CS 594 – Advanced Crypto (Spring 2026)

Alex Block

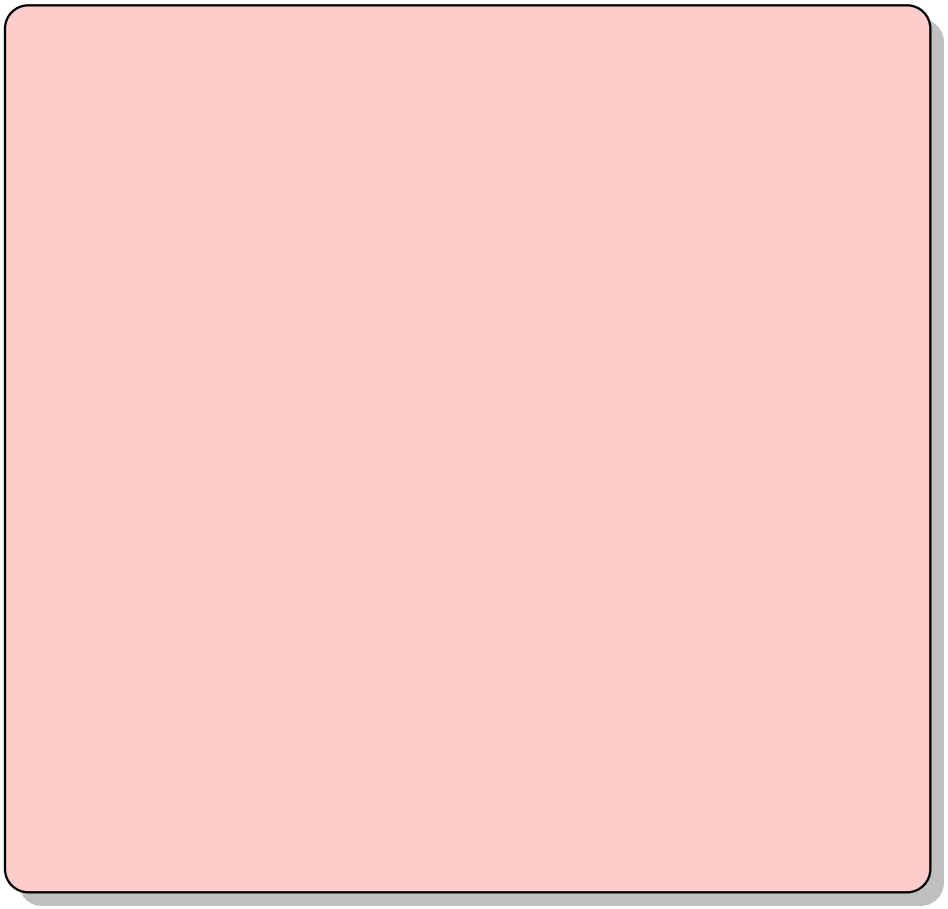
Lecture 16

March 11, 2026

Security of MPC

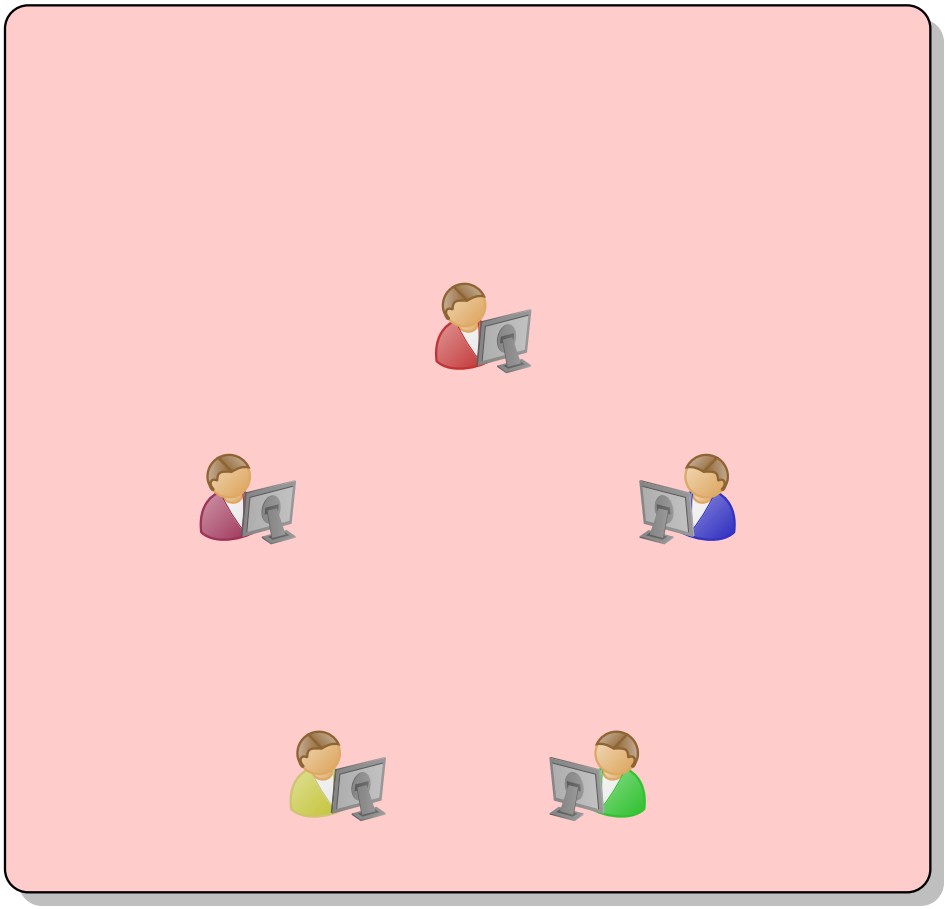
Real-Ideal Security Paradigm

Real-Ideal Security Paradigm



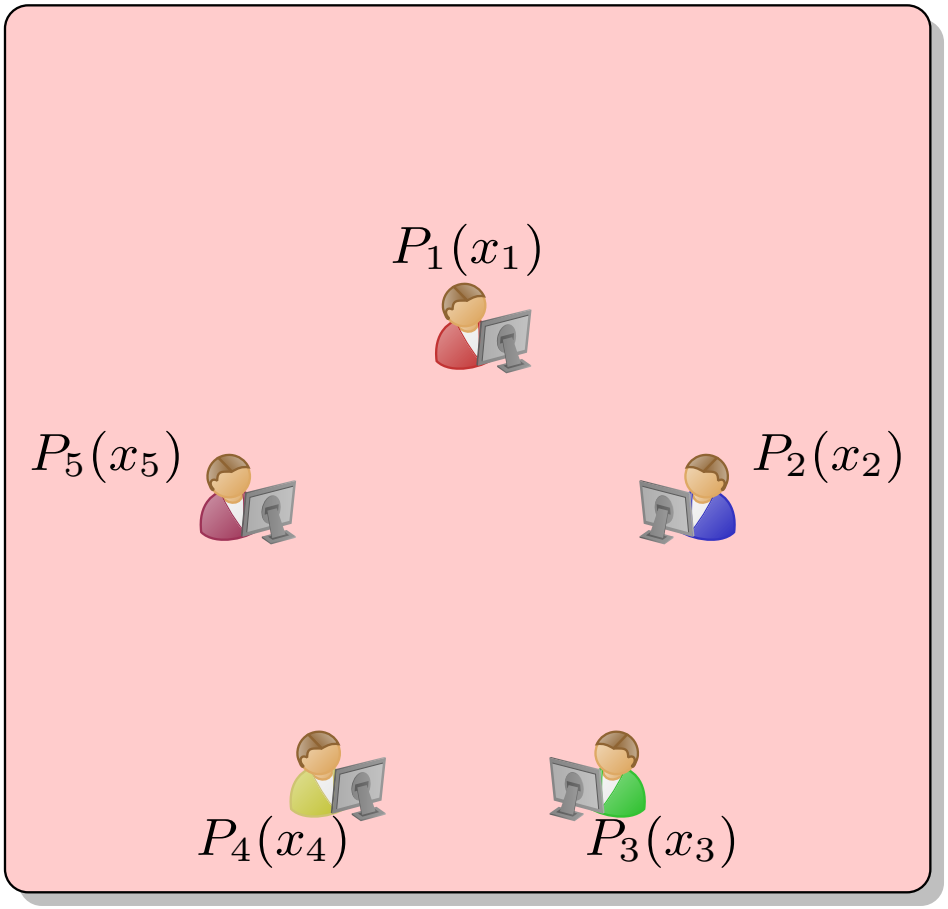
Real World

Real-Ideal Security Paradigm



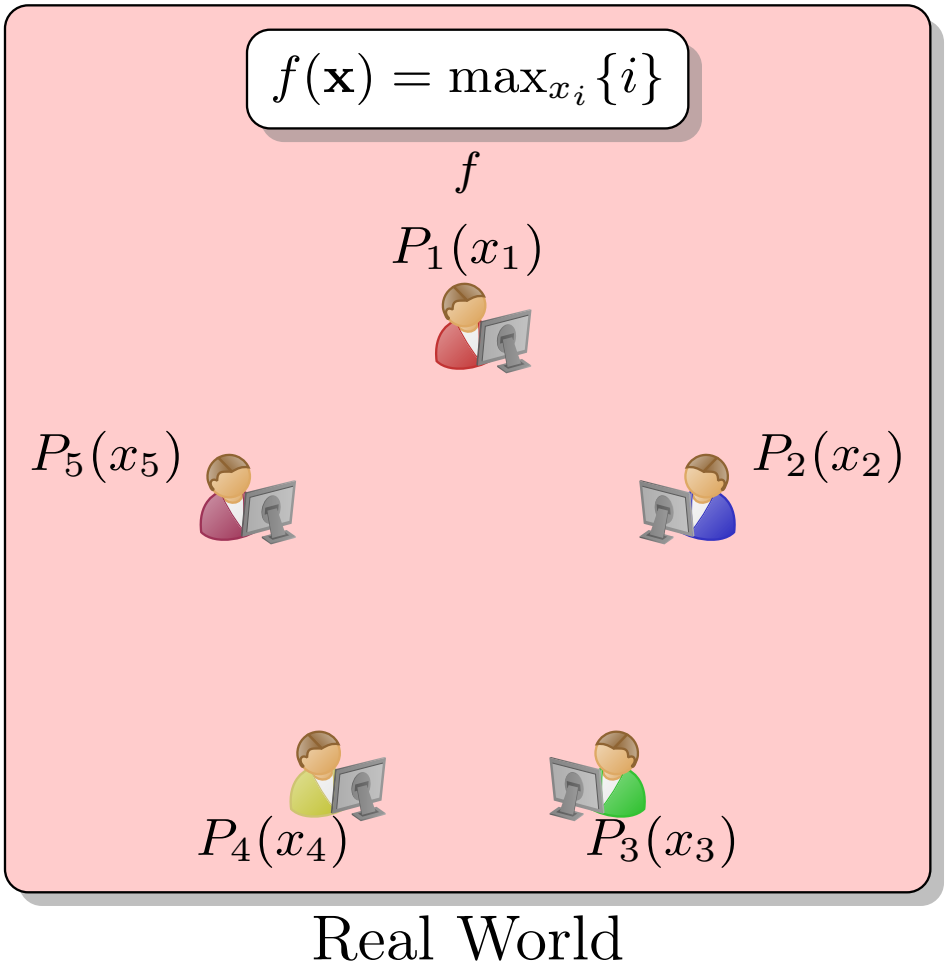
Real World

Real-Ideal Security Paradigm



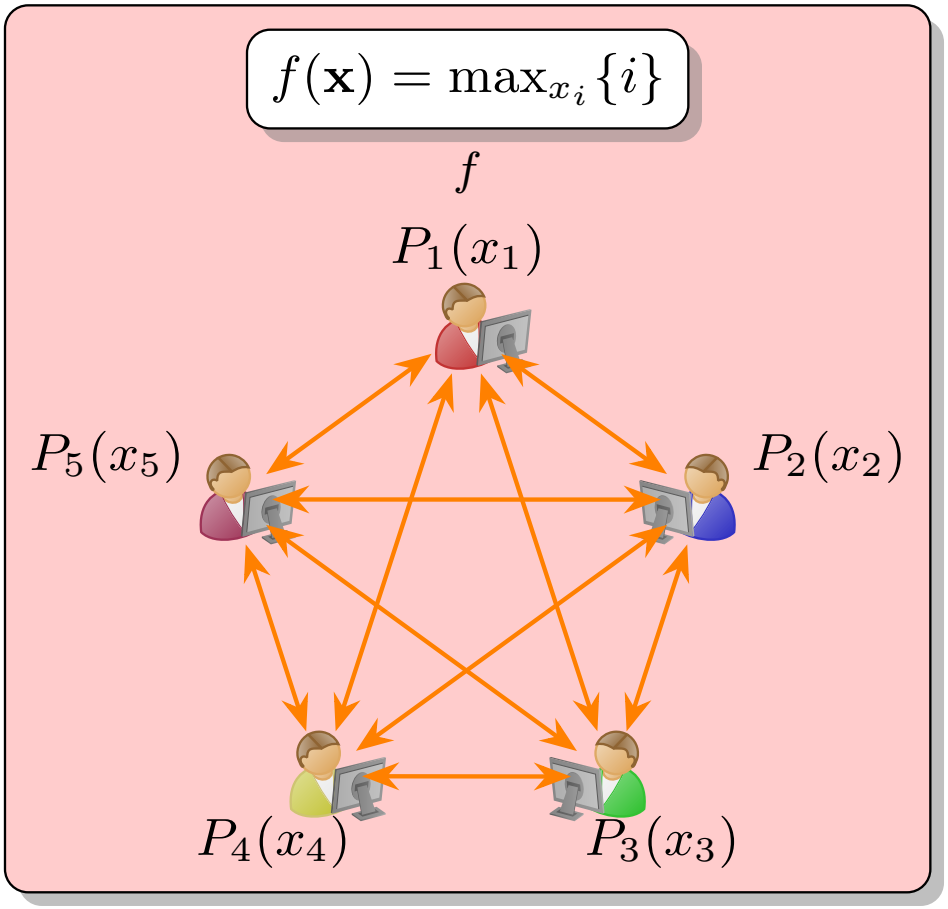
Real World

Real-Ideal Security Paradigm



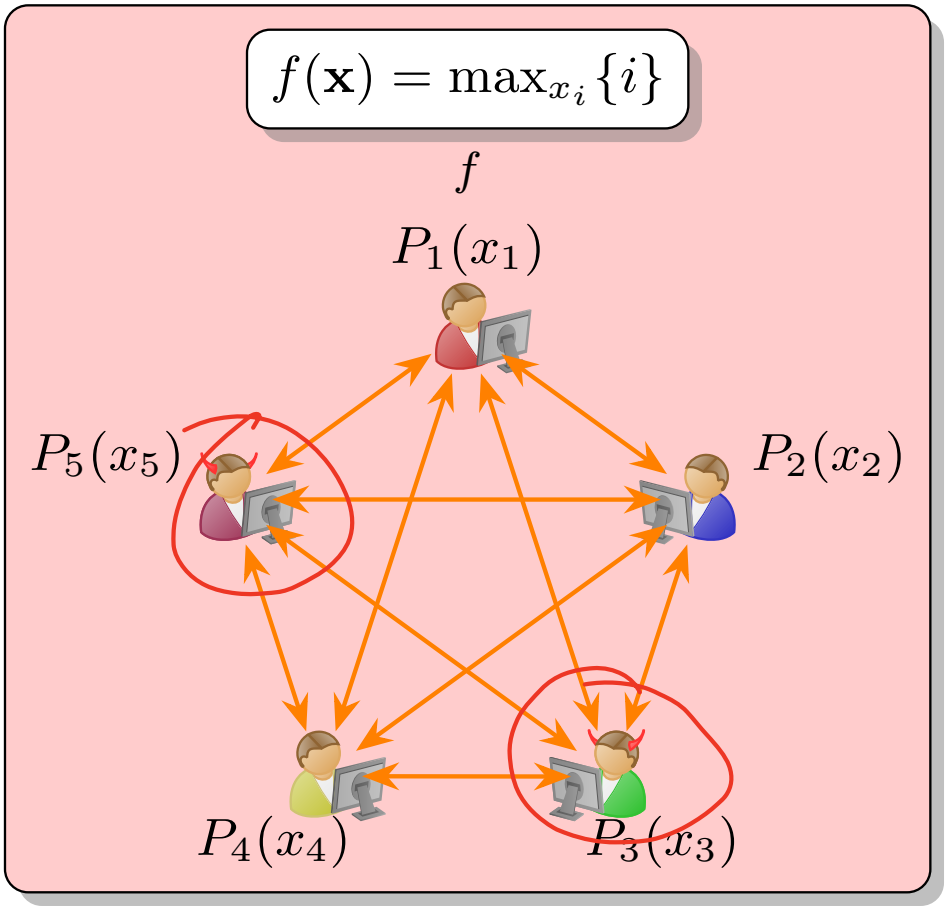
Real World

Real-Ideal Security Paradigm



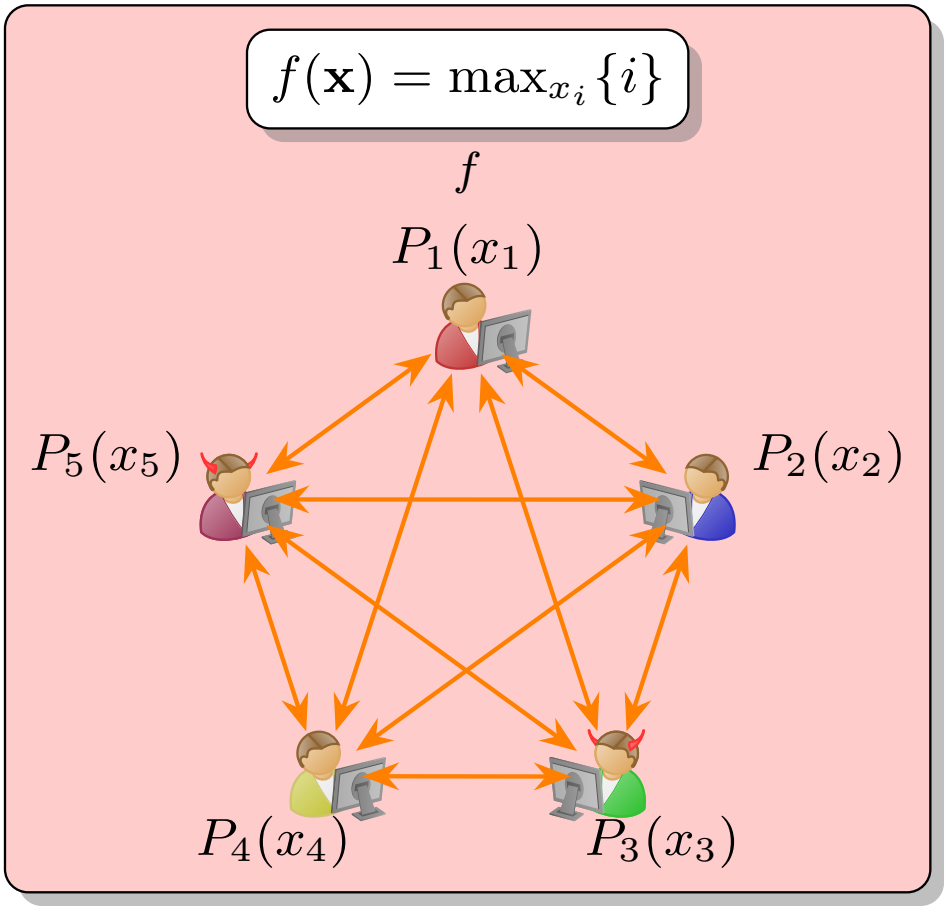
Real World

Real-Ideal Security Paradigm



Real World

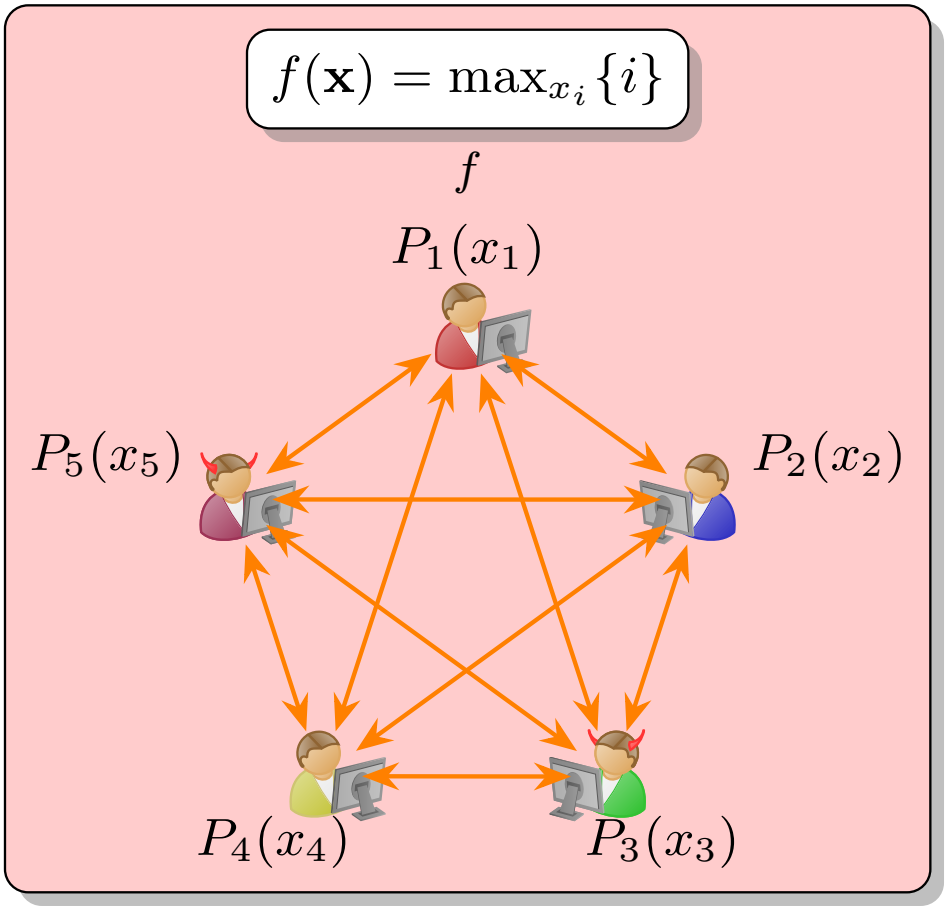
Real-Ideal Security Paradigm



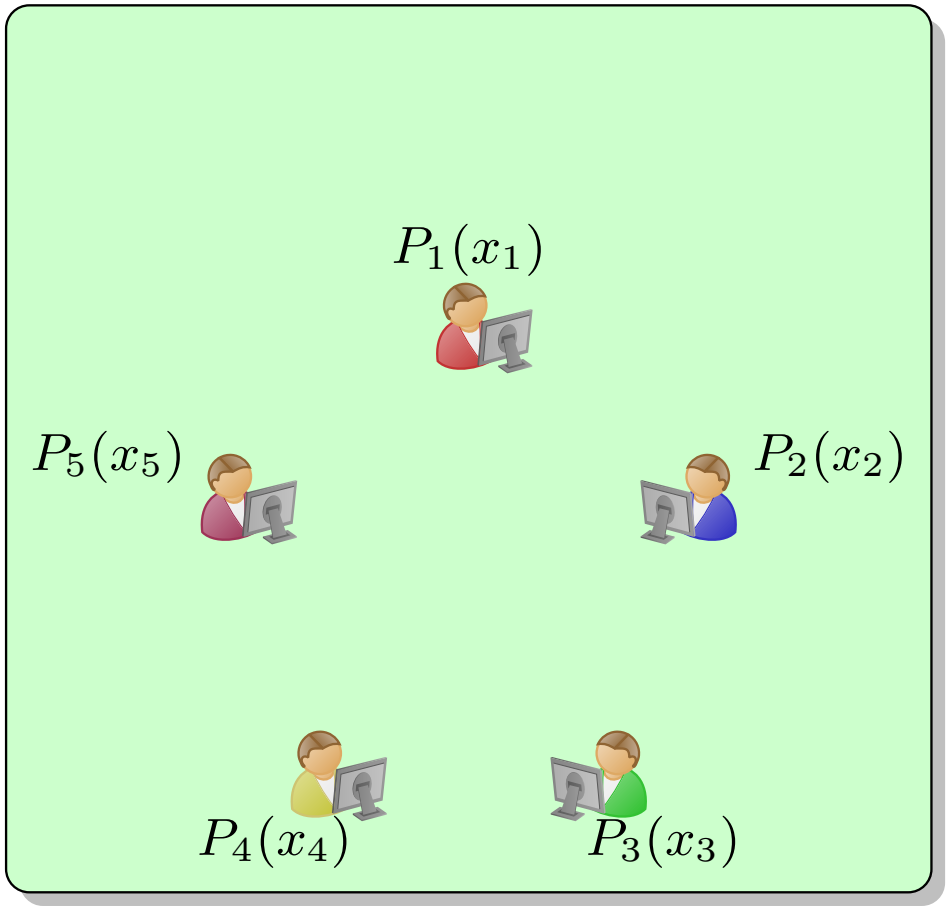
Real World

Ideal World

Real-Ideal Security Paradigm

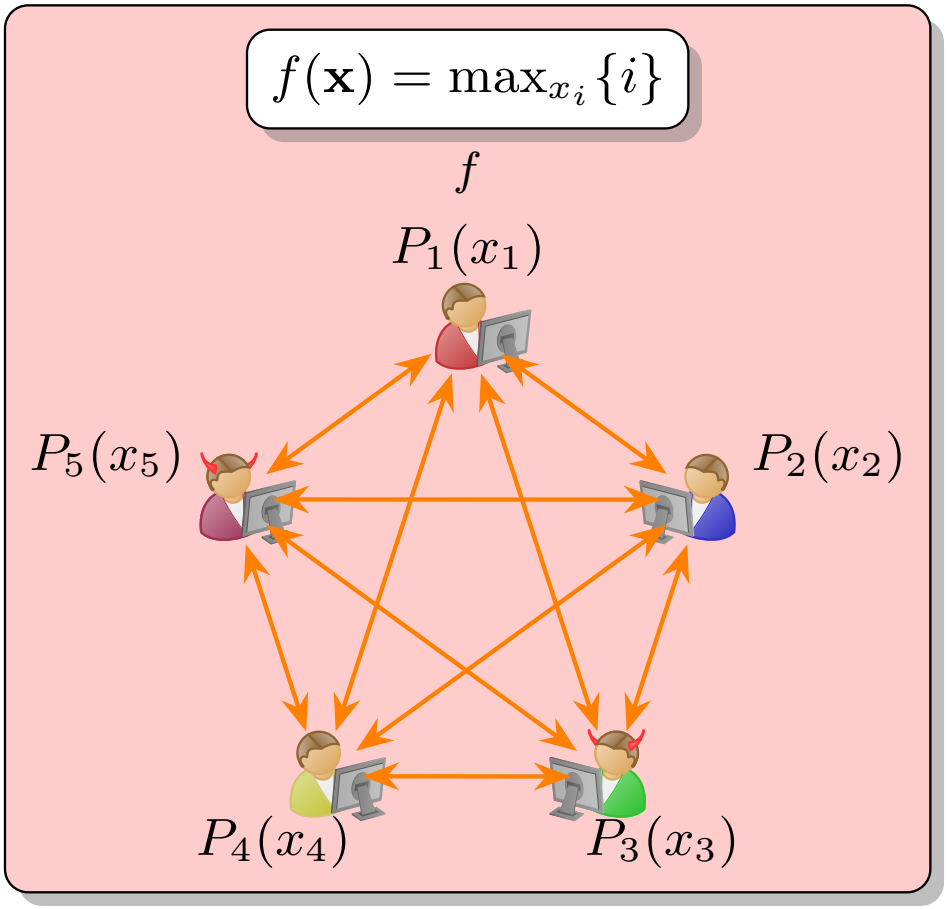


Real World

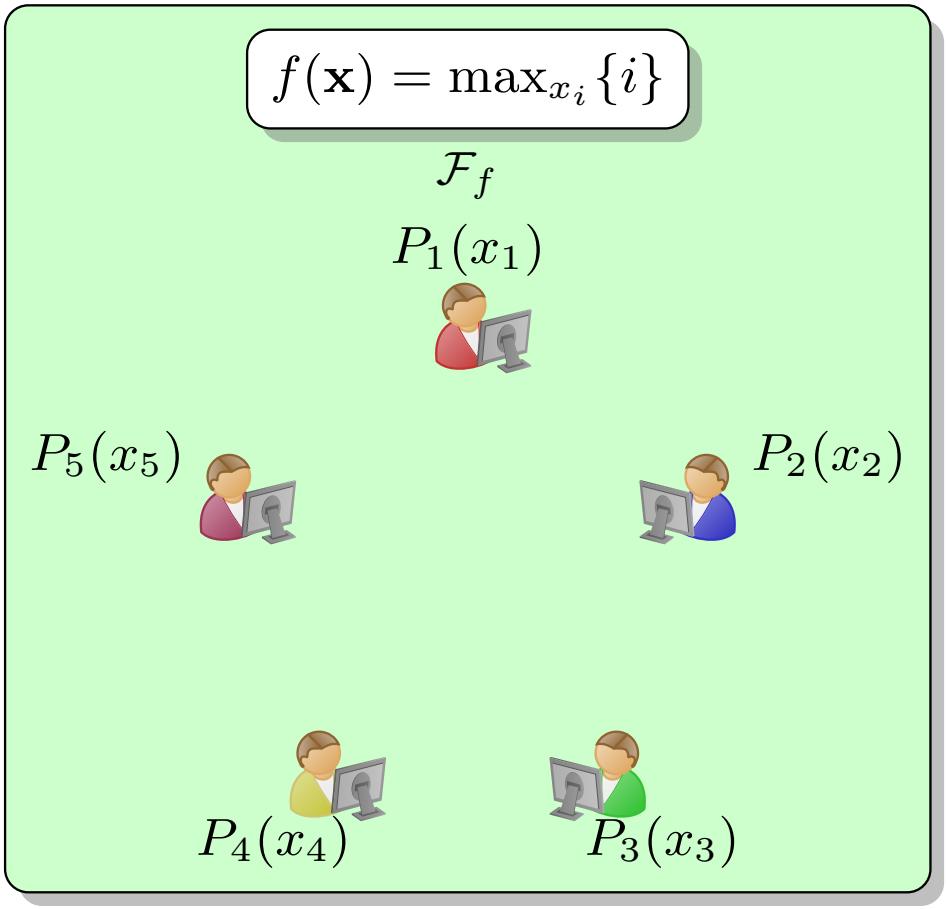


Ideal World

Real-Ideal Security Paradigm

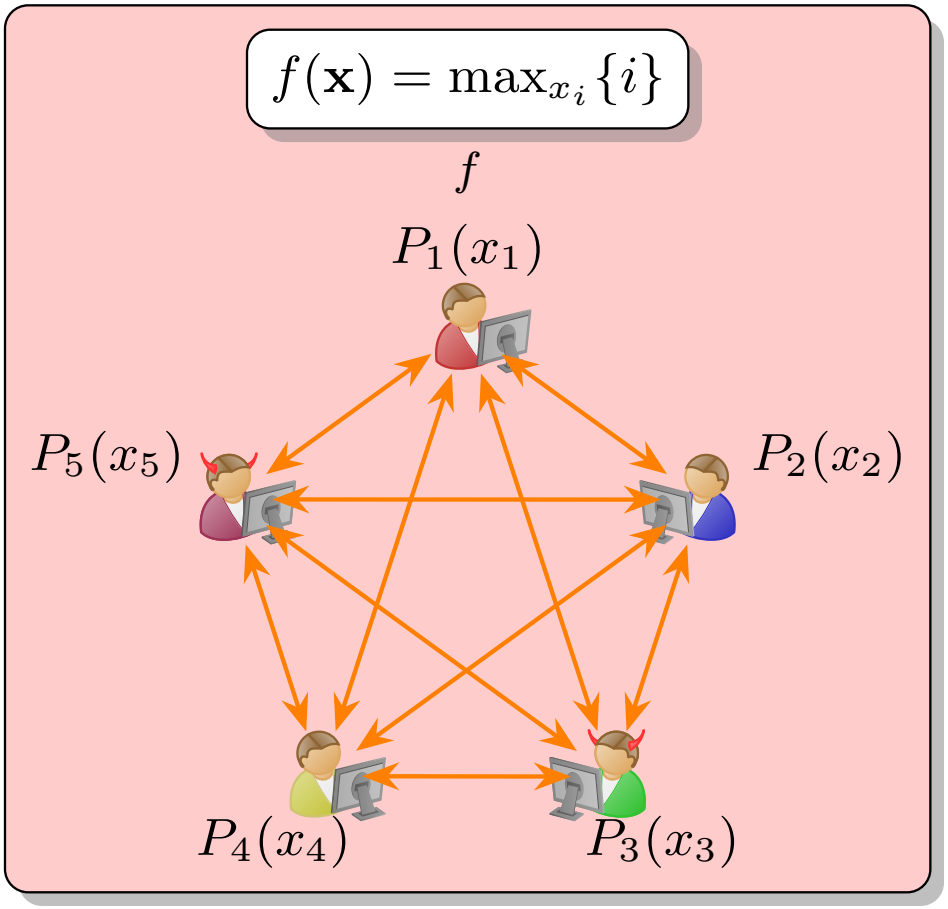


