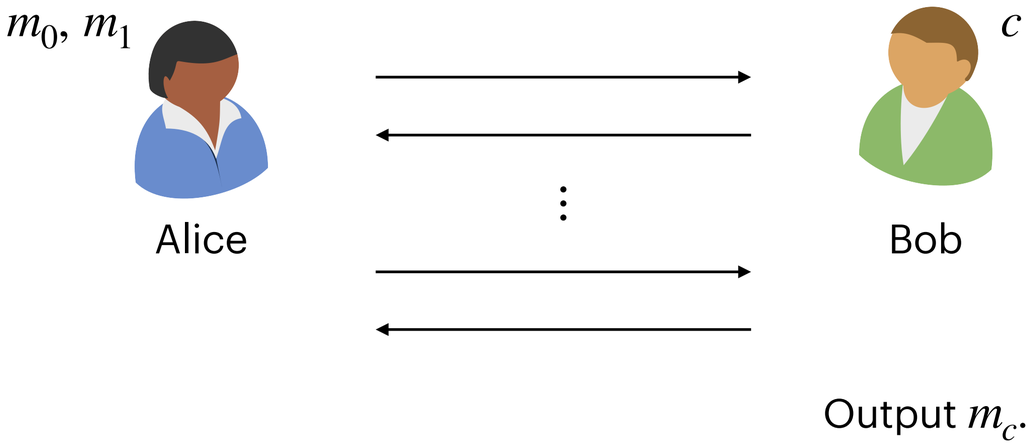
A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

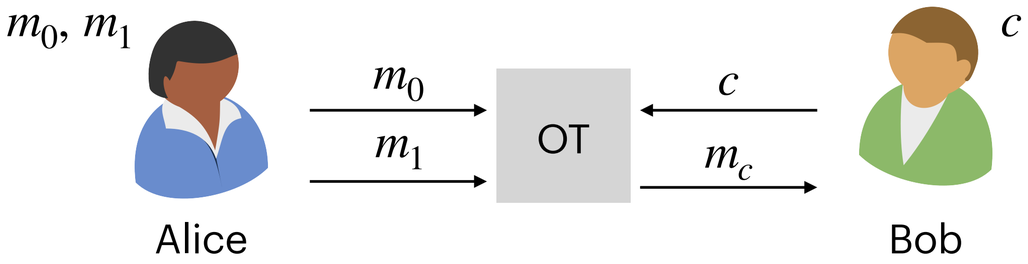
where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Recap: Oblivious Transfer (OT)



- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

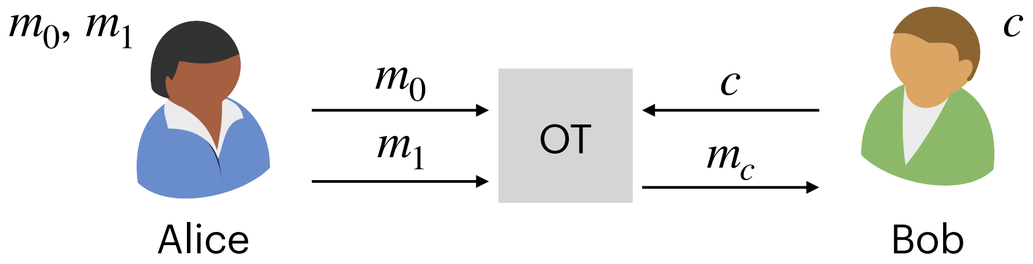
Recap: Oblivious Transfer (OT)



Output m_c.

- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

Recap: Oblivious Transfer (OT)



Output m_c.

- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

Theorem [Yao'86]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure two-party computation of any PPT program.

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

OT is necessary and complete for secure multi-party computation!

Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?

x

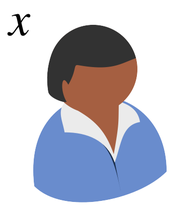
Alice

y

Bob

Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?



x

Alice

OT

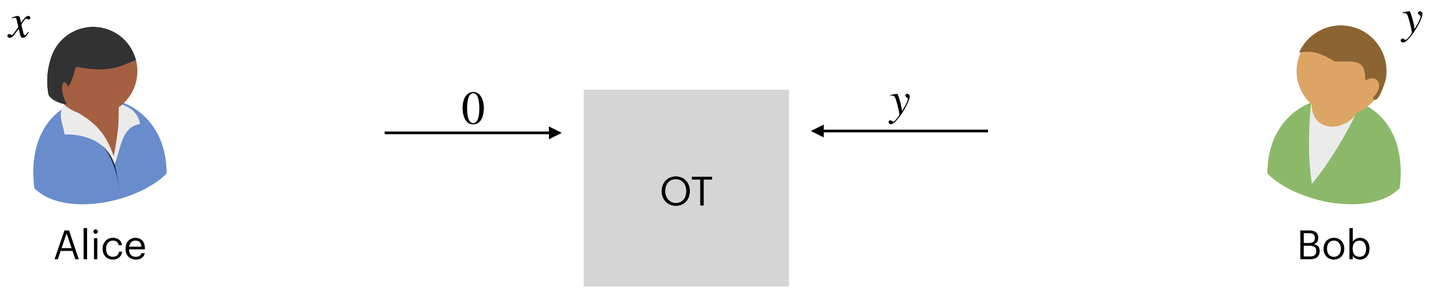
y

y

Bob

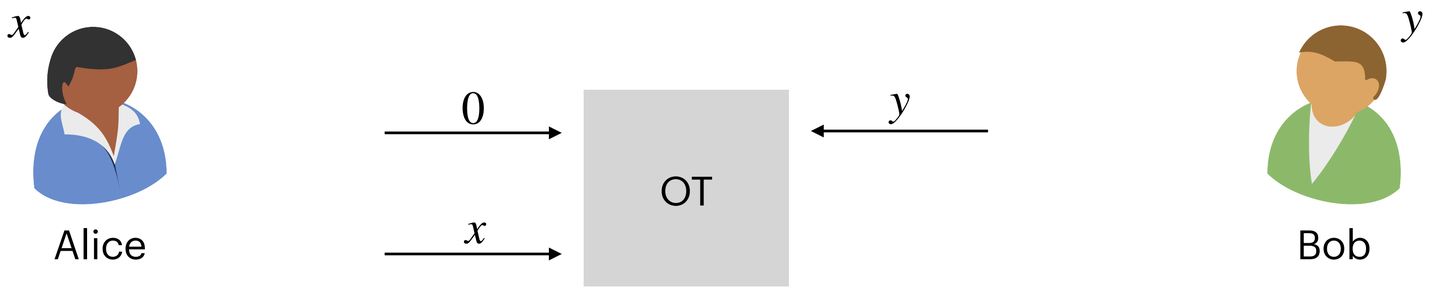
Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?



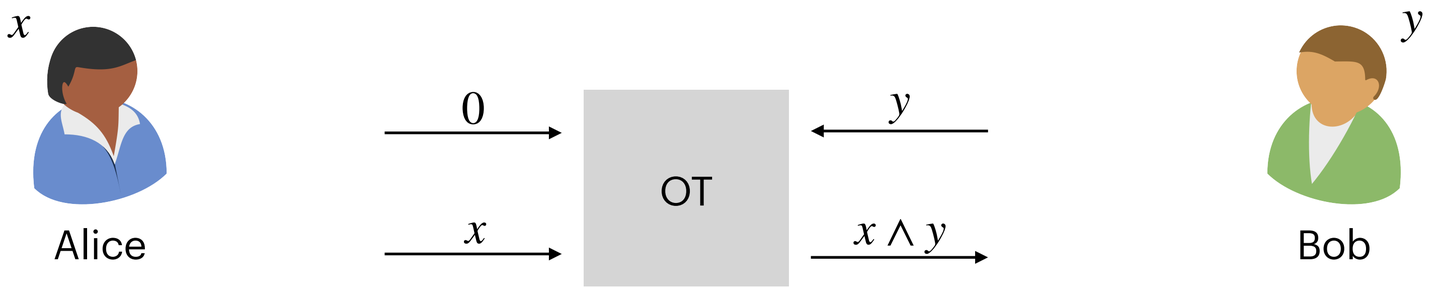
Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?



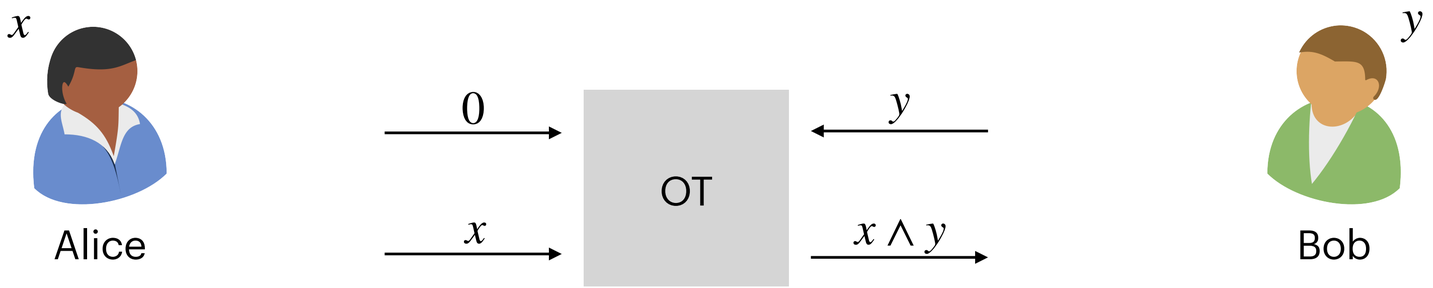
Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?



Towards General Secure Computation

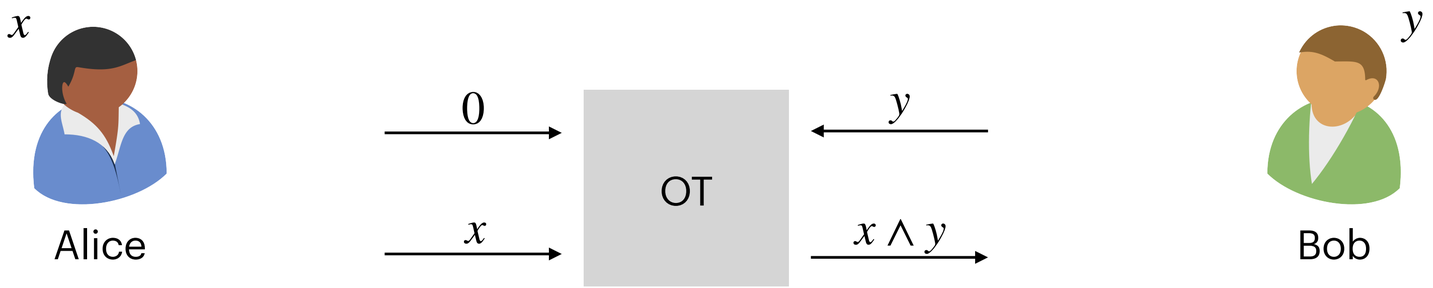
How to evaluate $f(x, y) = x \wedge y$ using OT?



- Bob can simply send the output to Alice.
- **Correctness:** If $y = 0$, Bob receives $f(x,0) = 0$; else, Bob receives $f(x,1) = x$ from OT correctness.
- **Security:** Follows directly from security of OT.

Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?

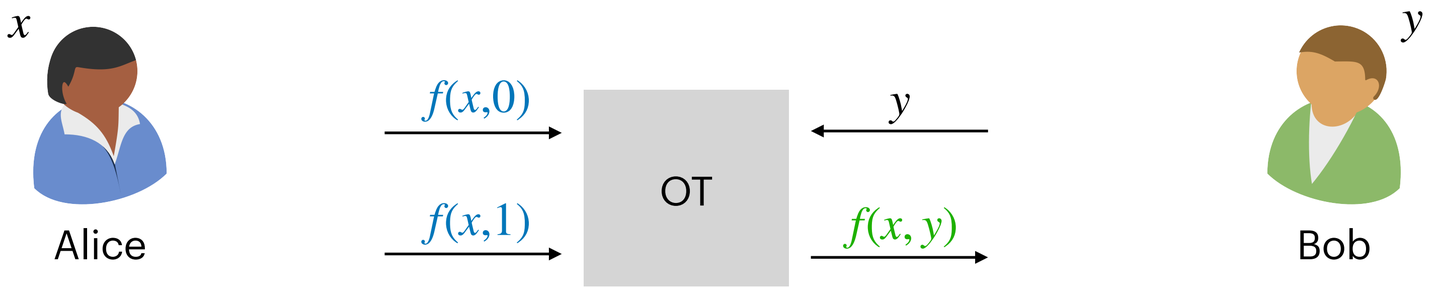


- Bob can simply send the output to Alice.
- **Correctness:** If $y = 0$, Bob receives $f(x,0) = 0$; else, Bob receives $f(x,1) = x$ from OT correctness.
- **Security:** Follows directly from security of OT.

Can we extend this idea to arbitrary functions?

Towards General Secure Computation

How to evaluate $f(x, y) = x \wedge y$ using OT?



Alice inputs the the truth table for $f(x, \cdot)$.
Bob selects the row to get $f(x, y)$.

- Bob can simply send the output to Alice.
- **Correctness:** If $y = 0$, Bob receives $f(x,0) = 0$; else, Bob receives $f(x,1) = x$ from OT correctness.
- **Security:** Follows directly from security of OT.

Can we extend this idea to arbitrary functions?

Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?

x

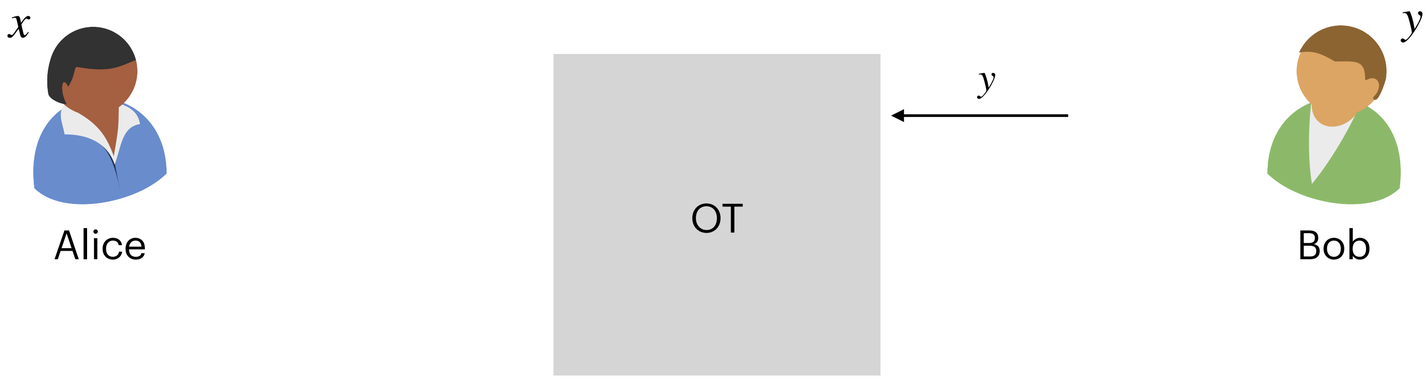
Alice

y

Bob

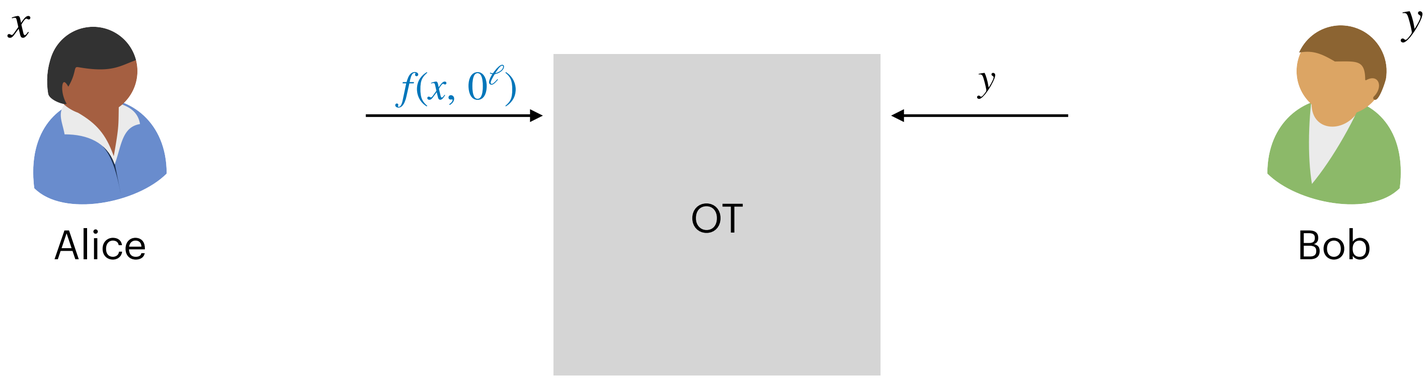
Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



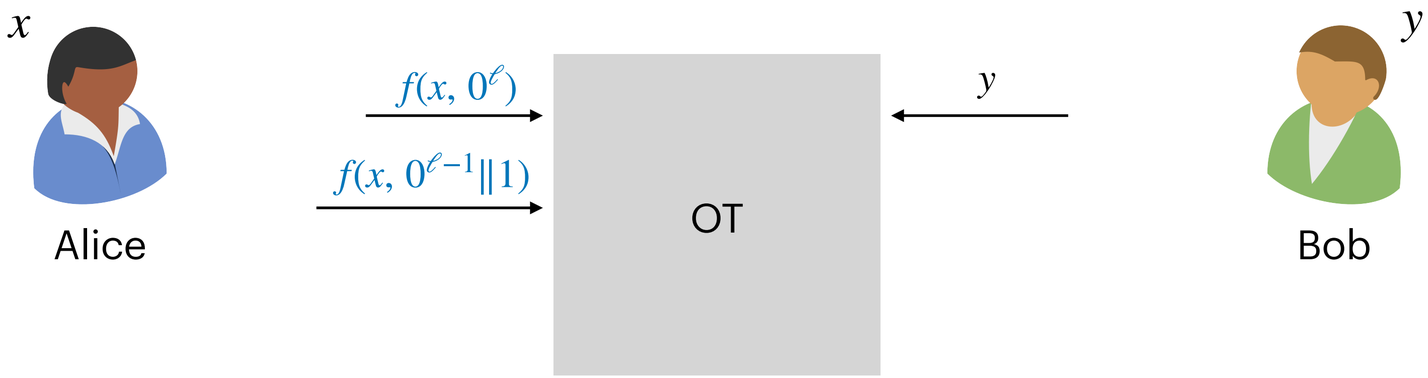
Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



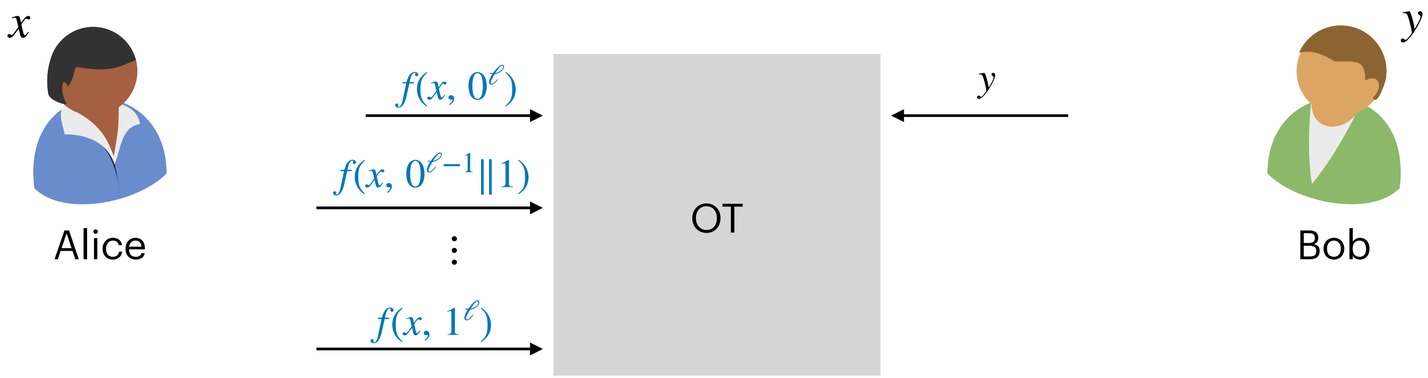
Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



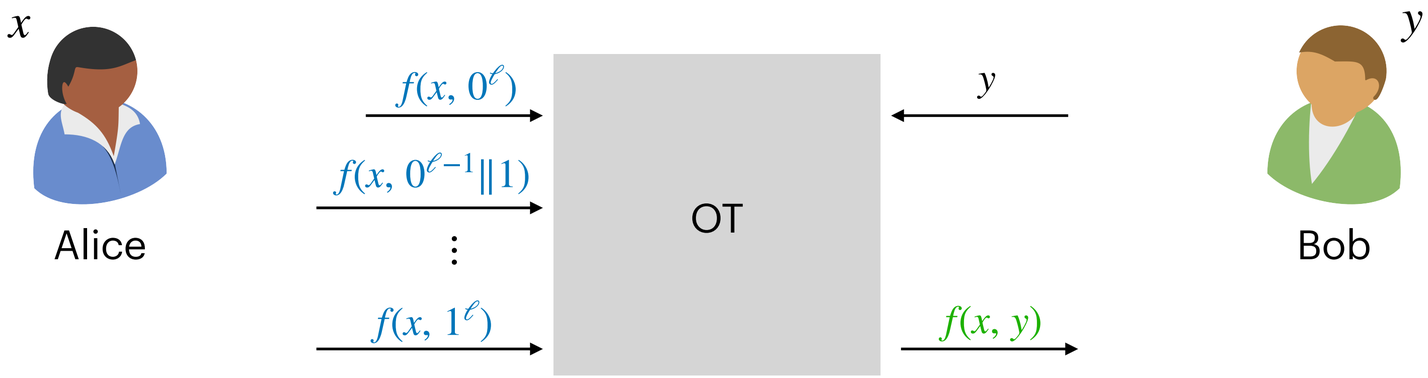
Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



