CS 594 – Advanced Crypto (Spring 2026)

Alex Block

Lecture 17

March 16, 2026

The GMW Protocol

GMW Protocol: Intuition

GMW Protocol: Intuition

- Yao's GC protocol: can think of it as secret sharing wire values.

GMW Protocol: Intuition

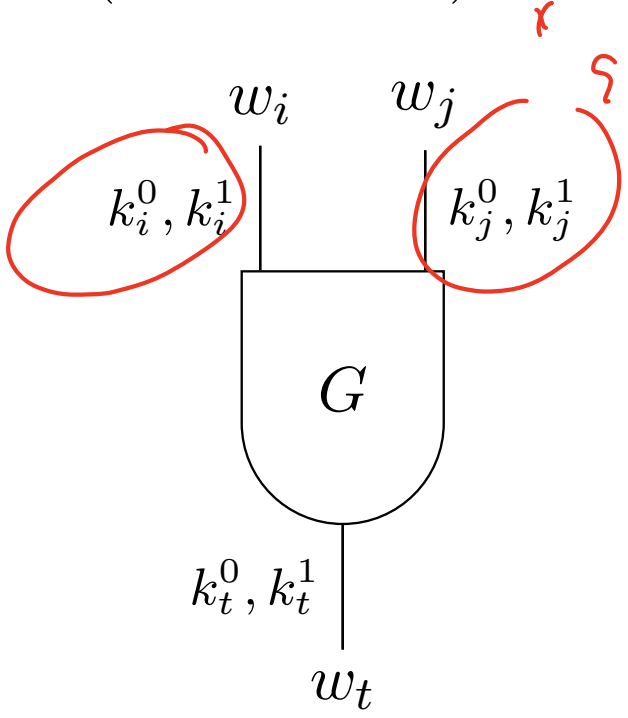
- Yao's GC protocol: can think of it as secret sharing wire values.
  - w_i^0, w_i^1 are two possible wire values known by P_1 (the generator), shared by k_i^0, k_i^1.

GMW Protocol: Intuition

- Yao's GC protocol: can think of it as secret sharing wire values.
  - w_i^0, w_i^1 are two possible wire values known by P_1 (the generator), shared by k_i^0, k_i^1.
  - P_2 (the evaluator) knows the true wire value w_i^b.

GMW Protocol: Intuition

- Yao's GC protocol: can think of it as secret sharing wire values.
  - w_i^0, w_i^1 are two possible wire values known by P_1 (the generator), shared by k_i^0, k_i^1.
  - P_2 (the evaluator) knows the true wire value w_i^b.

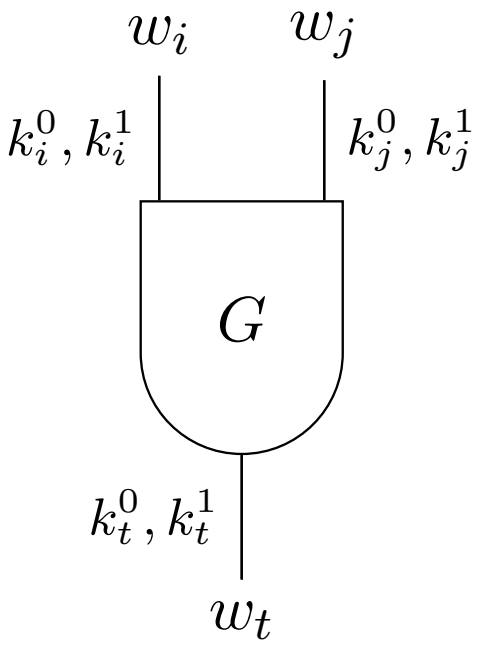


"secret shares"

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

GMW Protocol: Intuition

- Yao's GC protocol: can think of it as secret sharing wire values.
  - w_i^0, w_i^1 are two possible wire values known by P_1 (the generator), shared by k_i^0, k_i^1.
  - P_2 (the evaluator) knows the true wire value w_i^b.



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

- GMW Protocol: P_1 and P_2 both have *additive* secret shares of w_i for every wire.

GMW Protocol: Roadmap

GMW Protocol: Roadmap

- The GMW Protocol generalizes Yao's GC protocol to m parties and can operate over both Boolean and arithmetic circuits.

GMW Protocol: Roadmap

- The GMW Protocol generalizes Yao's GC protocol to m parties and can operate over both Boolean and arithmetic circuits.
  - Arithmetic circuit: circuits defined over $\mathbb{F}$ with ADD, MULT, and CONST (multiplication by a constant) gates.