Real World

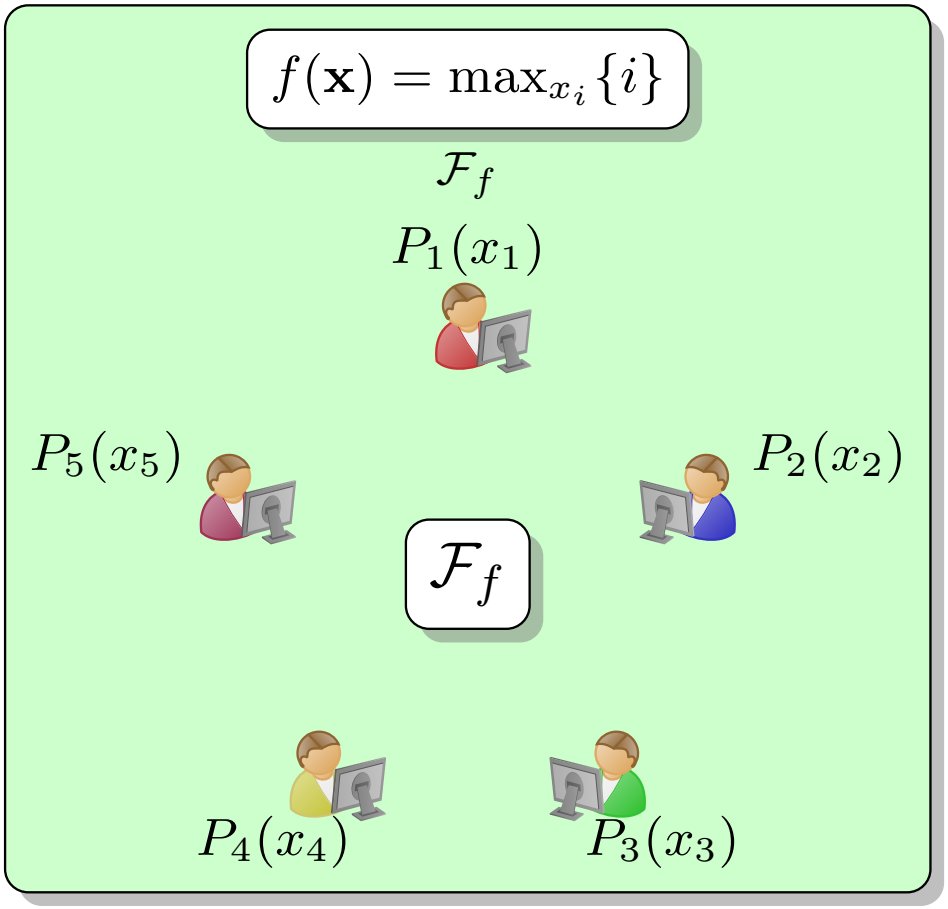


Ideal World

Real-Ideal Security Paradigm

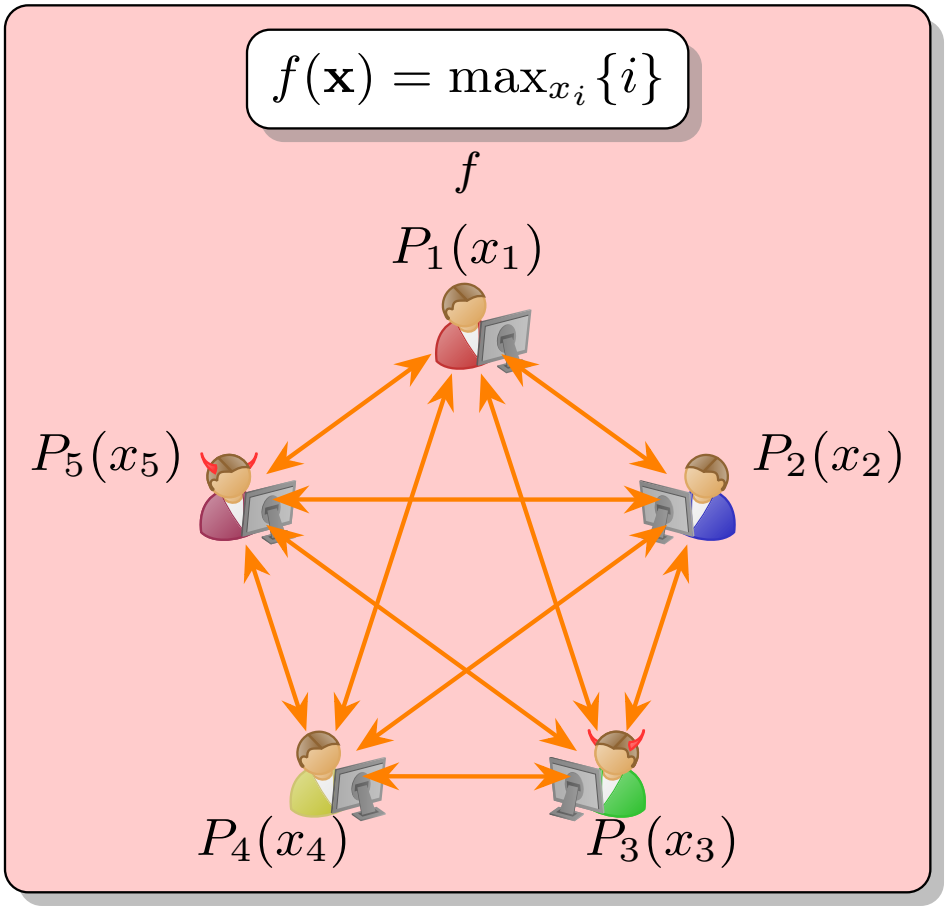


Real World

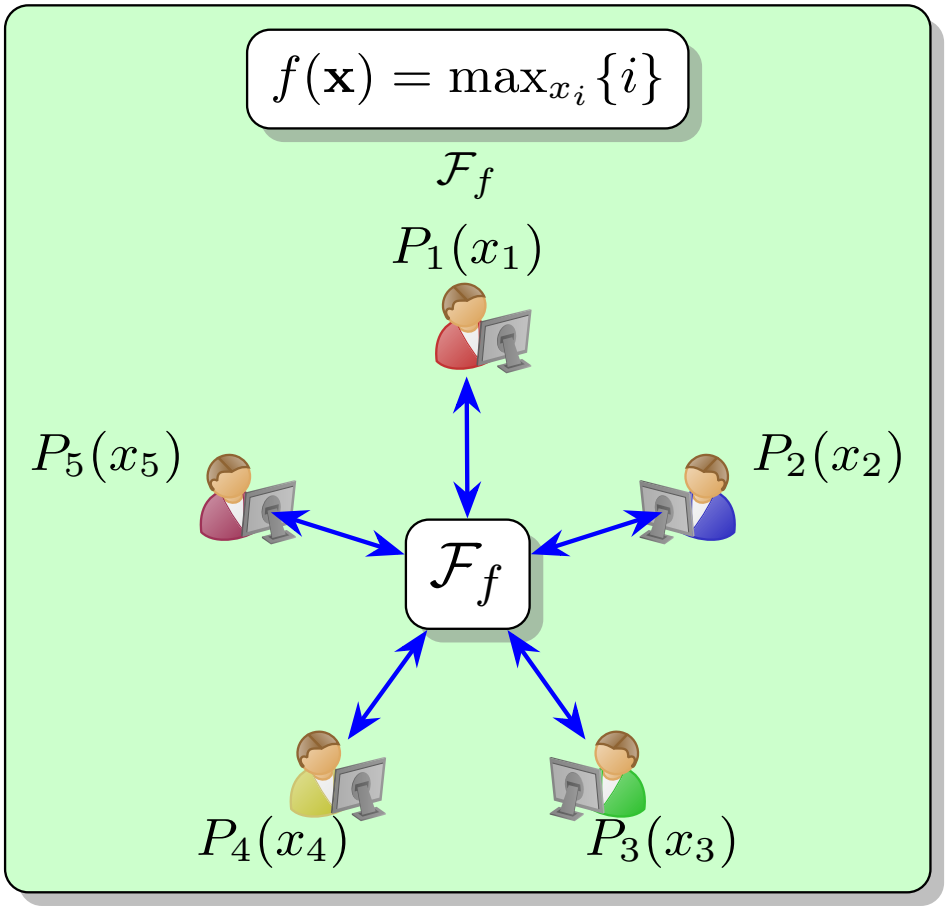


Ideal World

Real-Ideal Security Paradigm

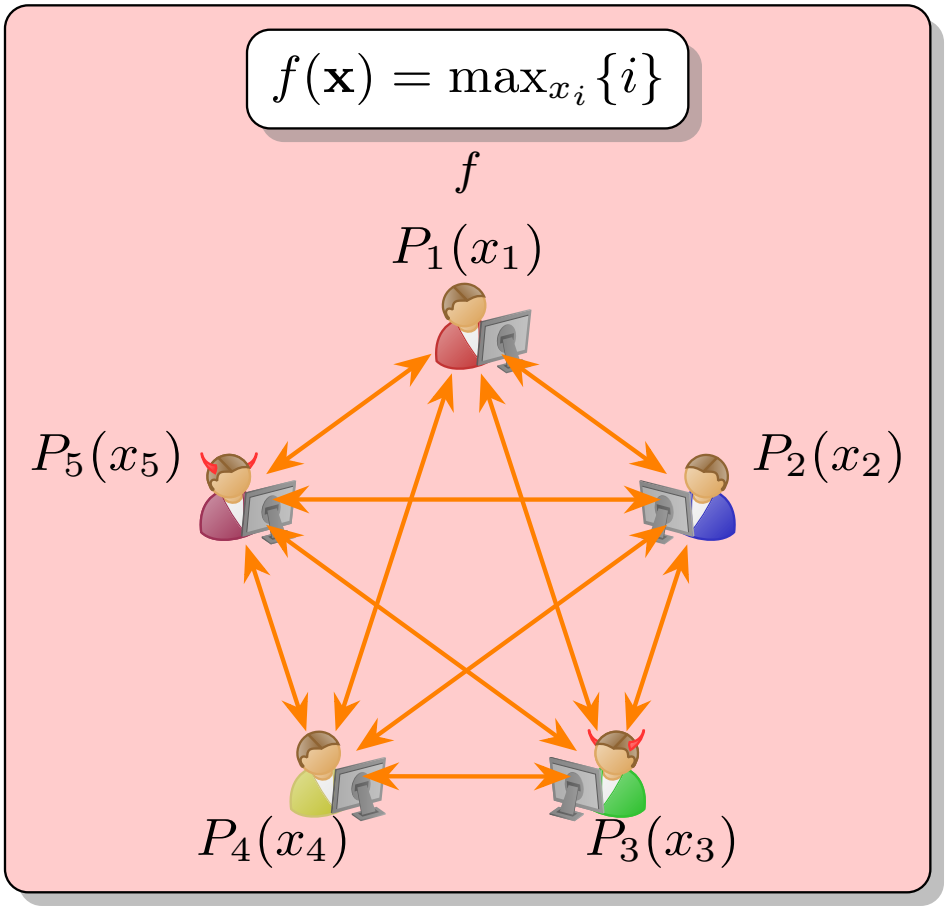


Real World

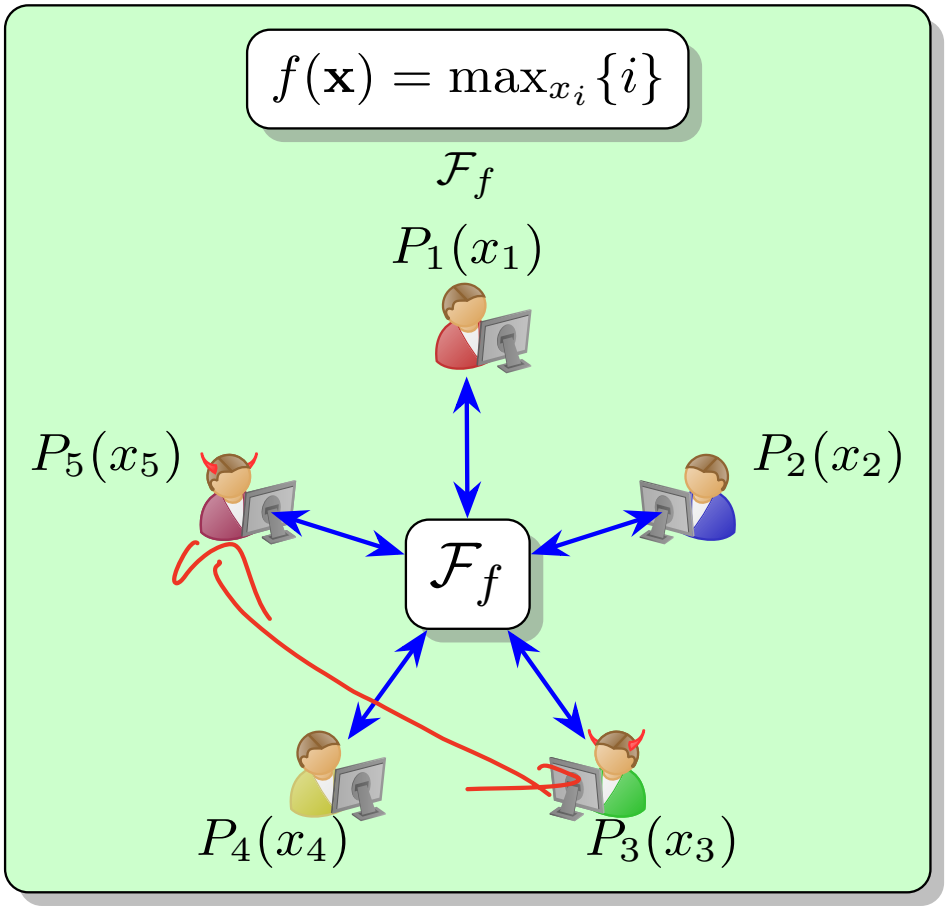


Ideal World

Real-Ideal Security Paradigm

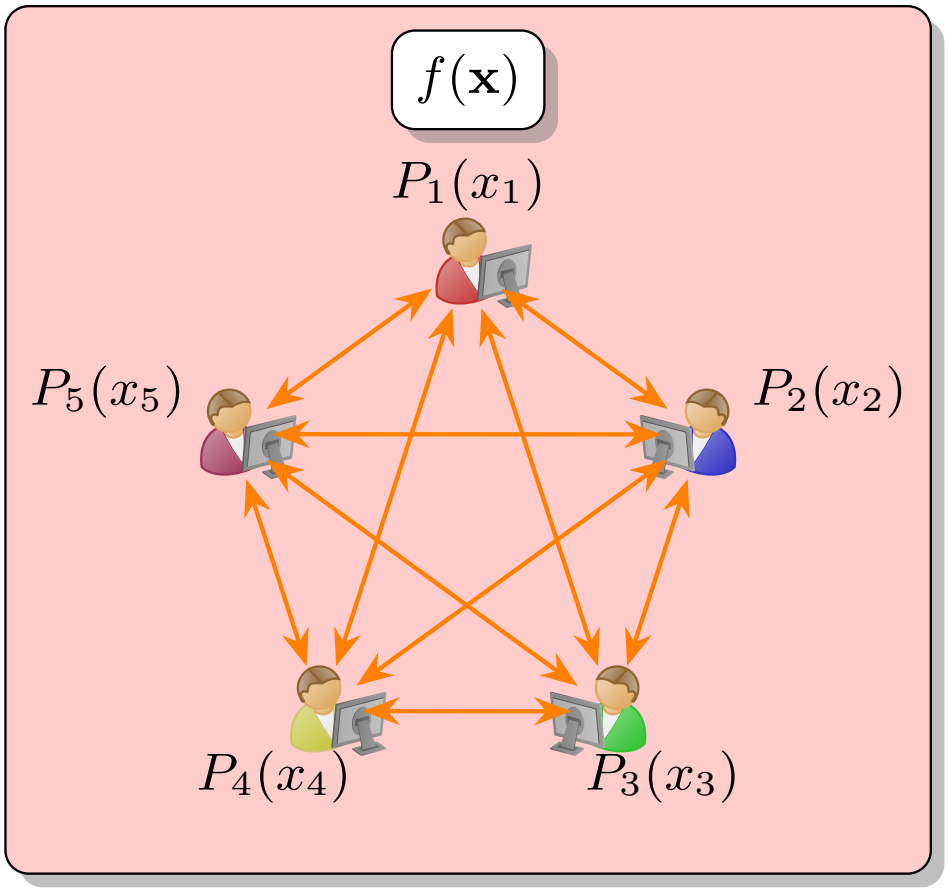


Real World

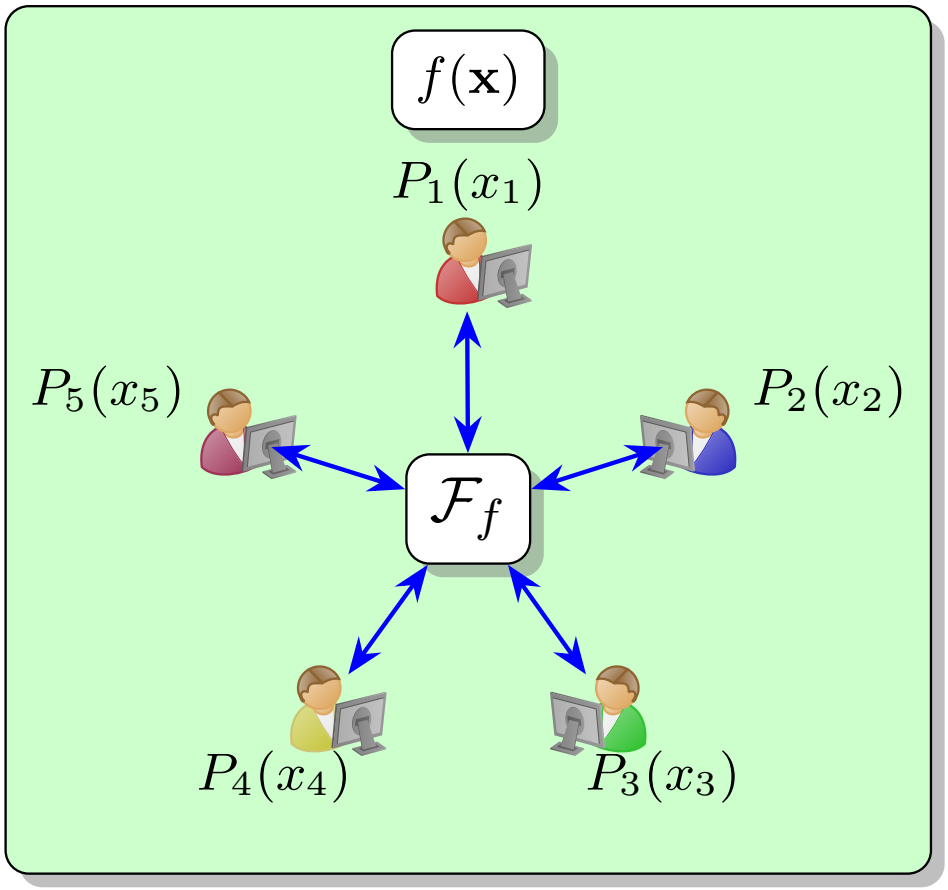


Ideal World

Semi-honest Security

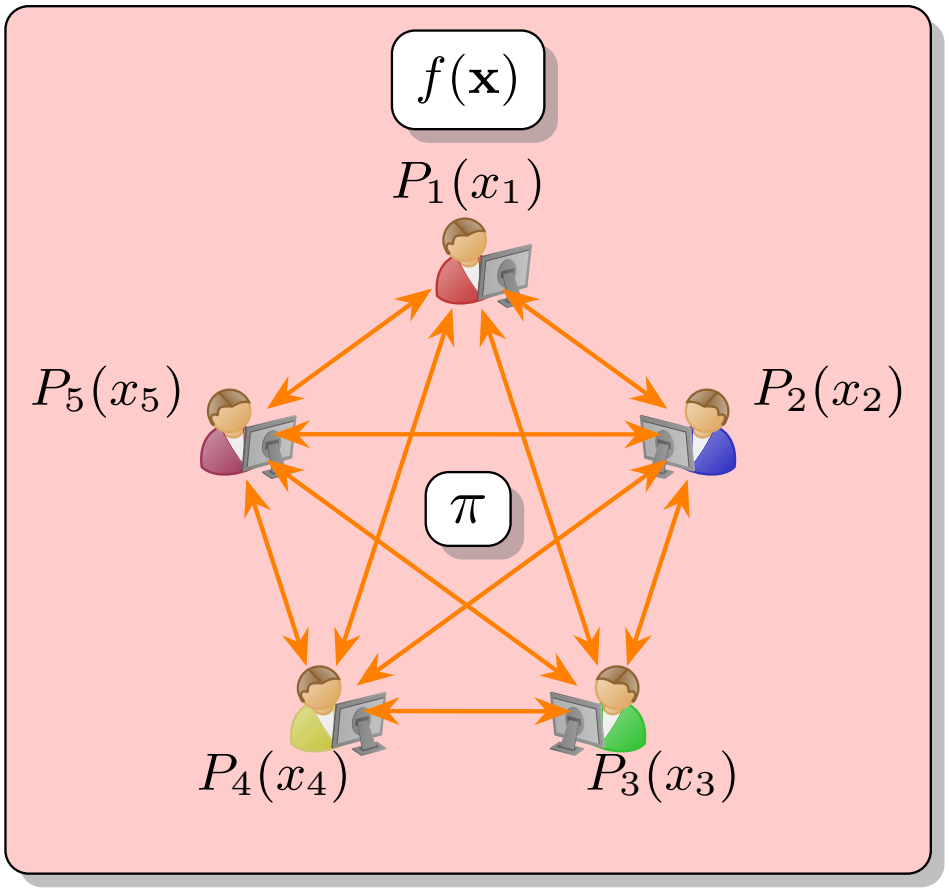


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

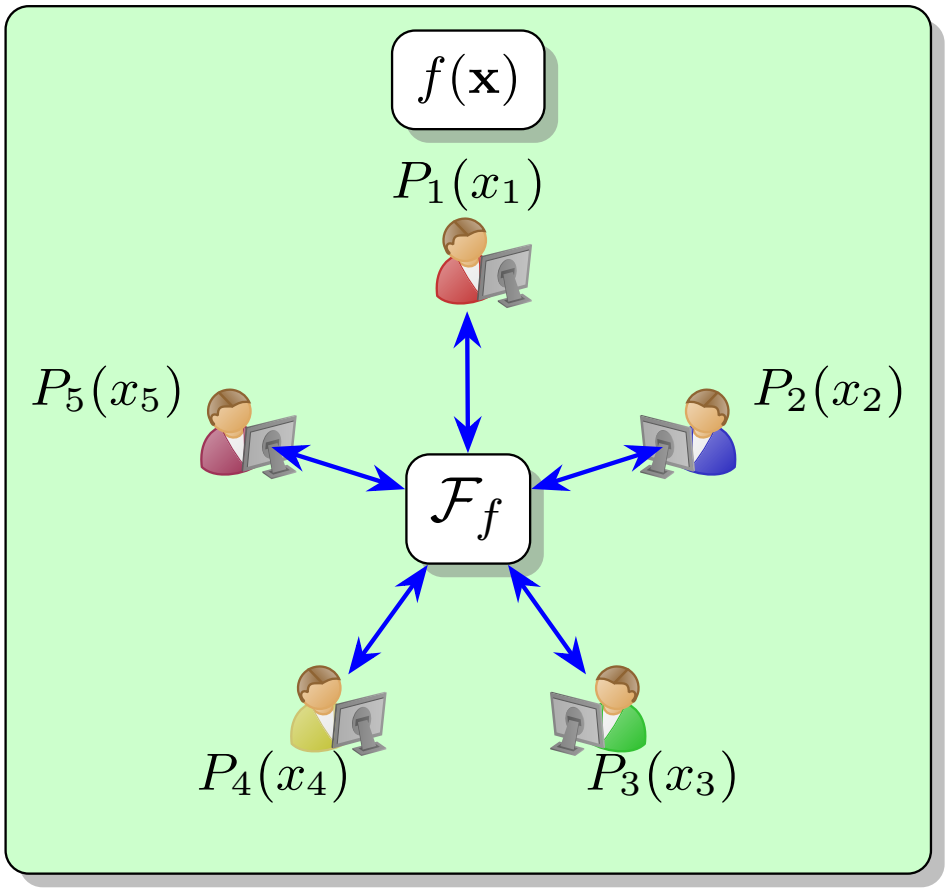


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

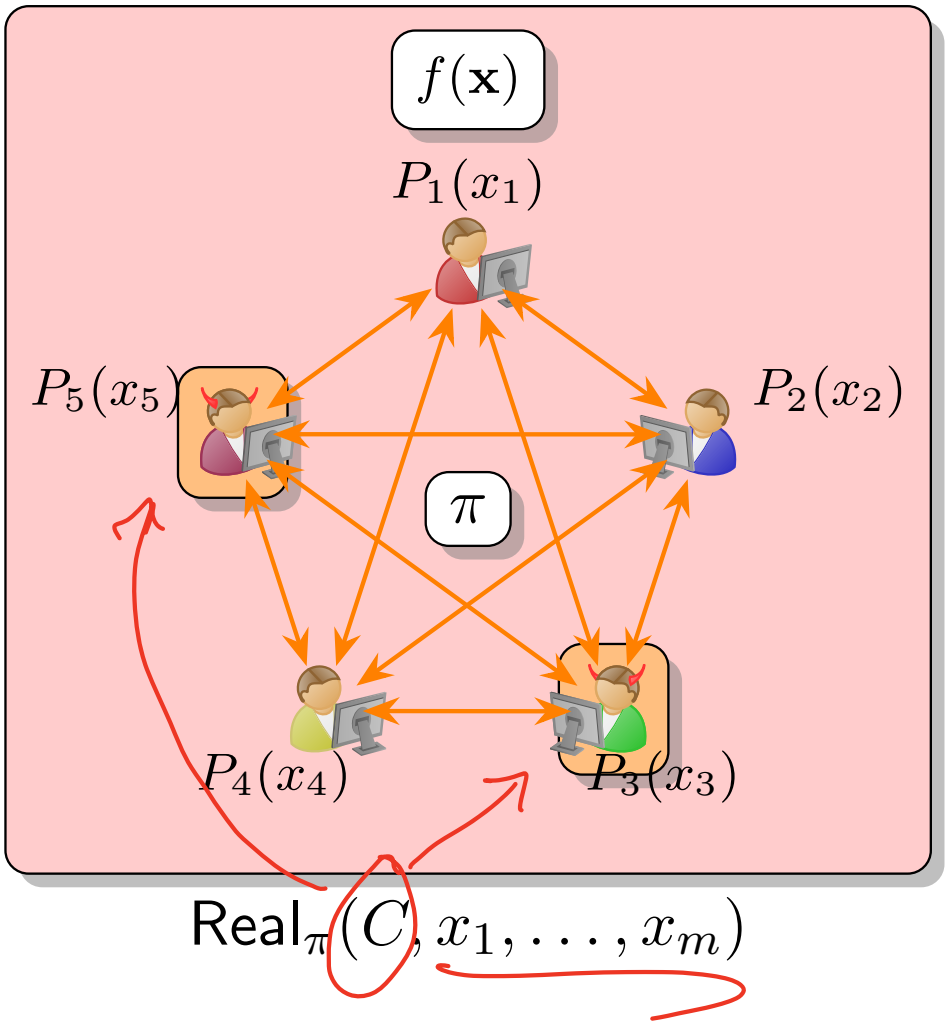


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

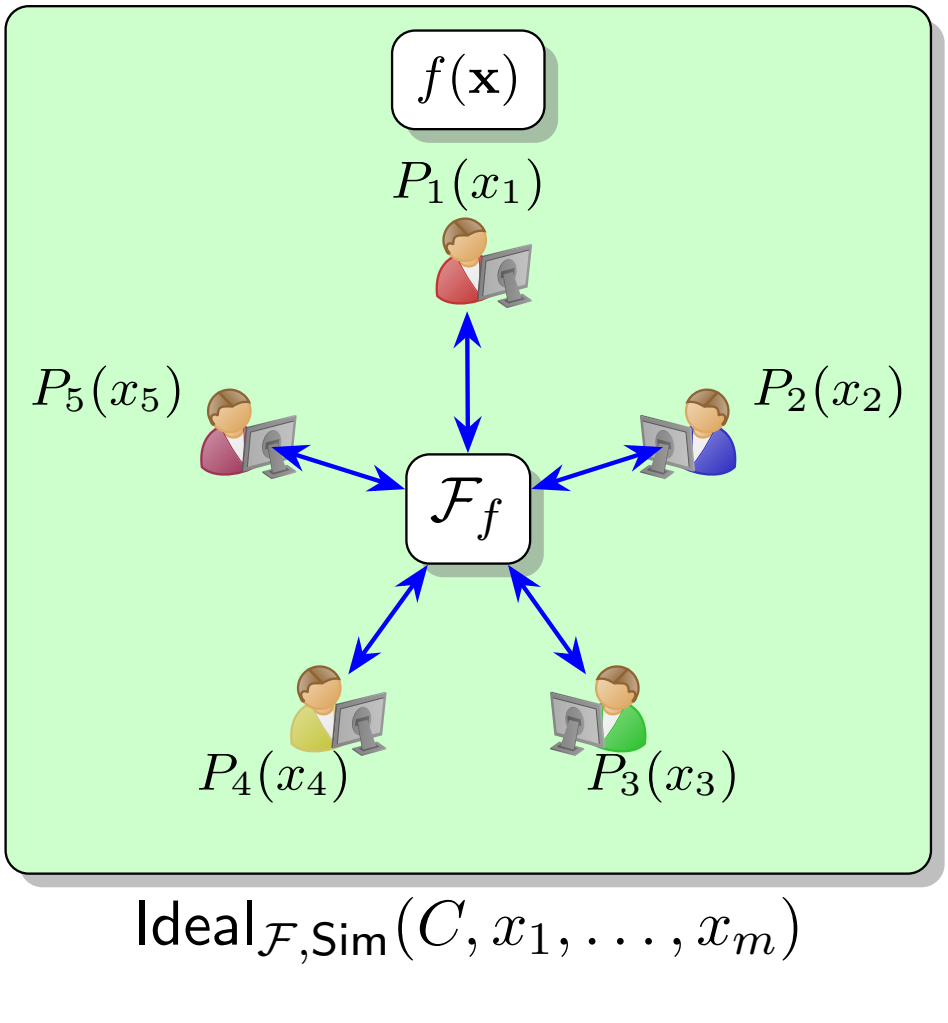


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

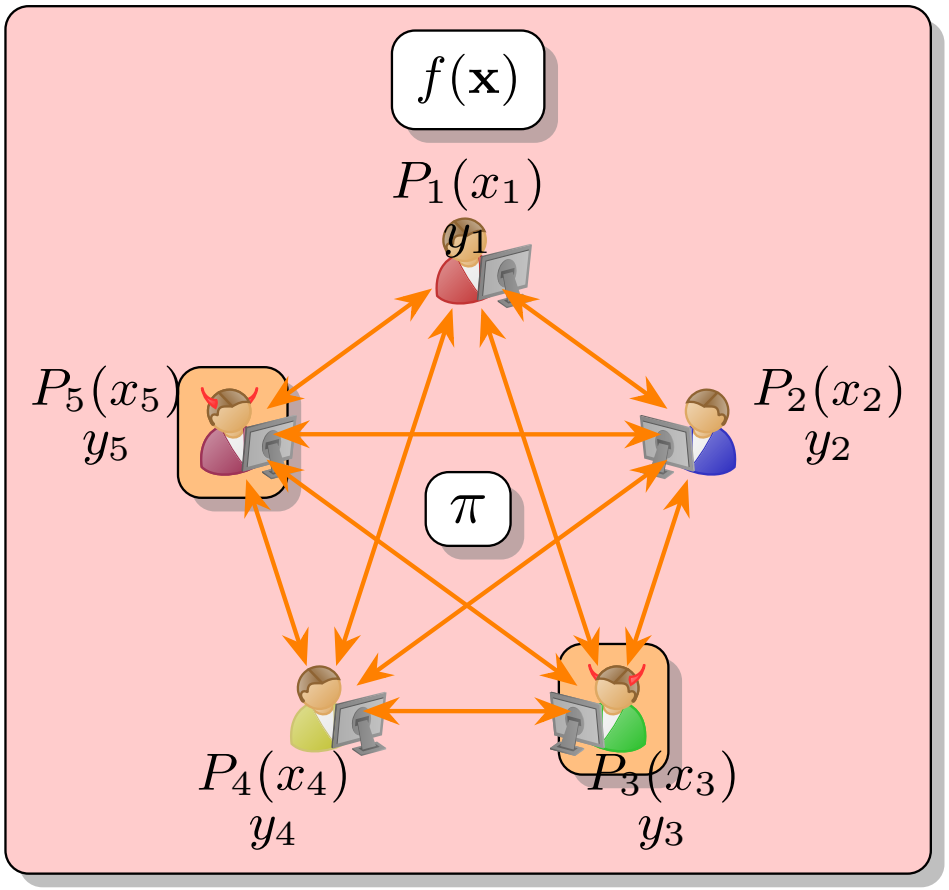


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