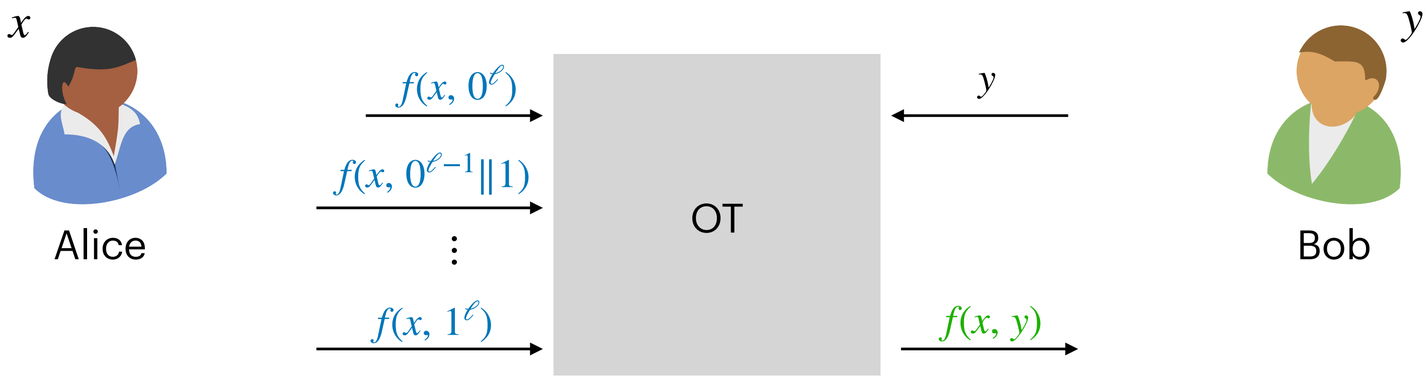
Evaluating Truth Tables

How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



Evaluating Truth Tables

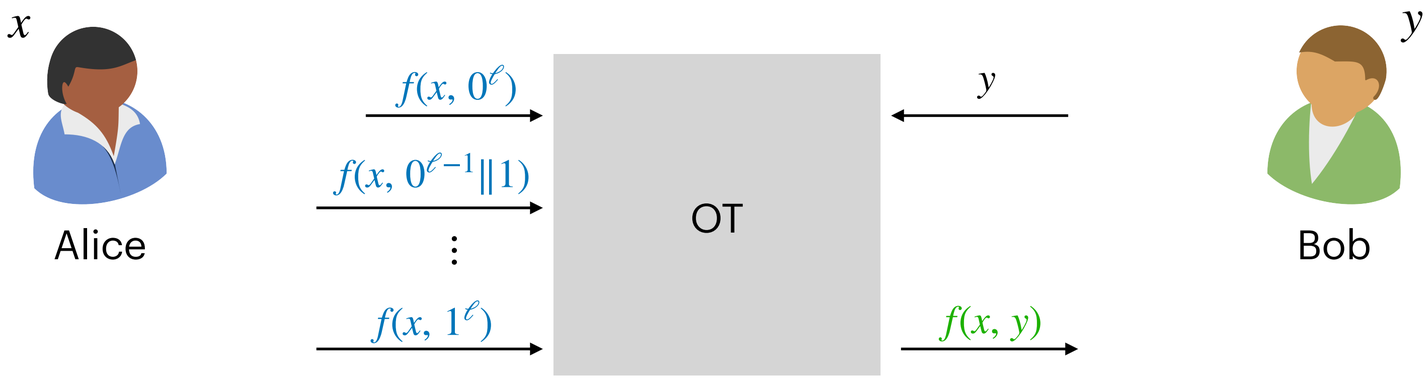
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Correctness and security follow from that of OT.

Evaluating Truth Tables

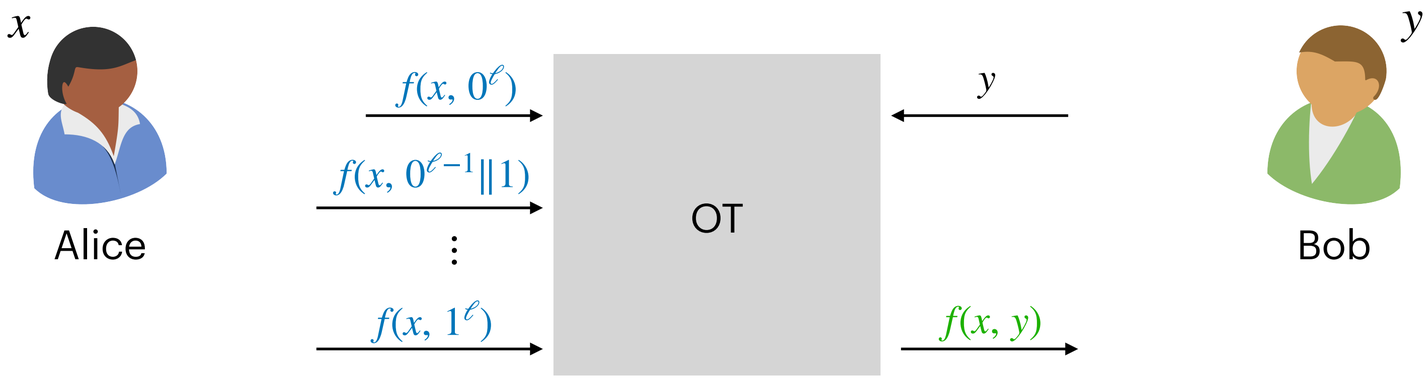
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Correctness and security follow from that of OT.
- Extends to randomized functions too.

Evaluating Truth Tables

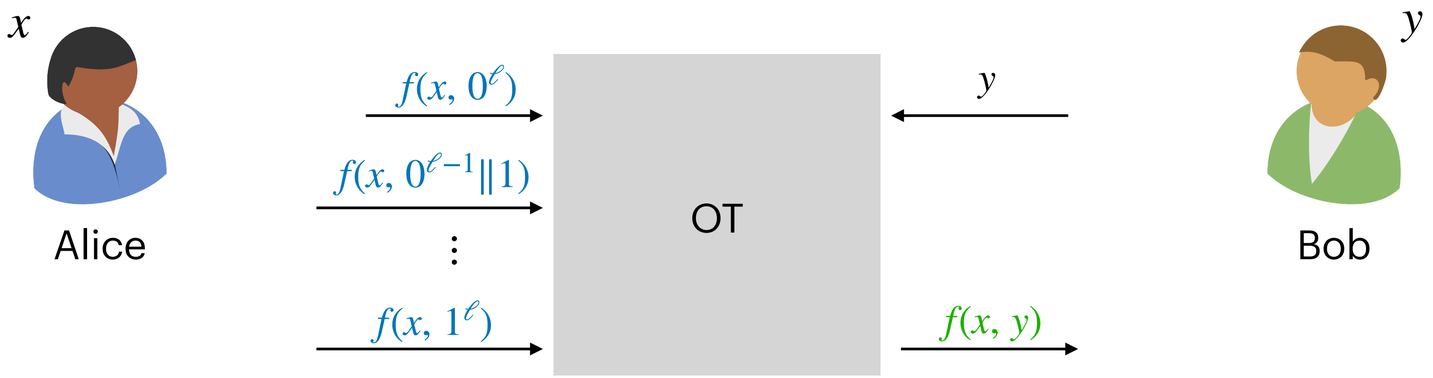
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Correctness and security follow from that of OT.
- Extends to randomized functions too.
  - To evaluate $f(x, y; r)$, Alice and Bob additionally input uniformly random r_1 and r_2 and we compute $f'(x, y, r_1, r_2) = f(x, y; r_1 \oplus r_2)$.

Evaluating Truth Tables

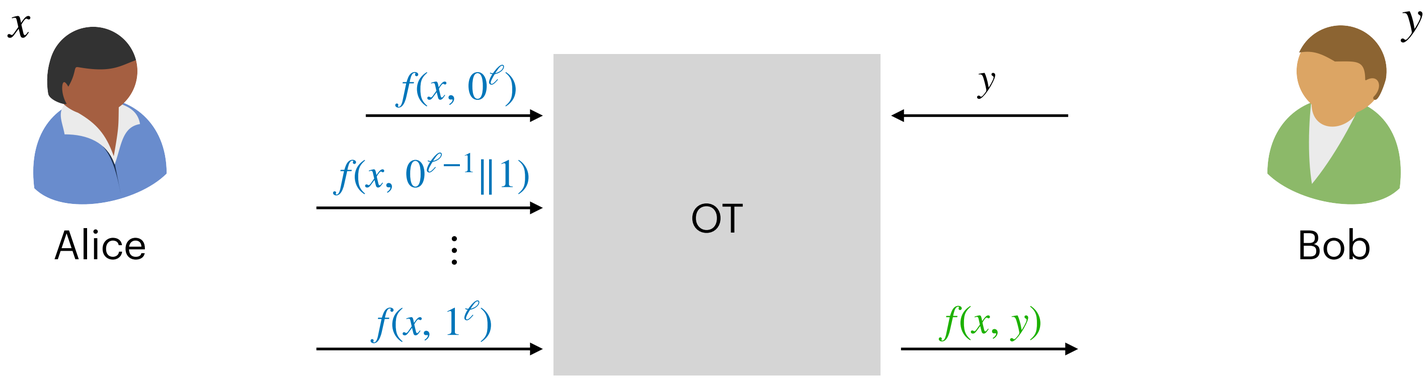
How to evaluate $f(x, y)$ using 1-out-of-2^ℓ OT for $x, y \in \{0,1\}^n$?



- Correctness and security follow from that of OT.
- Extends to randomized functions too.
  - To evaluate $f(x, y; r)$, Alice and Bob additionally input uniformly random r_1 and r_2 and we compute $f'(x, y, r_1, r_2) = f(x, y; r_1 \oplus r_2)$.
  - Suffices to focus on evaluating deterministic functions.

Evaluating Truth Tables

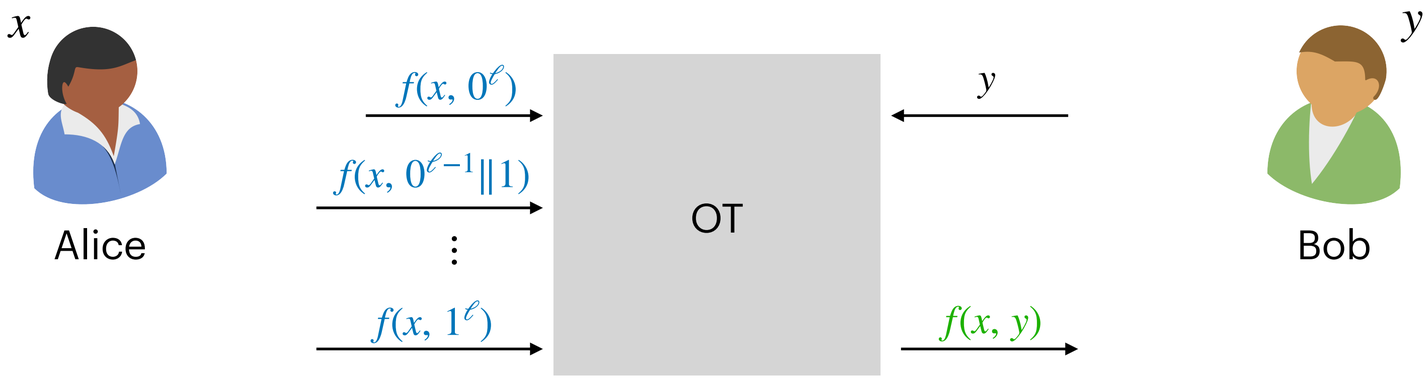
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Number of OT inputs is exponential in ℓ.
  - This approach is secure only for $\ell = O(\log(\lambda))$. We want to support $\ell = \text{poly}(\lambda)$.

Evaluating Truth Tables

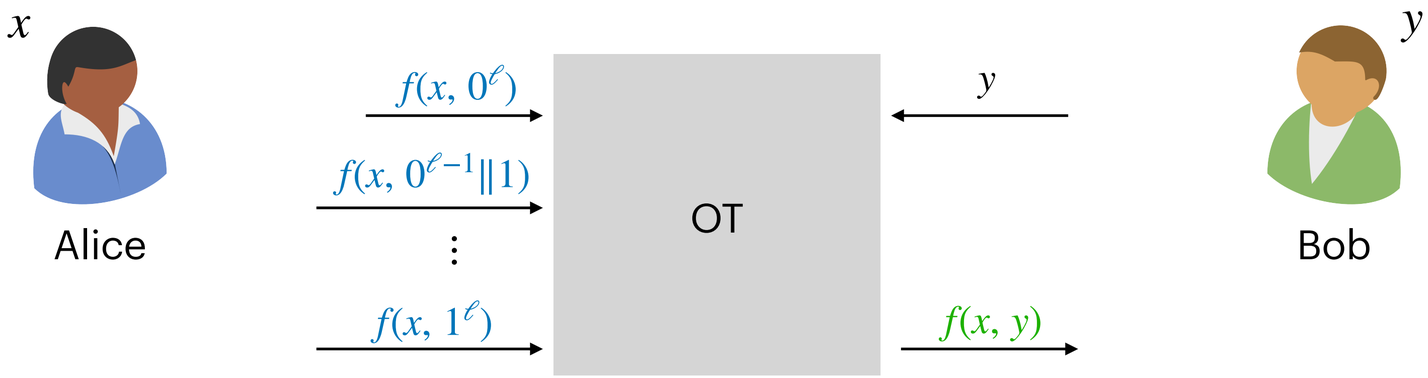
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Number of OT inputs is exponential in ℓ.
  - This approach is secure only for $\ell = O(\log(\lambda))$. We want to support $\ell = \text{poly}(\lambda)$.
- Truth tables are inefficient representations for arbitrary functions. We need a more efficient representation.

Evaluating Truth Tables

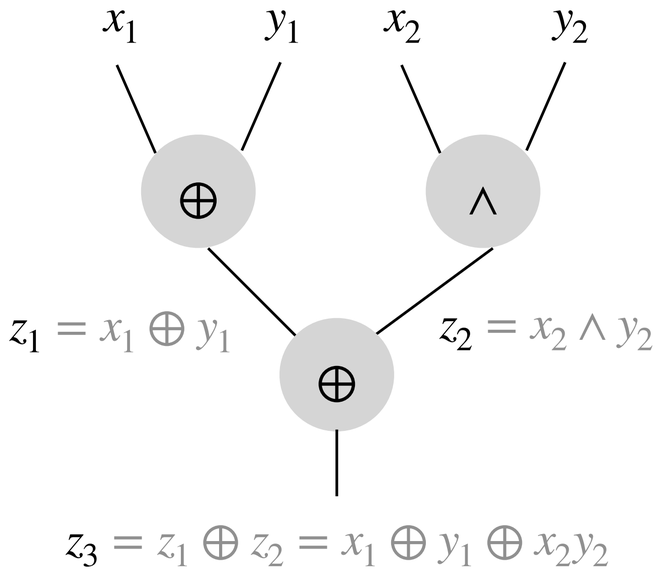
How to evaluate $f(x, y)$ using 1-out-of-2^{ℓ} OT for $x, y \in \{0,1\}^n$?



- Number of OT inputs is exponential in ℓ.
  - This approach is secure only for $\ell = O(\log(\lambda))$. We want to support $\ell = \text{poly}(\lambda)$.
- Truth tables are inefficient representations for arbitrary functions. We need a more efficient representation.
- We will evaluate circuits!
  - We will extend the above idea to evaluate each gate.

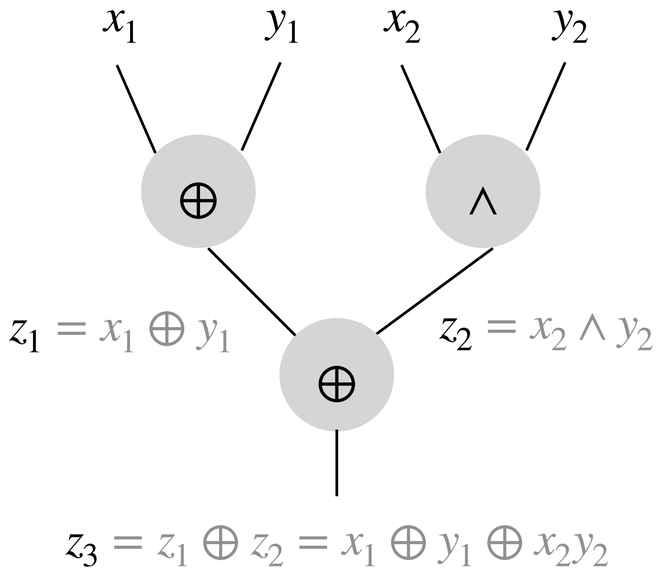
Towards Evaluating Circuits

- Circuits consist of 2-input AND and XOR gates.
  - Evaluating circuits with $\mathsf{poly}(\lambda)$ gates captures evaluating PPT algorithms.



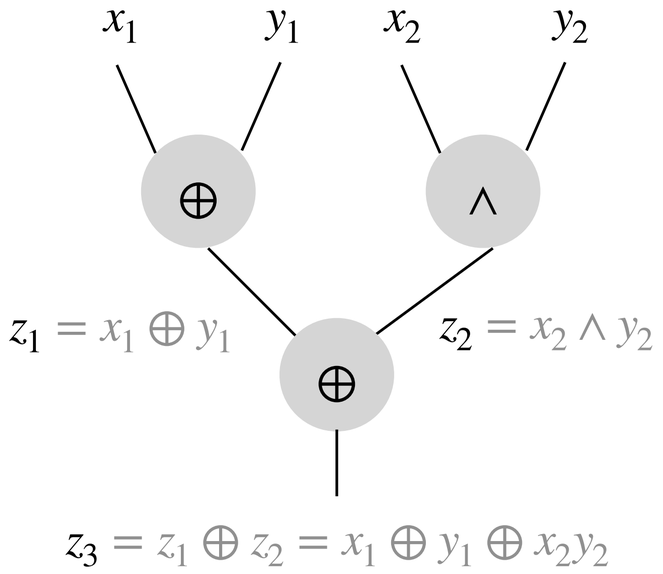
Towards Evaluating Circuits

- Circuits consist of 2-input AND and XOR gates.
  - Evaluating circuits with $\mathsf{poly}(\lambda)$ gates captures evaluating PPT algorithms.
- Secure computation of circuit:
  - Inputs are provided by one of the parties.
  - The parties must obtain the final output.
  - Honest party's input and internal wire values must be kept secret from the adversary.



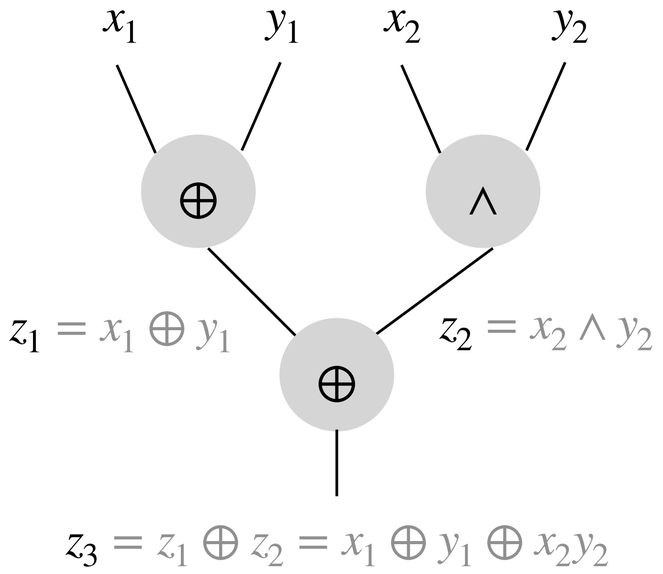
Towards Evaluating Circuits

- Circuits consist of 2-input AND and XOR gates.
  - Evaluating circuits with poly(λ) gates captures evaluating PPT algorithms.
- Secure computation of circuit:
  - Inputs are provided by one of the parties.
  - The parties must obtain the final output.
  - Honest party's input and internal wire values must be kept secret from the adversary.
- How to hide a secret across parties?
  - Encryption?



Towards Evaluating Circuits

- Circuits consist of 2-input AND and XOR gates.
  - Evaluating circuits with poly(λ) gates captures evaluating PPT algorithms.
- Secure computation of circuit:
  - Inputs are provided by one of the parties.
  - The parties must obtain the final output.
  - Honest party's input and internal wire values must be kept secret from the adversary.
- How to hide a secret across parties?
  - Encryption? Who has the secret key?



Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don’t learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.

Dealer

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.

Dealer

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.

Dealer

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.