GMW Protocol: Roadmap

- The GMW Protocol generalizes Yao's GC protocol to m parties and can operate over both Boolean and arithmetic circuits.
  - Arithmetic circuit: circuits defined over $\mathbb{F}$ with ADD, MULT, and CONST (multiplication by a constant) gates.

- We will first discuss the 2 player Boolean case, then generalize to m players.

GMW Protocol: Roadmap

- The GMW Protocol generalizes Yao's GC protocol to m parties and can operate over both Boolean and arithmetic circuits.
  - Arithmetic circuit: circuits defined over $\mathbb{F}$ with ADD, MULT, and CONST (multiplication by a constant) gates.
- We will first discuss the 2 player Boolean case, then generalize to m players.
- For GMW over finite fields, we'll instead look at another protocol and then cast it as GMW (next time).

GMW Protocol: Goal

GMW Protocol: Goal

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

GMW Protocol: Goal

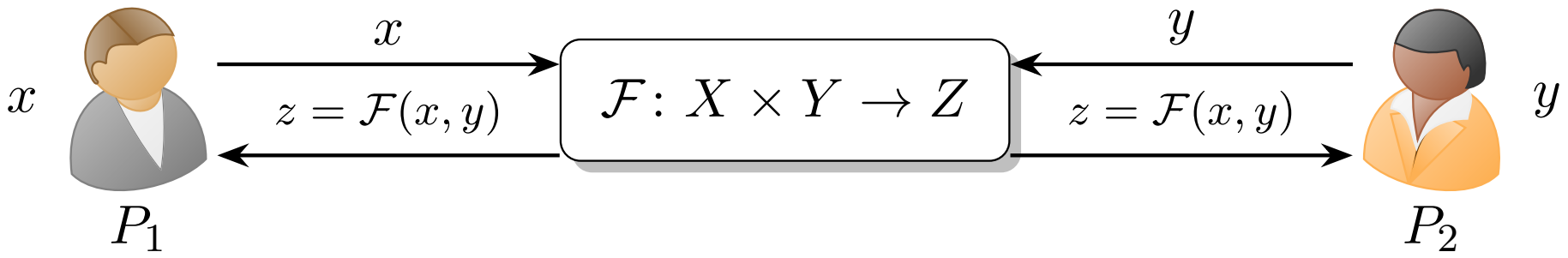
- **Goal:** Enable P_1 and P_2 with respective inputs x and y evaluate $\mathcal{F}(x, y)$, securely.
  - I.e., give a protocol for securely implementing the $\mathcal{F}$ functionality.

GMW Protocol: Goal

- **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.
  - I.e., the same goal as Yao's GC Protocol.

GMW Protocol: Goal

- **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.
  - I.e., the same goal as Yao's GC Protocol.



GMW Protocol: Initialization

GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.

GMW Protocol: Initialization

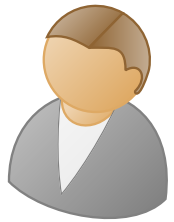
- First, each party will secret share their input bits x and y.

P_1

GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.

P_1



P_2

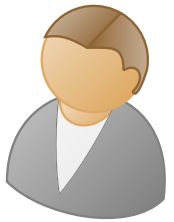
GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.

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

P_1

$\mathbf{y} \in \{0,1\}^n$



P_2

GMW Protocol: Initialization

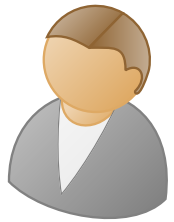
- First, each party will secret share their input bits x and y.