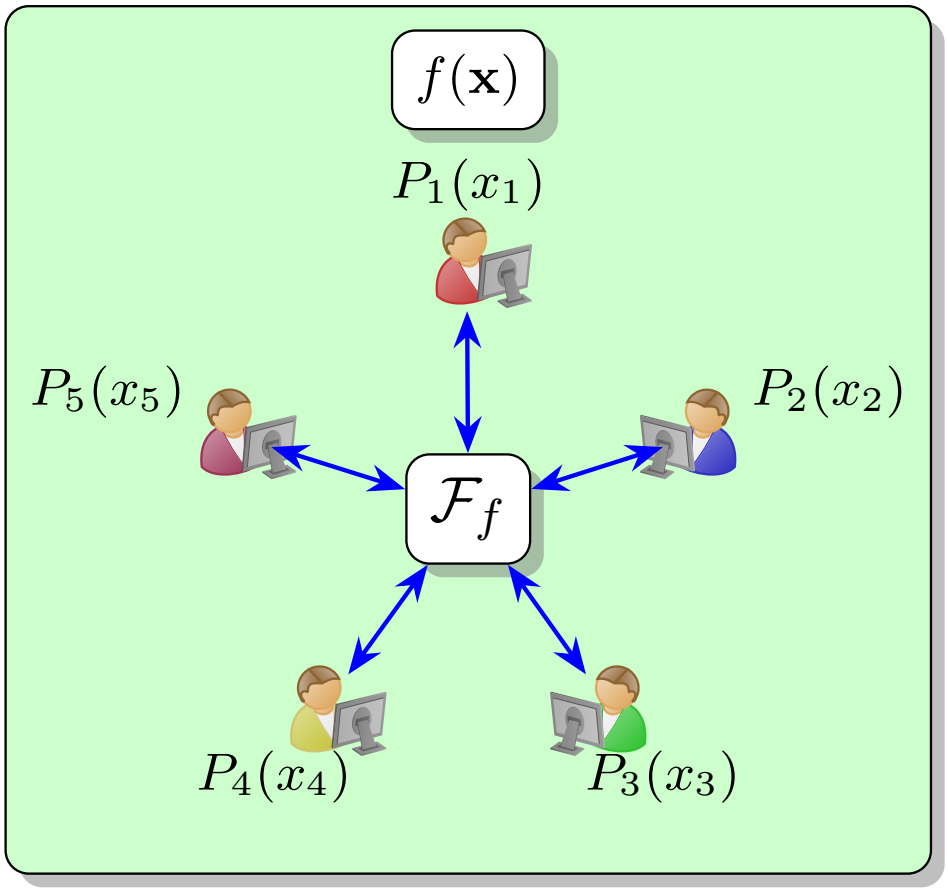


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

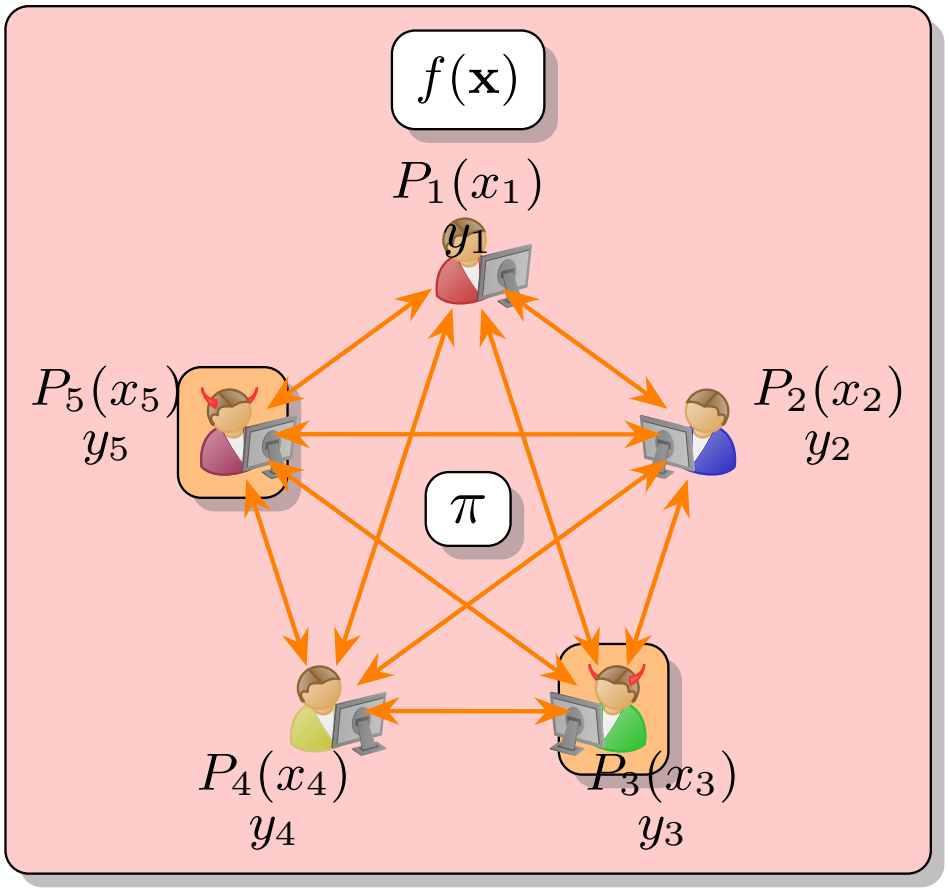


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

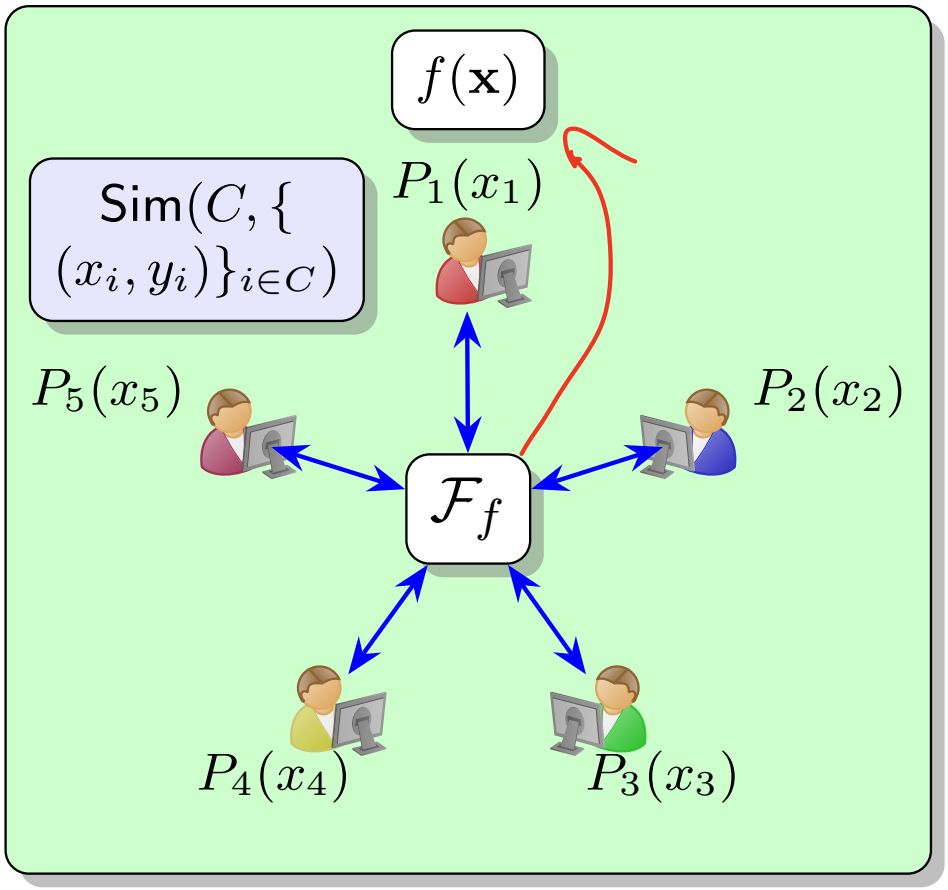


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

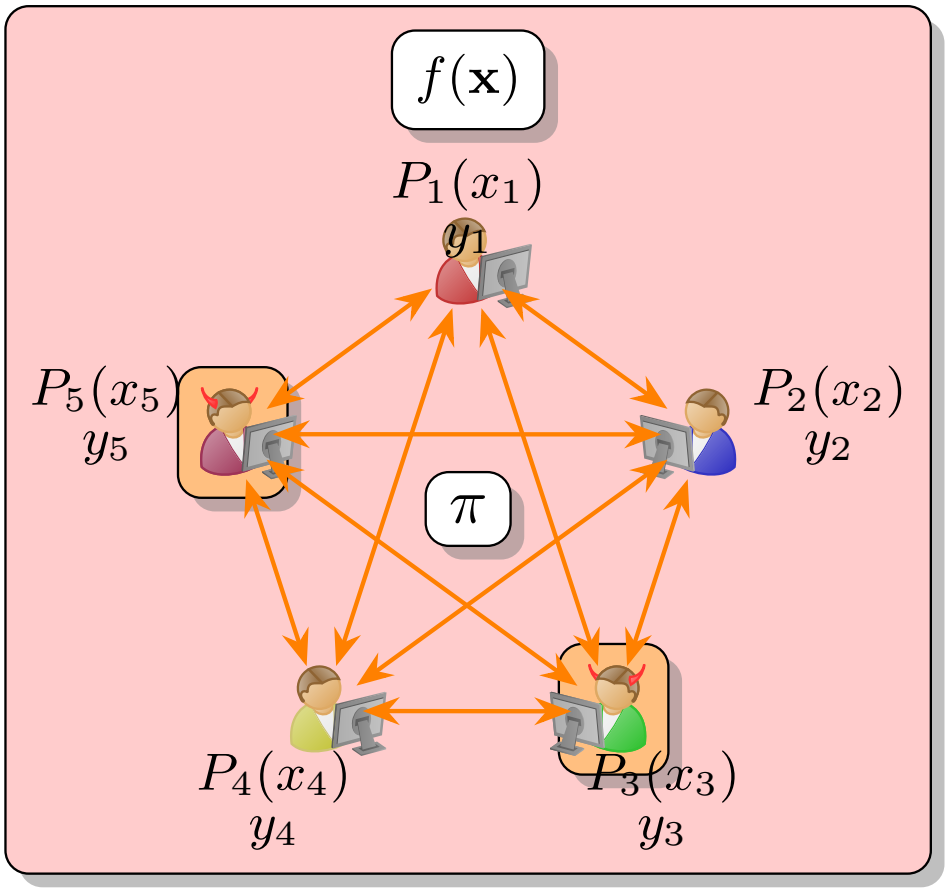


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

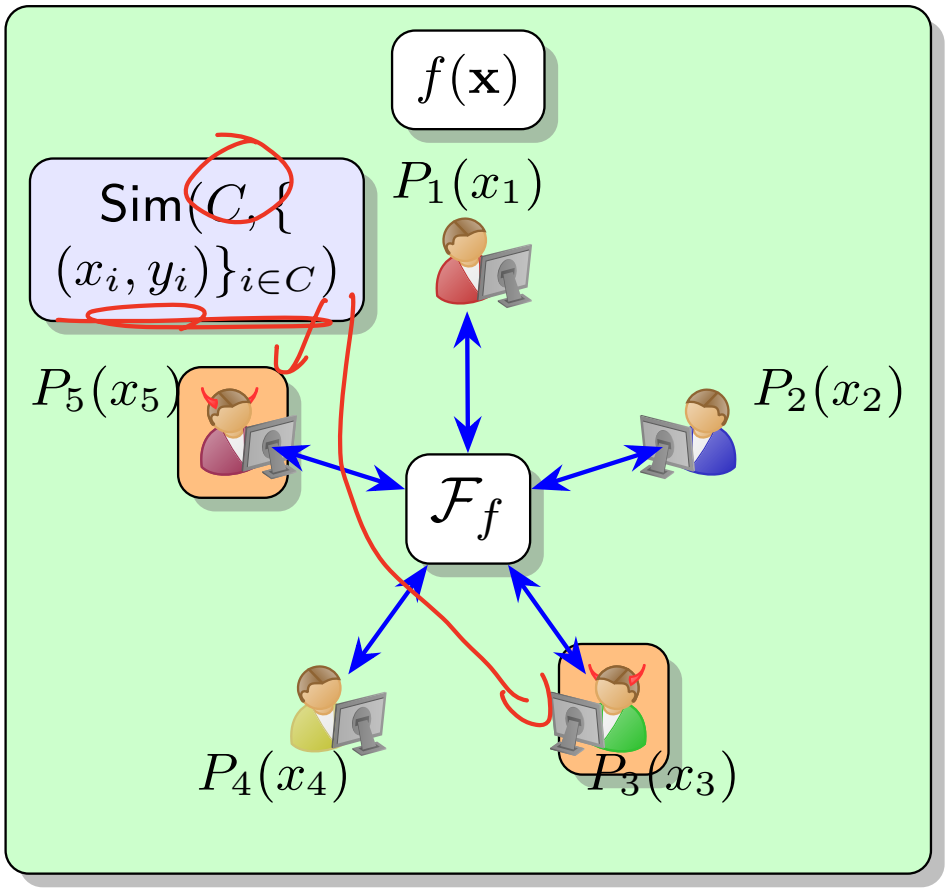


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

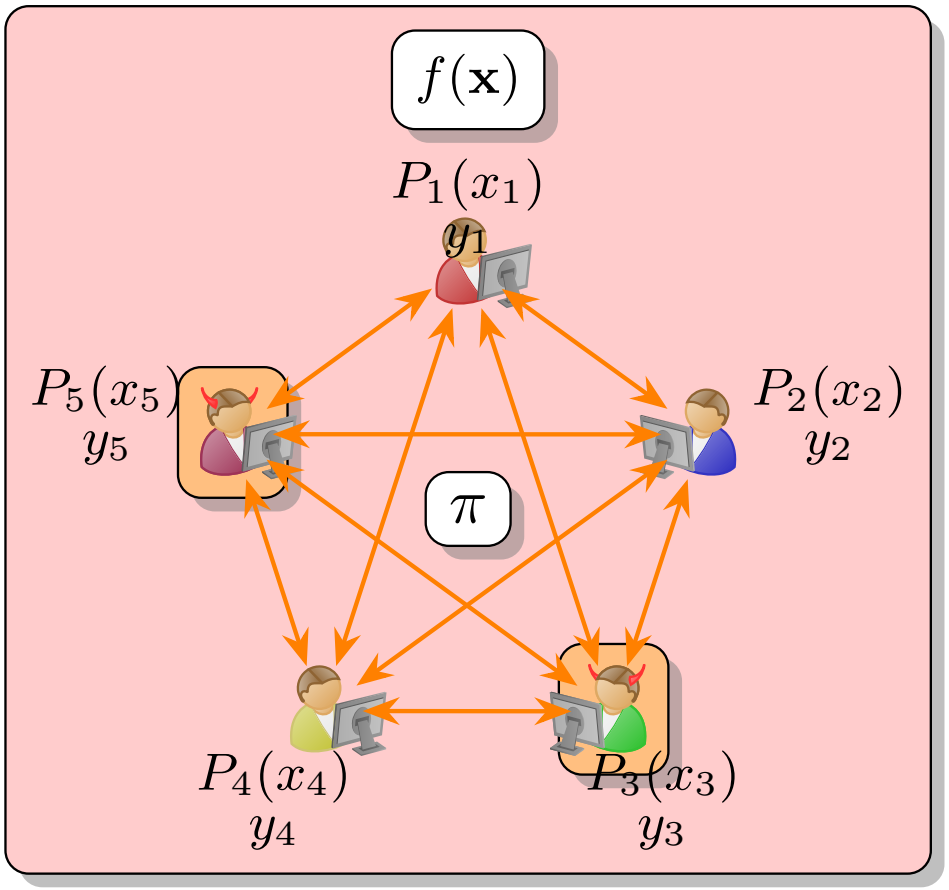


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

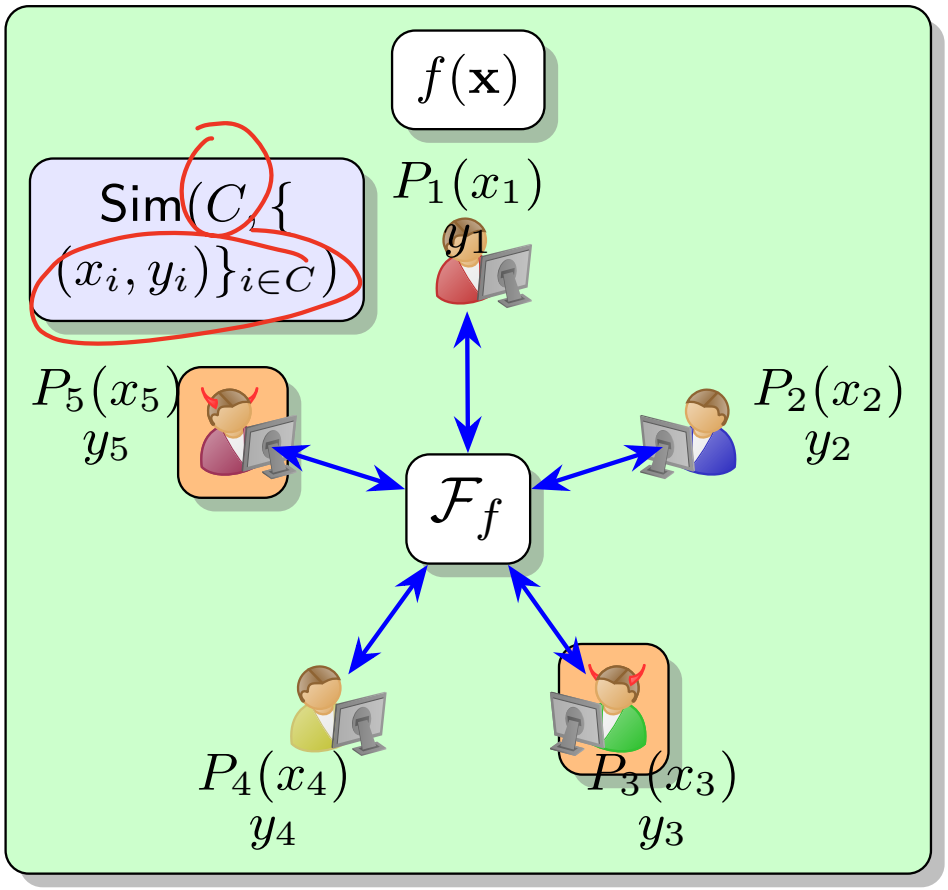


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security

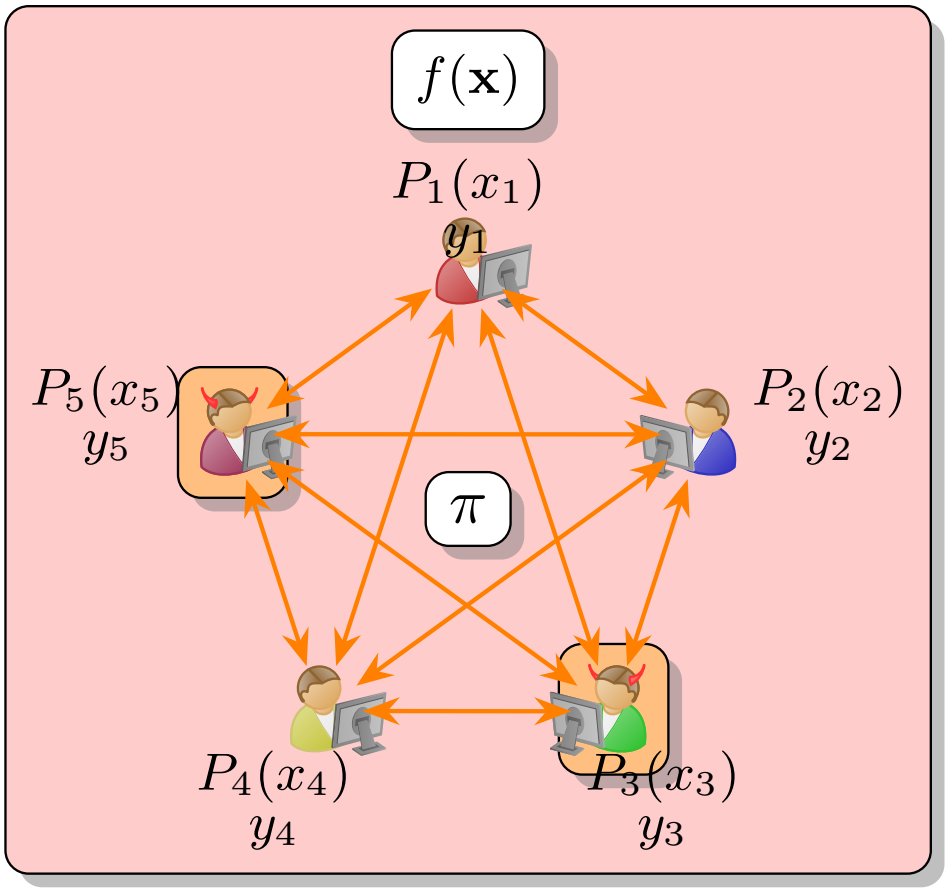


$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$

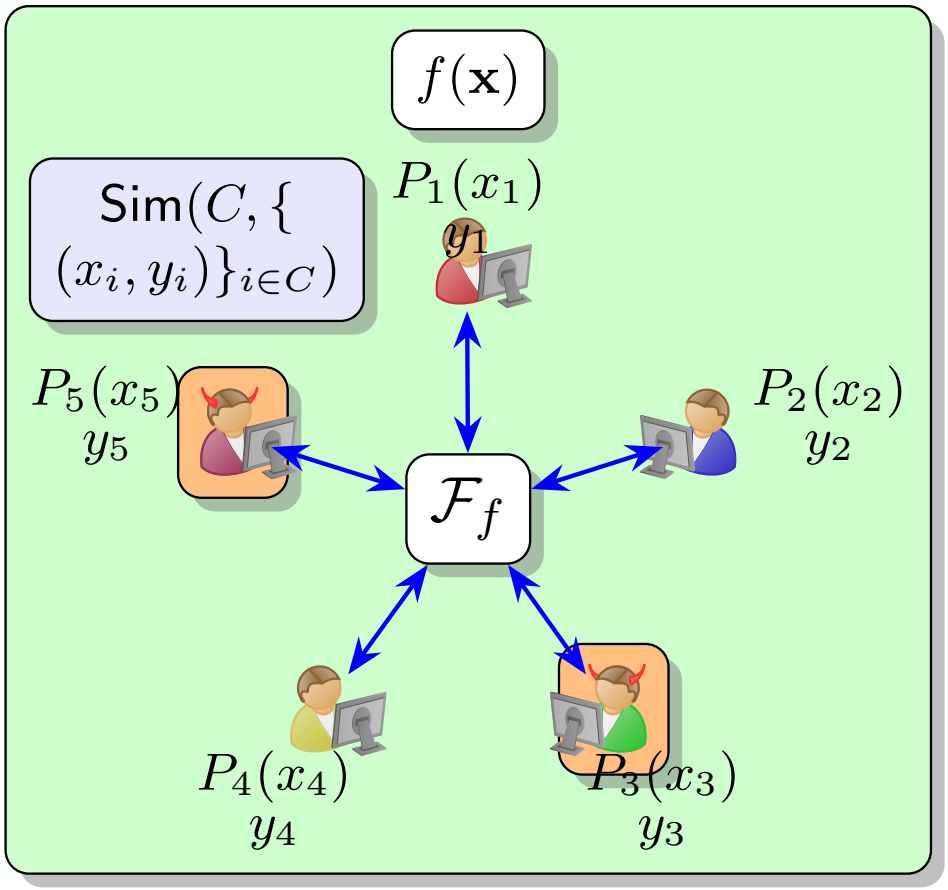


$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

Semi-honest Security



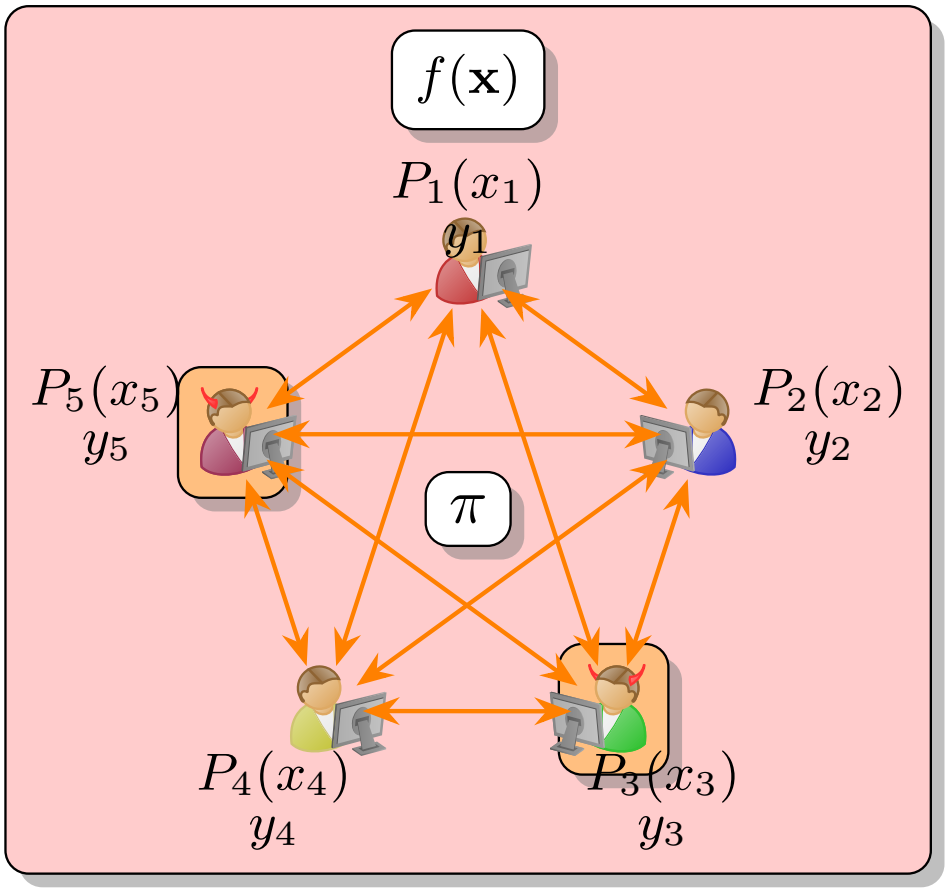
$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$