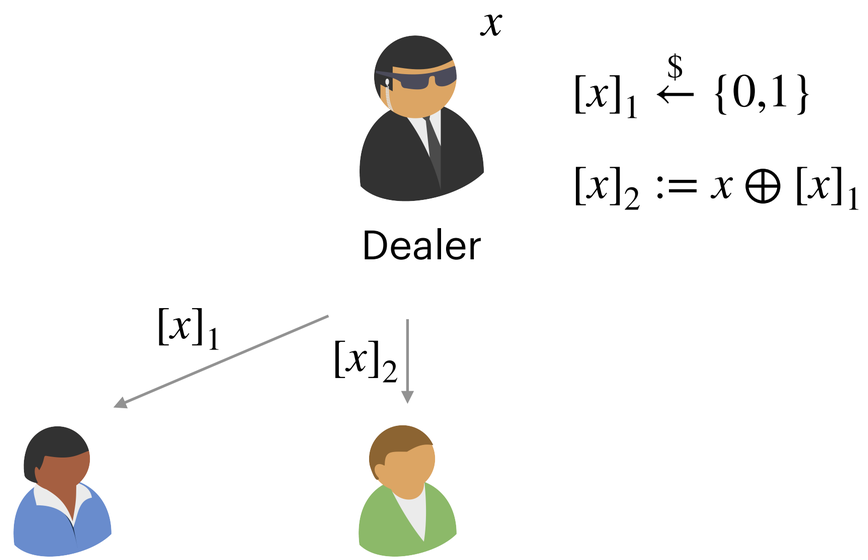
Dealer

$$[x]_1 \xleftarrow{\$} \{0,1\}$$

$$[x]_2 := x \oplus [x]_1$$

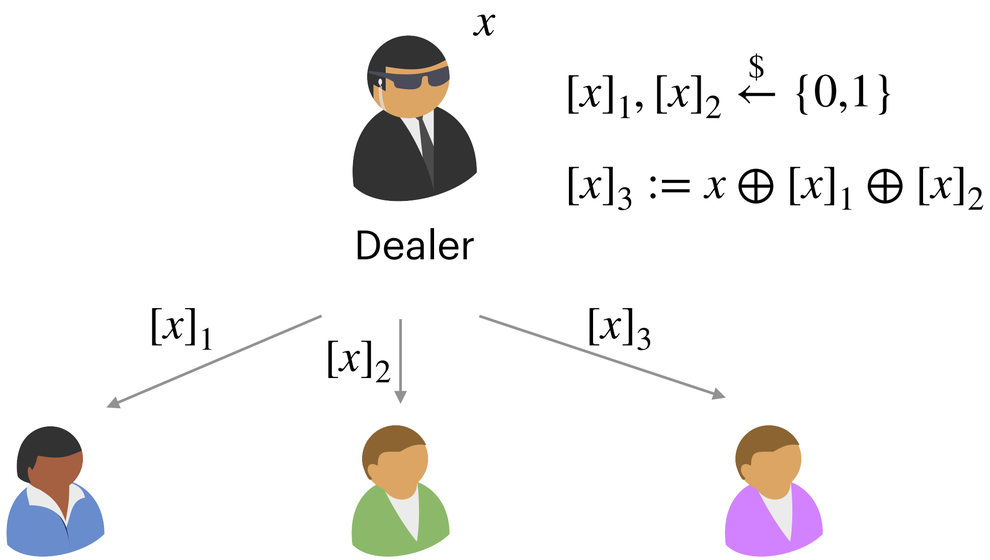
Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.



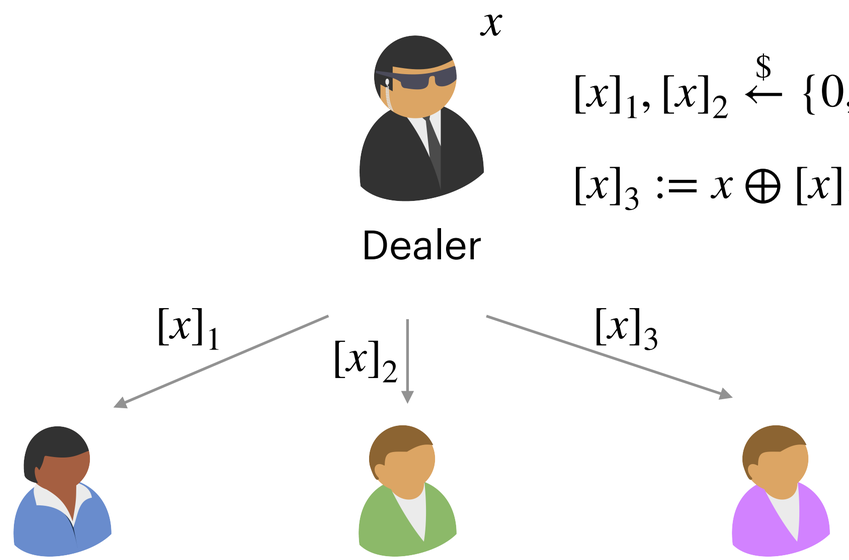
Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.



Secret Sharing

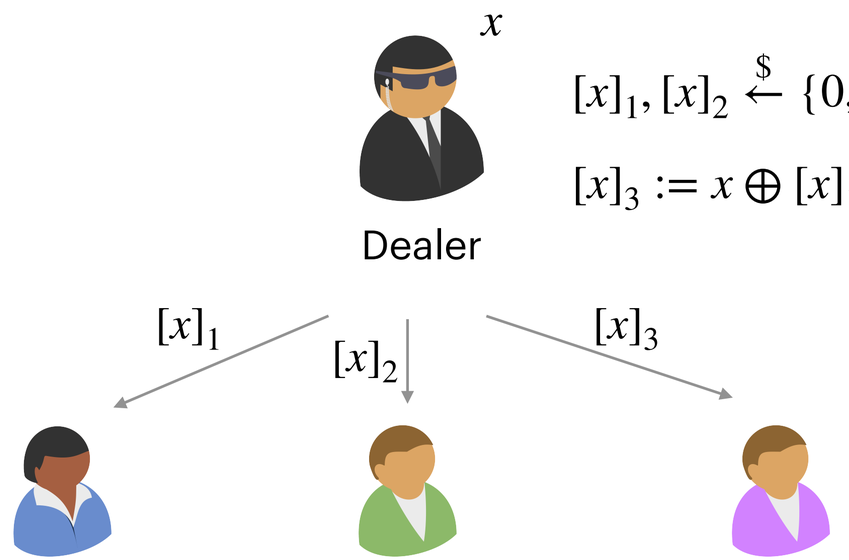
- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.



Correctness:

Secret Sharing

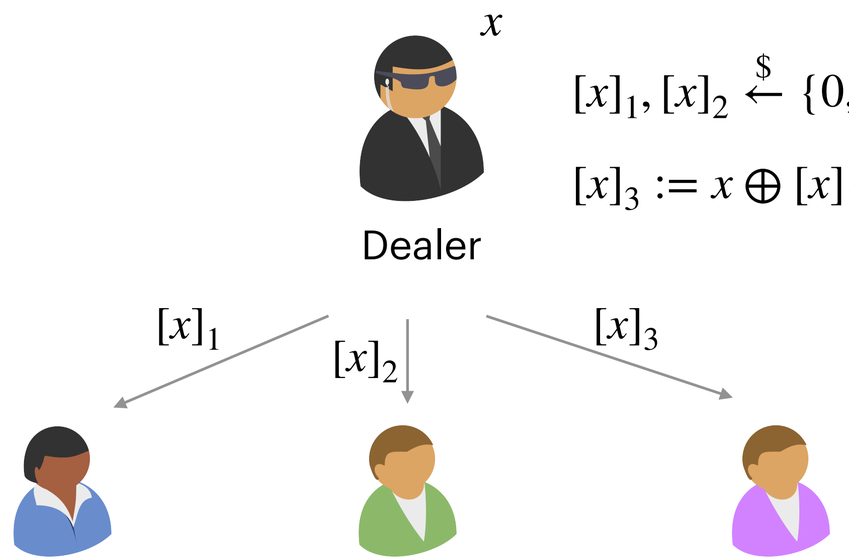
- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.



Correctness: To reconstruct compute $x := \oplus_{i=1}^{n} [x]_i$.

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.

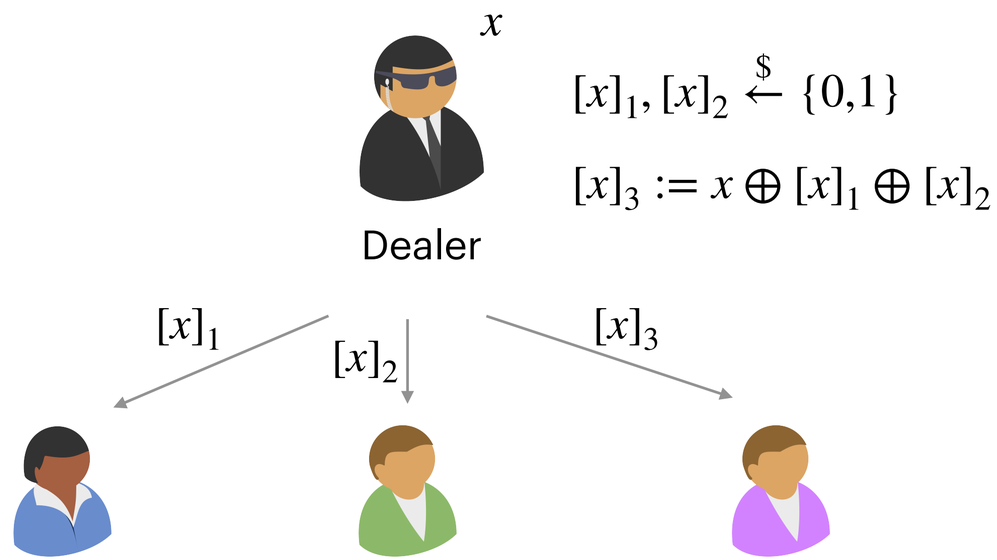


Correctness: To reconstruct compute $x := \oplus_{i=1}^{n} [x]_i$.

Security:

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.

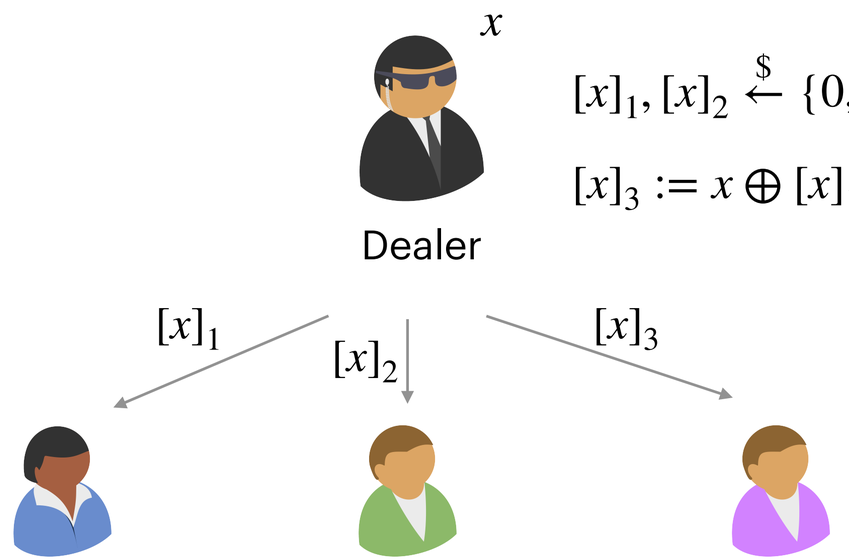


Correctness: To reconstruct compute $x := \oplus_{i=1}^{n} [x]_i$.

Security: Any set of $n - 1$shares is uniformly random.

Secret Sharing

- **Goal:** Distribute a secret among the parties such that
  - Corrupt parties don't learn anything about the secret from their shares, and
  - All parties can come together to reconstruct the secret.
- Can be extended to (k, n)-secret sharing.
  - Any k parties can reconstruct but a group of less than k corrupt parties cannot learn anything about the secret.
- Additive Secret Sharing
  - An n-out-of-n secret sharing scheme.
  - Sample $[x]_1, \ldots, [x]_{n-1} \xleftarrow{\$} \{0,1\}$.
  - Compute $[x]_n = x \oplus_{i=1}^{n-1} [x]_i$.



Correctness: To reconstruct compute $x := \oplus_{i=1}^{n} [x]_i$.

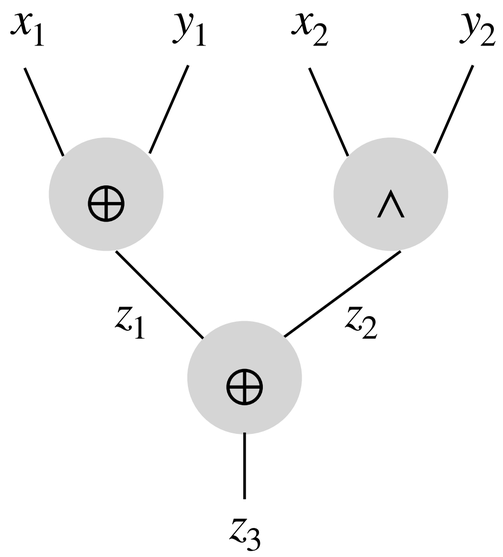
Security: Any set of $n - 1$shares is uniformly random.

Simulator?

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

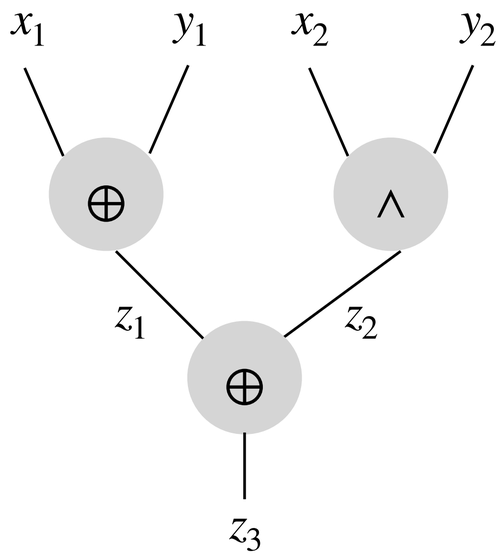
Plaintext Circuit Evaluation



Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation

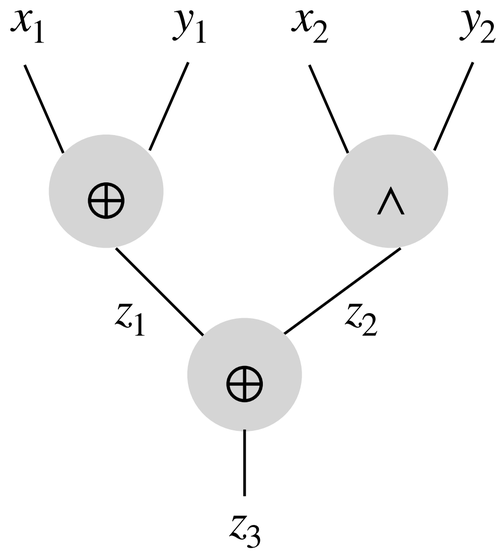


Secure Computation

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



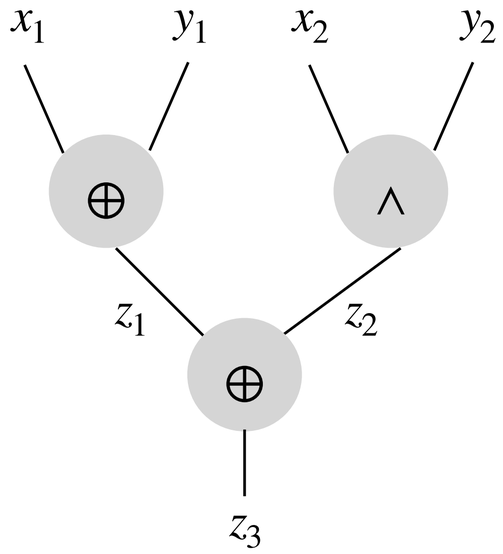
Secure Computation

- Each party can secret share its inputs.

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

$[x_1]_A$

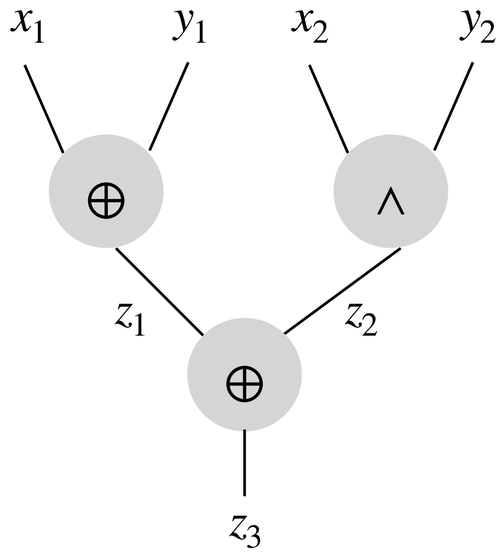
$[x_1]_B$

- Each party can secret share its inputs.

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

$[x_1]_A$ $[y_1]_A$

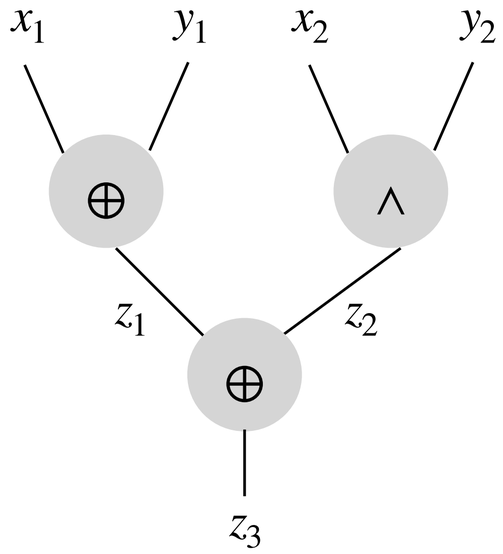
$[x_1]_B$ $[y_1]_B$

- Each party can secret share its inputs.

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

$[x_1]_A$ $[y_1]_A$ $[x_2]_A$ $[y_2]_A$

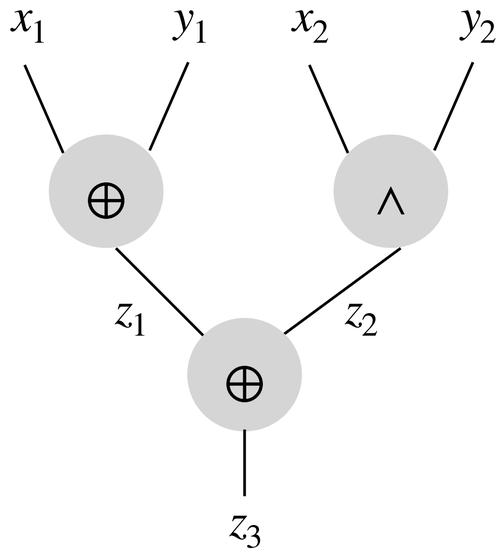
$[x_1]_B$ $[y_1]_B$ $[x_2]_B$ $[y_2]_B$

- Each party can secret share its inputs.

Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

$[x_1]_A$ $[y_1]_A$ $[x_2]_A$ $[y_2]_A$

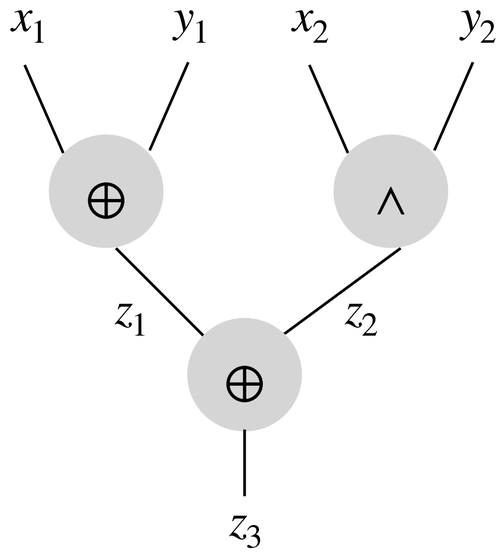
$[x_1]_B$ $[y_1]_B$ $[x_2]_B$ $[y_2]_B$

- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.

Template for Secure Computation

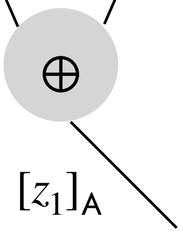
Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

$[x_1]_A$ $[y_1]_A$ $[x_2]_A$ $[y_2]_A$



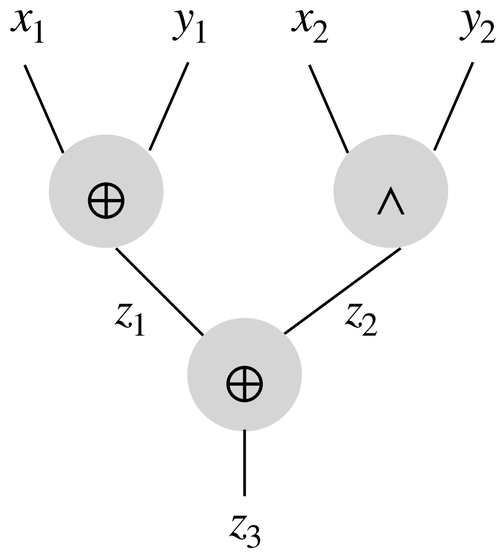
$[x_1]_B$ $[y_1]_B$ $[x_2]_B$ $[y_2]_B$

- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.