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

P_1

$\mathbf{r}^{(1)} \xleftarrow{\$} \{0,1\}^n$

$\mathbf{y} \in \{0,1\}^n$

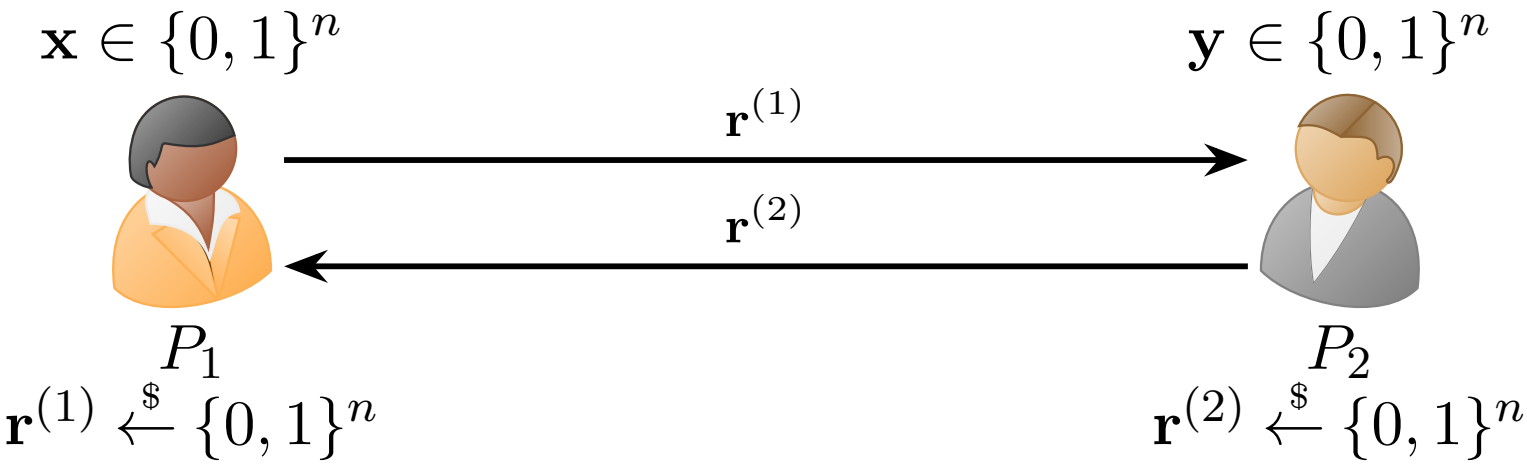


P_2

$\mathbf{r}^{(2)} \xleftarrow{\$} \{0,1\}^n$

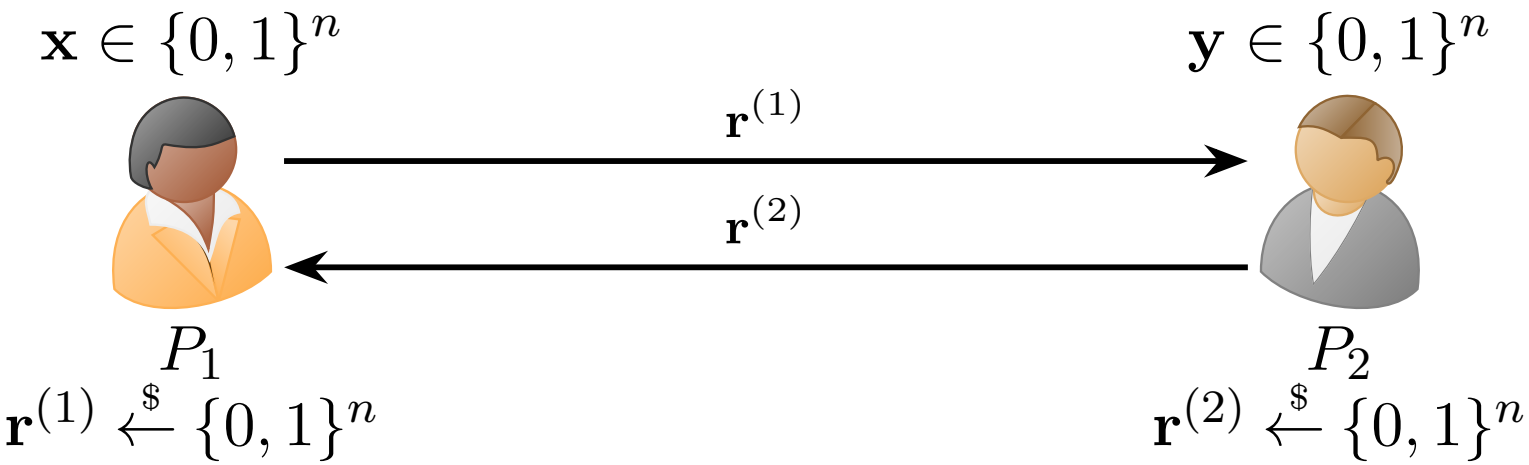
GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.



GMW Protocol: Initialization

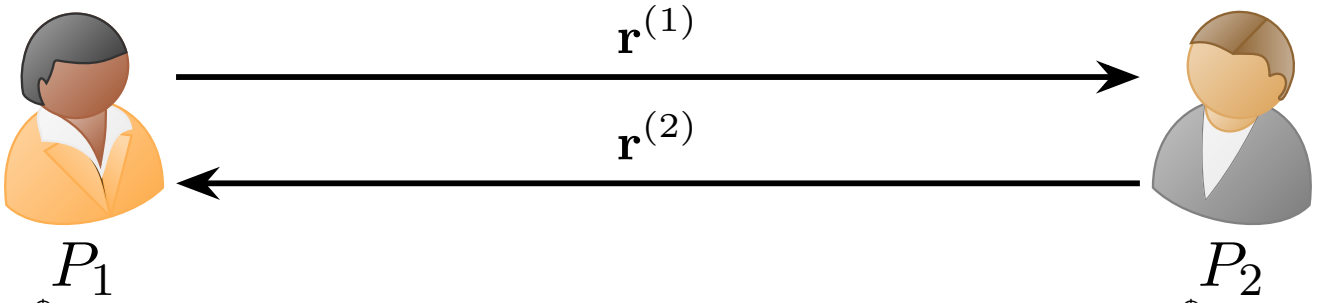
- First, each party will secret share their input bits x and y.