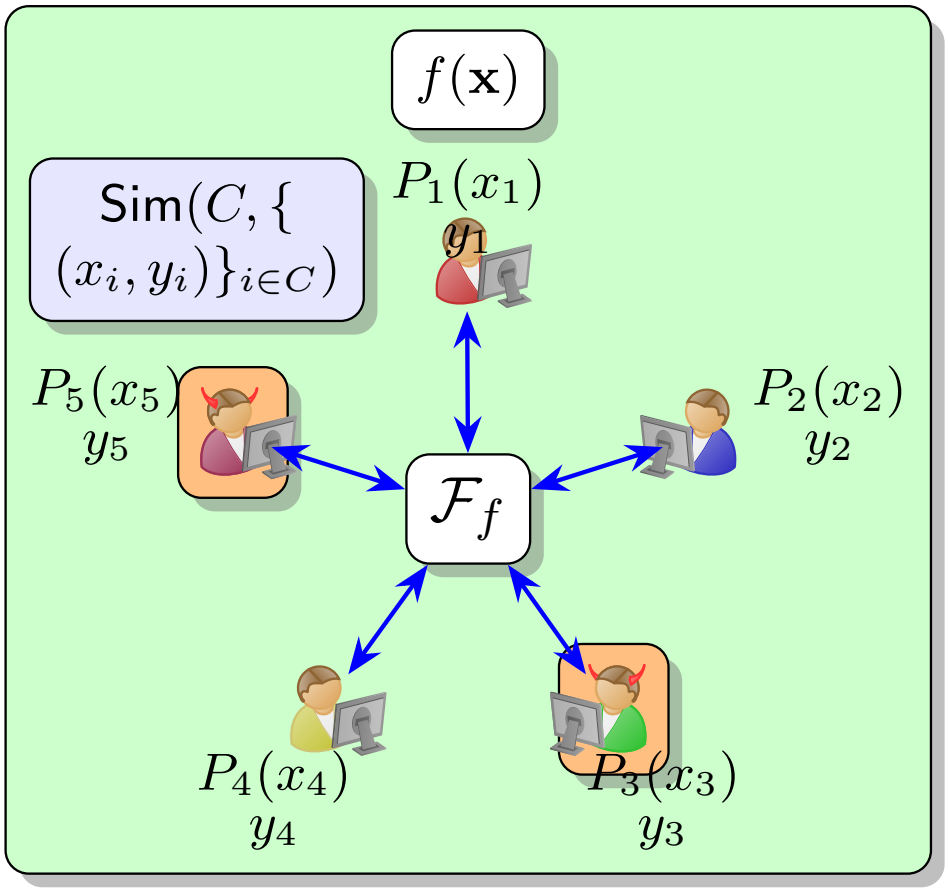
$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

- Sim is Ideal World "adversary"

Semi-honest Security



$\mathsf{Real}_{\pi}(C, x_1, \ldots, x_m)$



$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$

- Sim is Ideal World "adversary"
- Tries to mimic (i.e., simulate) view of adversaries in Real World while living in Ideal World.

Semi-honest Secure MPC

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $C \subset [m]$ denote the set of corrupt parties in π.
- We say that π *securely realizes* $\mathcal{F}$ *in the presence of semi-honest adversaries* if there exists a simulator Sim such that $\forall C \subset [m]$ and all inputs $x_1 \ldots, x_m$, the distributions $\mathsf{Real}_\pi(C, x_1, \ldots, x_m)$ and $\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(C, x_1, \ldots, x_m)$ are indistinguishable.

$\mathsf{Real}_\pi(C, x_1, \ldots, x_m)$

- Run protocol π where each party P_i honestly behaves with input x_i.
- Let V_i denote the final view of P_i.
- Output $\{V_i : i \in C\}, (y_1, \ldots, y_m)$.

$\mathsf{Ideal}(C, x_1, \ldots, x_m)$

- Compute $(y_1, \ldots, y_m) \leftarrow \mathcal{F}(x_1, \ldots, x_m)$.
- Output $\mathsf{Sim}(C, \{(x_i, y_i) : i \in C\}), (y_1, \ldots, y_m)$

Why Semi-honest Security?

Why Semi-honest Security?

- First glance: weak definition of security!

Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.

Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.

- However, even semi-honest security is far from trivial.

Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.
- However, even semi-honest security is far from trivial.
- Semi-honest secure protocols π are often used to build more *powerful* protocols π'.

Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.
- However, even semi-honest security is far from trivial.
- Semi-honest secure protocols π are often used to build more *powerful* protocols π'.
  - E.g., malicious security.

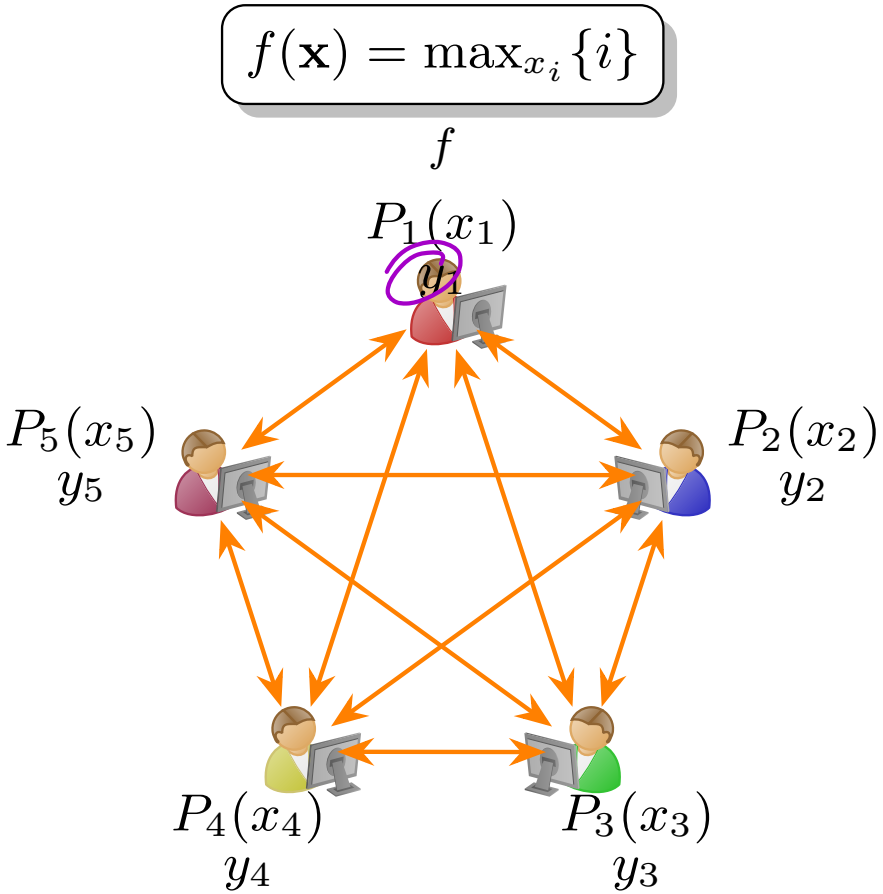
Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.
- However, even semi-honest security is far from trivial.
- Semi-honest secure protocols π are often used to build more *powerful* protocols π'.
  - E.g., malicious security.
- Even semi-honest security has realistic real-world scenarios.

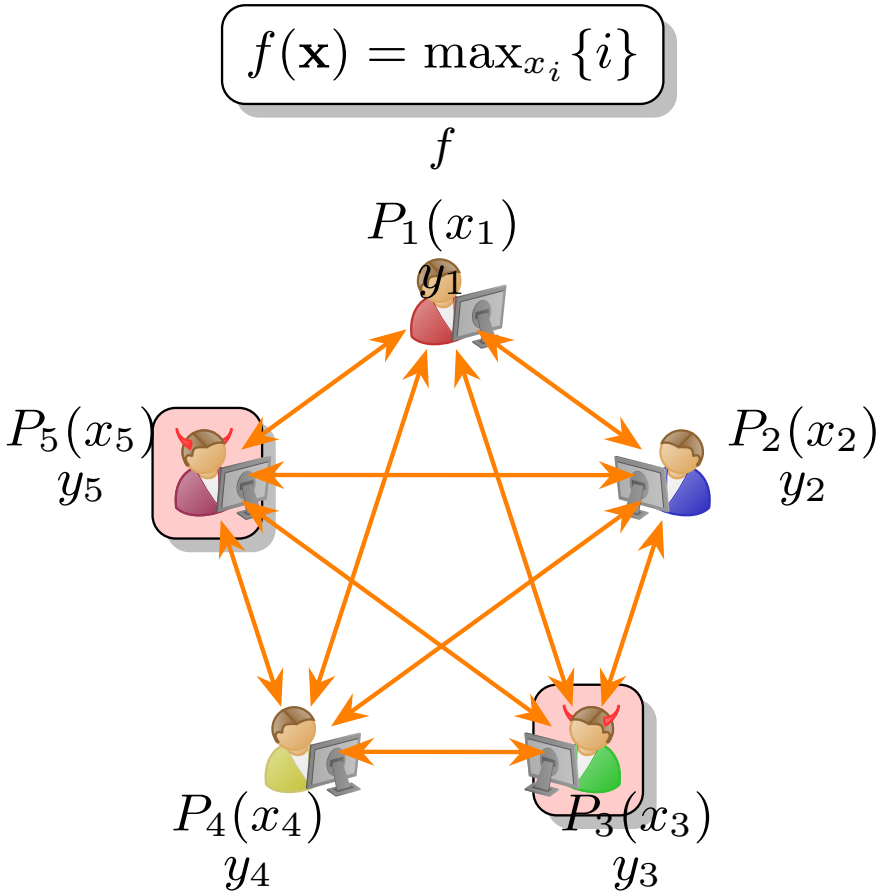
Why Semi-honest Security?

- First glance: weak definition of security!
  - Analogous to HVZK.
- However, even semi-honest security is far from trivial.
- Semi-honest secure protocols π are often used to build more *powerful* protocols π'.
  - E.g., malicious security.
- Even semi-honest security has realistic real-world scenarios.
  - Security of protocols with parties who are trusted (or have incentives) to act honestly, but may become compromised later.

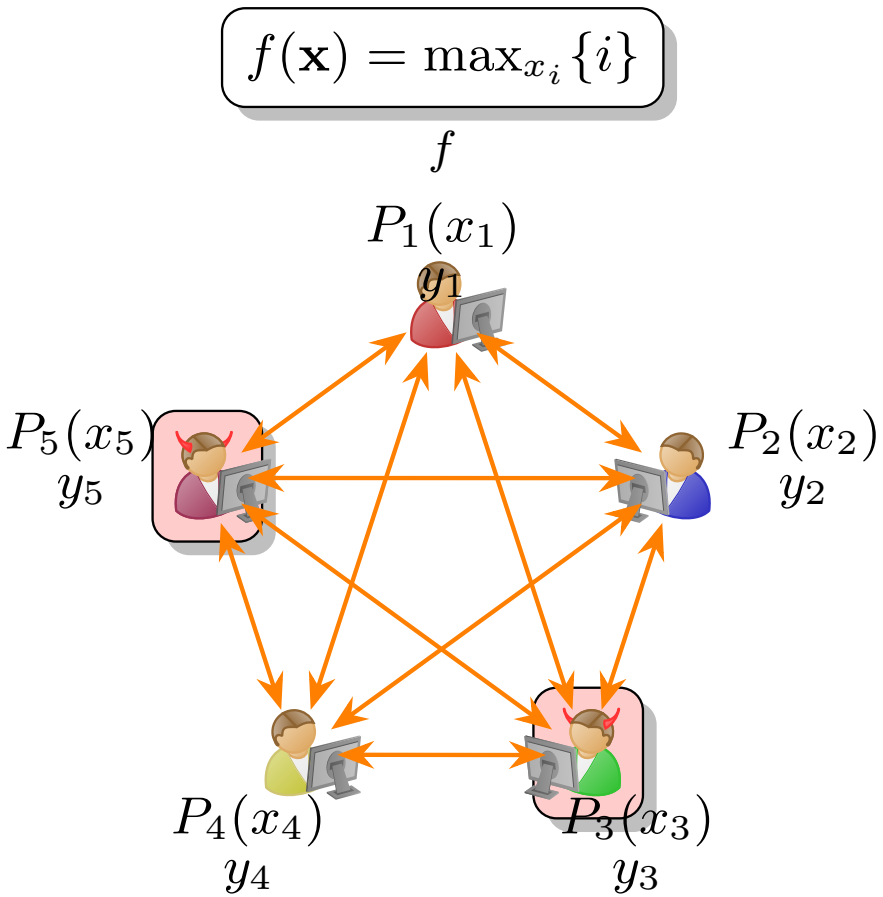
Malicious Corruptions



Malicious Corruptions

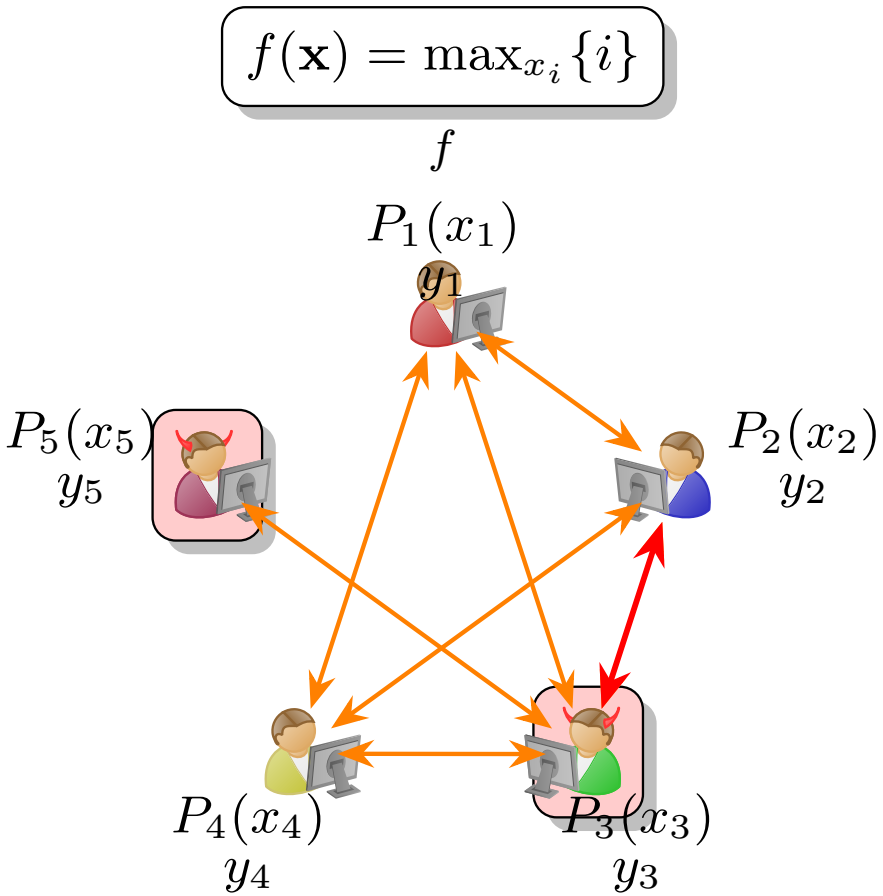


Malicious Corruptions



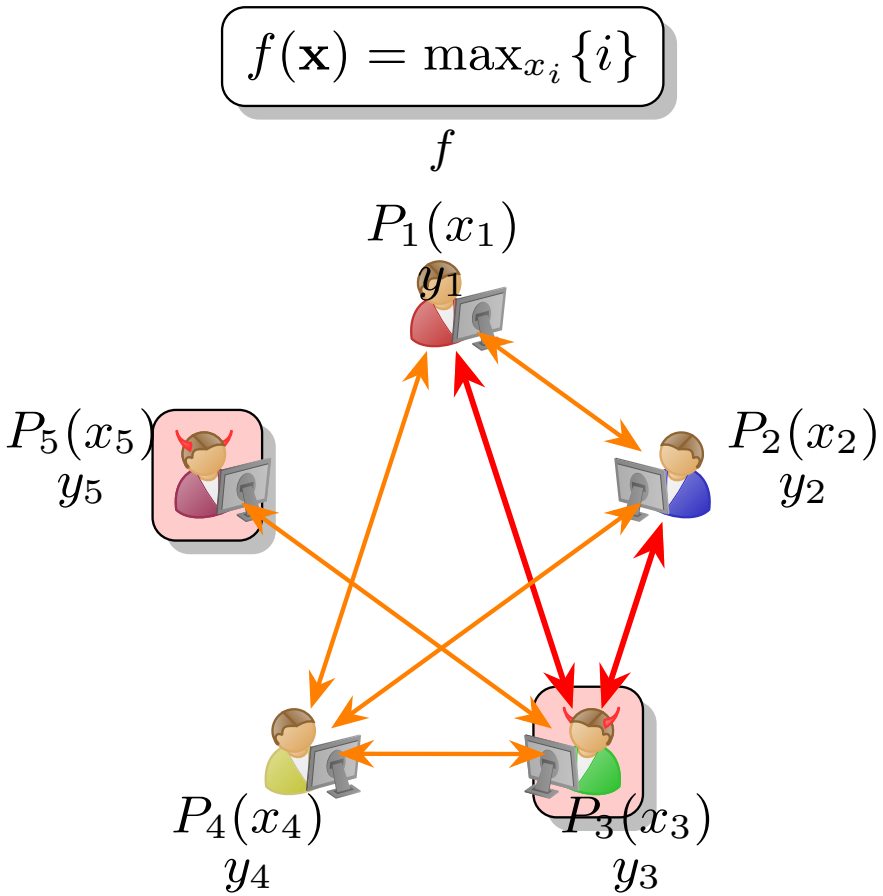
- Corrupt parties can deviate *arbitrarily* from protocol.

Malicious Corruptions



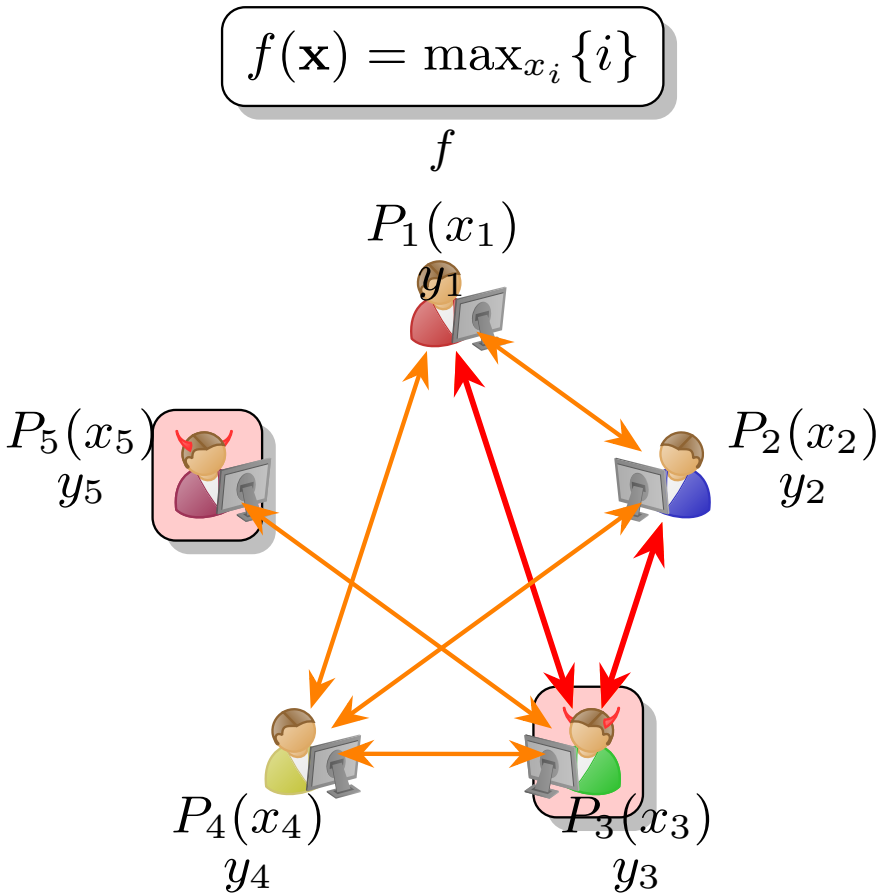
- Corrupt parties can deviate *arbitrarily* from protocol.

Malicious Corruptions



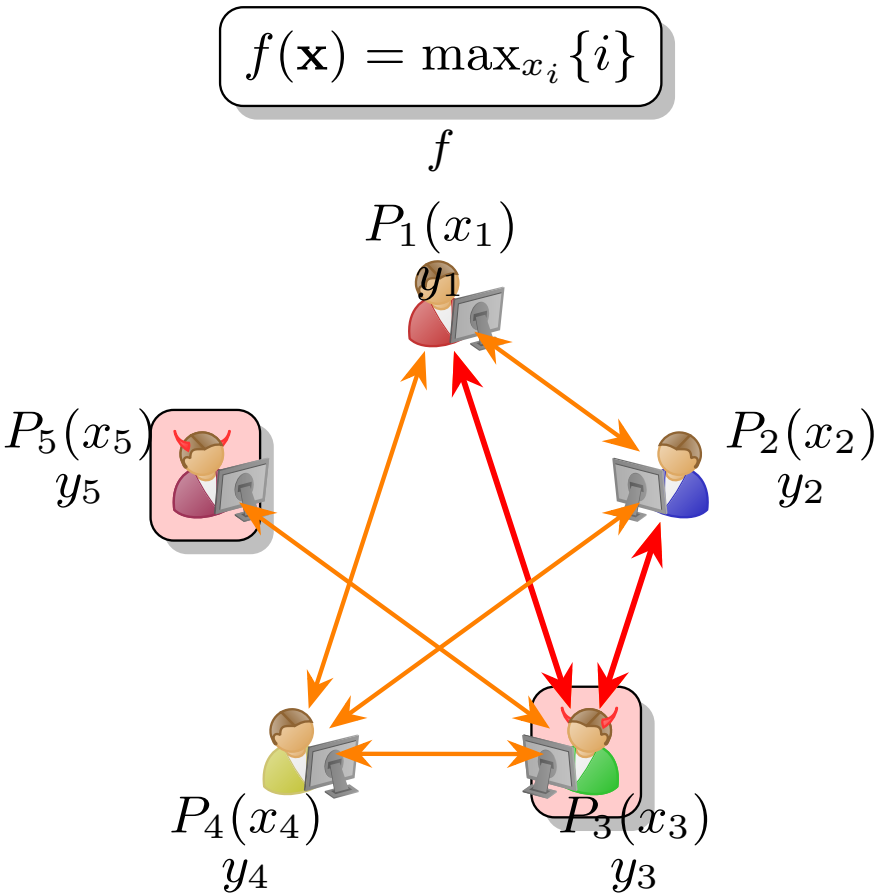
- Corrupt parties can deviate *arbitrarily* from protocol.

Malicious Corruptions



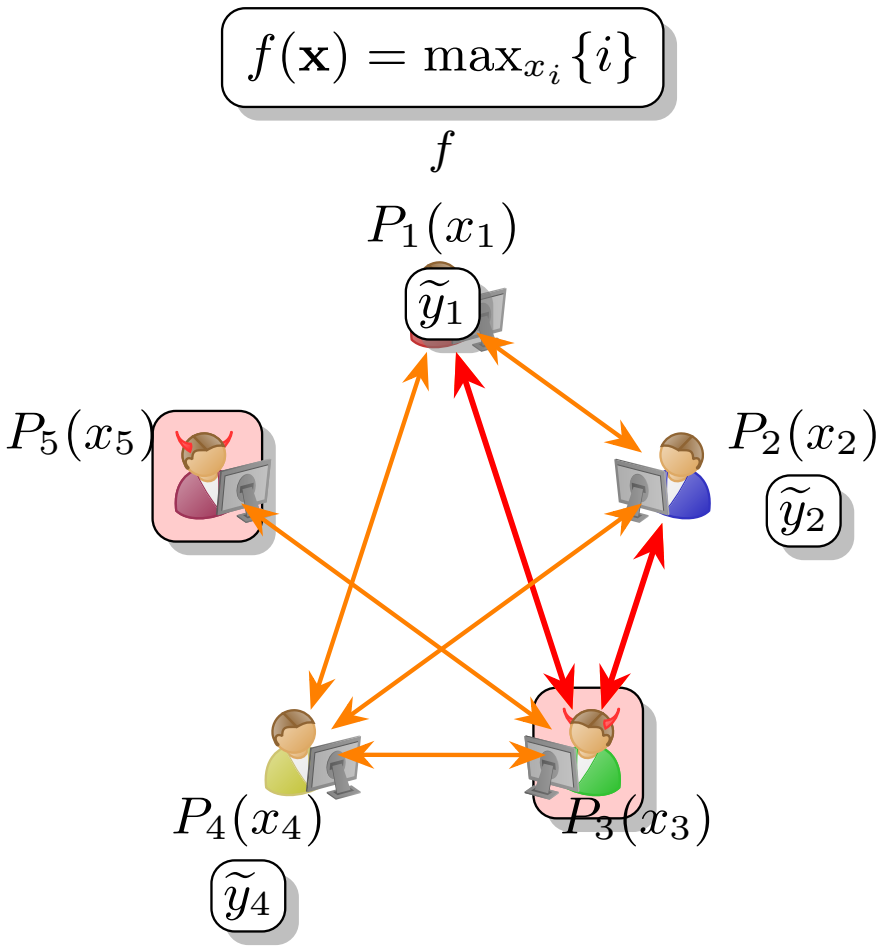
- Corrupt parties can deviate *arbitrarily* from protocol.
  - Can send garbage, different messages to different parties, etc.

Malicious Corruptions



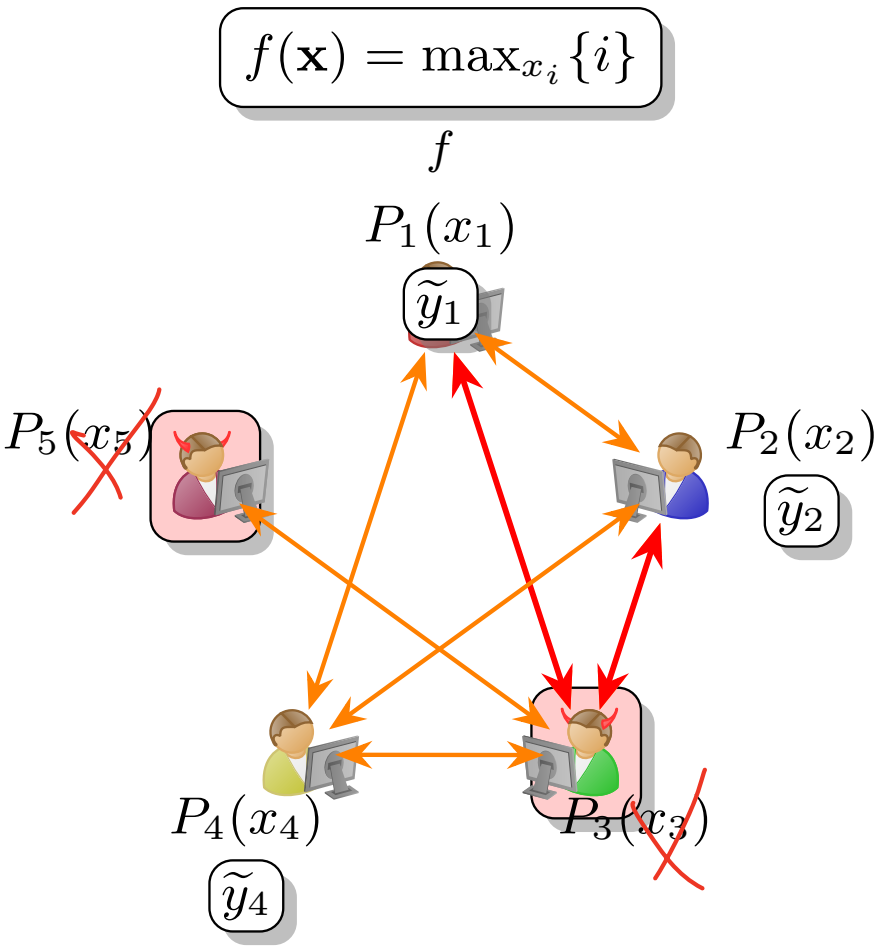
- Corrupt parties can deviate *arbitrarily* from protocol.
  - Can send garbage, different messages to different parties, etc.
- How does this affect *honest parties'* outputs?