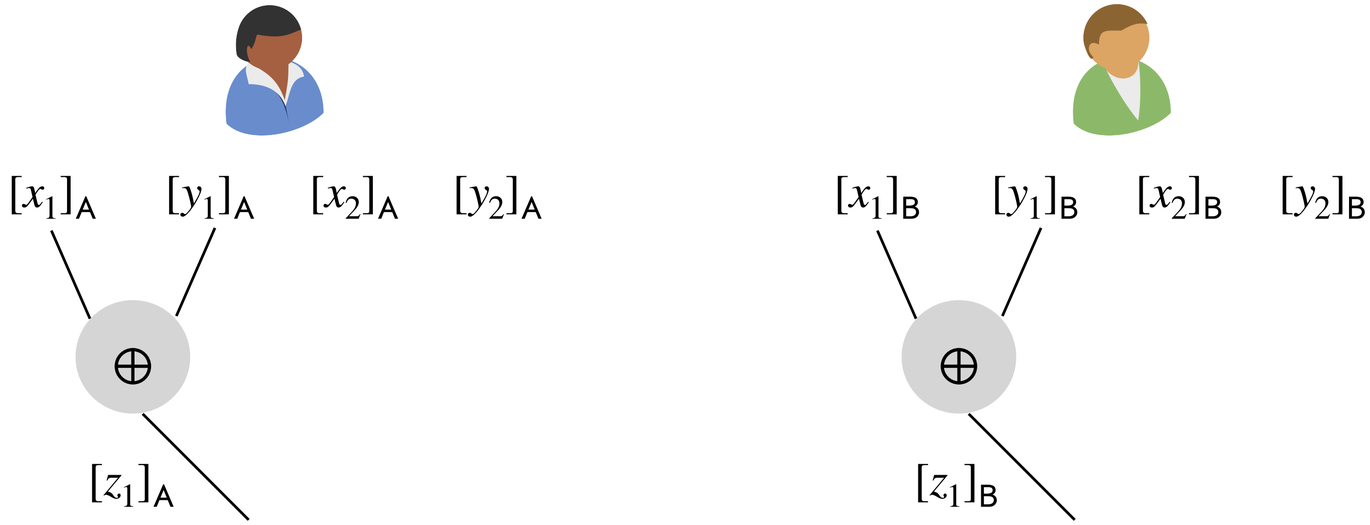
Template for Secure Computation

Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

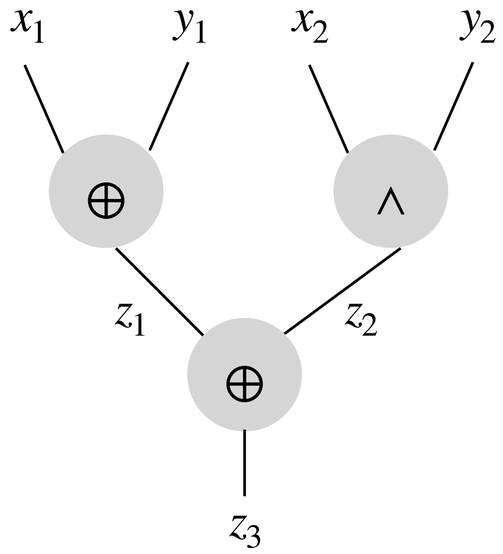


- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.

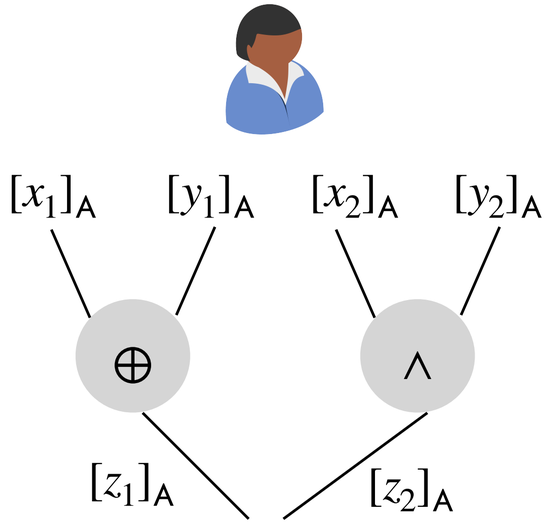
Template for Secure Computation

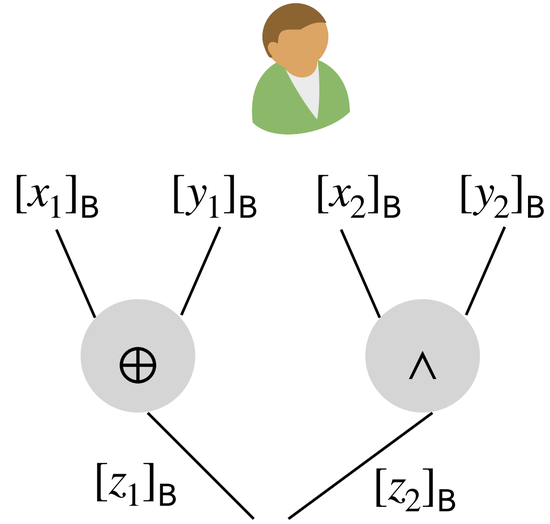
Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation



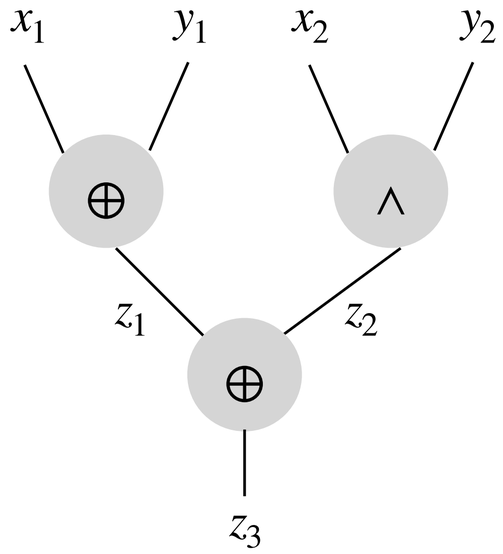


- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.

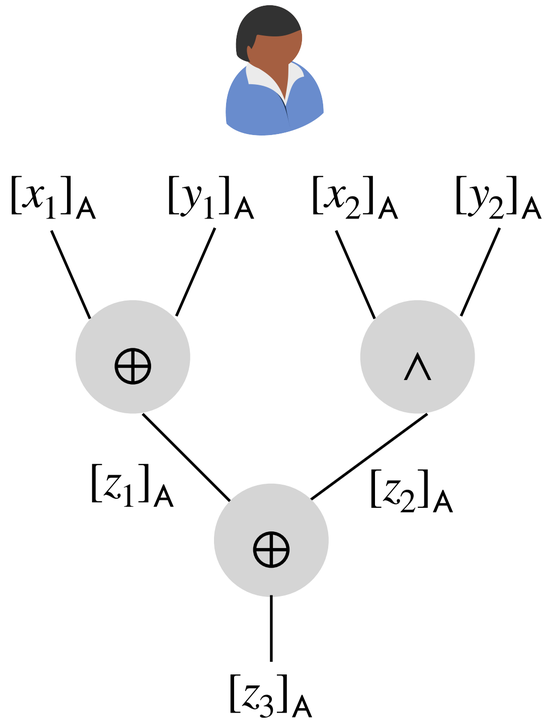
Template for Secure Computation

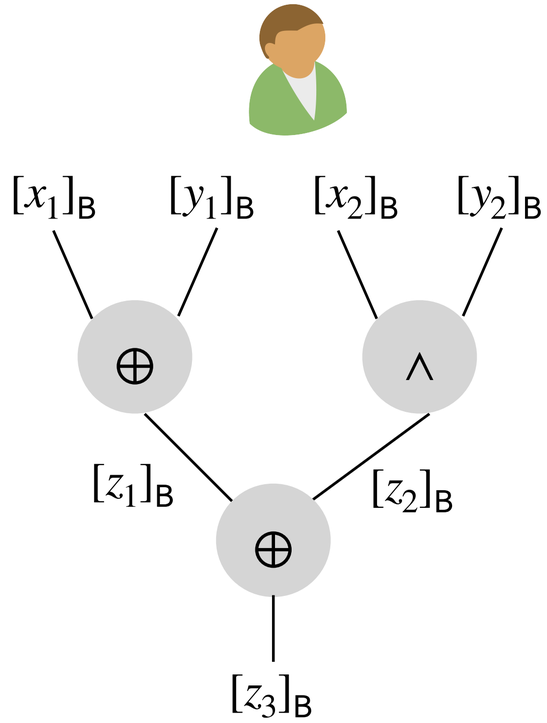
Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation



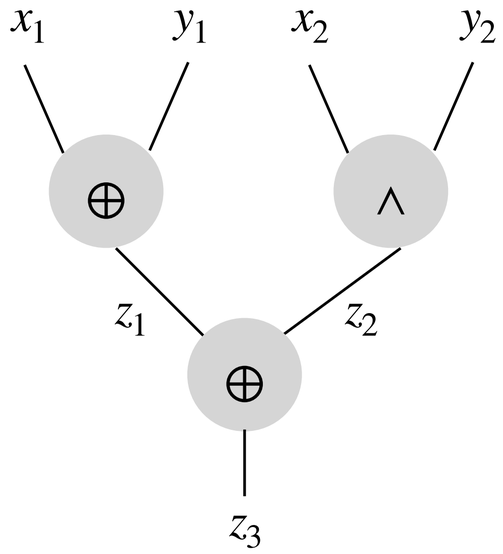


- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.

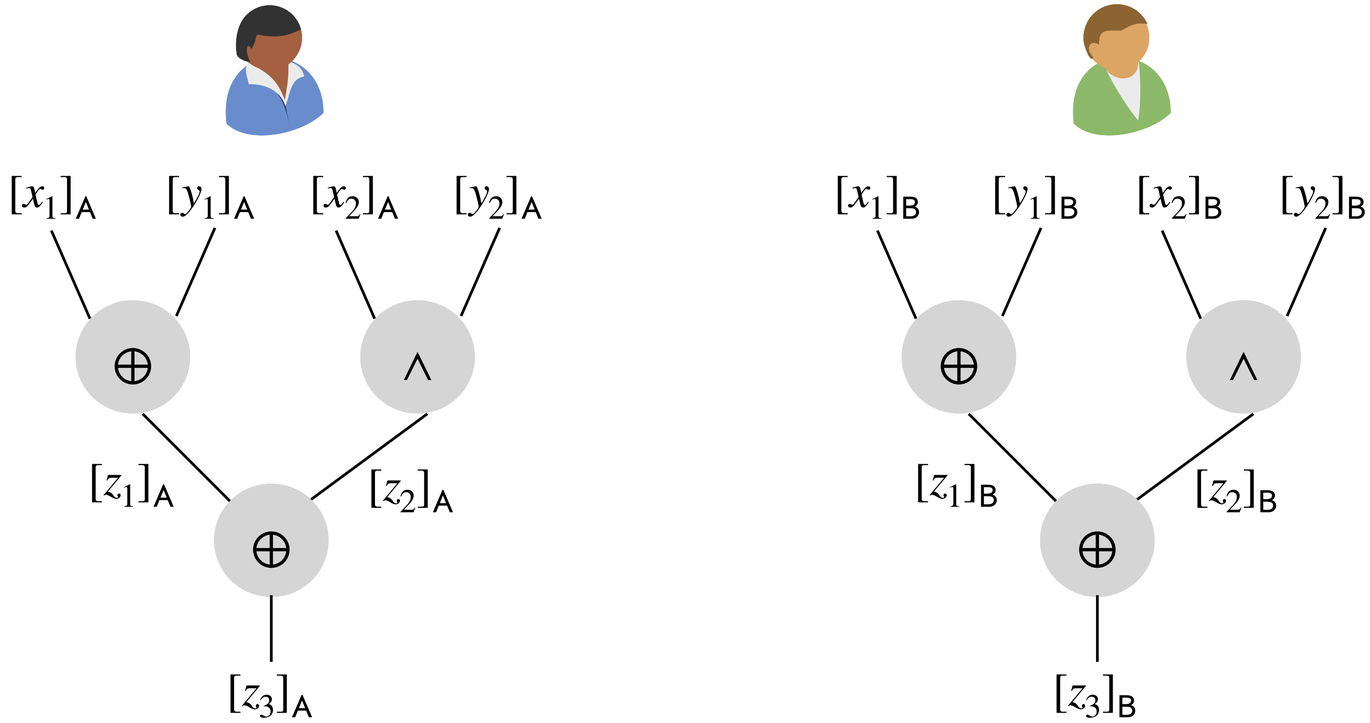
Template for Secure Computation

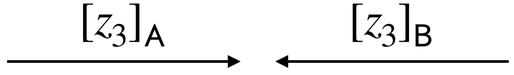
Perform secret-shared evaluation of the circuit.

Plaintext Circuit Evaluation



Secure Computation

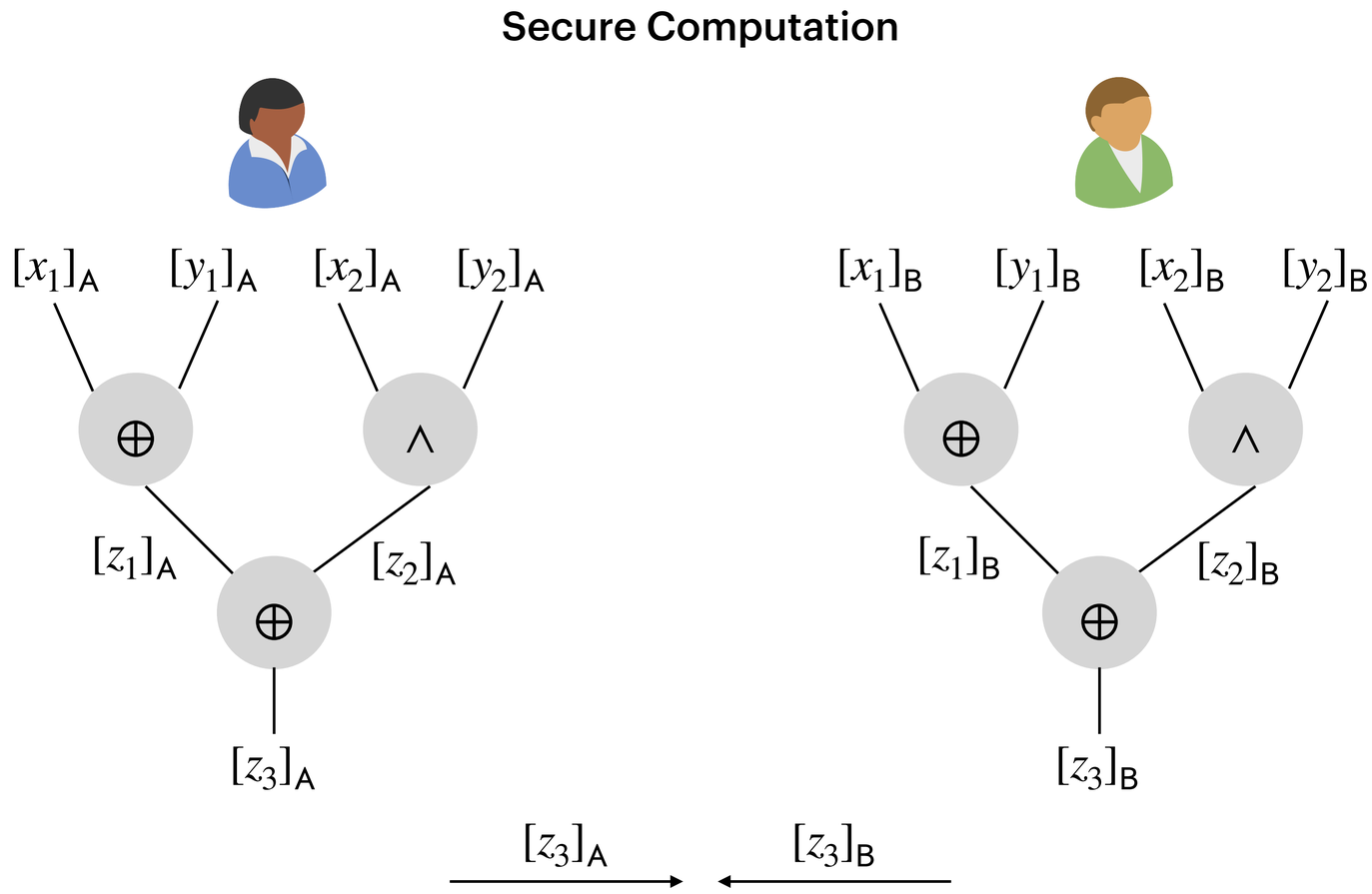




- Each party can secret share its inputs.
- **Invariant:** Parties have shares of each wire value.
- Parties can reconstruct the output share.

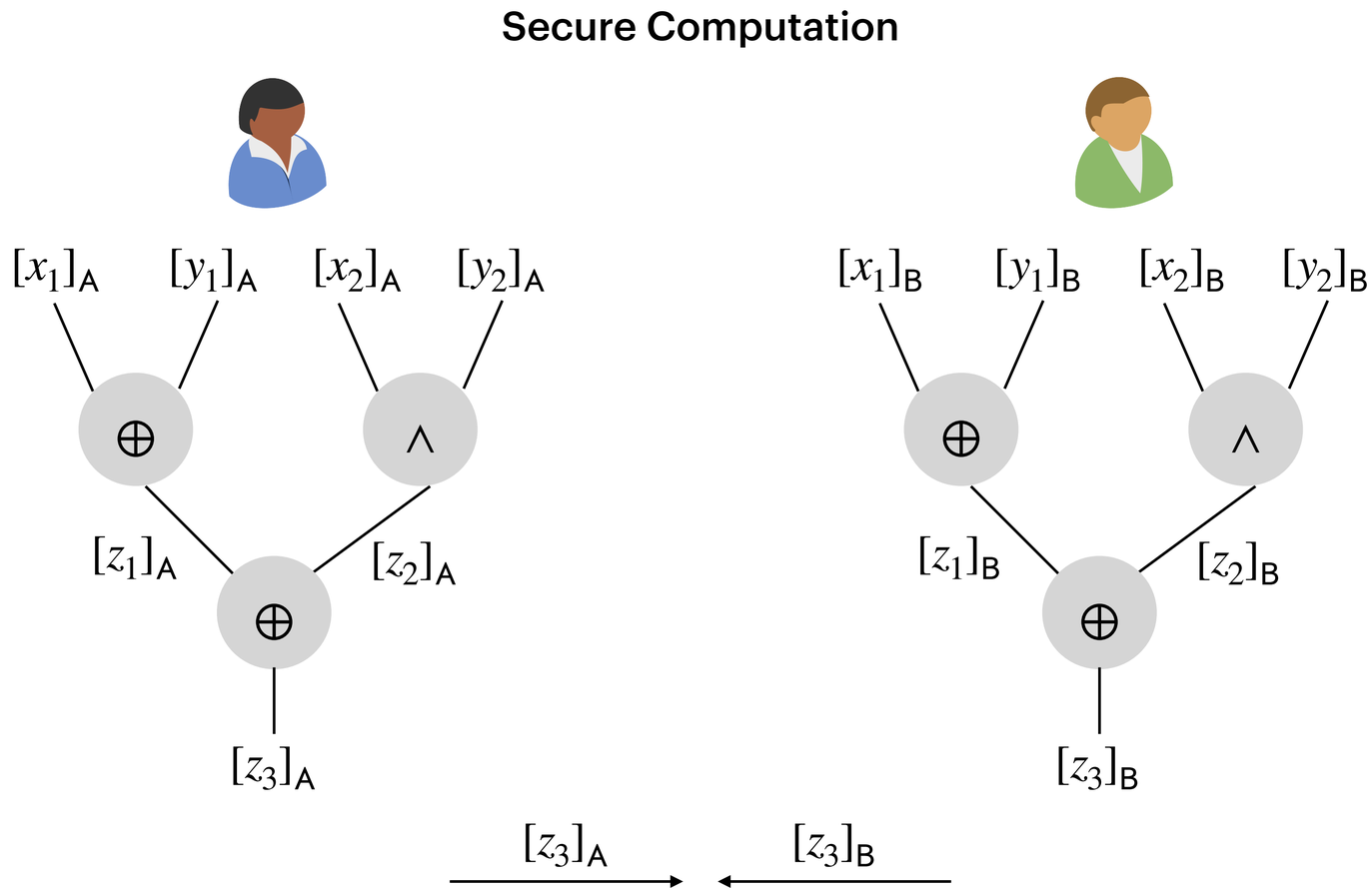
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.