Malicious Corruptions



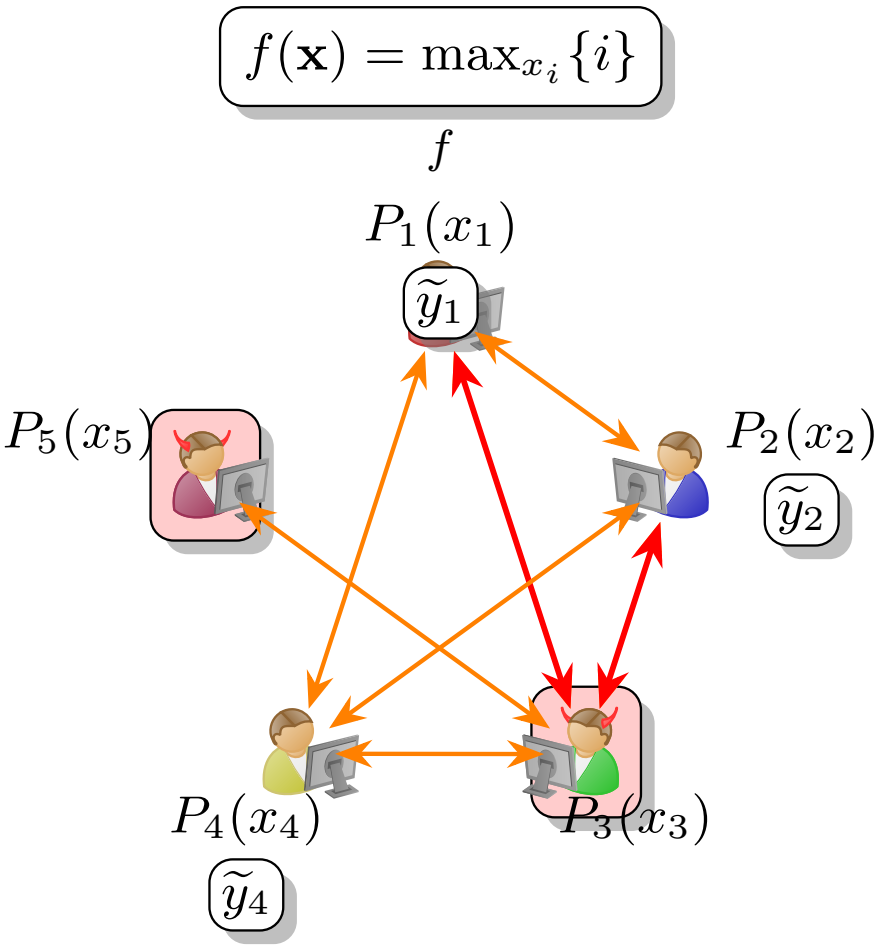
- Corrupt parties can deviate *arbitrarily* from protocol.
  - Can send garbage, different messages to different parties, etc.
- How does this affect *honest parties'* outputs?
- What is the adversary's input in the *ideal world*?

Malicious Corruptions



- Corrupt parties can deviate *arbitrarily* from protocol.
  - Can send garbage, different messages to different parties, etc.
- How does this affect *honest parties'* outputs?
- What is the adversary's input in the *ideal world*?
  - Arbitrary deviation means adversaries can simply ignore their inputs.

Malicious Corruptions



- Corrupt parties can deviate *arbitrarily* from protocol.
  - Can send garbage, different messages to different parties, etc.
- How does this affect *honest parties'* outputs?
- What is the adversary's input in the *ideal world*?
  - Arbitrary deviation means adversaries can simply ignore their inputs.
  - What does Sim give the ideal functionality?

Malicious Security Definition

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

- We say that π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

- We say that π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if for all $\mathcal{A}$

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

- We say that π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if for all $\mathcal{A}$ there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ with $\mathsf{corr}(\mathcal{A}) = \mathsf{corr}(Sim)$

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

- We say that π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if for all $\mathcal{A}$ there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ with $\mathsf{corr}(\mathcal{A}) = \mathsf{corr}(Sim)$ such that for all honest inputs $\{x_i : i \notin \mathsf{corr}(\mathcal{A})\}$,

Malicious Security Definition

- Let π be a protocol and $\mathcal{F}$ be a functionality.
- Let $\mathcal{A}$ denote the adversary and let $\mathsf{corr}(\mathcal{A}) \subset [m]$ denote the corrupt parties.
- For simulator Sim, we also let $\mathsf{corr}(\mathsf{Sim}) \subset [m]$ denote the set of parties corrupted by Sim in the ideal world.

- We say that π *securely realizes* $\mathcal{F}$ *in the presence of malicious adversaries* if for all $\mathcal{A}$ there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ with $\mathsf{corr}(\mathcal{A}) = \mathsf{corr}(Sim)$ such that for all honest inputs $\{x_i : i \notin \mathsf{corr}(\mathcal{A})\}$, the distributions following distributions (on the next slide) are indistinguishable.

Malicious Security Definition

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.

Malicious Security Definition

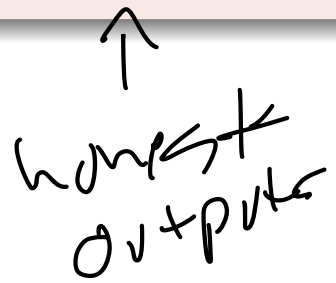
$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$



Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$

$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\{x_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$:

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$

$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\{x_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$:

- Obtain $\{x_j : j \in \mathsf{corr}(\mathsf{Sim})\} \leftarrow \mathsf{Sim}_1(1^?)$

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$

$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\{x_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$:

- Obtain $\{x_j^* : j \in \mathsf{corr}(\mathsf{Sim})\} \leftarrow \mathsf{Sim}_1$.
- Compute $(y_1, \ldots, y_m) \leftarrow \mathcal{F}(x_1, \ldots, x_m)$.

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$

$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\{x_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$:

- Obtain $\{x_j : j \in \mathsf{corr}(\mathsf{Sim})\} \leftarrow \mathsf{Sim}_1$.
- Compute $(y_1, \ldots, y_m) \leftarrow \mathcal{F}(x_1, \ldots, x_m)$.
- Obtain $V^* \leftarrow \mathsf{Sim}_2(\{y_j : j \in \mathsf{corr}(\mathsf{Sim})\})$.

Malicious Security Definition

$\mathsf{Real}_{\pi,\mathcal{A}}(\{x_i : i \notin \mathsf{corr}(\mathcal{A})\})$:

- Run π such that each honest P_i has input x_i and behaves honestly, and $\mathcal{A}$ chooses the next messages of all corrupt parties $\widetilde{P}_j$.
- Let y_i be the output of each honest P_i, and let V_j denote the final view of every party $j \in [m]$.
- Output $(\{V_j : j \in \mathsf{corr}(\mathcal{A})\}, \{y_i : i \notin \mathsf{corr}(\mathcal{A})\})$

$\mathsf{Ideal}_{\mathcal{F},\mathsf{Sim}}(\{x_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$:

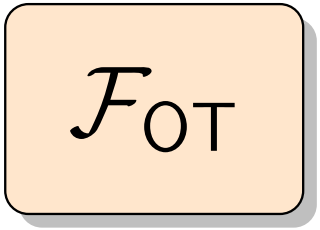
- Obtain $\{x_j : j \in \mathsf{corr}(\mathsf{Sim})\} \leftarrow \mathsf{Sim}_1$.
- Compute $(y_1, \ldots, y_m) \leftarrow \mathcal{F}(x_1, \ldots, x_m)$.
- Obtain $V^* \leftarrow \mathsf{Sim}_2(\{y_j : j \in \mathsf{corr}(\mathsf{Sim})\})$.
- Output $(V^*, \{y_i : i \notin \mathsf{corr}(\mathsf{Sim})\})$.

same len/ c' is

Yao's Garbled Circuits

Building Block: Oblivious Transfer

Building Block: Oblivious Transfer



Building Block: Oblivious Transfer

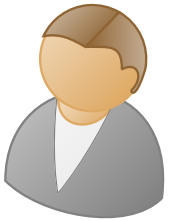
Sender $\mathcal{S}$

$\mathcal{F}_{\mathsf{OT}}$

Building Block: Oblivious Transfer

Sender $\mathcal{S}$

$\mathcal{F}_{\mathsf{OT}}$



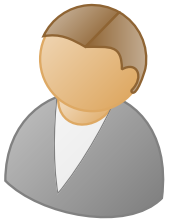
Receiver $\mathcal{R}$

Building Block: Oblivious Transfer

$\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^n$

Sender $\mathcal{S}$

$\mathcal{F}_{\mathsf{OT}}$



Receiver $\mathcal{R}$

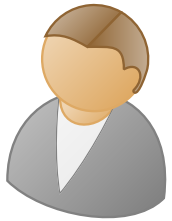
Building Block: Oblivious Transfer

$\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^n$

Sender $\mathcal{S}$

$\mathcal{F}_{\mathsf{OT}}$

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

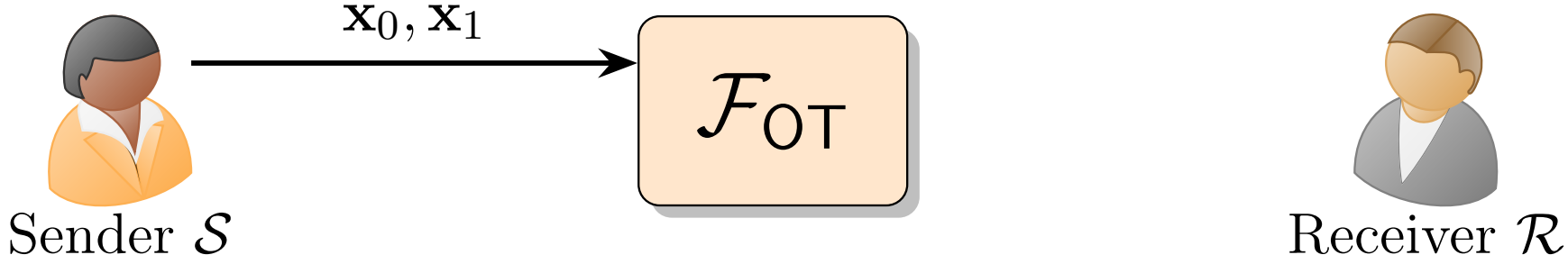


Receiver $\mathcal{R}$

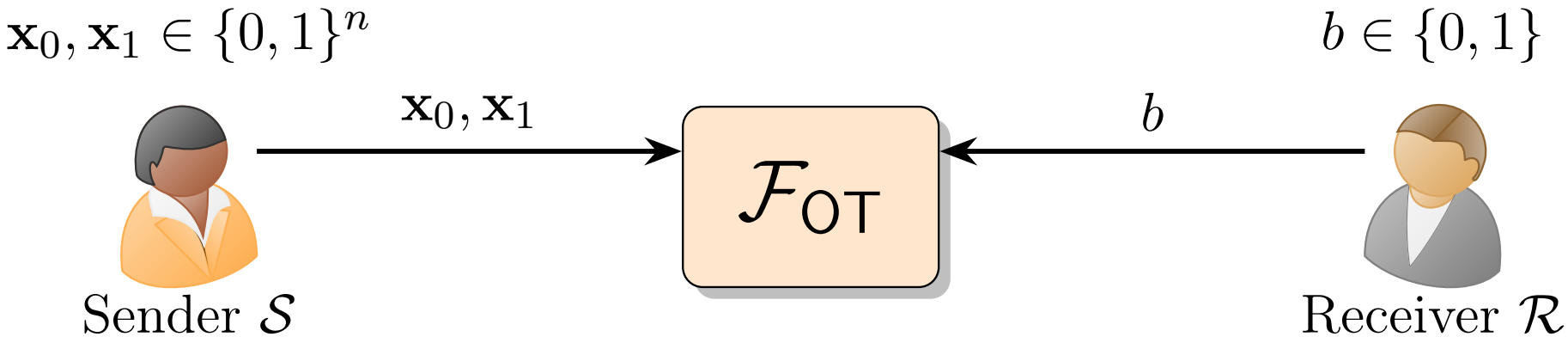
Building Block: Oblivious Transfer

$\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^n$

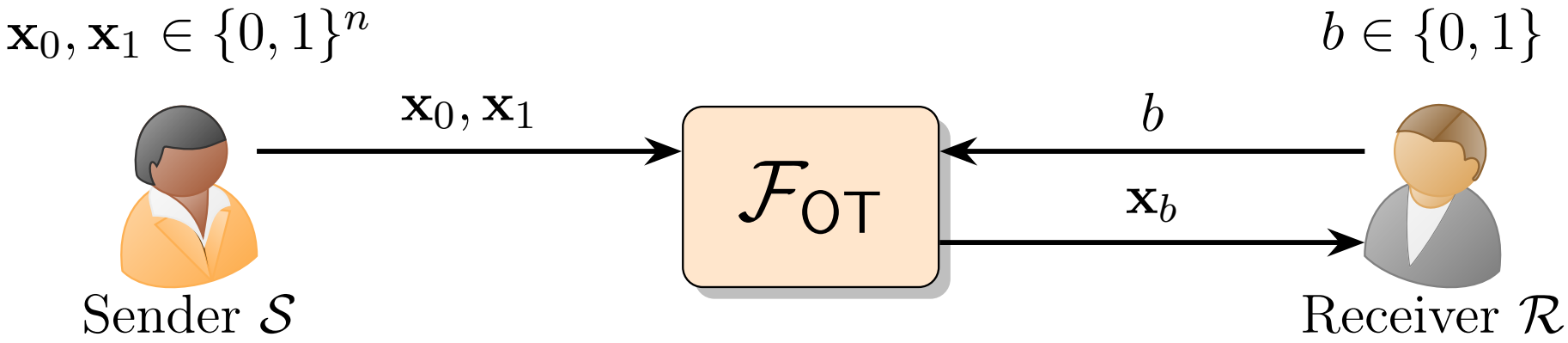
$b \in \{0,1\}$



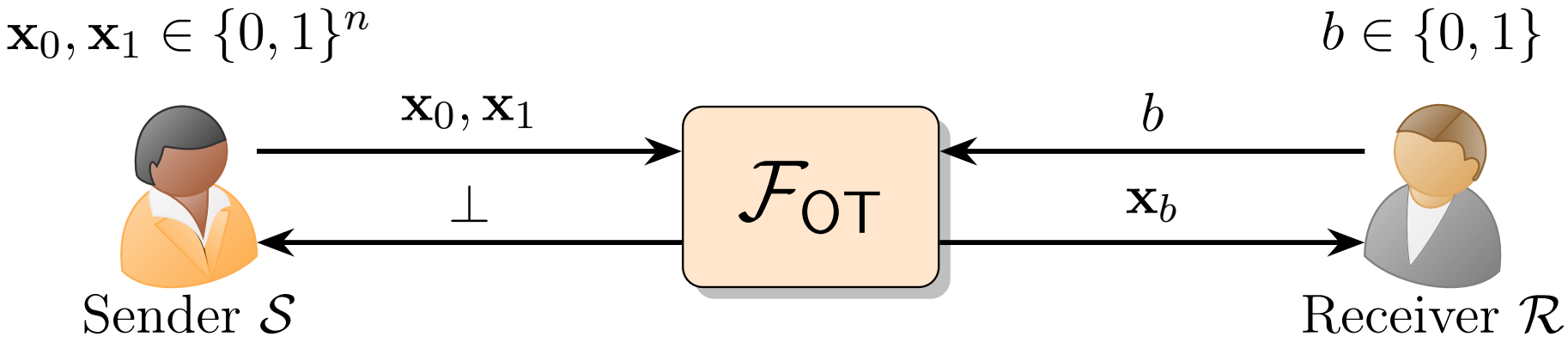
Building Block: Oblivious Transfer



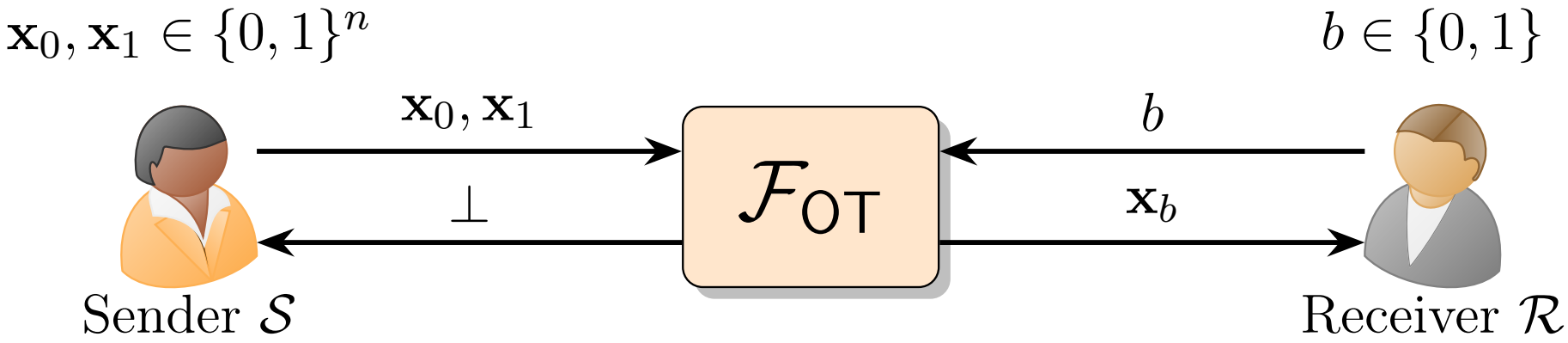
Building Block: Oblivious Transfer



Building Block: Oblivious Transfer

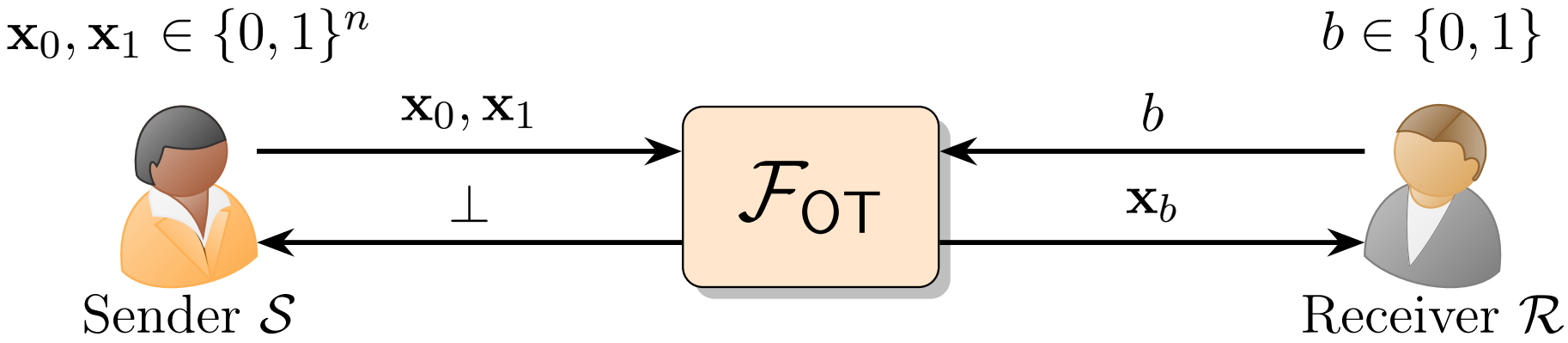


Building Block: Oblivious Transfer



Key Properties

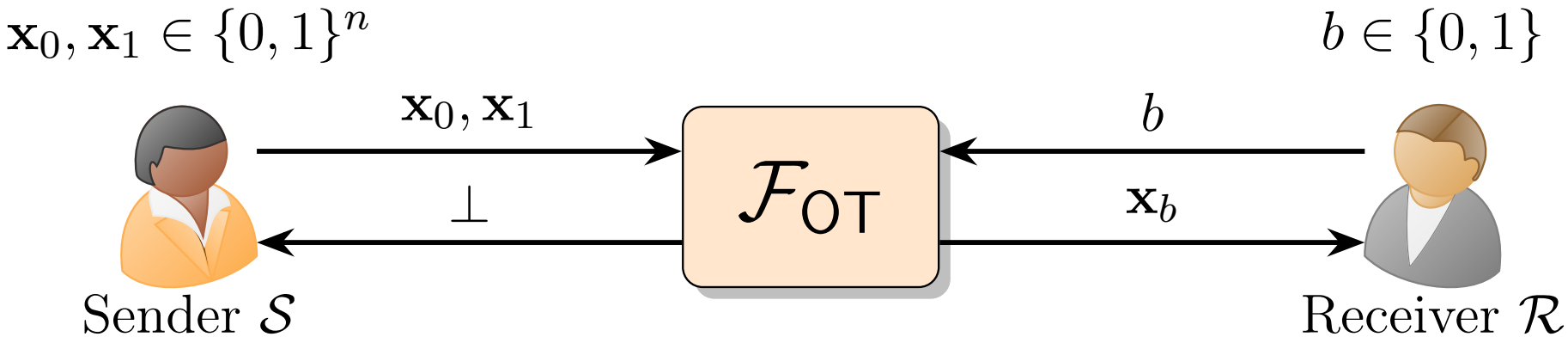
Building Block: Oblivious Transfer



Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.

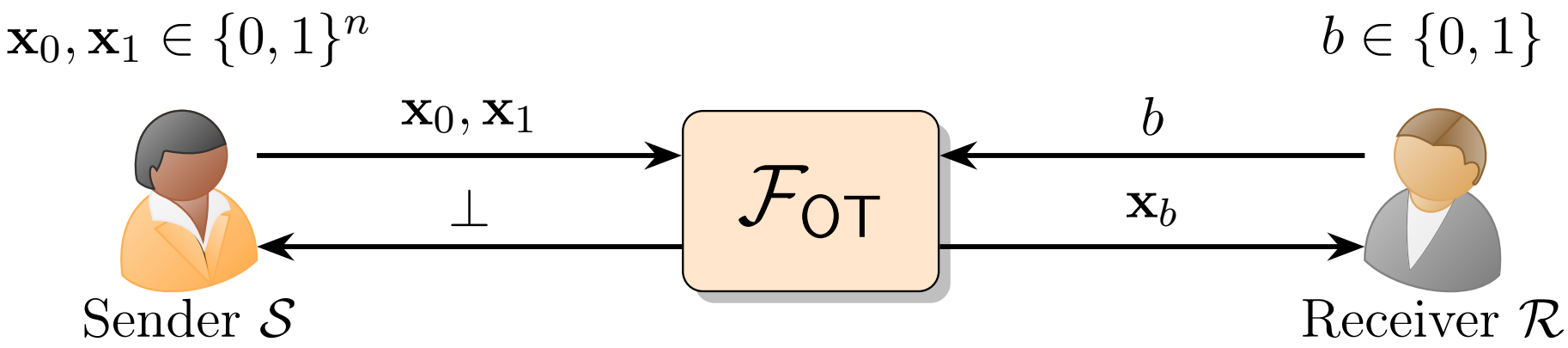
Building Block: Oblivious Transfer



Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

Building Block: Oblivious Transfer

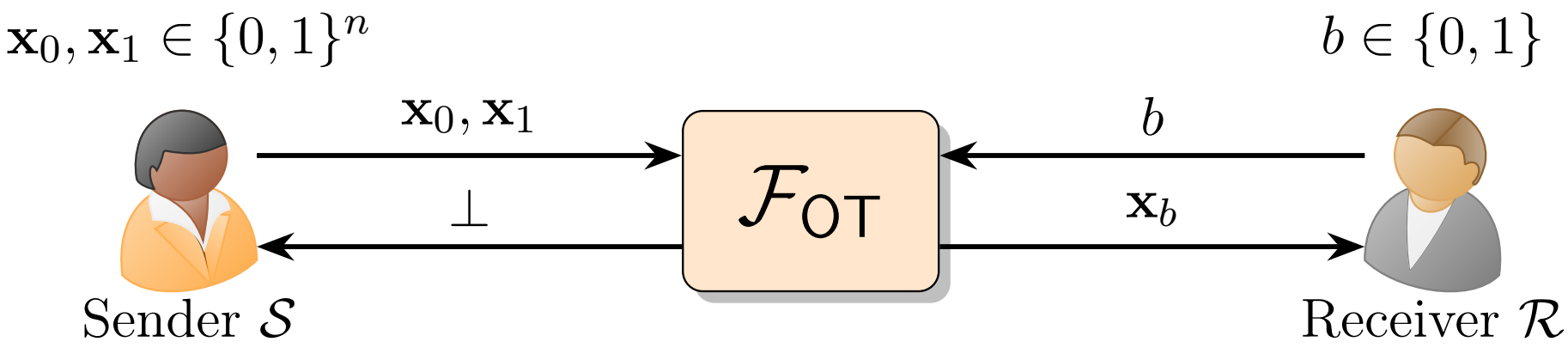


Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*.

Building Block: Oblivious Transfer

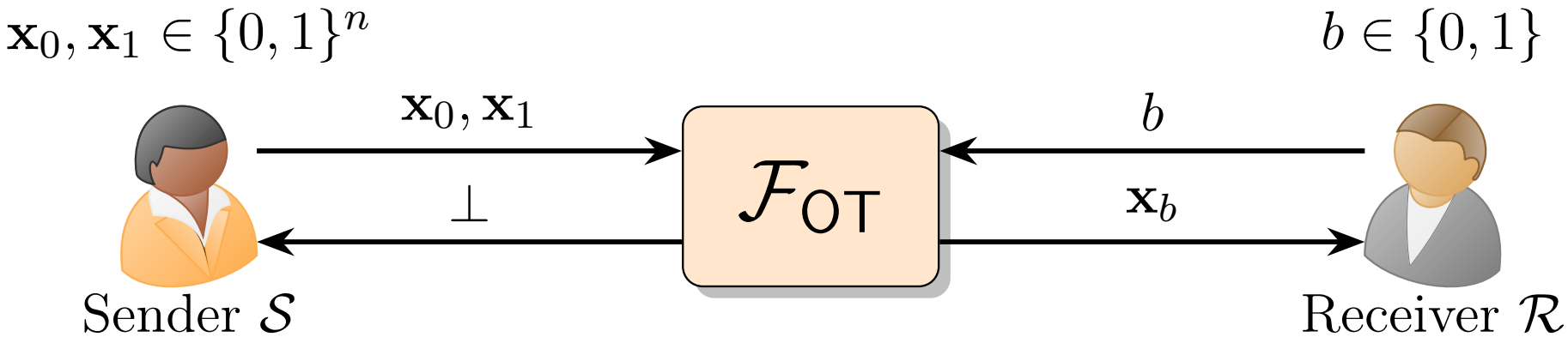


Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.

Building Block: Oblivious Transfer



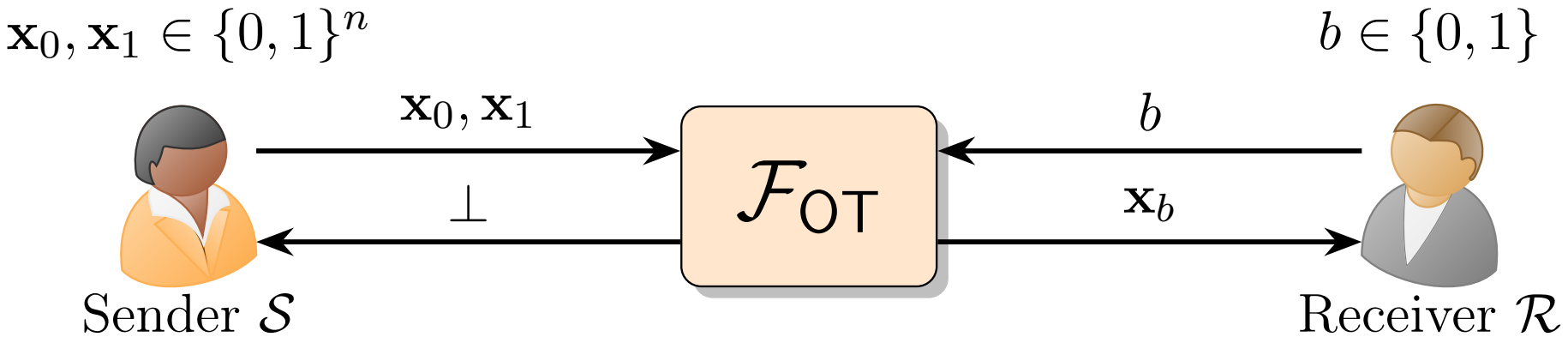
Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.

$$\mathcal{F}_{\mathsf{OT}}^{(1,k)}$$

Building Block: Oblivious Transfer



Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

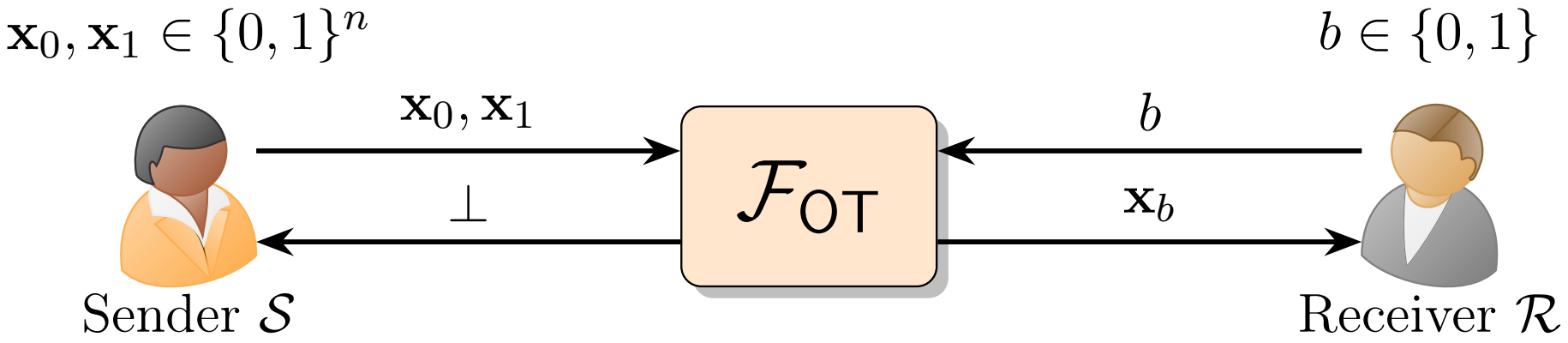
- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.