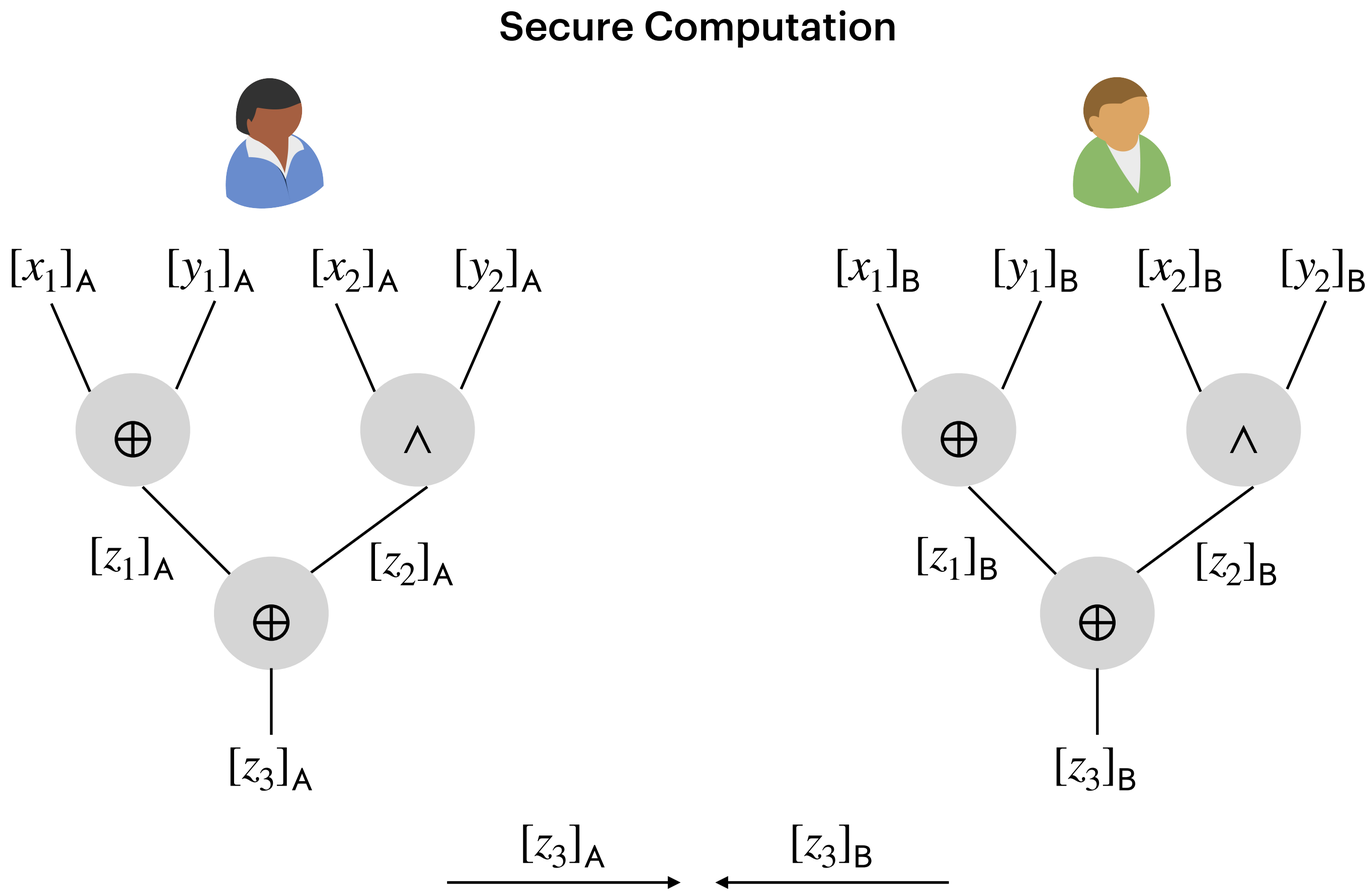
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?



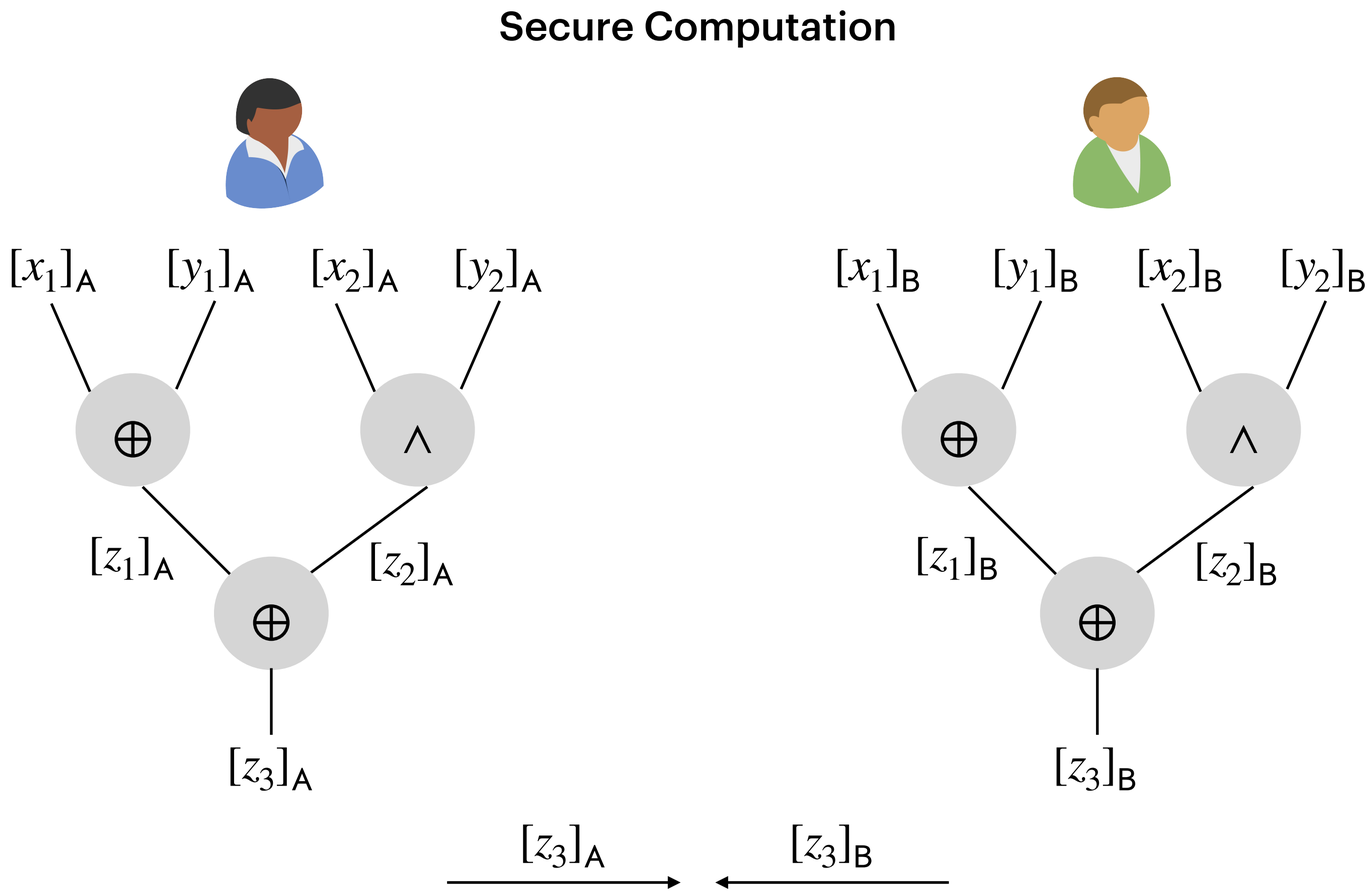
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.



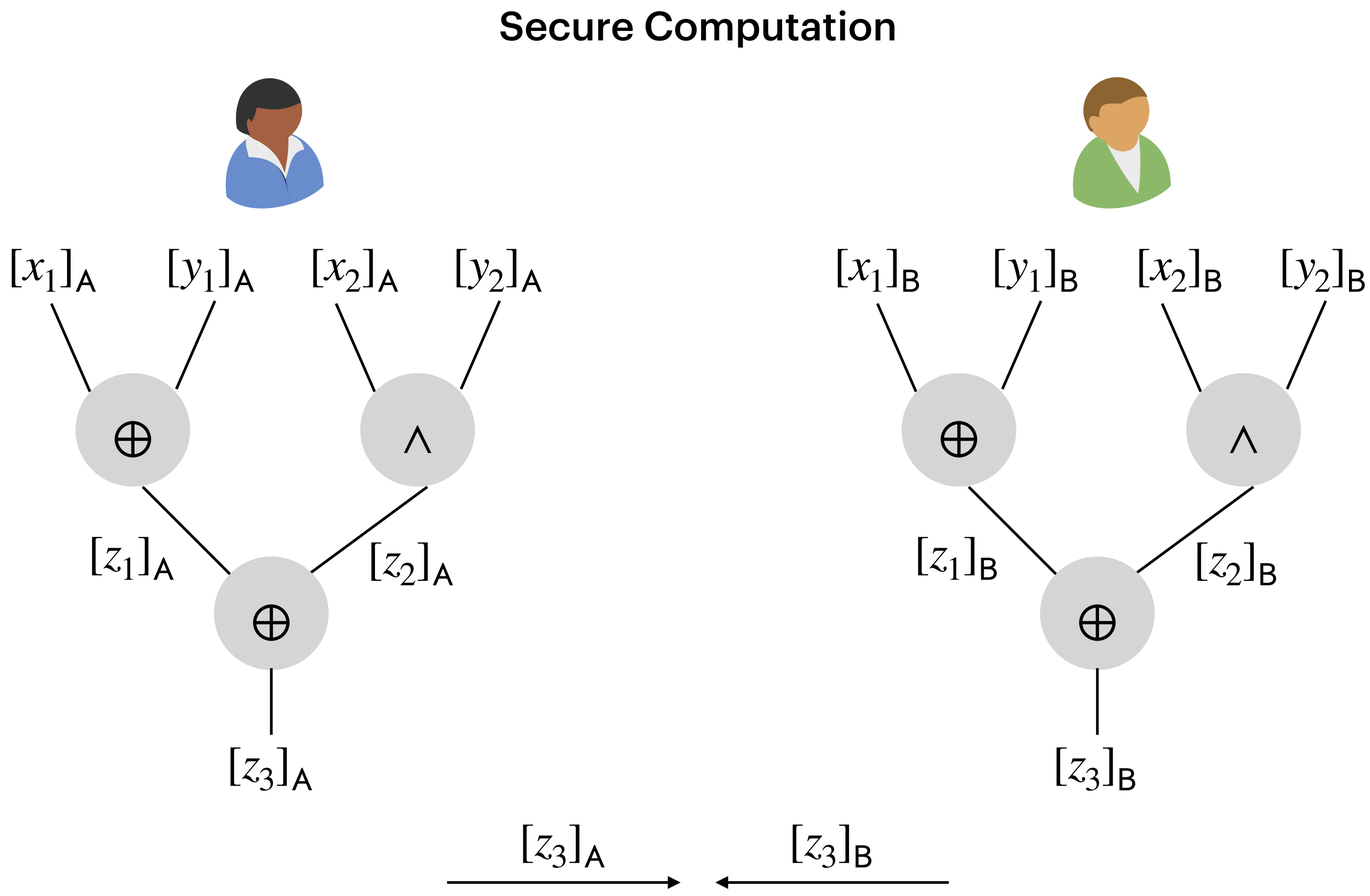
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.
  - Simulator for Alice $S(x_1, x_2, f(x_1, x_2, y_1, y_2))$



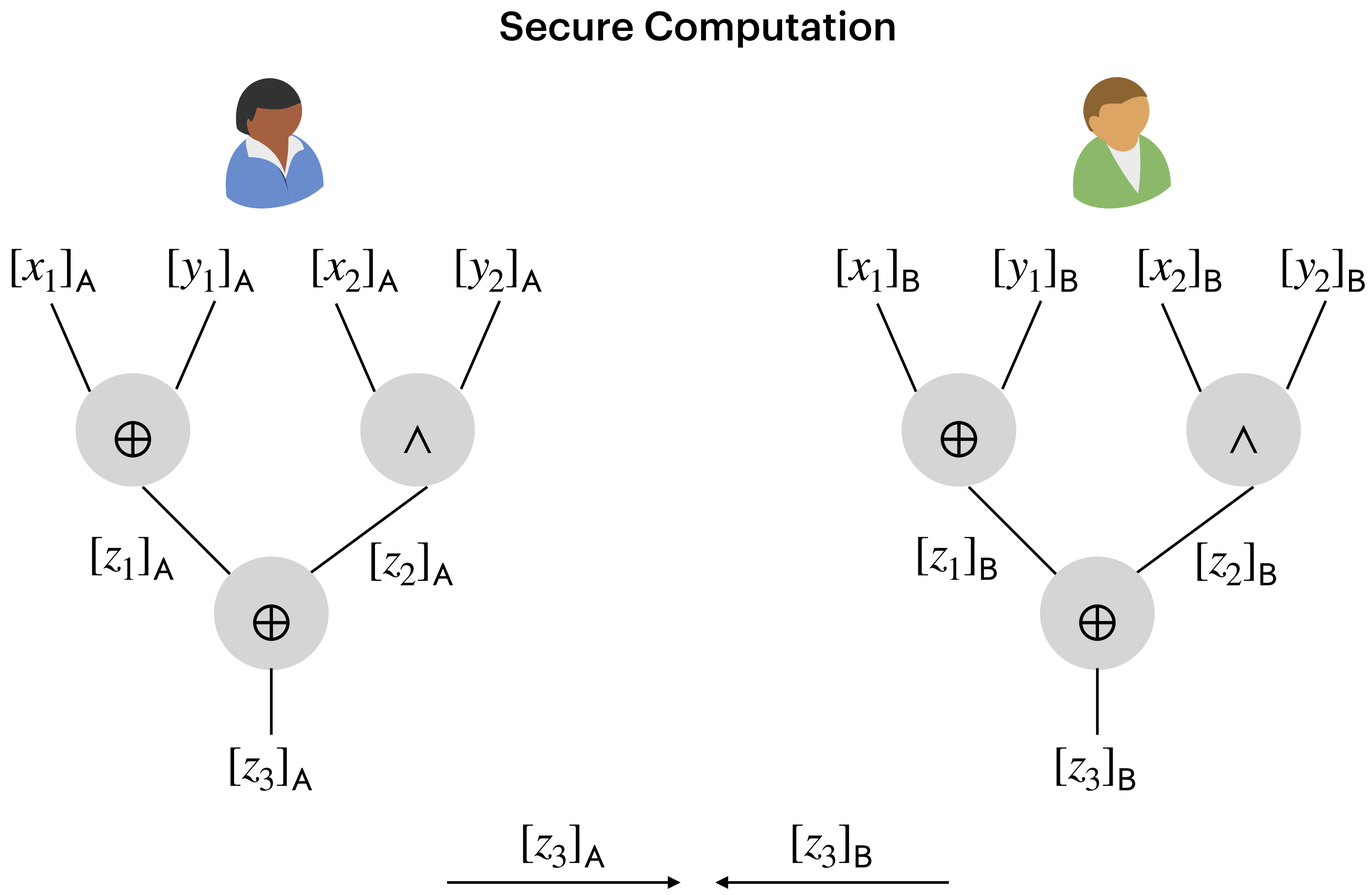
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.
  - Simulator for Alice $S(x_1, x_2, f(x_1, x_2, y_1, y_2))$
    - Use secret-sharing simulator for input shares.



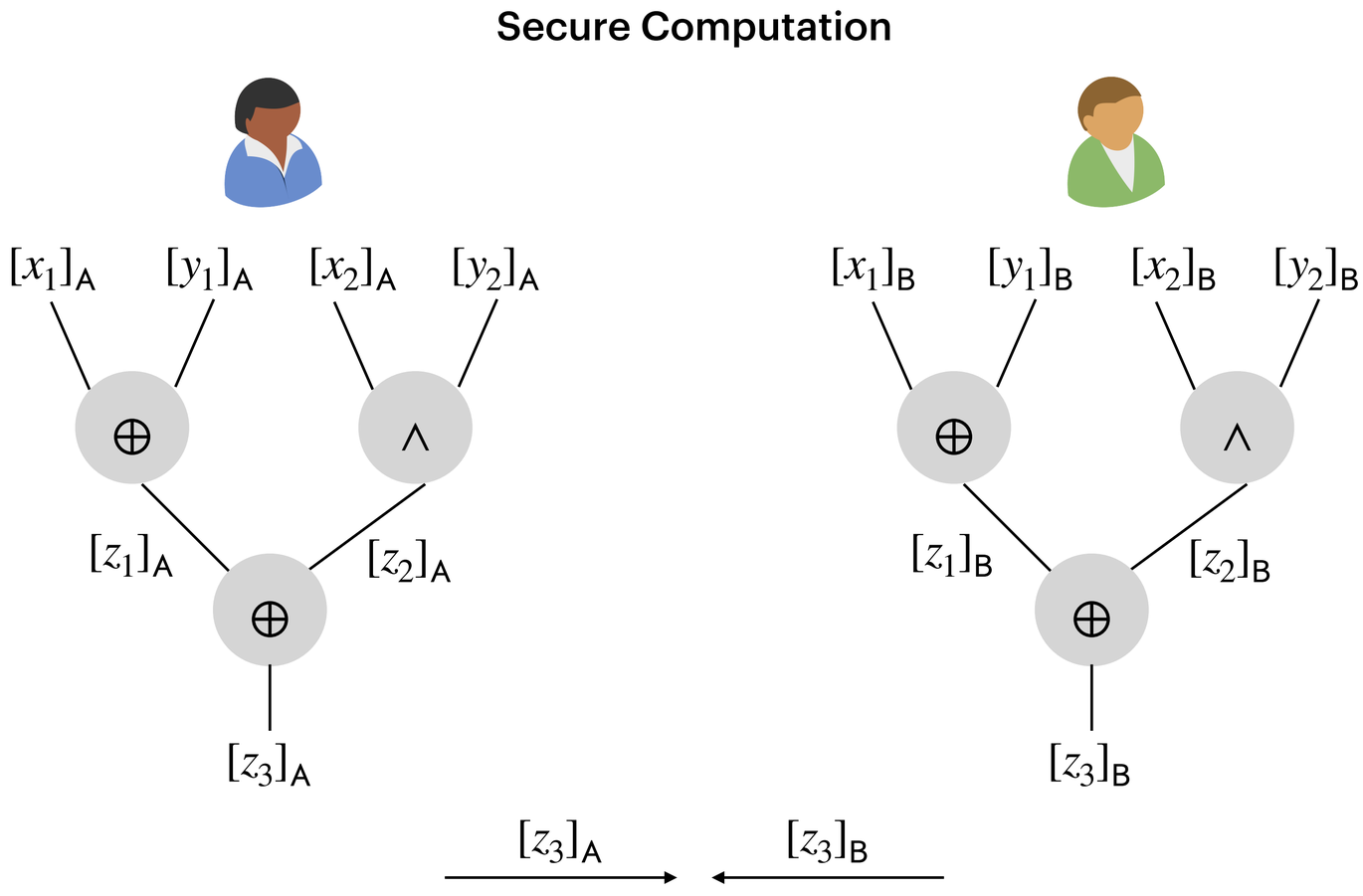
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.
  - Simulator for Alice $S(x_1, x_2, f(x_1, x_2, y_1, y_2))$
    - Use secret-sharing simulator for input shares.
    - **Simulator for each gate:** Given shares on input wires, compute share on output wire.



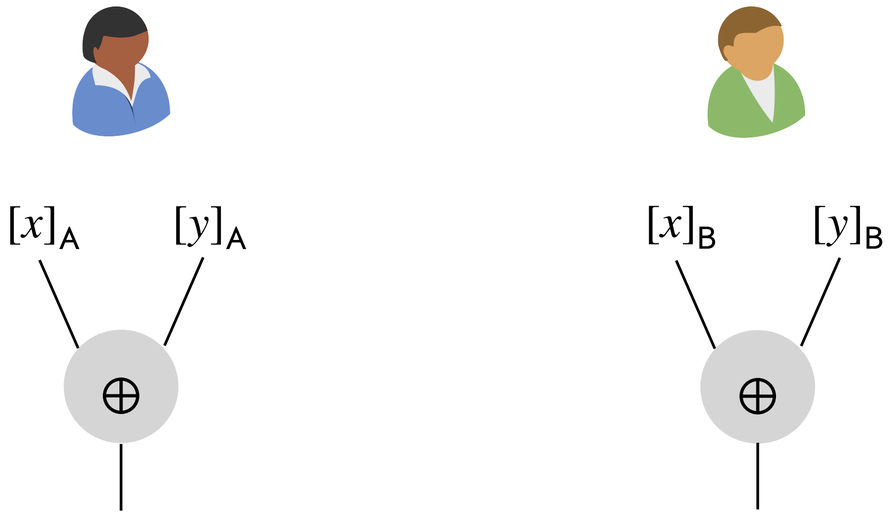
Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.
  - Simulator for Alice $S(x_1, x_2, f(x_1, x_2, y_1, y_2))$
    - Use secret-sharing simulator for input shares.
    - **Simulator for each gate:** Given shares on input wires, compute share on output wire.
    - Compute output share $[z_3]_A$ and simulate $[z_3]_B := f(x_1, x_2, y_1, y_2) \oplus [z_3]_A$.



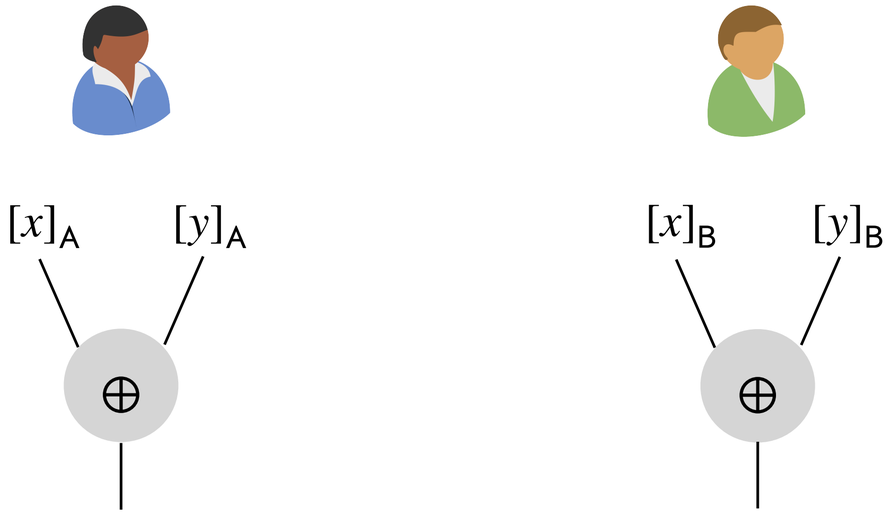
Evaluating XOR Gates

- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.



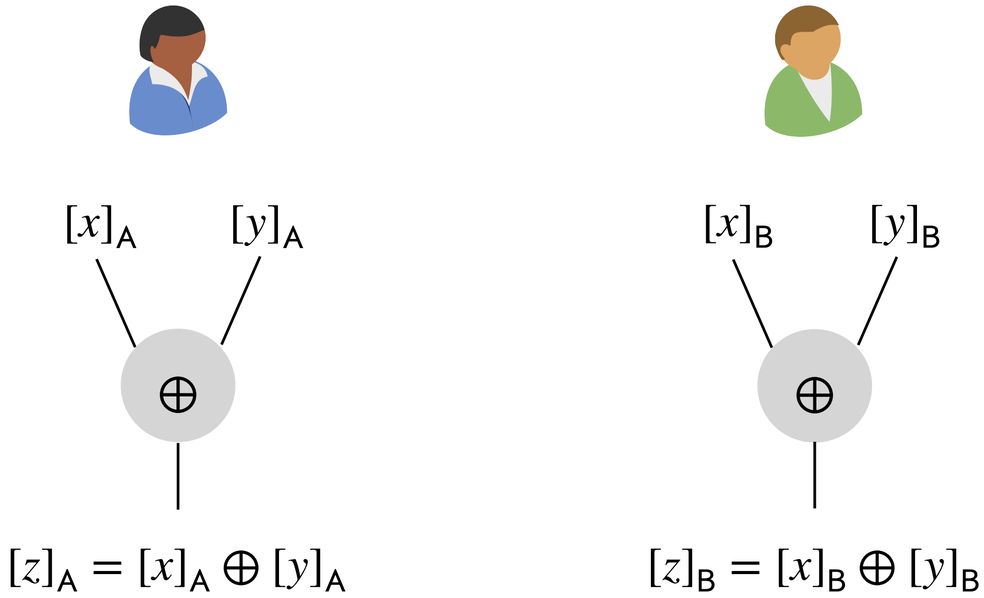
Evaluating XOR Gates

- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.
- Each party XORs its shares.
  - Additive secret sharing supports evaluating XOR gates for “free” – it does not require any communication.



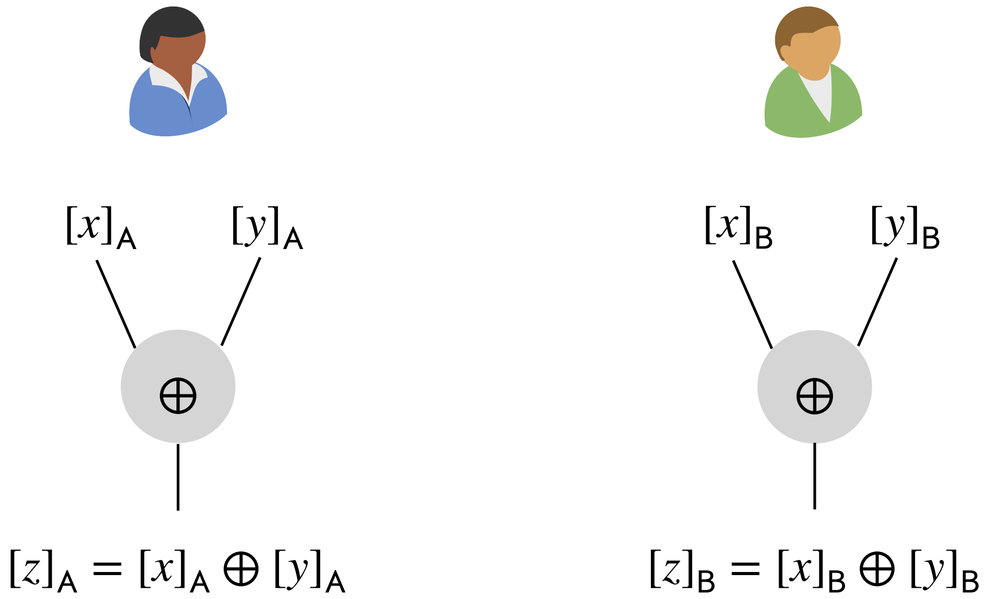
Evaluating XOR Gates

- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.
- Each party XORs its shares.
  - Additive secret sharing supports evaluating XOR gates for “free” – it does not require any communication.



Evaluating XOR Gates

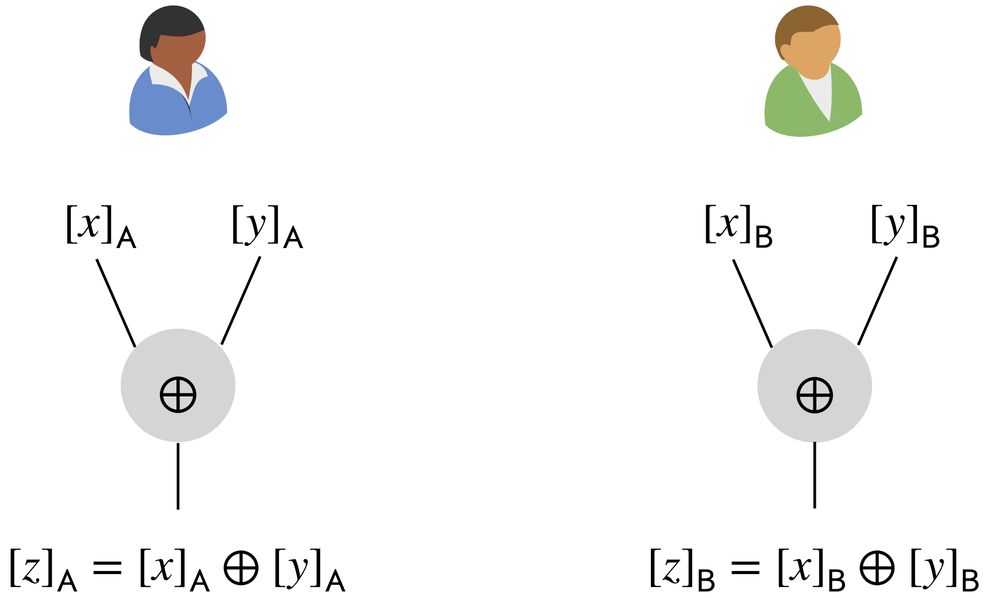
- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.
- Each party XORs its shares.
  - Additive secret sharing supports evaluating XOR gates for "free" – it does not require any communication.



Correctness: $z = [z]_A \oplus [z]_B = [x]_A \oplus [y]_A \oplus [x]_B \oplus [y]_B = x \oplus y\,.$

Evaluating XOR Gates

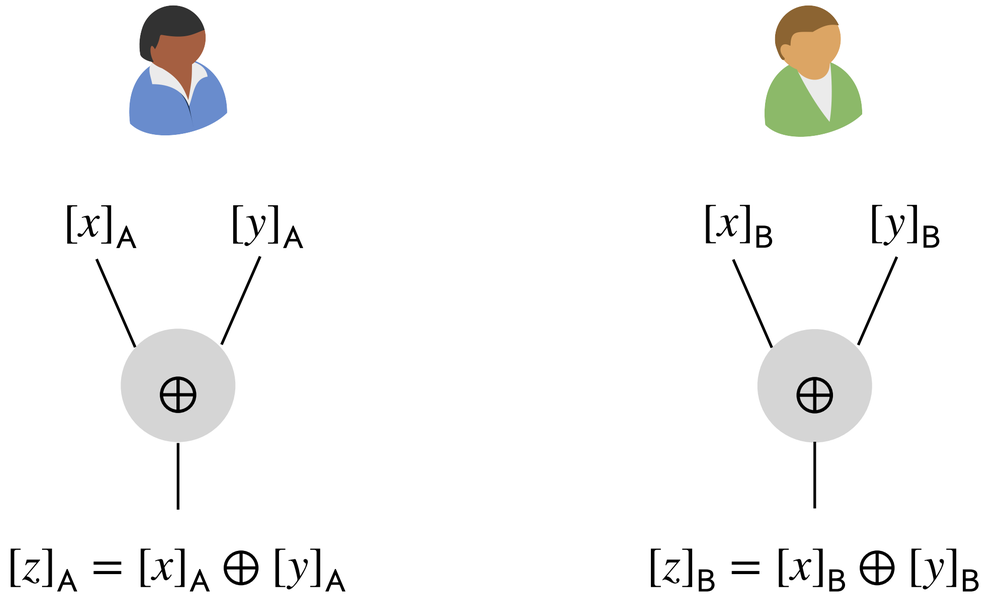
- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.
- Each party XORs its shares.
  - Additive secret sharing supports evaluating XOR gates for “free” – it does not require any communication.



Security:

Evaluating XOR Gates

- **Goal:** Given shares of two inputs, compute share of the XOR of the inputs.
- Each party XORs its shares.
  - Additive secret sharing supports evaluating XOR gates for "free" – it does not require any communication.



Security: Local computation $\Longrightarrow$ Compute $[z]_A = [x]_A \oplus [y]_A$.

Evaluating AND Gates

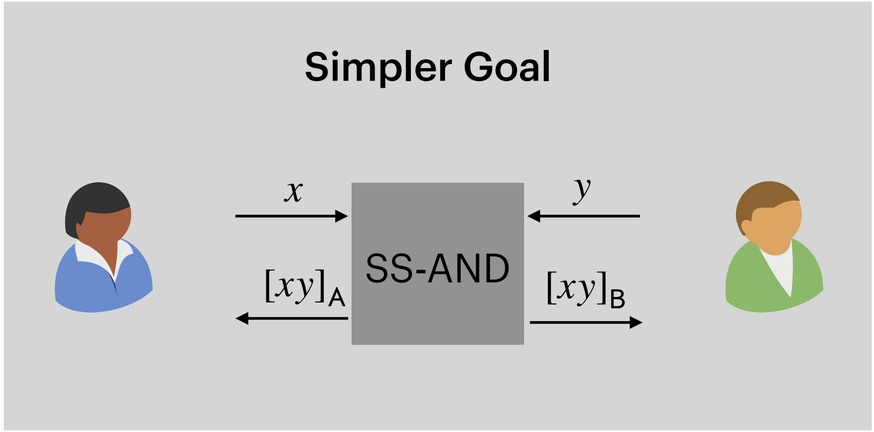
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.

Evaluating AND Gates

- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.

Evaluating AND Gates

- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.

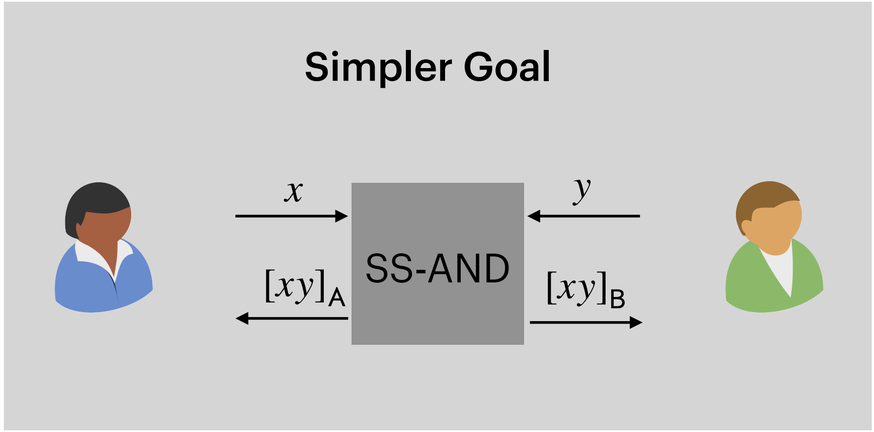


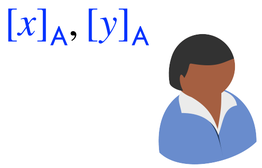
Evaluating AND Gates

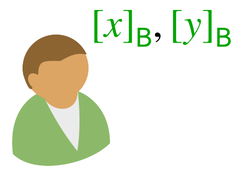
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$







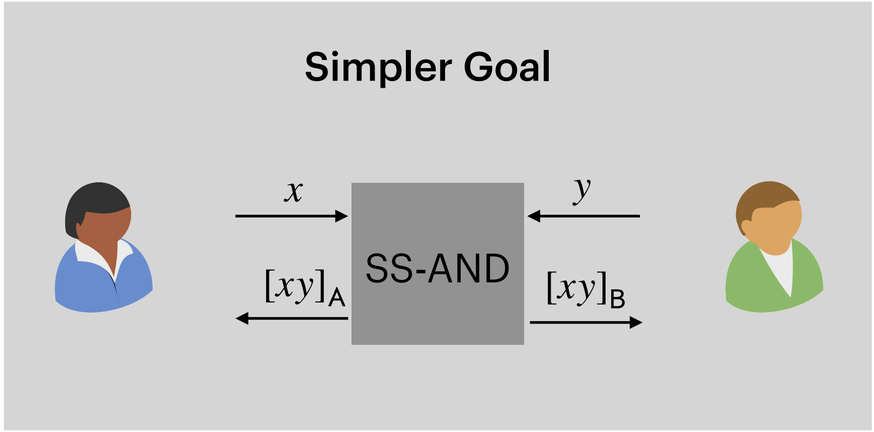
Evaluating AND Gates

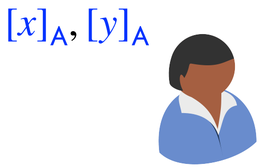
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

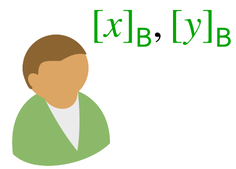
$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

  - Shares of $[x]_A[y]_B$ and $[x]_B[y]_A$ can be computed using the simpler primitive.







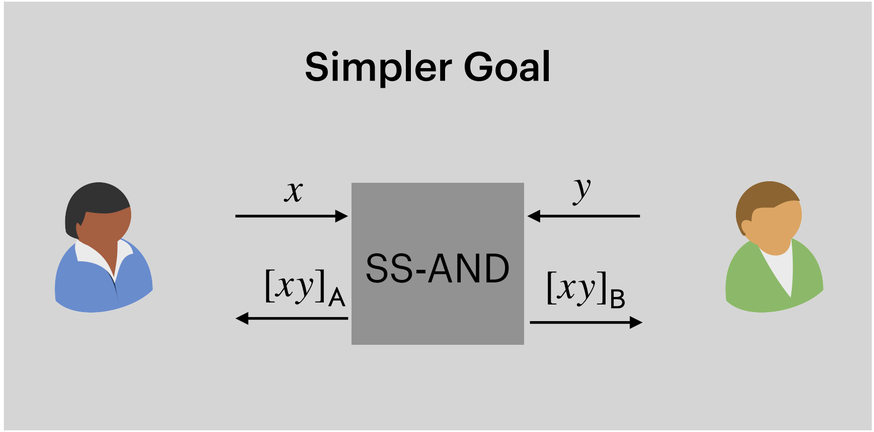
Evaluating AND Gates

- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

  - Shares of $[x]_A[y]_B$ and $[x]_B[y]_A$ can be computed using the simpler primitive.



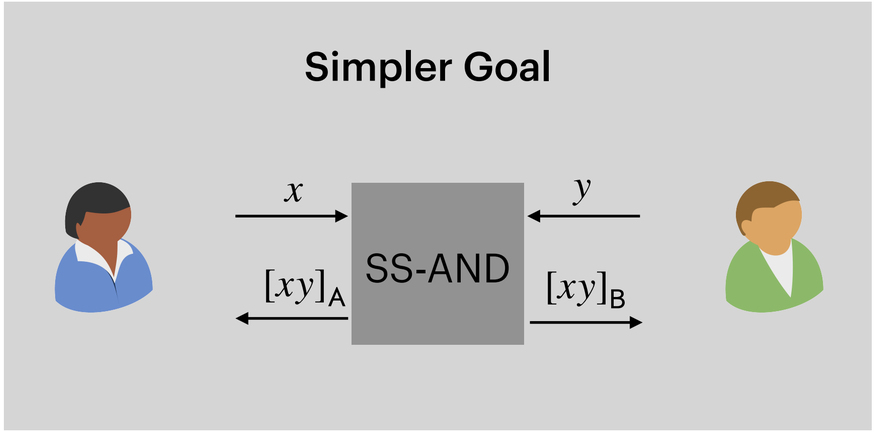
Evaluating AND Gates

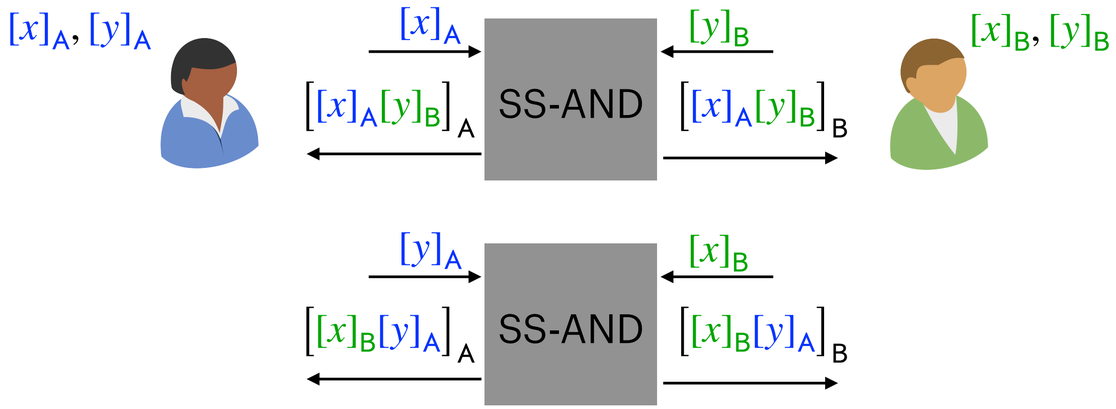
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

  - Shares of $[x]_A[y]_B$ and $[x]_B[y]_A$ can be computed using the simpler primitive.





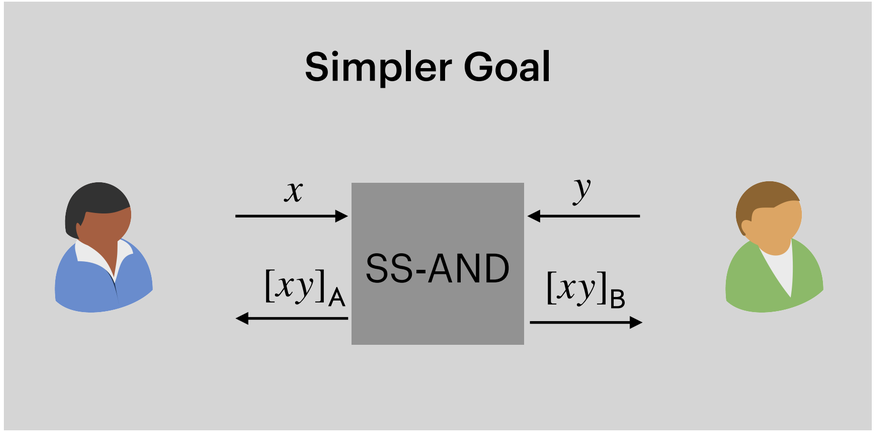
Evaluating AND Gates

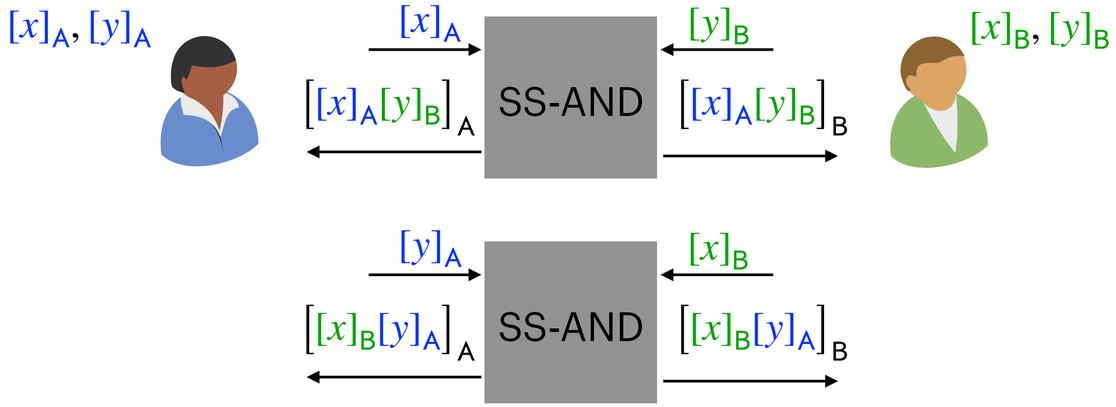
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

  - Shares of $[x]_A[y]_B$ and $[x]_B[y]_A$ can be computed using the simpler primitive.
  - $[x]_A[y]_A$ and $[x]_B[y]_B$ can be locally computed by Alice and Bob.