Sender $\mathcal{S}$

$\mathcal{F}_{\mathsf{OT}}^{(1,k)}$

Receiver $\mathcal{R}$

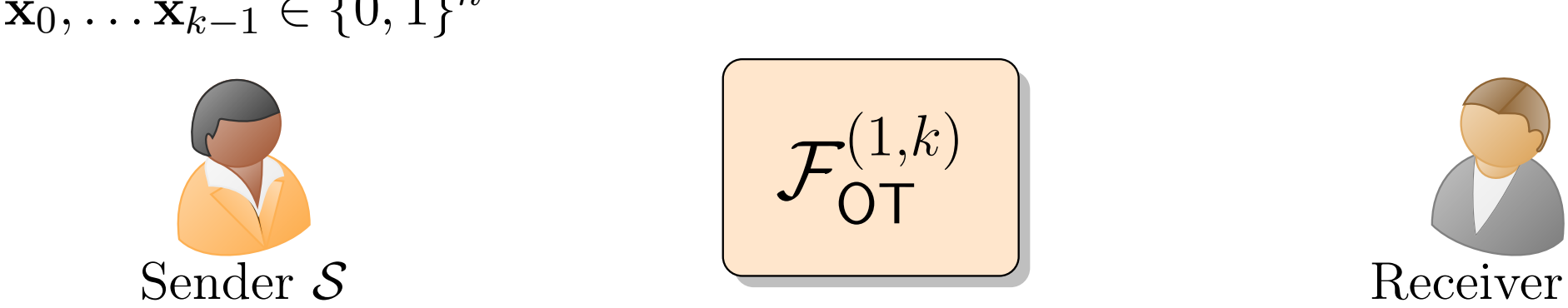
Building Block: Oblivious Transfer



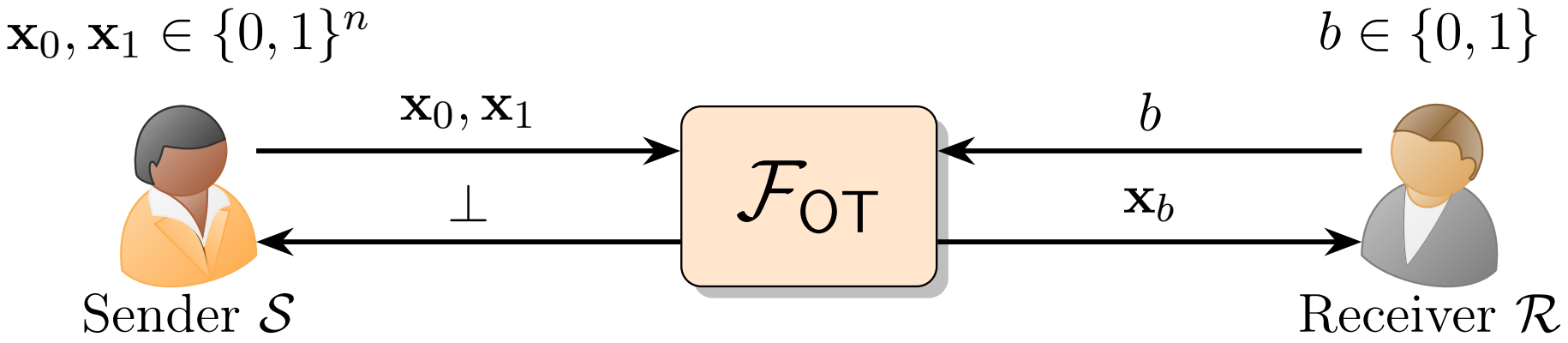
Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.



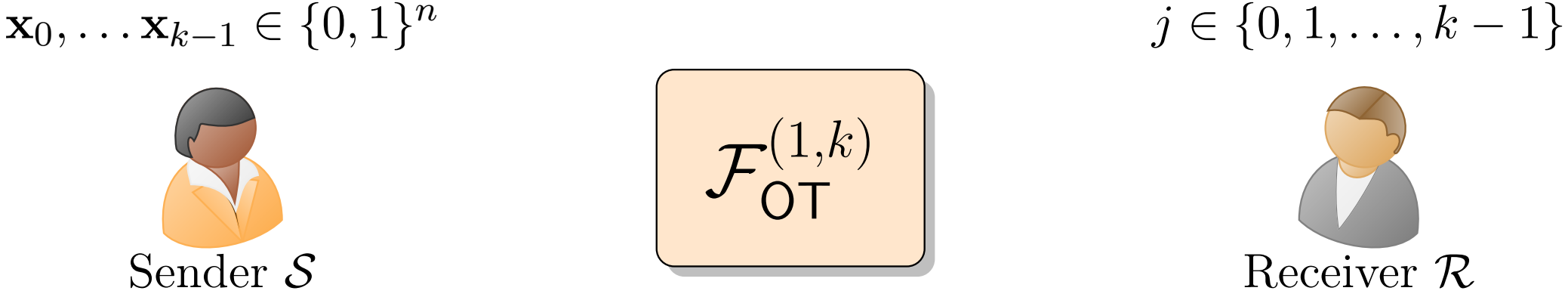
Building Block: Oblivious Transfer



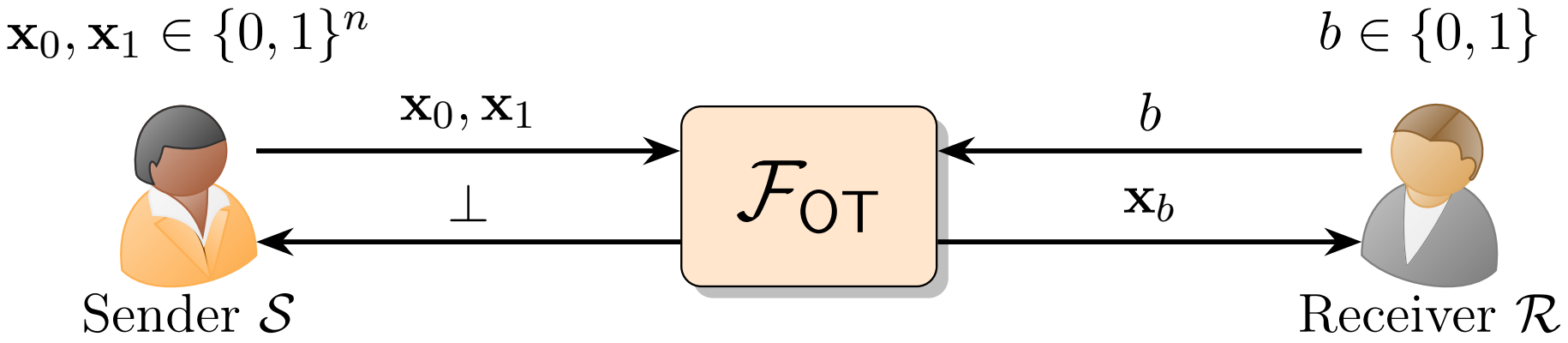
Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.



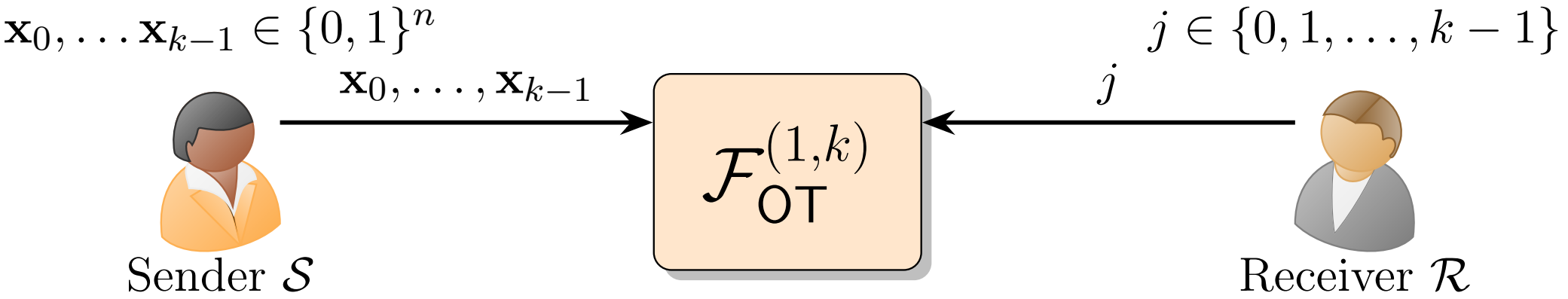
Building Block: Oblivious Transfer



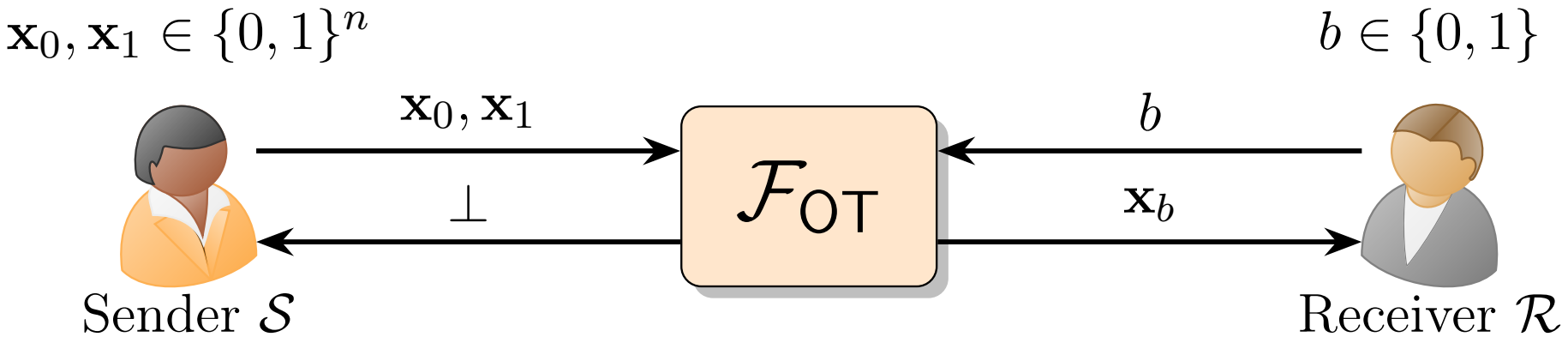
Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.



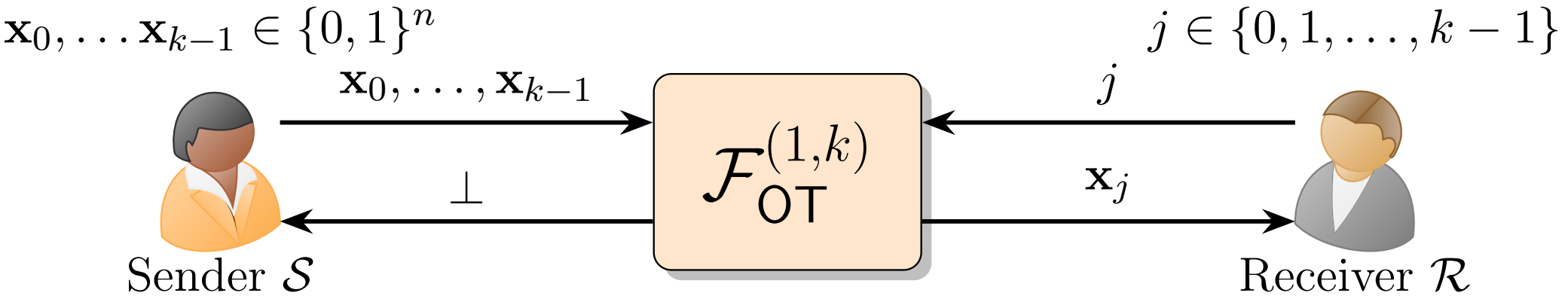
Building Block: Oblivious Transfer



Key Properties

- $\mathcal{R}$ learns *nothing* about $\mathbf{x}_{1-b}$.
- $\mathcal{S}$ learns *nothing* about b.

- Above is called *1-out-of-2 OT Functionality*. Can extend to *1-out-of-k OT*.



Aside: Facts about OT

Aside: Facts about OT

- OT is a very powerful cryptographic object.

Aside: Facts about OT

- OT is a very powerful cryptographic object.
- Kilian (1988) showed it is equivalent to MPC.

Aside: Facts about OT

- OT is a very powerful cryptographic object.
- Kilian (1988) showed it is equivalent to MPC.
- Impagliazzo and Rudich (1989) showed that a (black-box) reduction from OT to symmetric-key primitives (i.e., OWFs or PRFs) implies that $\mathbf{P} \neq \mathbf{NP}$.

Aside: Facts about OT

- OT is a very powerful cryptographic object.
- Kilian (1988) showed it is equivalent to MPC.
- Impagliazzo and Rudich (1989) showed that a (black-box) reduction from OT to symmetric-key primitives (i.e., OWFs or PRFs) implies that $\mathbf{P} \neq \mathbf{NP}$.
- We will use OT as a crucial building block for Yao's Garbled Circuit Protocol.

Goal of Yao's GC Protocol

Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.

Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.

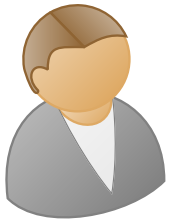
Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.

$$\mathcal{F}\colon X \times Y \to Z$$

Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.

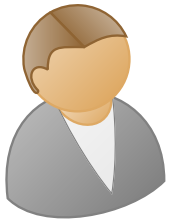


P_1

$$\mathcal{F}\colon X \times Y \to Z$$

Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



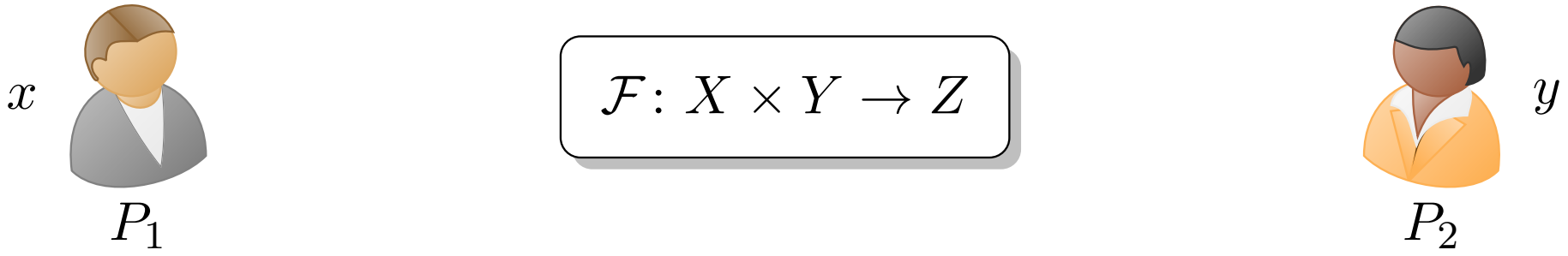
P_1

$$\mathcal{F}\colon X \times Y \to Z$$

P_2

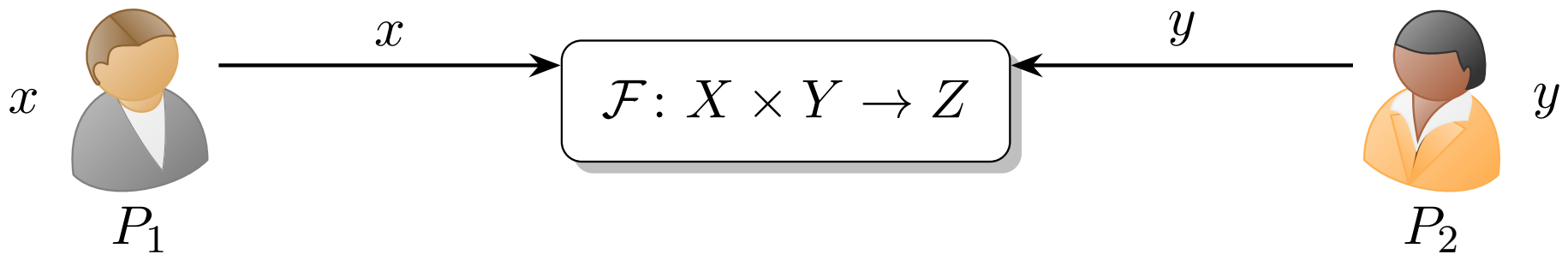
Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



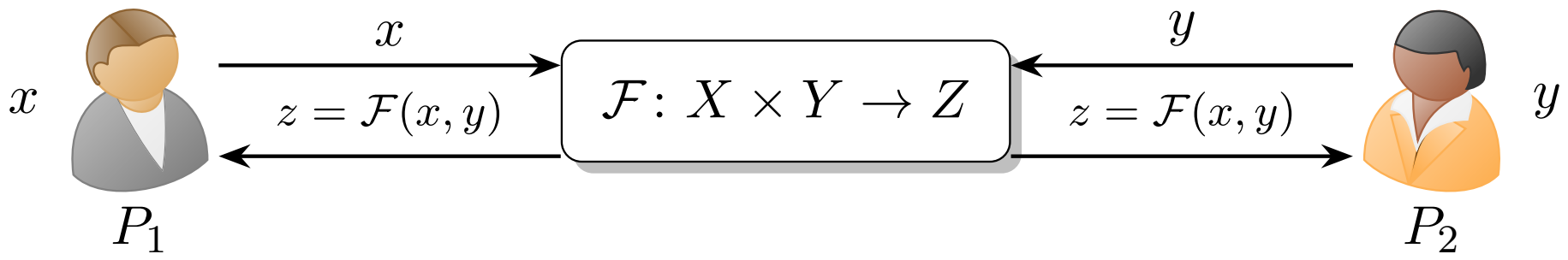
Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



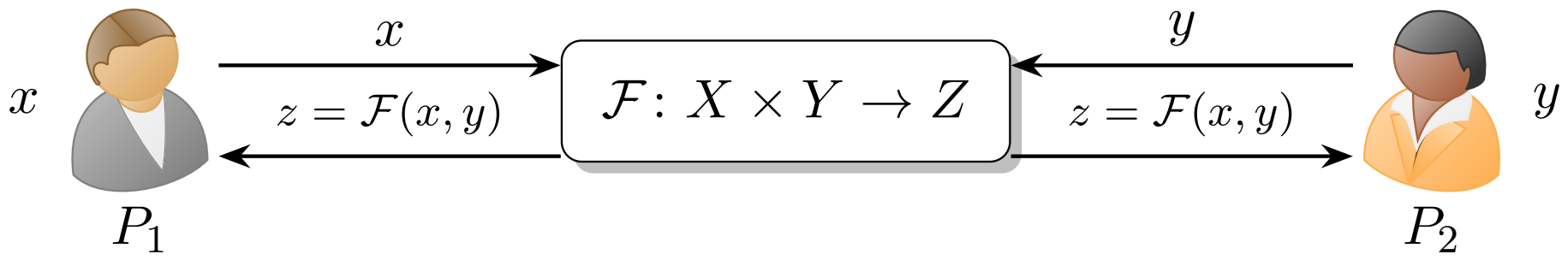
Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



Goal of Yao's GC Protocol

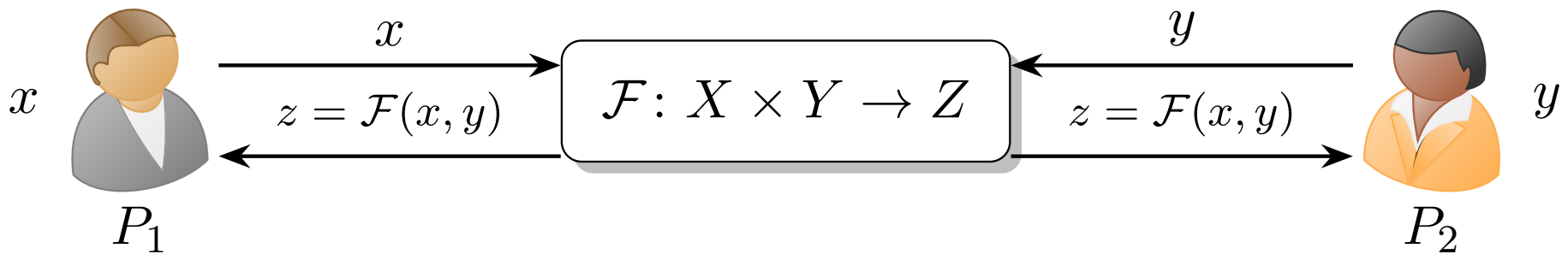
- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



- We only care about semi-honest security.

Goal of Yao's GC Protocol

- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.



- We only care about semi-honest security.
- Final protocol will be *constant round*.

Setup: Functions as (Look-up) Tables

$$\mathcal{F} \colon X \times Y \to Z$$

Setup: Functions as (Look-up) Tables

$\mathcal{F} \colon X \times Y \to Z$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

Setup: Functions as (Look-up) Tables

$\mathcal{F}\colon X \times Y \to Z$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

T

Setup: Functions as (Look-up) Tables

$\mathcal{F}\colon X \times Y \to Z$

$\mathcal{F}(x_i, y_j)$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

T

Setup: Functions as (Look-up) Tables

$\mathcal{F}\colon X \times Y \to Z$

$\mathcal{F}(x_i, y_j)$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

T

Setup: Functions as (Look-up) Tables

$\mathcal{F}\colon X \times Y \to Z$

$\mathcal{F}(x_i, y_j)$

$T_{x_i,y_j} = z_{i,j}$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

T

Setup: Functions as (Look-up) Tables

$\mathcal{F}\colon X \times Y \to Z$

$\mathcal{F}(x_i, y_j)$ → $T_{x_i,y_j} = z_{i,j}$

x	y	z
x_1	y_1	$z_{1,1}$
x_1	y_2	$z_{1,2}$
	$\vdots$	
x_1	$y_{\lvert Y\rvert}$	$z_{1,\lvert Y\rvert}$
x_2	y_1	$z_{2,1}$
	$\vdots$	
$x_{\lvert X\rvert}$	$y_{\lvert Y\rvert}$	$z_{\lvert X\rvert,\lvert Y\rvert}$

T

GC Intuition: Encrypting a Function Table

GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.

GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.

GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.

P_1

GC Intuition: Encrypting a Function Table

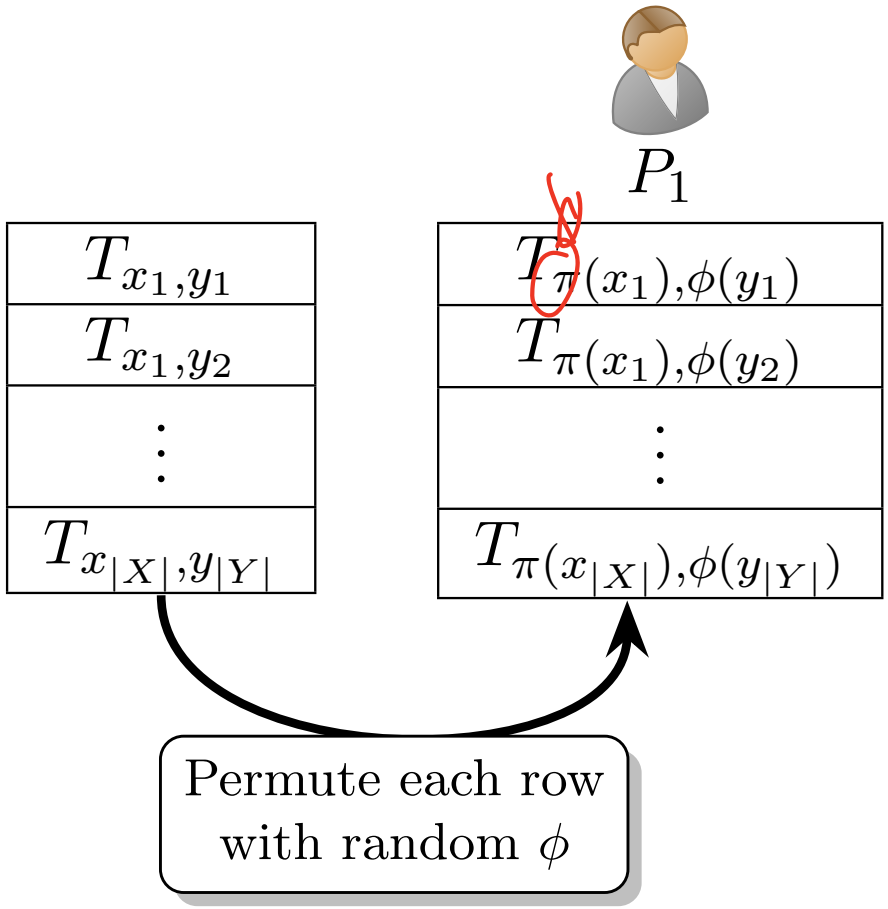
- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.

P_1

T_{x_1,y_1}
T_{x_1,y_2}
$\vdots$
$T_{x_{\lvert X\rvert},y_{\lvert Y\rvert}}$

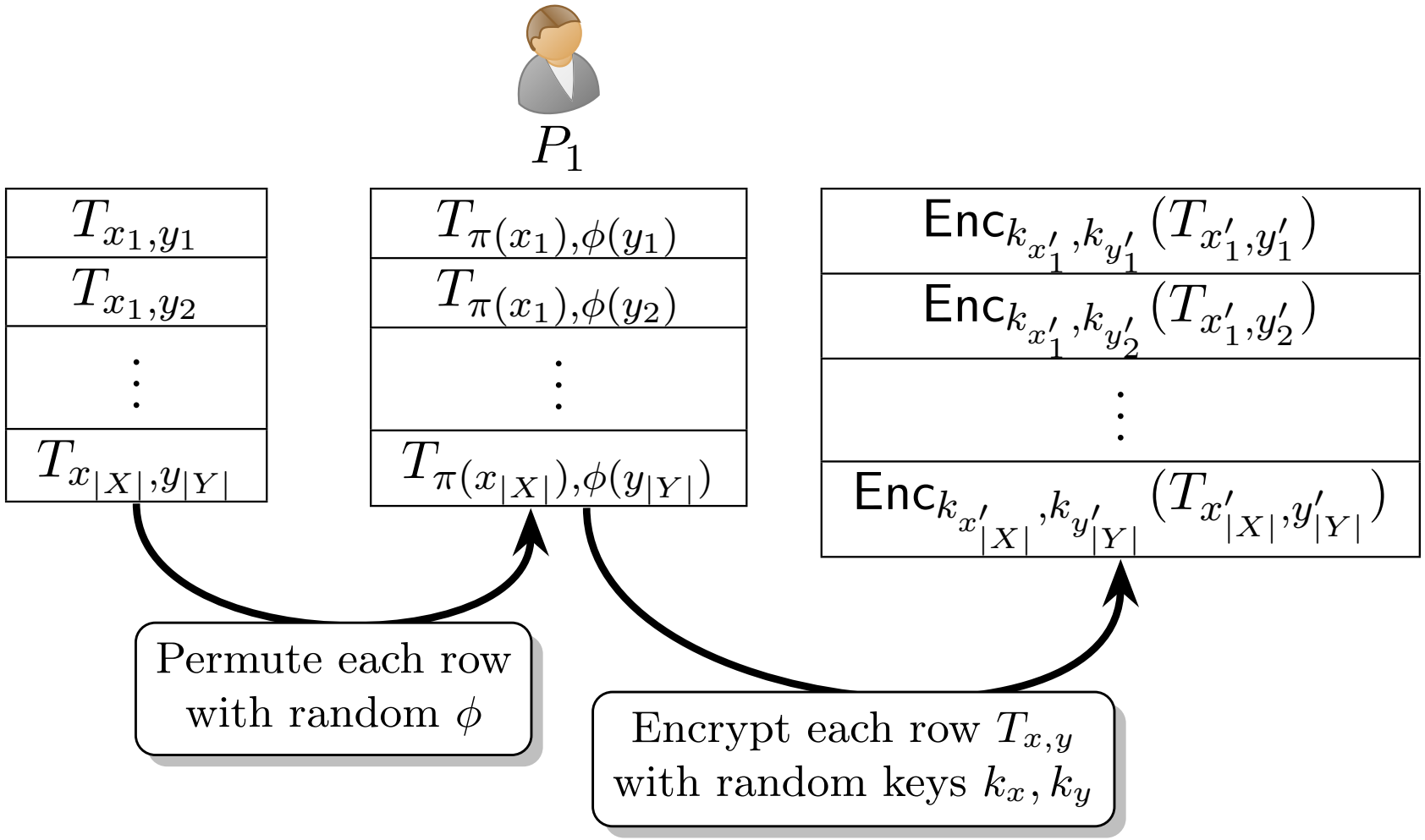
GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.



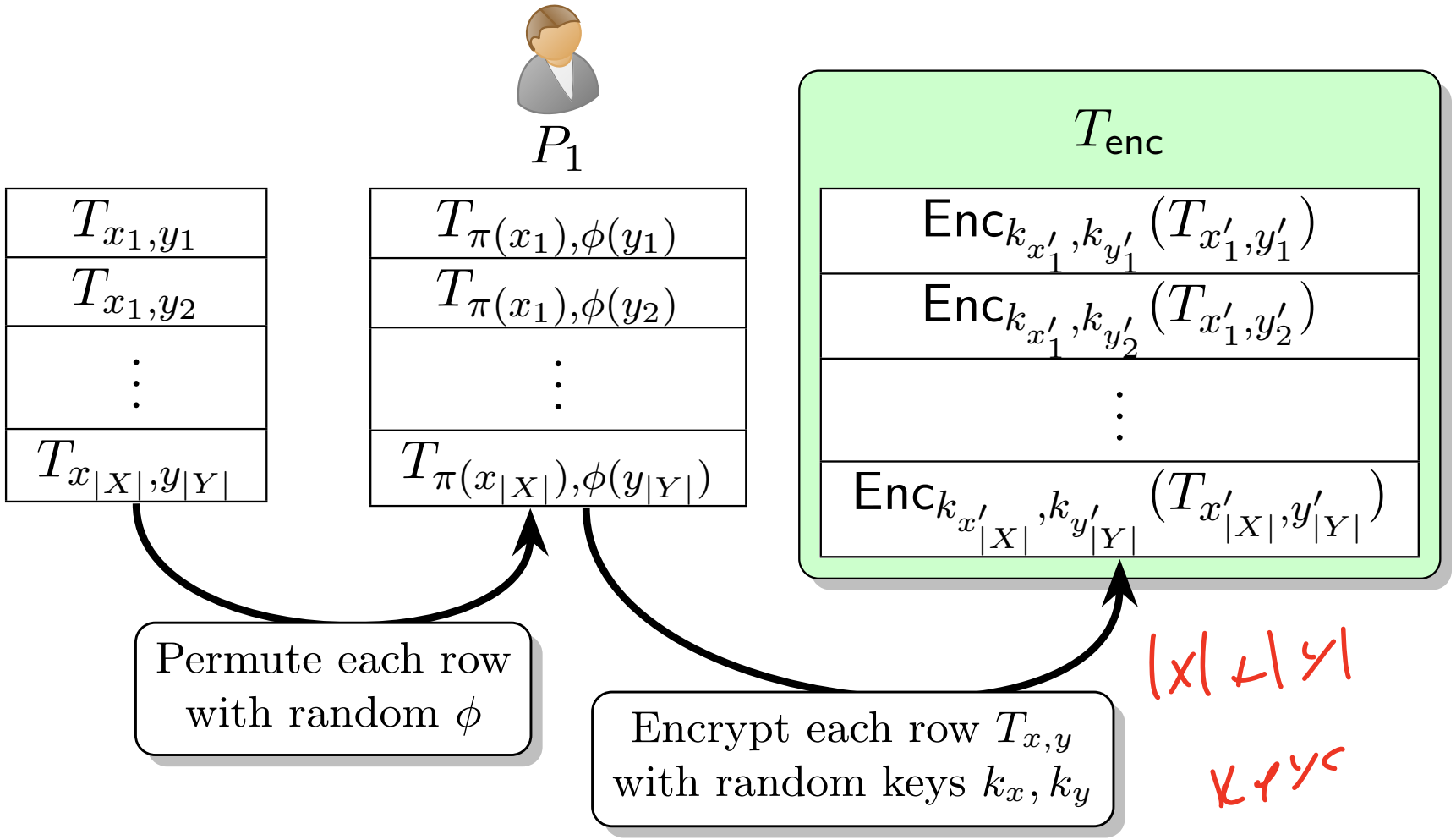
GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.



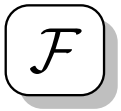
GC Intuition: Encrypting a Function Table

- Viewing $\mathcal{F}$ as a table gives a simple way to implement $\mathcal{F}$ securely.
- P_1's computation is as follows.



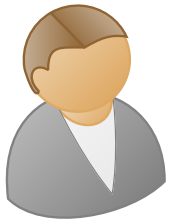
GC Intuition: Encrypting a Function Table

GC Intuition: Encrypting a Function Table



GC Intuition: Encrypting a Function Table

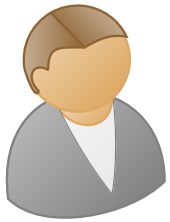
$\mathcal{F}$



P_1

GC Intuition: Encrypting a Function Table

$\mathcal{F}$



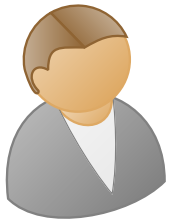
P_1

P_2

GC Intuition: Encrypting a Function Table

$\mathcal{F}$

x_i

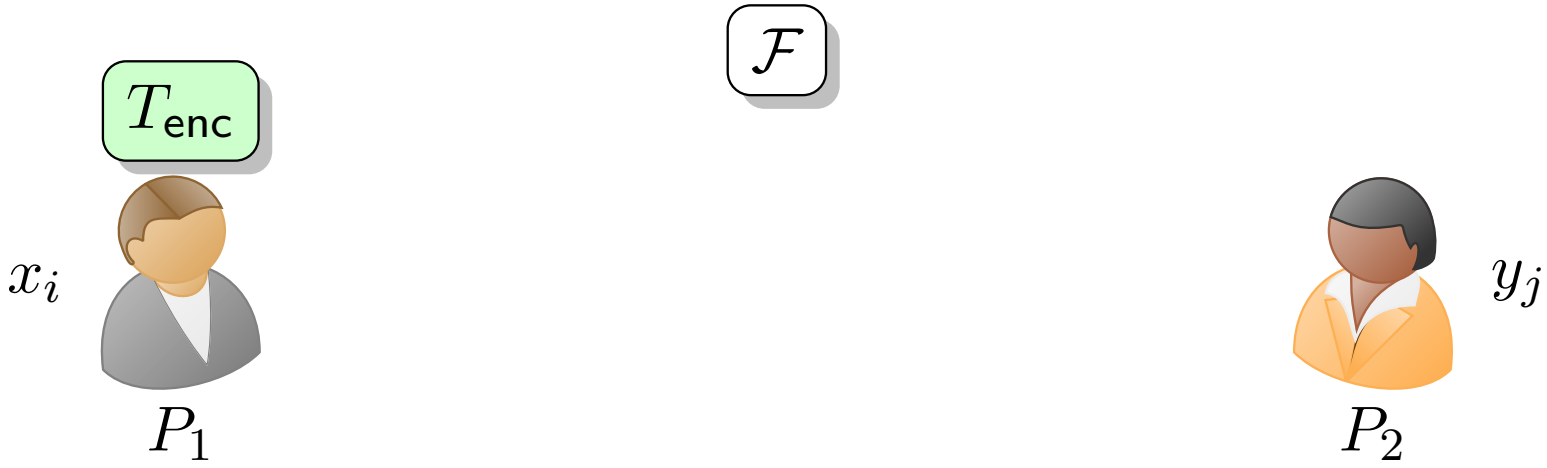


P_1

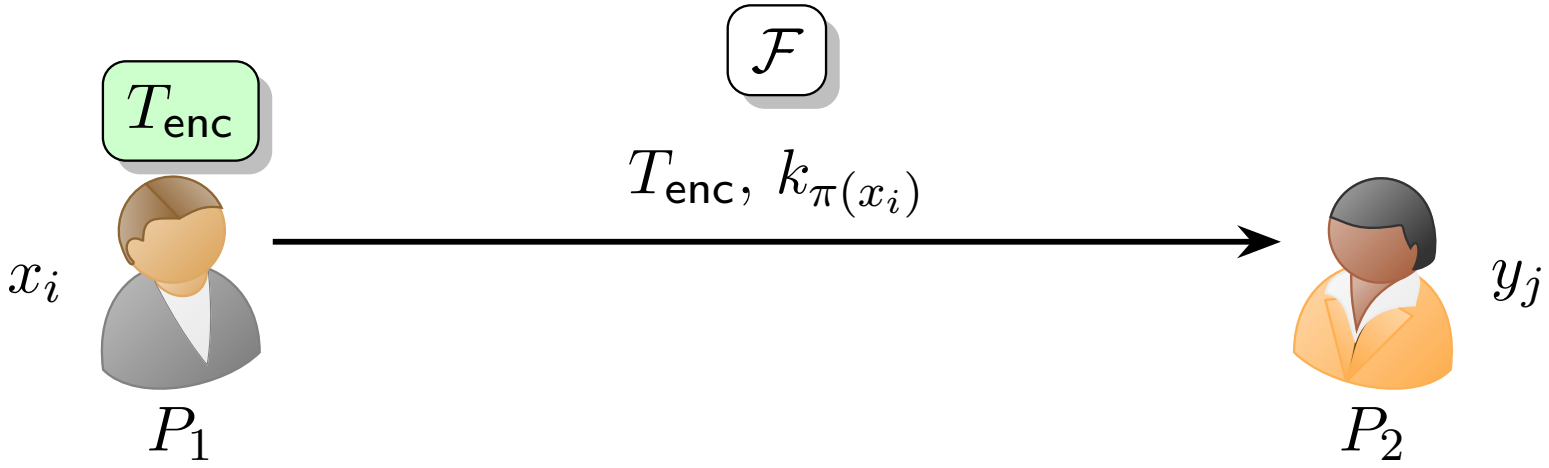
y_j

P_2

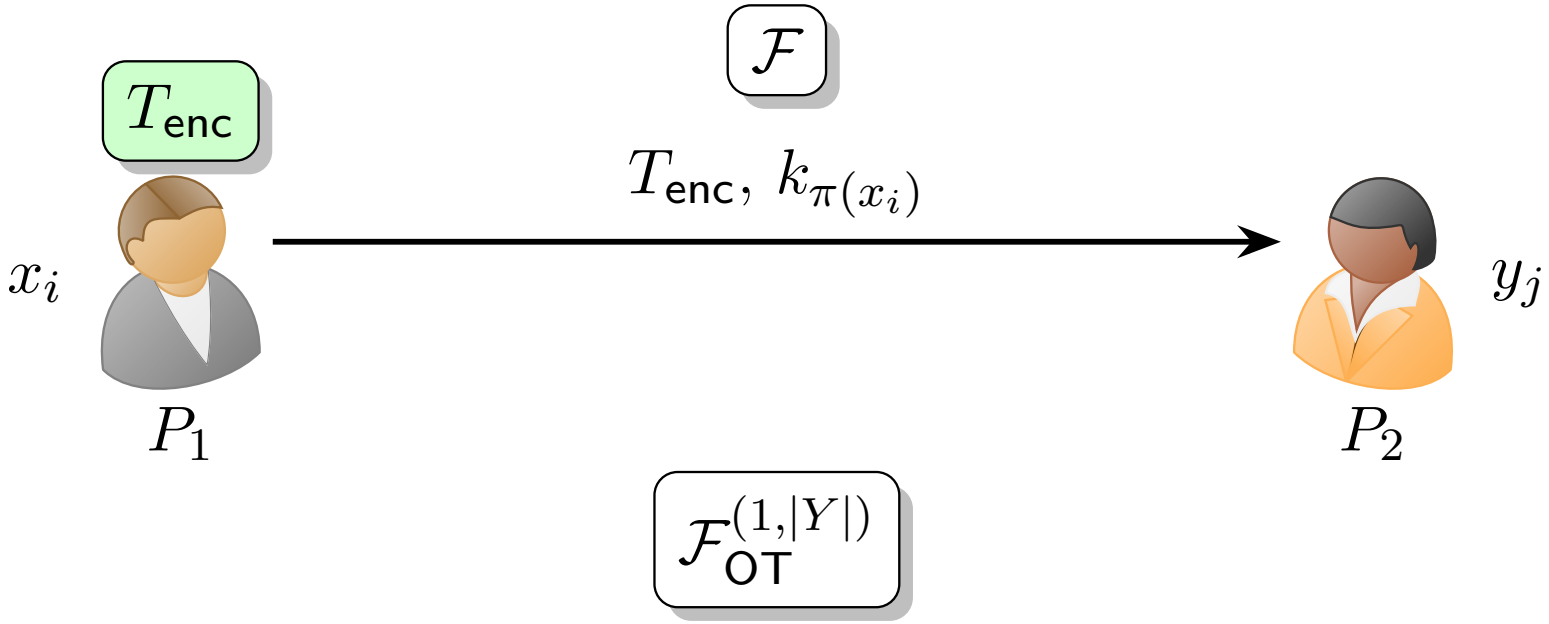
GC Intuition: Encrypting a Function Table



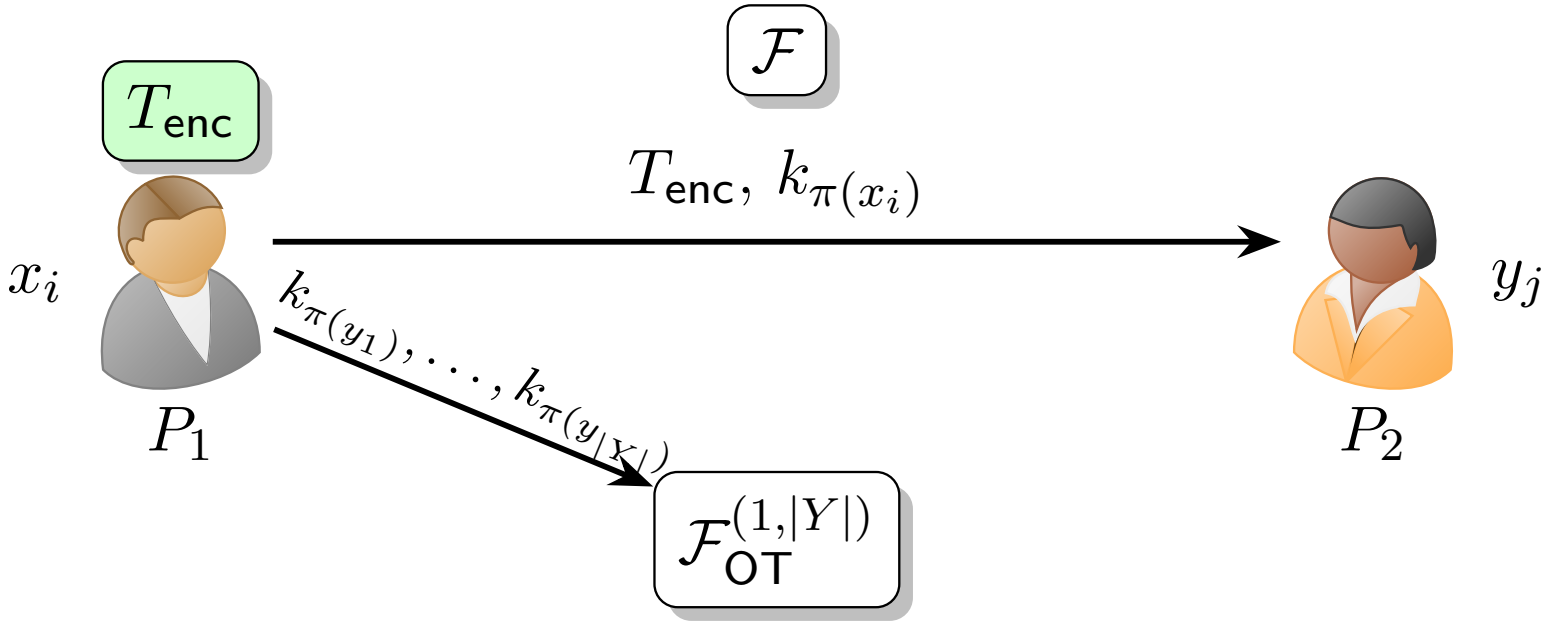
GC Intuition: Encrypting a Function Table



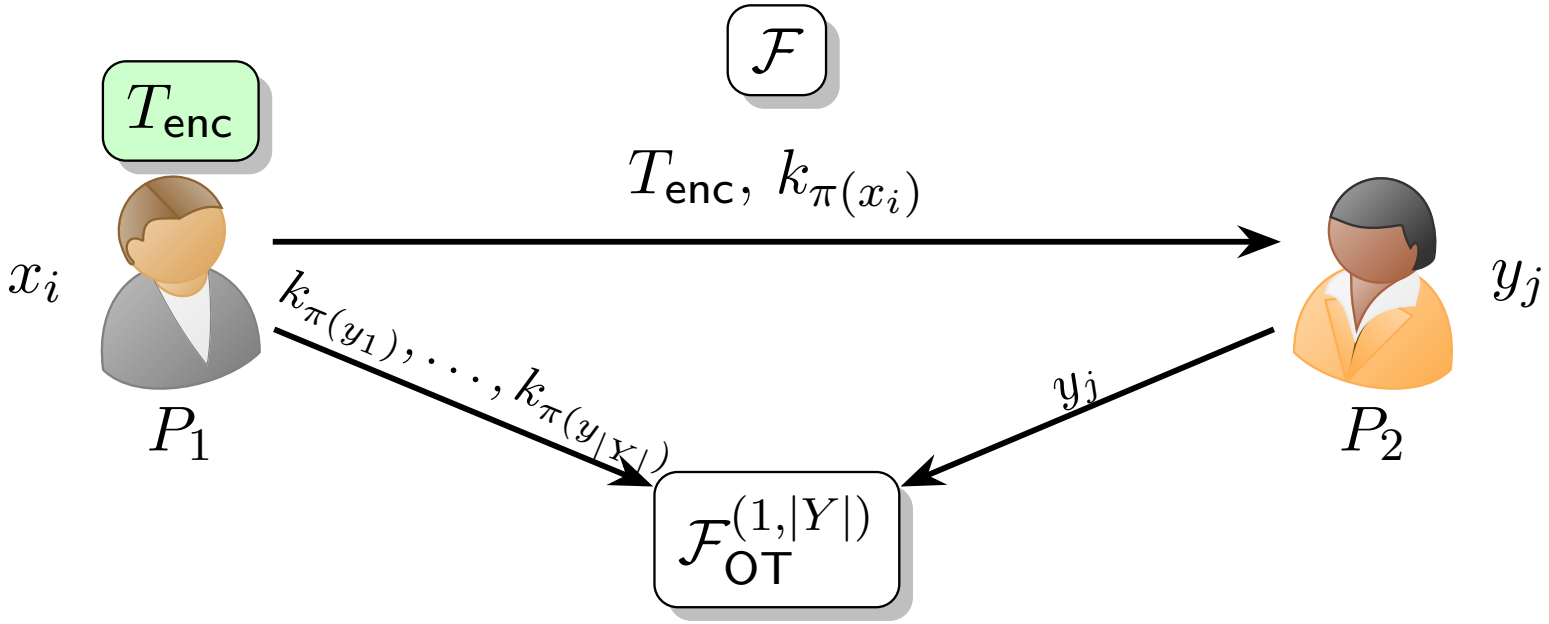
GC Intuition: Encrypting a Function Table



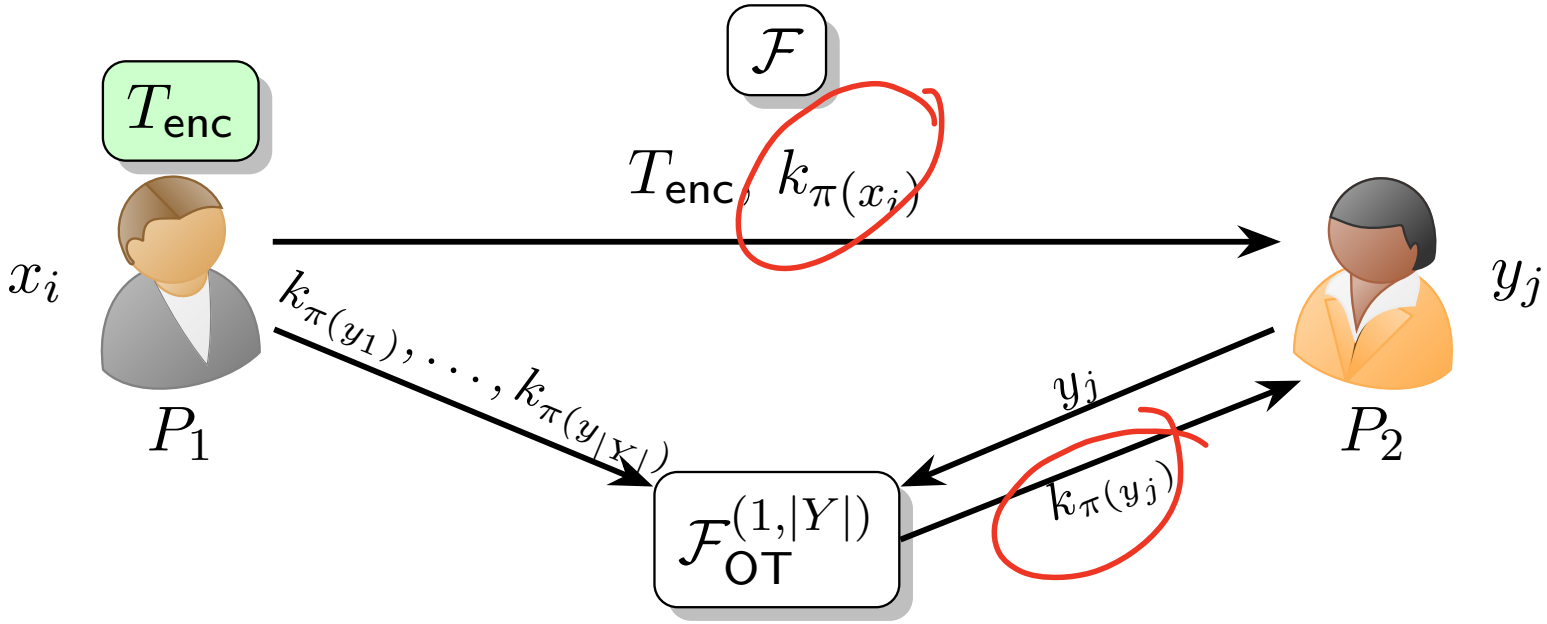
GC Intuition: Encrypting a Function Table



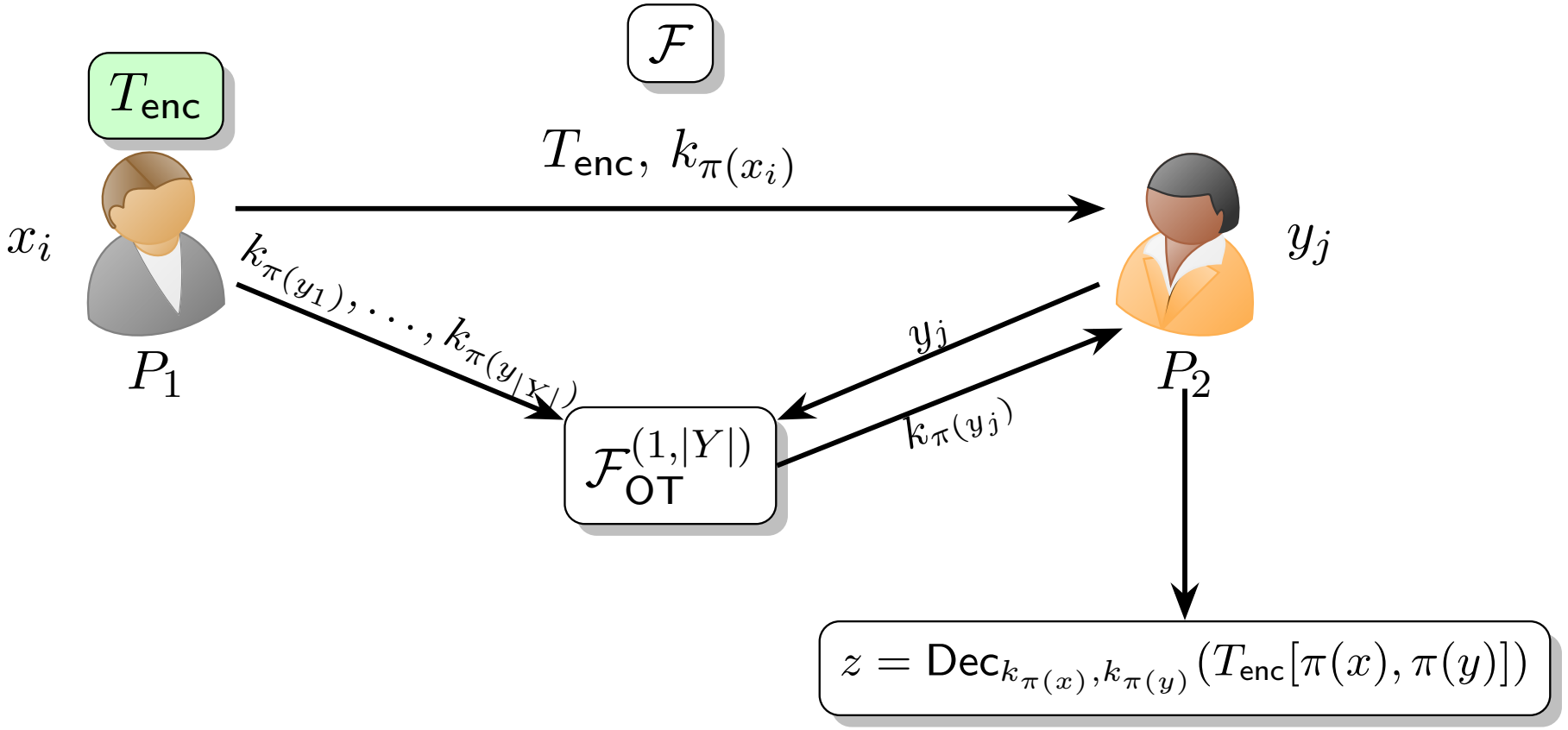
GC Intuition: Encrypting a Function Table



GC Intuition: Encrypting a Function Table



GC Intuition: Encrypting a Function Table



Setup: Which Row to Decrypt?

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.
  - Additionally compute $p_x \in \{0,1\}^{\log |X|}$ and $p_y \in \{0,1\}^{\log |Y|}$ such that $\phi(T)[p_x, p_y] = T_{x,y}$.

Setup: Which Row to Decrypt?

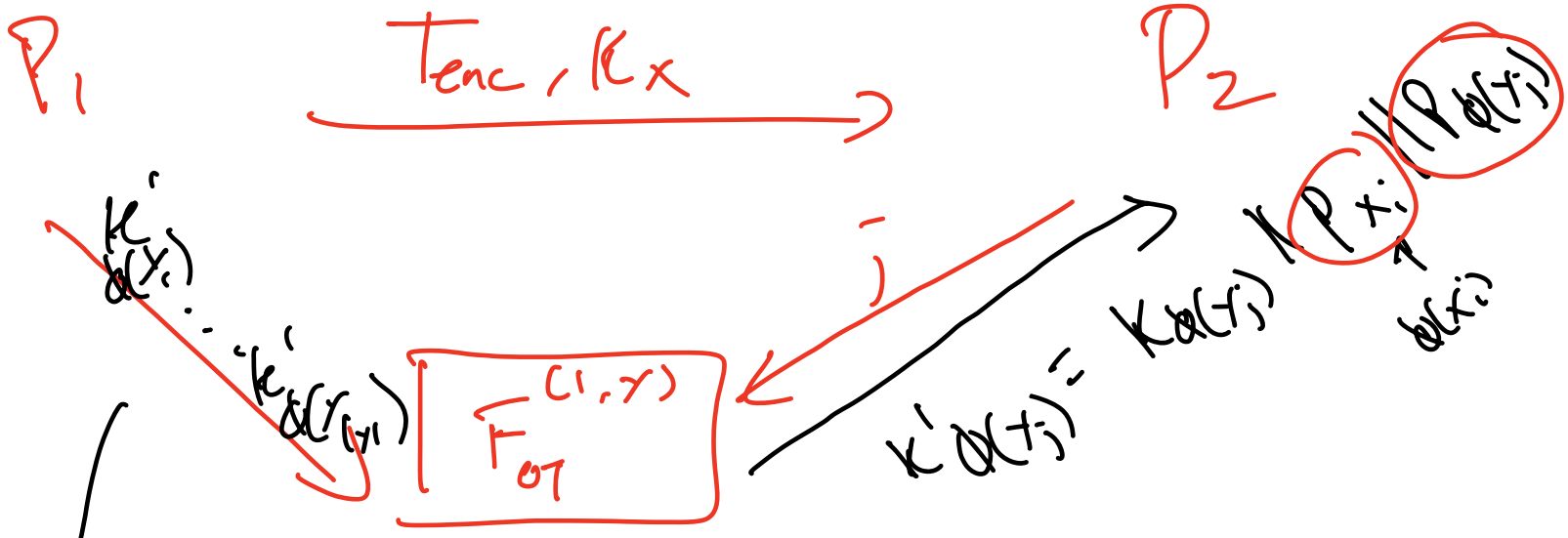
- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.
  - Additionally compute $p_x \in \{0,1\}^{\log |X|}$ and $p_y \in \{0,1\}^{\log |Y|}$ such that $\phi(T)[p_x, p_y] = T_{x,y}$.
  - Then, (p_x, p_y) are *pointers* (or together are a single pointer) into the permuted table.

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.
  - Additionally compute $p_x \in \{0,1\}^{\log |X|}$ and $p_y \in \{0,1\}^{\log |Y|}$ such that $\phi(T)[p_x, p_y] = T_{x,y}$.
  - Then, (p_x, p_y) are *pointers* (or together are a single pointer) into the permuted table.
  - Construct new keys $k'_x = k_x \| p_x$ and $k'_y = k_y \| p_y$.

Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.
  - Additionally compute $p_x \in \{0,1\}^{\log |X|}$ and $p_y \in \{0,1\}^{\log |Y|}$ such that $\phi(T)[p_x, p_y] = T_{x,y}$.
  - Then, (p_x, p_y) are *pointers* (or together are a single pointer) into the permuted table.
  - Construct new keys $k'_x = k_x \| p_x$ and $k'_y = k_y \| p_y$.
- Now, P_2 receives k'_x and obtains k'_y via the OT functionality.



$$\hookrightarrow K'_Y = K_Y \| P_X \| P_T$$

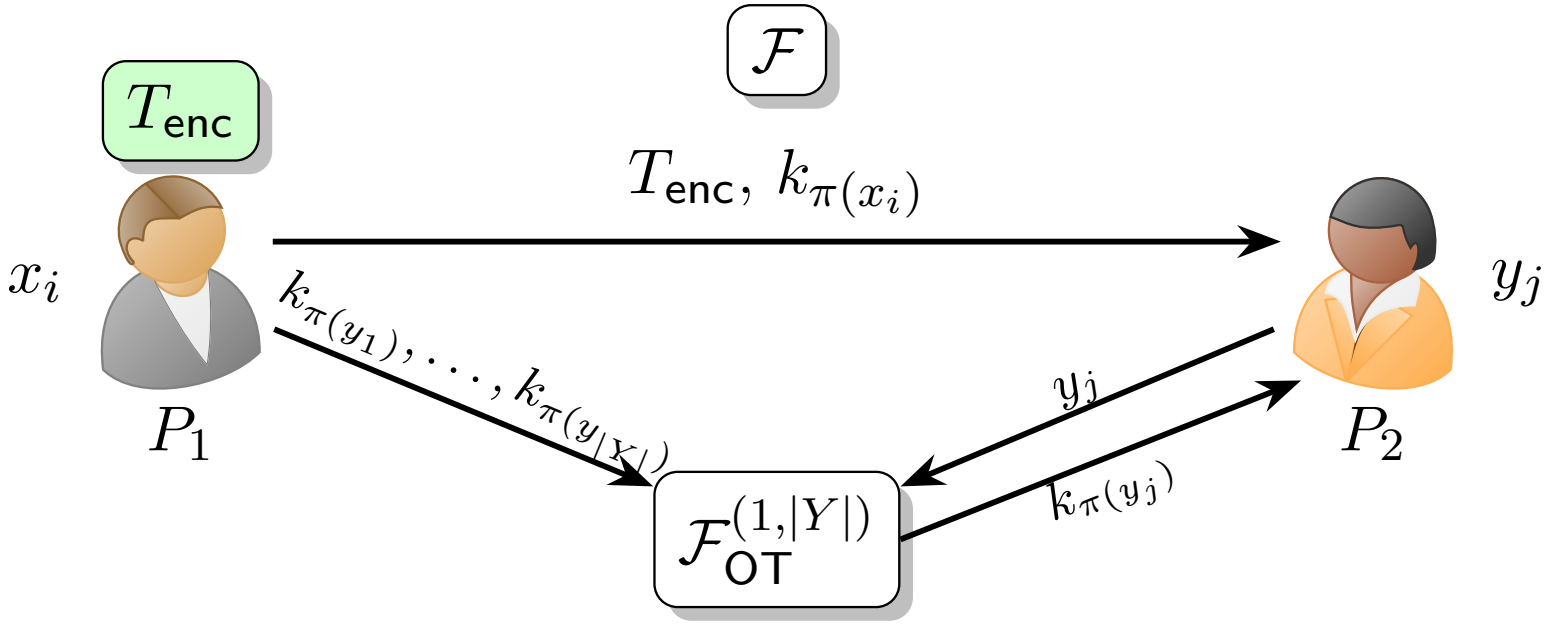
$$s.t. \quad Q(T)\left[P_X, P_T\right] = F(X,Y)$$

$$P_{X_i} = Q(i) \quad (\ i\ )$$

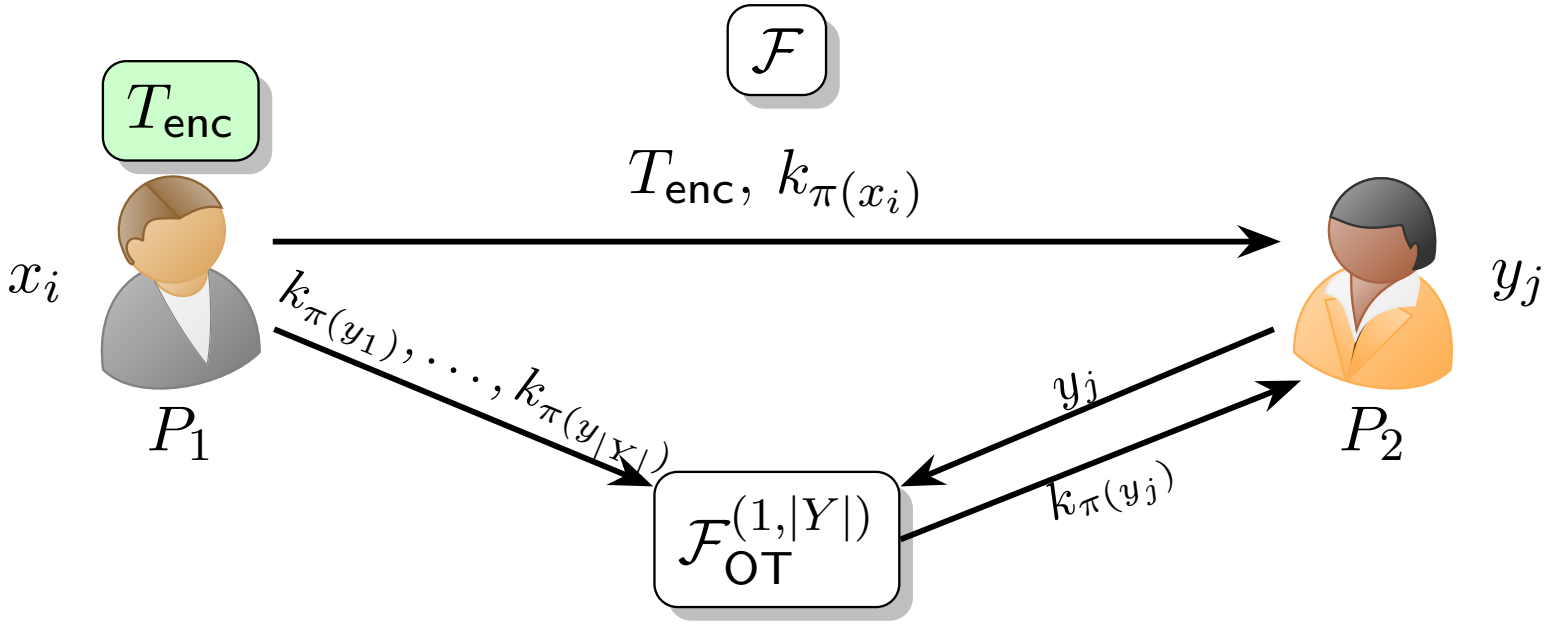
Setup: Which Row to Decrypt?

- In above picture, how does P_2 know which row to decrypt?
  - T_{enc} is the encryption of T *after* randomly permuting the rows.
- On average, P_2 has to try to perform $|X| \cdot |Y|/2$ decryptions!
  - Example: $X = Y = \{0,1\}^n$, so this is exponential time!
- Fix: *point-and-permute* (Beaver et al., 1990).
  - For every $x \in X$ and $y \in Y$, P_1 samples each encryption key k_x, k_y as normal.
  - Additionally compute $p_x \in \{0,1\}^{\log |X|}$ and $p_y \in \{0,1\}^{\log |Y|}$ such that $\phi(T)[p_x, p_y] = T_{x,y}$.
  - Then, (p_x, p_y) are *pointers* (or together are a single pointer) into the permuted table.
  - Construct new keys $k'_x = k_x \| p_x$ and $k'_y = k_y \| p_y$.
- Now, P_2 receives k'_x and obtains k'_y via the OT functionality.
  - Knows exactly which row in T_{enc} to decrypt now!

Managing Look-up Table Size

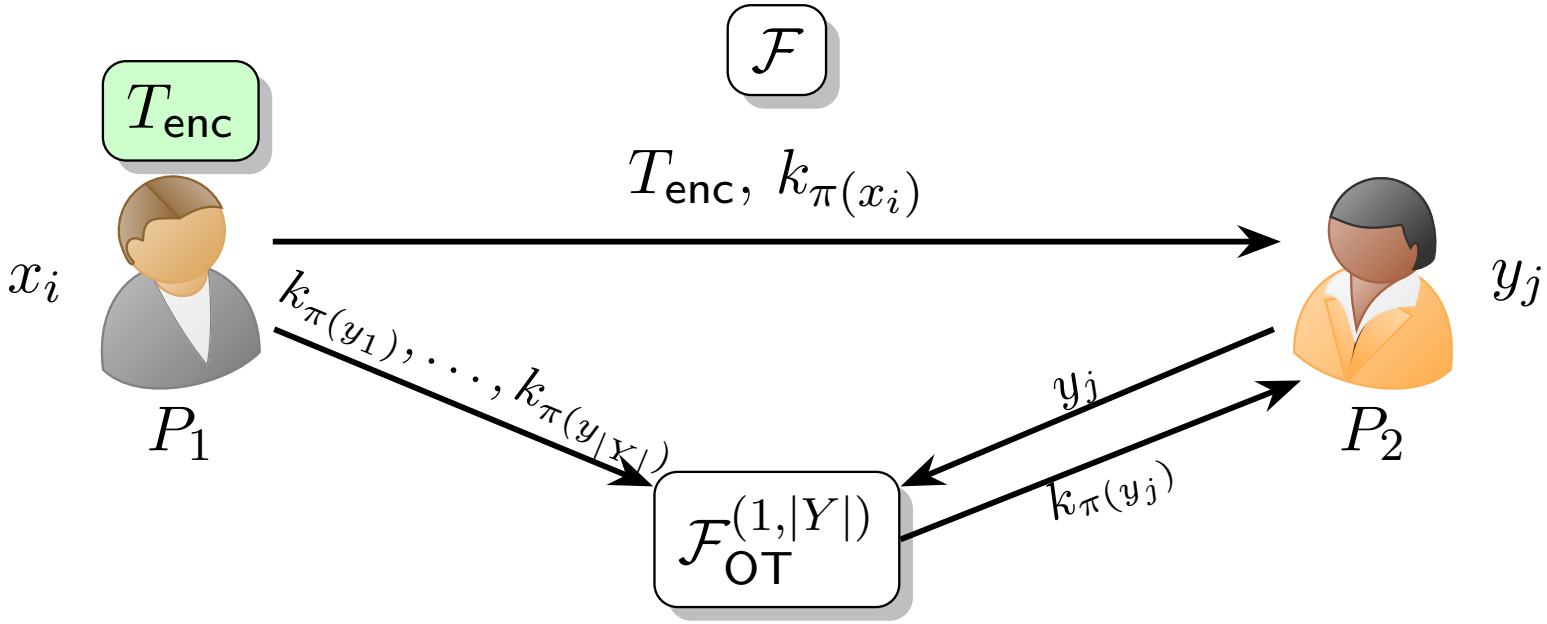


Managing Look-up Table Size



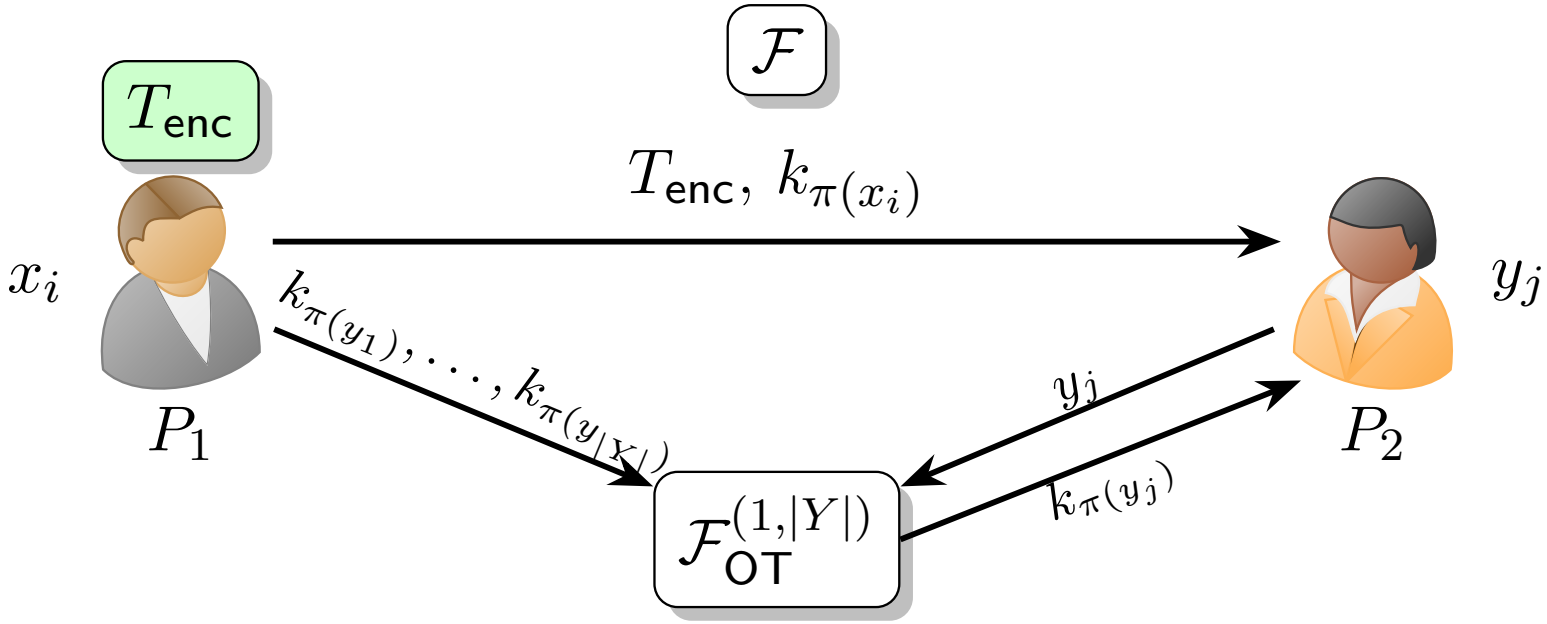
- Pitfalls of this solution?

Managing Look-up Table Size



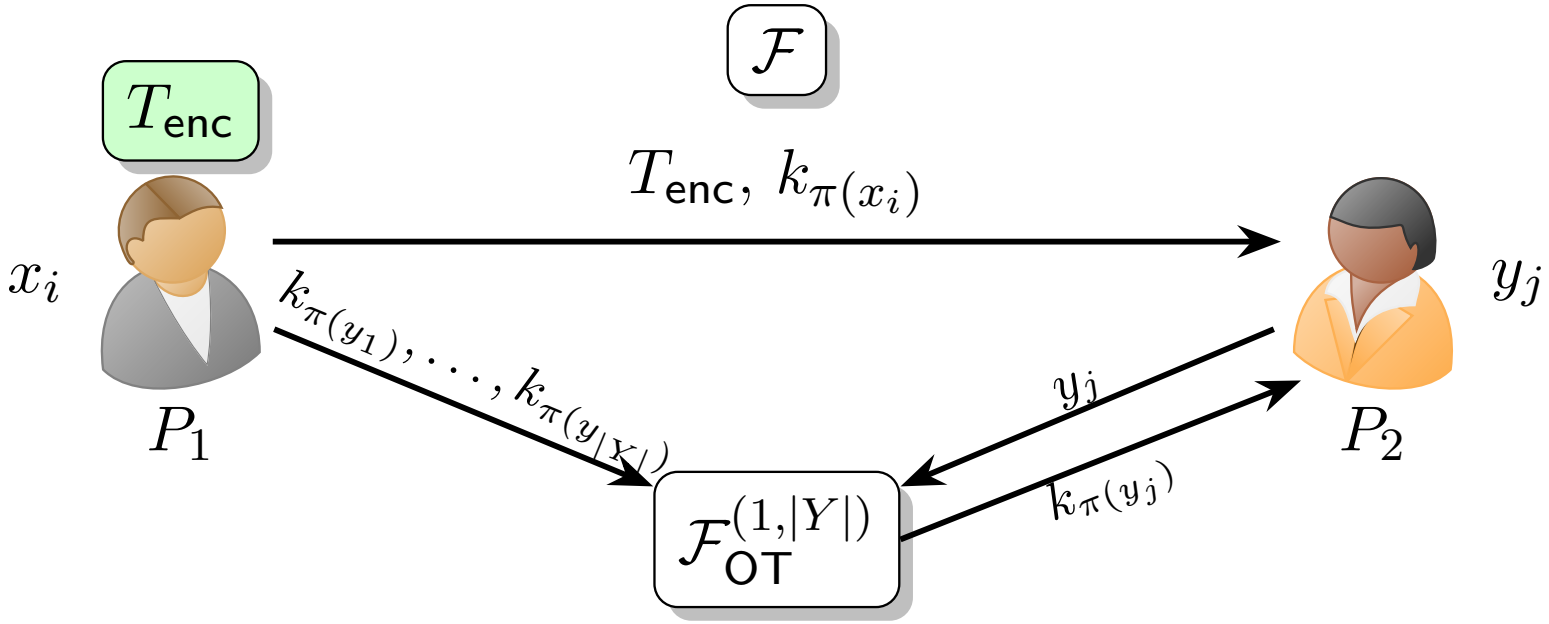
- Pitfalls of this solution?
  - $|T_{\mathsf{enc}}|$ is huge!

Managing Look-up Table Size



- Pitfalls of this solution?
  - $|T_{\mathsf{enc}}|$ is huge!
  - $|T_{\mathsf{enc}}| = \Omega(|X| \cdot |Y|)$; $\Omega(2^{2n})$ for $X = Y = \{0,1\}^n$.

Managing Look-up Table Size



- Pitfalls of this solution?
  - $|T_{\mathsf{enc}}|$ is huge!
  - $|T_{\mathsf{enc}}| = \Omega(|X| \cdot |Y|)$; $\Omega(2^{2n})$ for $X = Y = \{0,1\}^n$.

- If $\mathcal{F}$ has a *small* (e.g., constant-sized) table, then this would be much better.

Managing Look-up Table Size

- Idea:

Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).

Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

Managing Look-up Table Size

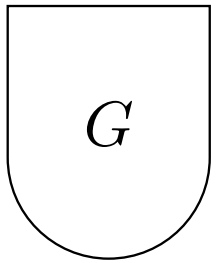
- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

- Boolean circuit gates have *4 table entries* (or 2 for NOT).

Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

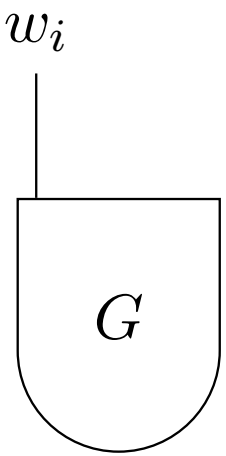
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

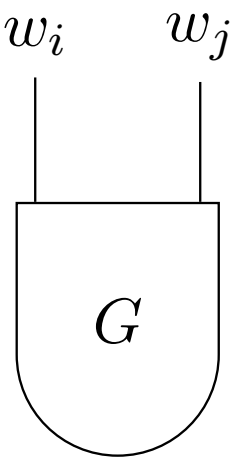
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

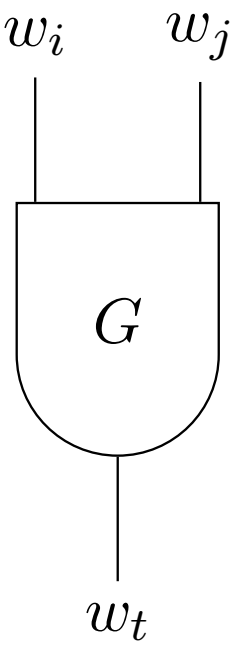
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

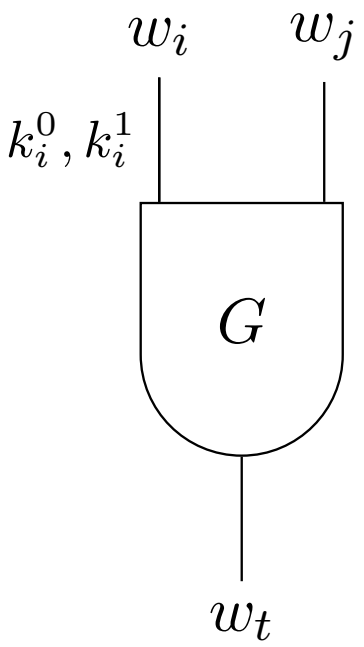
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

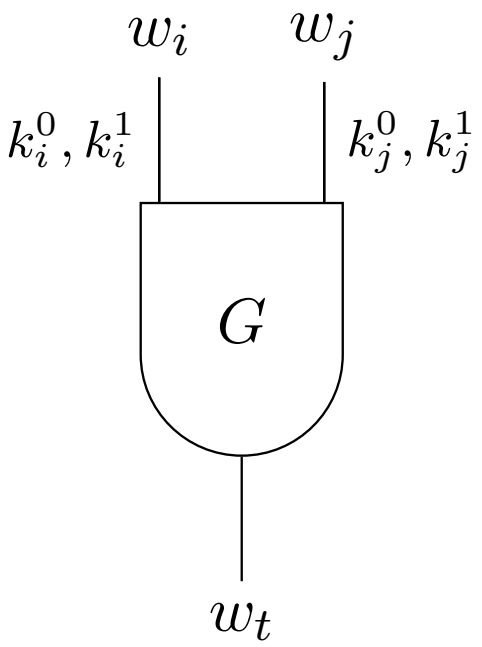
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

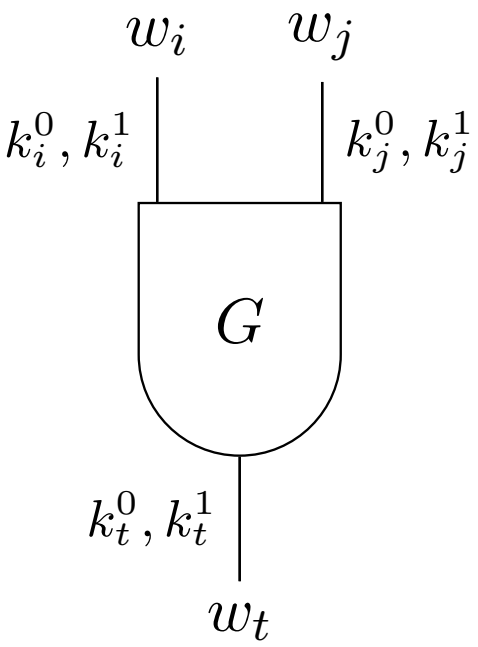
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

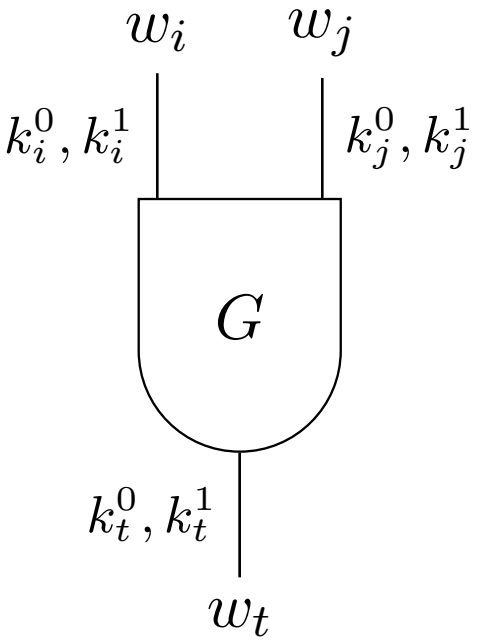
- Boolean circuit gates have *4 table entries* (or 2 for NOT).



Managing Look-up Table Size

- Idea:
  - Represent $\mathcal{F}$ as a *Boolean* circuit (AND, OR, XOR, NOT, etc.).
  - Apply the previous function table encryption idea to *each gate*.

- Boolean circuit gates have *4 table entries* (or 2 for NOT).



$$T_G = \begin{bmatrix} \mathsf{Enc}_{k_i^0,k_j^0}(k_t^{G(0,0)}) \\ \mathsf{Enc}_{k_i^0,k_j^1}(k_t^{G(0,1)}) \\ \mathsf{Enc}_{k_i^1,k_j^0}(k_t^{G(1,0)}) \\ \mathsf{Enc}_{k_i^1,k_j^1}(k_t^{G(1,1)}) \end{bmatrix}$$

AND

$$\begin{bmatrix} \mathsf{Enc}(k_t^0) \\ \mathsf{Enc}(k_t^0) \\ \mathsf{Enc}(k_t^0) \\ \mathsf{Enc}(k_t^1) \end{bmatrix}$$

$\updownarrow$

$Q_G(T_G)$

Yao's GC Protocol

Yao's GC Protocol

- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.

Yao's GC Protocol

- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.
- P_1 does the following.

Yao's GC Protocol

- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.
- P_1 does the following.
  - For each gate G in topological order, let w_i and w_j be the input wires and w_t be the output wire.

Yao's GC Protocol

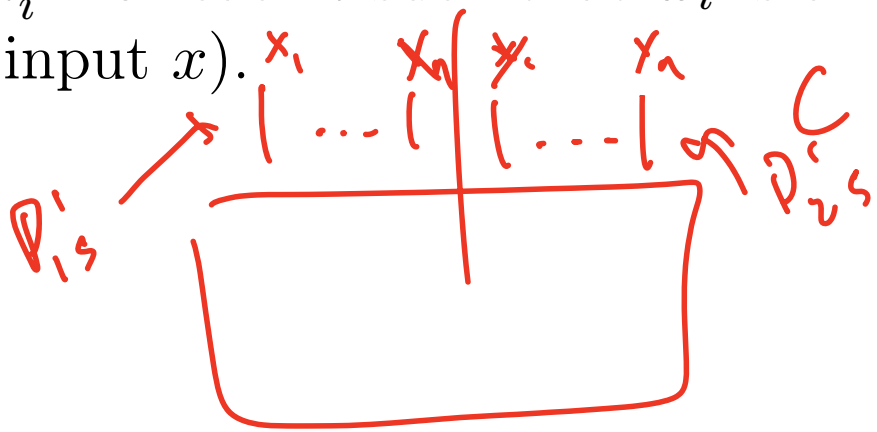
- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.
- P_1 does the following.
  - For each gate G in topological order, let w_i and w_j be the input wires and w_t be the output wire.
  - Construct T_G according to the previous slide by sampling two keys per wire i, j, t (one for each of 0 and 1), along with permuting the table each time.

Yao's GC Protocol

- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.
- P_1 does the following.
  - For each gate G in topological order, let w_i and w_j be the input wires and w_t be the output wire.
  - Construct T_G according to the previous slide by sampling two keys per wire i, j, t (one for each of 0 and 1), along with permuting the table each time.
  - Also apply the point-and-permute technique to all keys sampled.

Yao's GC Protocol

- Let $\mathcal{C}$ be the Boolean circuit representing $\mathcal{F}$.
- P_1 does the following.
  - For each gate G in topological order, let w_i and w_j be the input wires and w_t be the output wire.
  - Construct T_G according to the previous slide by sampling two keys per wire i, j, t (one for each of 0 and 1), along with permuting the table each time.
  - Also apply the point-and-permute technique to all keys sampled.
- P_1 then sends each (encrypted and permuted) T_G to P_2, along with key $k_i^{x_i}$ for each i such that w_i is an *input wire* (corresponding to P_1's input x).



$|C| = s$ $\quad \#T = s$

$C : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$

Yao's GC Protocol

- P_1 and P_2 then perform a 1-out-of-2 OT for each of P_2's input wires to obtain the appropriate keys $k_j^{y_j}$.

Yao's GC Protocol

- P_1 and P_2 then perform a 1-out-of-2 OT for each of P_2's input wires to obtain the appropriate keys $k_j^{y_j}$.

- Given $\mathcal{C}$ and T_G for every gate $G \in \mathcal{C}$, P_2 then evaluates the circuit $\mathcal{C}$ using each T_G (the garbled gates).

Yao's GC Protocol

- P_1 and P_2 then perform a 1-out-of-2 OT for each of P_2's input wires to obtain the appropriate keys $k_j^{y_j}$.

- Given $\mathcal{C}$ and T_G for every gate $G \in \mathcal{C}$, P_2 then evaluates the circuit $\mathcal{C}$ using each T_G (the garbled gates).

- At the end, P_2 obtains one key k_ℓ^b for each output wire w_ℓ, which P_2 sends to P_1, and P_1 responds with b_ℓ (the correct bit).

Yao's GC Protocol

- P_1 and P_2 then perform a 1-out-of-2 OT for each of P_2's input wires to obtain the appropriate keys $k_j^{y_j}$.

- Given $\mathcal{C}$ and T_G for every gate $G \in \mathcal{C}$, P_2 then evaluates the circuit $\mathcal{C}$ using each T_G (the garbled gates).

- At the end, P_2 obtains one key k_ℓ^b for each output wire w_ℓ, which P_2 sends to P_1, and P_1 responds with b_ℓ (the correct bit).
  - To save one round of communication, P_1 can instead send the decoding tables for each output wire w_ℓ (e.g., send $(0, k_\ell^0), (1, k_\ell^1)$).

Yao's GC Protocol: Complexity

Yao's GC Protocol: Complexity

- Suppose $|\mathcal{C}| = s$.

Yao's GC Protocol: Complexity

- Suppose $|\mathcal{C}| = s$.

- P_1 sends s garbled tables, each of size at most 4, n keys $k_i^{x_i}$ corresponding to P_1's input $x \in \{0,1\}^n$, and s_{out} pairs of keys k_ℓ^0, k_ℓ^1 for each output wire w_ℓ.

$O(\lambda(s+n))$ bits

Yao's GC Protocol: Complexity

- Suppose $|\mathcal{C}| = s$.
- P_1 sends s garbled tables, each of size at most 4, n keys $k_i^{x_i}$ corresponding to P_1's input $x \in \{0,1\}^n$, and s_{out} pairs of keys k_ℓ^0, k_ℓ^1 for each output wire w_ℓ.
- P_1 and P_2 make n calls to $\mathcal{F}_{\mathsf{OT}}$, giving n more keys to P_2.

Yao's GC Protocol: Complexity

- Suppose $|\mathcal{C}| = s$.
- P_1 sends s garbled tables, each of size at most 4, n keys $k_i^{x_i}$ corresponding to P_1's input $x \in \{0,1\}^n$, and s_{out} pairs of keys k_ℓ^0, k_ℓ^1 for each output wire w_ℓ.
- P_1 and P_2 make n calls to $\mathcal{F}_{\mathsf{OT}}$, giving n more keys to P_2.
- P_2 computes s decryptions by evaluating the circuit $\mathcal{C}$.

Yao's GC Protocol: Complexity

- Suppose $|\mathcal{C}| = s$.
- P_1 sends s garbled tables, each of size at most 4, n keys $k_i^{x_i}$ corresponding to P_1's input $x \in \{0,1\}^n$, and s_{out} pairs of keys k_ℓ^0, k_ℓ^1 for each output wire w_ℓ.
- P_1 and P_2 make n calls to $\mathcal{F}_{\mathsf{OT}}$, giving n more keys to P_2.
- P_2 computes s decryptions by evaluating the circuit $\mathcal{C}$.
- Finally, P_2 outputs s_{out} bits.

Next Time: the GMW Protocol