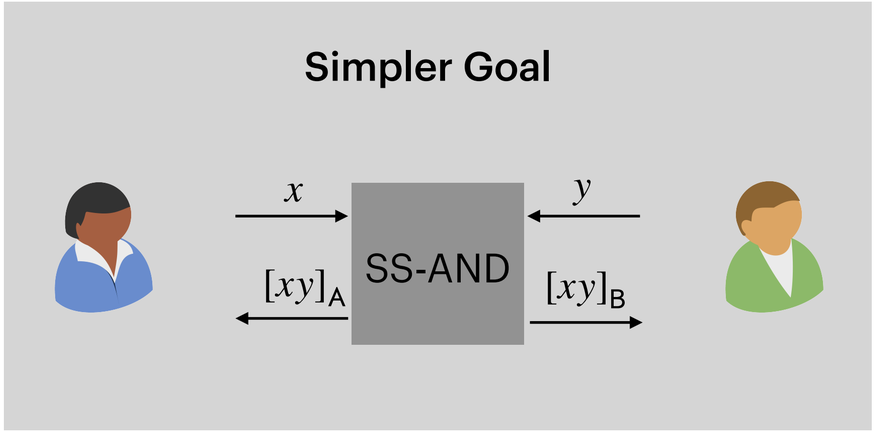
Evaluating AND Gates

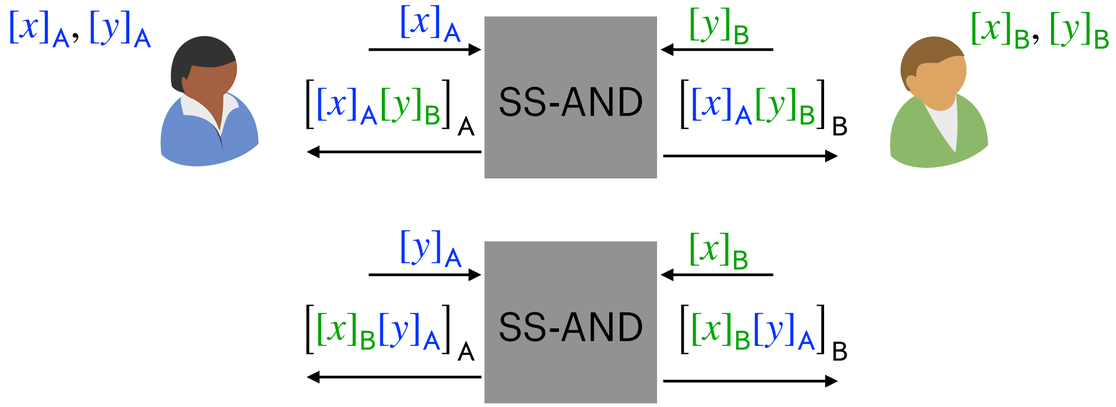
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

- Shares of $[x]_A[y]_B$ and $[x]_B[y]_A$ can be computed using the simpler primitive.
- $[x]_A[y]_A$ and $[x]_B[y]_B$ can be locally computed by Alice and Bob.
- XOR of all terms can be locally computed over shares.





$$[z]_A := \left[[x]_A[y]_B\right]_A \oplus \left[[x]_B[y]_A\right]_A \oplus [x]_A[y]_A$$

$$[z]_B := \left[[x]_A[y]_B\right]_B \oplus \left[[x]_B[y]_A\right]_B \oplus [x]_B[y]_B$$

Evaluating AND Gates

- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

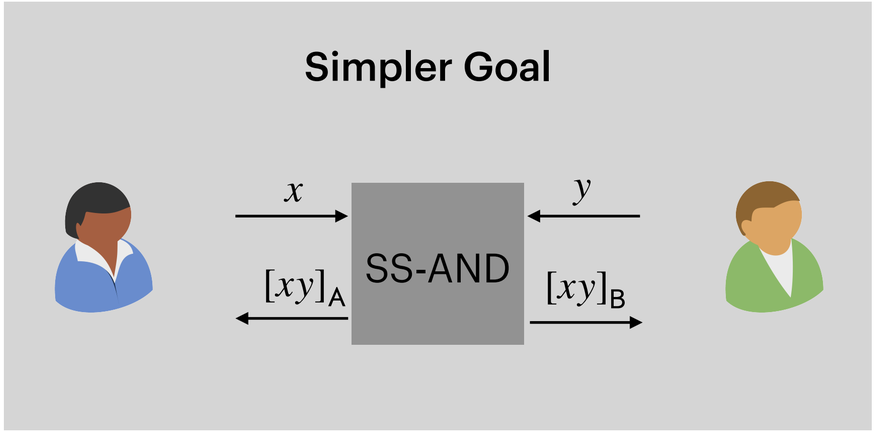
$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

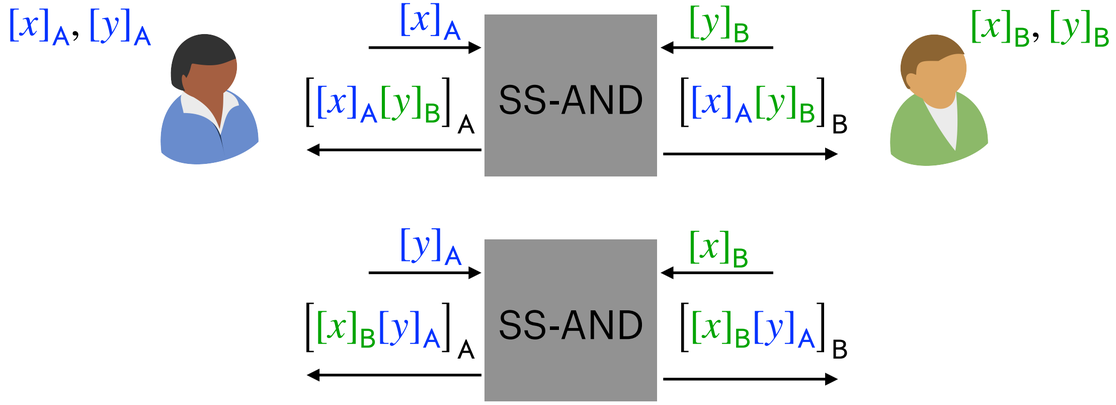
Correctness

$$z = [z]_A \oplus [z]_B = \left[[x]_A[y]_B\right]_A \oplus \left[[x]_B[y]_A\right]_A \oplus [x]_A[y]_A \oplus \left[[x]_A[y]_B\right]_B \oplus \left[[x]_B[y]_A\right]_B \oplus [x]_B[y]_B$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

$$= xy$$





$$[z]_A := \left[[x]_A[y]_B\right]_A \oplus \left[[x]_B[y]_A\right]_A \oplus [x]_A[y]_A$$

$$[z]_B := \left[[x]_A[y]_B\right]_B \oplus \left[[x]_B[y]_A\right]_B \oplus [x]_B[y]_B$$

Evaluating AND Gates

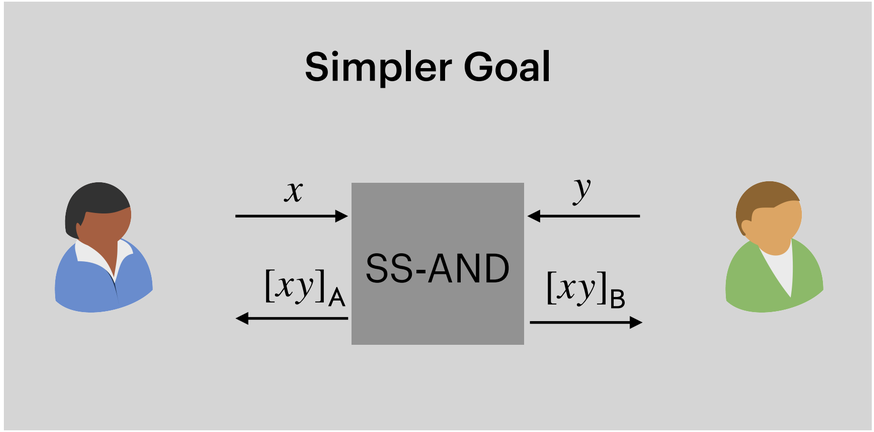
- **Goal:** Given shares of two inputs, compute share of the AND of the inputs.
- **Simpler Goal:** Compute share of AND of each party's input.
- Suffices to achieve main goal.

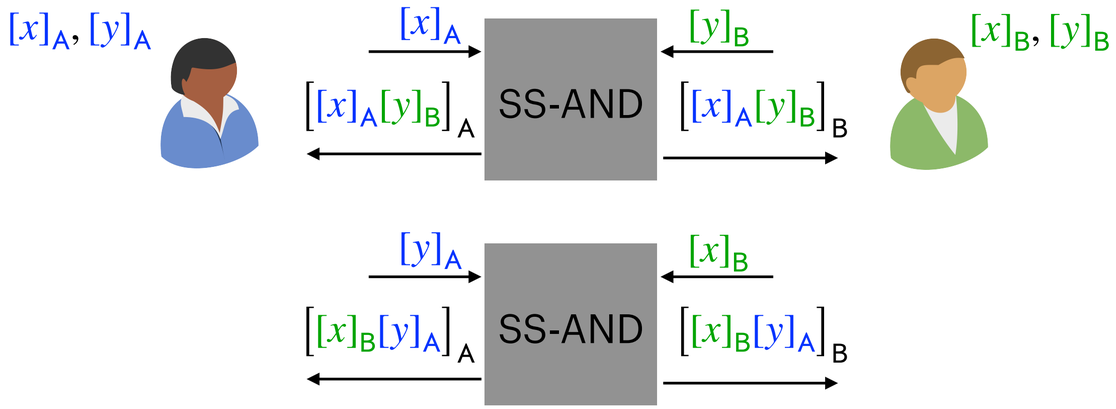
$$xy = \left([x]_A \oplus [x]_B\right)\left([y]_A \oplus [y]_B\right)$$

$$= [x]_A[y]_A \oplus [x]_A[y]_B \oplus [x]_B[y]_A \oplus [x]_B[y]_B$$

Security

- Suffices to compute output of SS-AND using only corrupt parties input.
- $[z]_A$ is then computed using local computation.





$$[z]_A := \left[[x]_A[y]_B\right]_A \oplus \left[[x]_B[y]_A\right]_A \oplus [x]_A[y]_A$$

$$[z]_B := \left[[x]_A[y]_B\right]_B \oplus \left[[x]_B[y]_A\right]_B \oplus [x]_B[y]_B$$

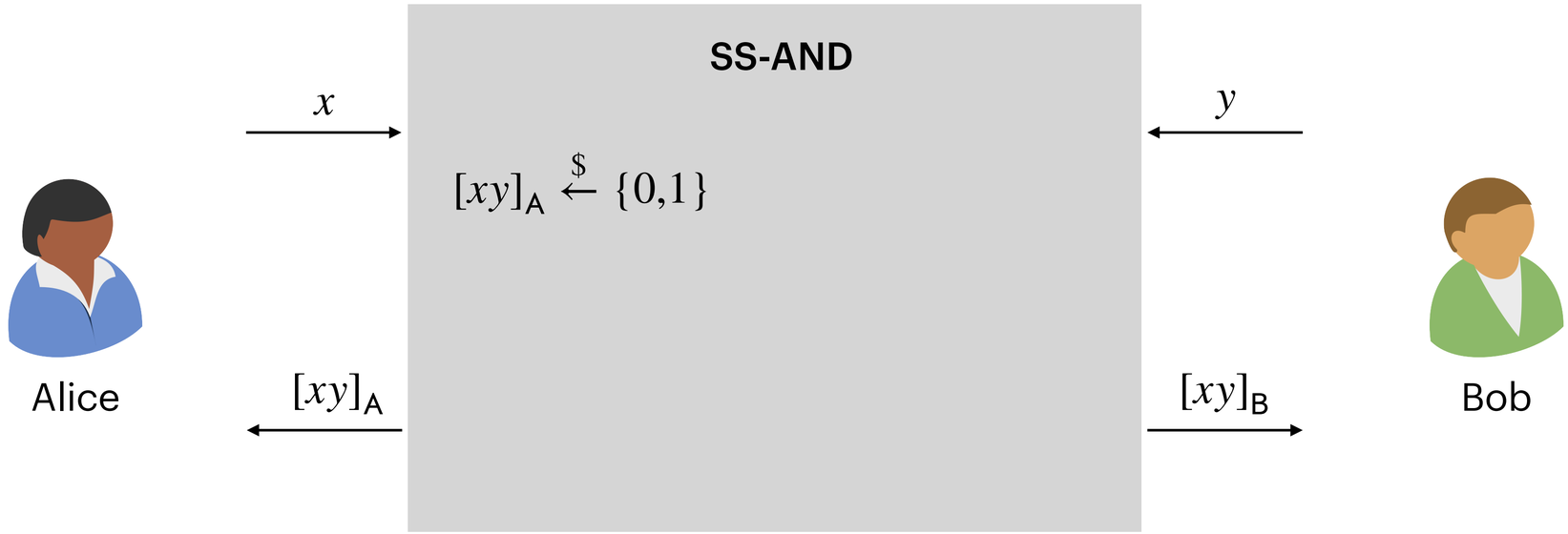
Evaluating AND Gates

- **Goal:** Compute share of AND of each party's input.

Evaluating AND Gates

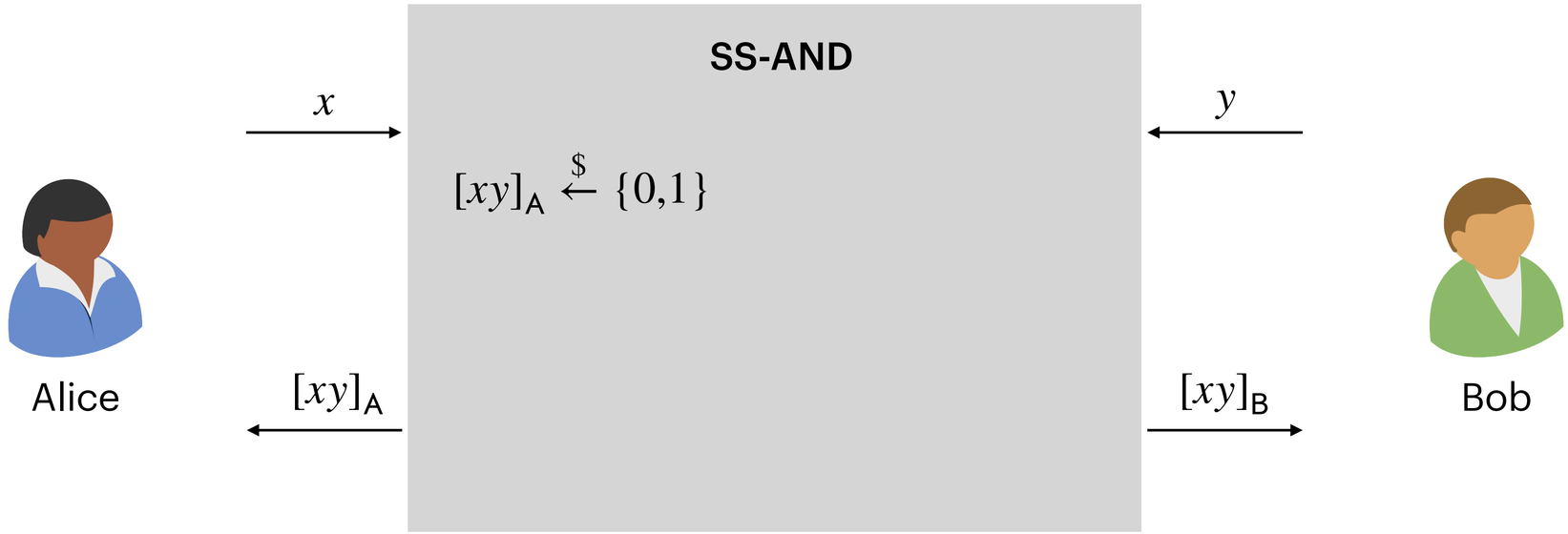
- **Goal:** Compute share of AND of each party's input.
- Construction

Evaluating AND Gates



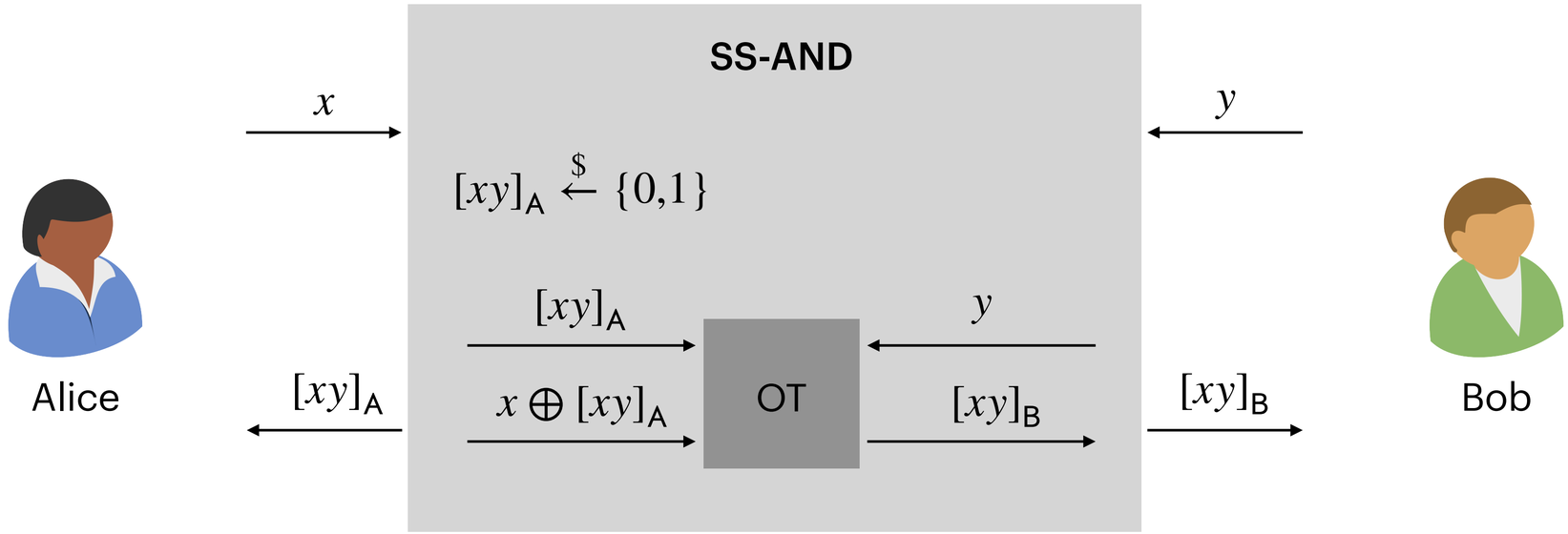
- **Goal:** Compute share of AND of each party's input.
- Construction
  - Alice samples her share $[xy]_A$ uniformly at random.

Evaluating AND Gates



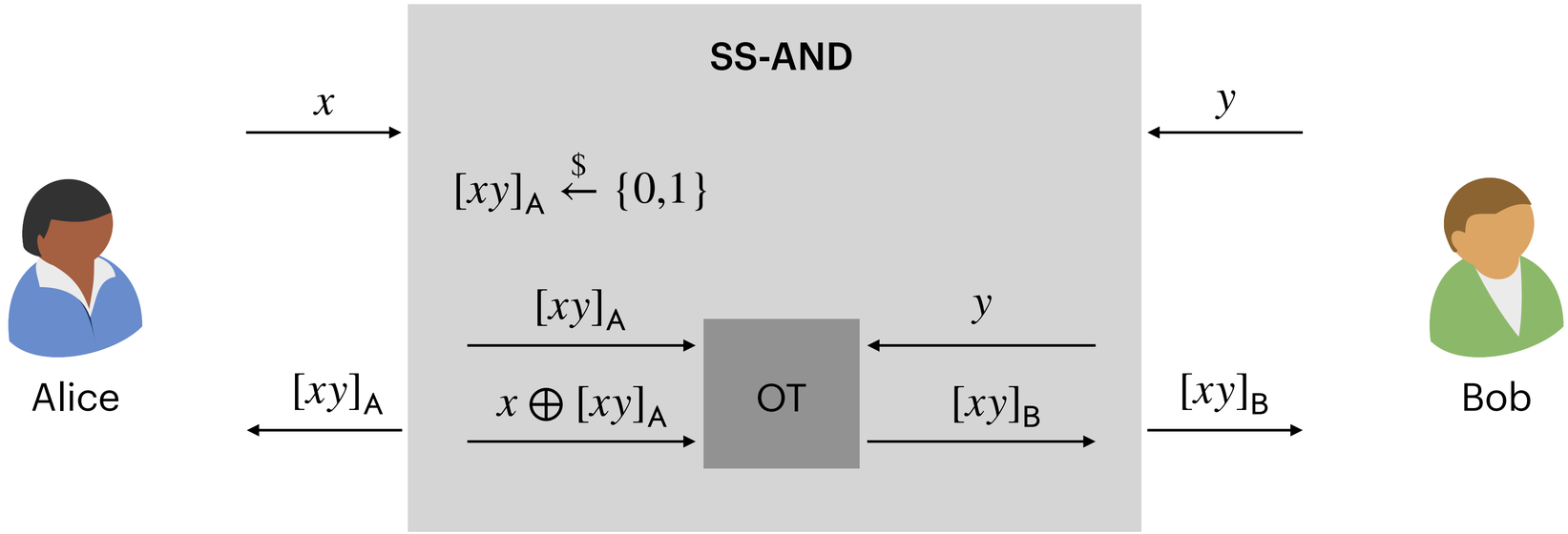
- **Goal:** Compute share of AND of each party's input.
- Construction
  - Alice samples her share $[xy]_A$ uniformly at random.
  - Compute the function for Bob's share $f\left(x, y, [xy]_A\right) = xy \oplus [xy]_A$ using OT-based truth table evaluation.

Evaluating AND Gates



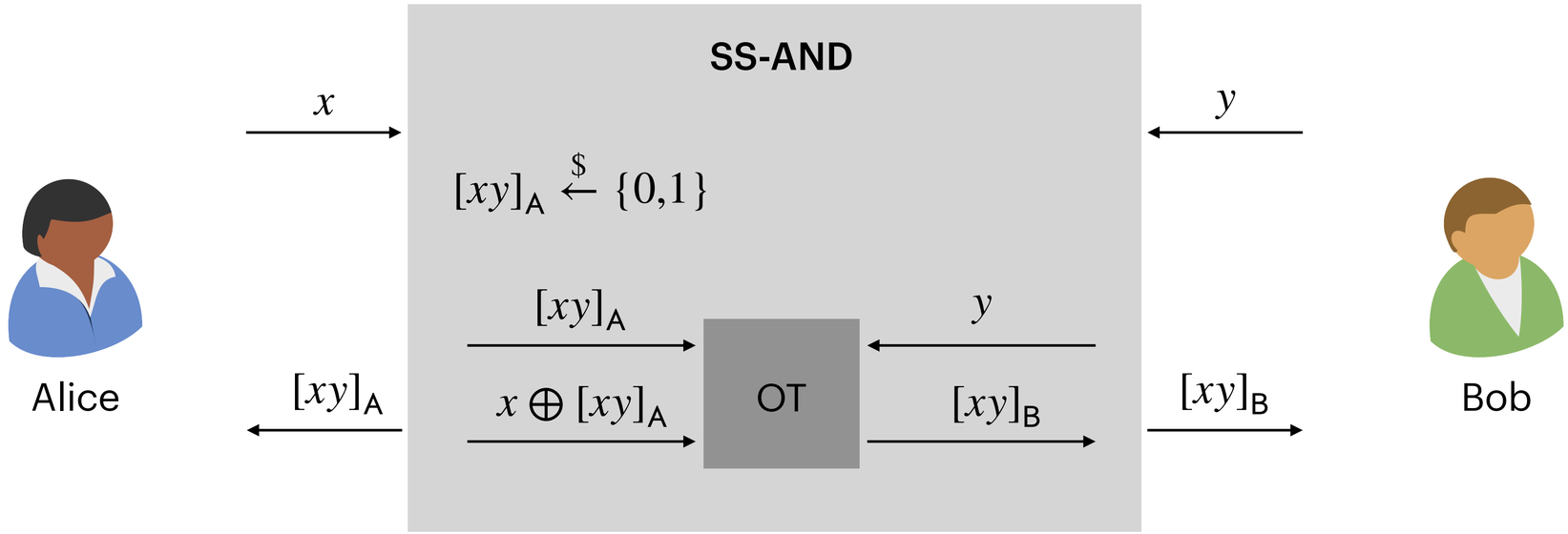
- **Goal:** Compute share of AND of each party's input.
- Construction
  - Alice samples her share $[xy]_A$ uniformly at random.
  - Compute the function for Bob's share $f\left(x, y, [xy]_A\right) = xy \oplus [xy]_A$ using OT-based truth table evaluation.

Evaluating AND Gates



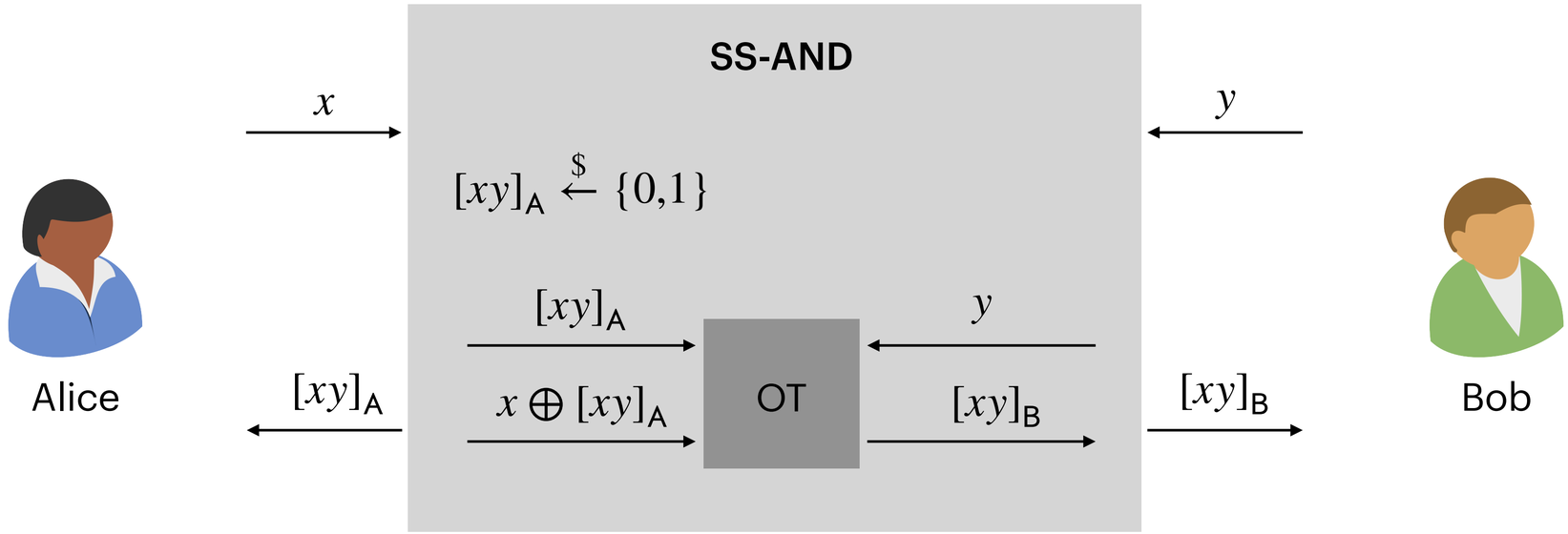
- **Security:** Simulate output of SS-AND using only corrupt parties input.

Evaluating AND Gates



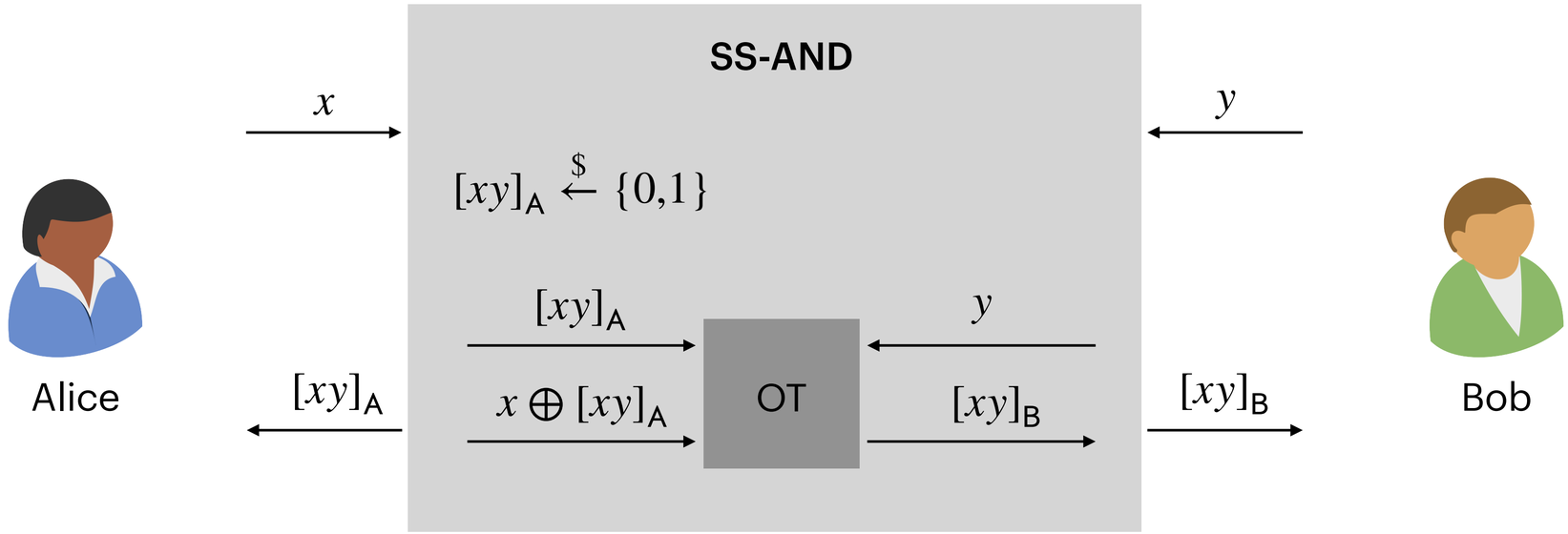
- **Security:** Simulate output of SS-AND using only corrupt parties input.
  - Output is a fresh sharing of xy since $[xy]_A$ is sampled uniformly at random.

Evaluating AND Gates



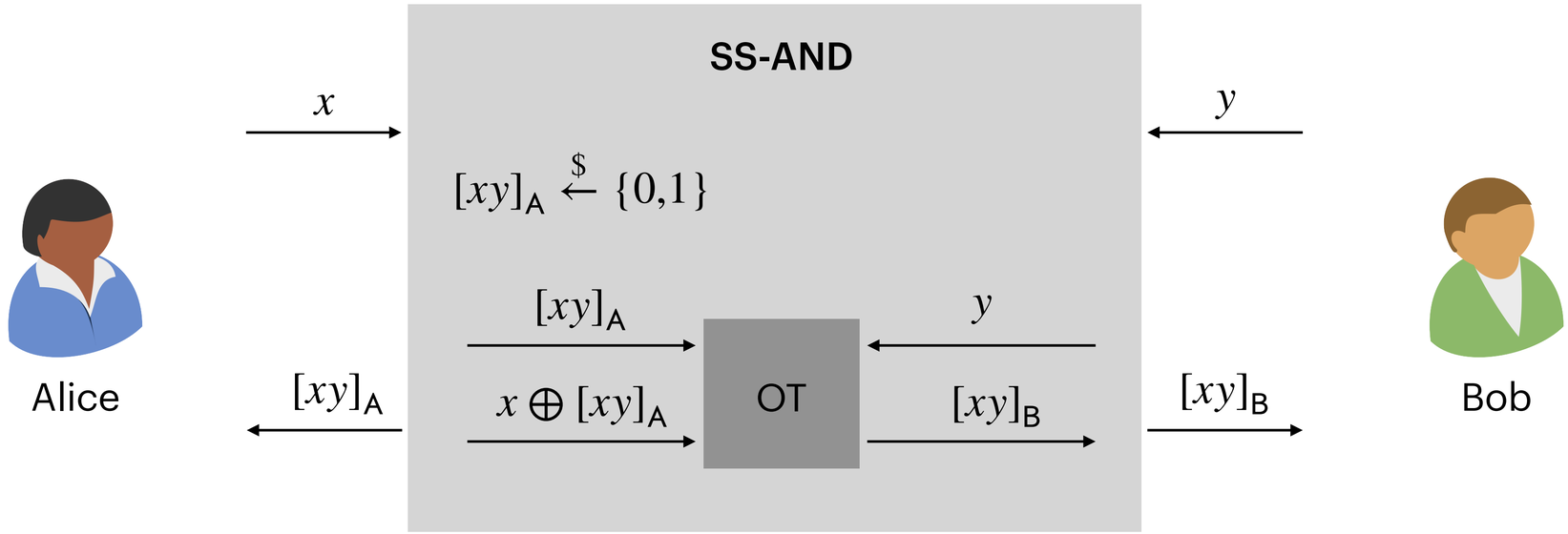
- **Security:** Simulate output of SS-AND using only corrupt parties input.
  - Output is a fresh sharing of xy since $[xy]_A$ is sampled uniformly at random.
  - The simulator can sample $[xy]_A$ as a uniformly random bit. Simulator for Bob samples $[xy]_B$ uniformly at random too.

Evaluating AND Gates



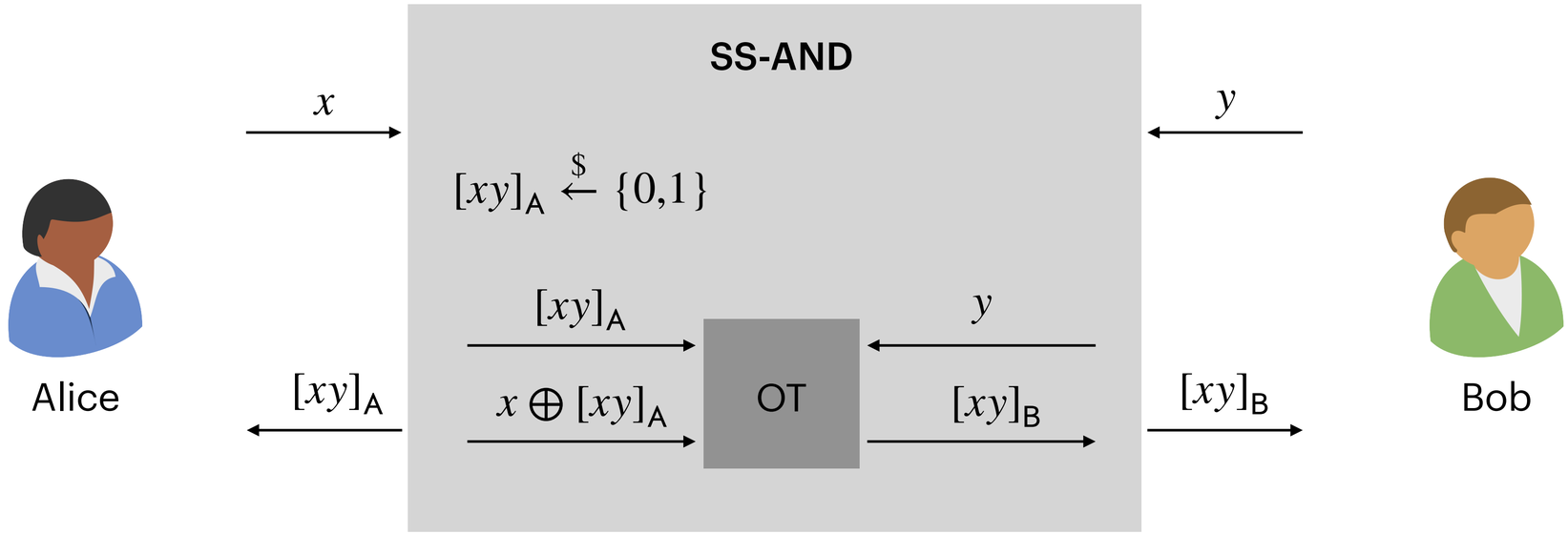
- **Security:** Simulate output of SS-AND using only corrupt parties input.
  - Output is a fresh sharing of xy since $[xy]_A$ is sampled uniformly at random.
  - The simulator can sample $[xy]_A$ as a uniformly random bit. Simulator for Bob samples $[xy]_B$ uniformly at random too.
- Compare to simulator for OT-based truth table computation of $f(x, y) = xy$?

Evaluating AND Gates



- **Security:** Simulate output of SS-AND using only corrupt parties input.
  - Output is a fresh sharing of xy since $[xy]_A$ is sampled uniformly at random.
  - The simulator can sample $[xy]_A$ as a uniformly random bit. Simulator for Bob samples $[xy]_B$ uniformly at random too.
- Compare to simulator for OT-based truth table computation of $f(x, y) = xy$?
  - OT simulator needs output. To compute $f(x, y) = xy$, simulator needs the output xy.

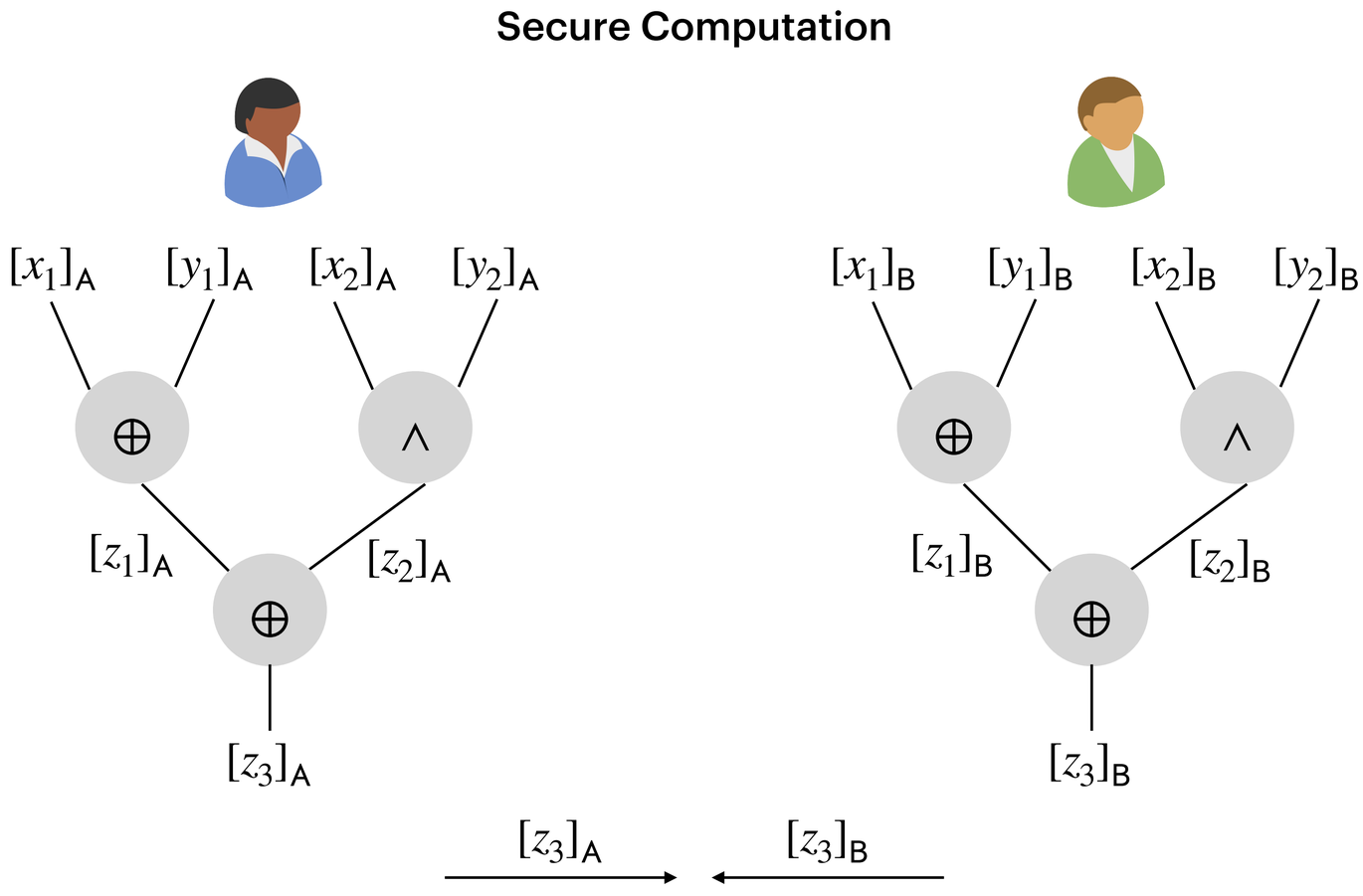
Evaluating AND Gates



- **Security:** Simulate output of SS-AND using only corrupt parties input.
  - Output is a fresh sharing of xy since $[xy]_A$ is sampled uniformly at random.
  - The simulator can sample $[xy]_A$ as a uniformly random bit. Simulator for Bob samples $[xy]_B$ uniformly at random too.
- Compare to simulator for OT-based truth table computation of $f(x, y) = xy$?
  - OT simulator needs output. To compute $f(x, y) = xy$, simulator needs the output xy.
  - But since Alice XORs with a random bit $[xy]_A$, we can run the OT simulator on a random output bit.

Template for Secure Computation

- **Invariant:** For each wire value z, Alice has $[z]_A$ and Bob has $[z]_B$ such that $[z]_A \oplus [z]_B = z$.
- **Correctness:** How to maintain the invariant at each gate?
- **Security:** Requires constructing a simulator for the protocol.
  - Simulator for Alice $S(x_1, x_2, f(x_1, x_2, y_1, y_2))$
    - Use secret-sharing simulator for input shares.
    - **Simulator for each gate:** Given shares on input wires, compute share on output wire.
    - Compute output share $[z_3]_A$ and simulate $[z_3]_B := f(x_1, x_2, y_1, y_2) \oplus [z_3]_A$.



Extension to Multiple Parties

Extension to Multiple Parties

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

Extension to Multiple Parties

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

- Secret share each input to all n parties.
  - Shares of x are $x = \bigoplus_{i=1}^{n} [x]_i$.

Extension to Multiple Parties

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

- Secret share each input to all n parties.
  - Shares of x are $x = \oplus_{i=1}^{n} [x]_i$.
- XOR gates can be evaluated locally.
  - $[z]_i := [x]_i \oplus [y]_i$.

Extension to Multiple Parties

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

- Secret share each input to all n parties.
  - Shares of x are $x = \oplus_{i=1}^{n} [x]_i$.
- XOR gates can be evaluated locally.
  - $[z]_i := [x]_i \oplus [y]_i$.
- AND gates can be evaluated by using SS-AND between every pair of parties.
  - $xy = \left(\oplus_{i=1}^{n} [x]_i \right) \left(\oplus_{j=1}^{n} [y]_j \right) = \oplus_{i=1}^{n} \oplus_{j=1}^{n} [x]_i \, [y]_j$.
  - Party-i and Party-j use SS-AND to compute shares of $[x]_i \, [y]_j$.
  - XOR all resulting shares to get shares of the output.