$$\mathbf{s}^{(1)} = (\mathbf{x} \oplus \mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \{0,1\}^{2n}$$

GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.

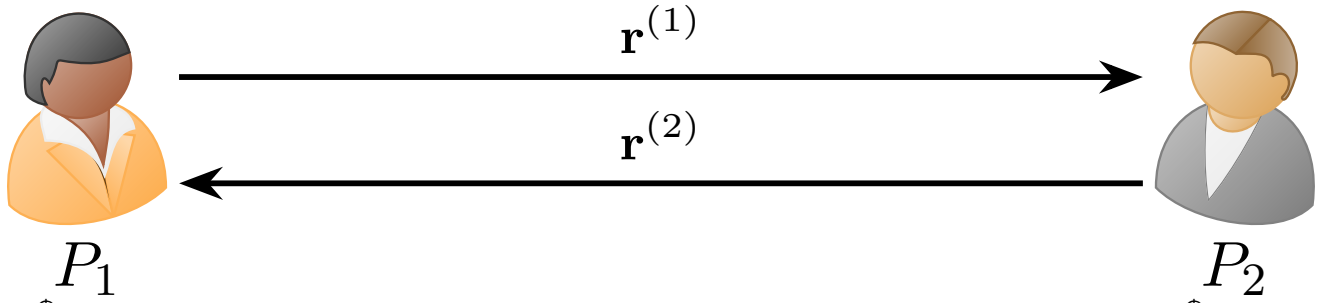


$$\mathbf{s}^{(1)} = (\mathbf{x} \oplus \mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \{0,1\}^{2n}$$

$$\mathbf{s}^{(2)} = (\mathbf{r}^{(1)}, \mathbf{y} \oplus \mathbf{r}^{(2)}) \in \{0,1\}^{2n}$$

GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.

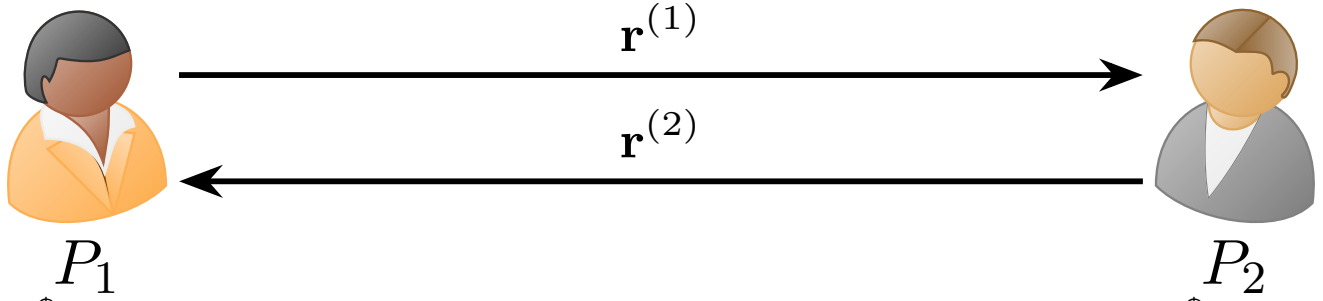


$$\mathbf{s}^{(1)} = (\mathbf{x} \oplus \mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \{0,1\}^{2n} \qquad \mathbf{s}^{(2)} = (\mathbf{r}^{(1)}, \mathbf{y} \oplus \mathbf{r}^{(2)}) \in \{0,1\}^{2n}$$

$$\mathbf{s}^{(1)} \oplus \mathbf{s}^{(2)} = (\mathbf{x}, \mathbf{y})$$

GMW Protocol: Initialization

- First, each party will secret share their input bits x and y.



$$\mathbf{s}^{(1)} = (\mathbf{x} \oplus \mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \{0,1\}^{2n} \qquad \mathbf{s}^{(2)} = (\mathbf{r}^{(1)}, \mathbf{y} \oplus \mathbf{r}^{(2)}) \in \{0,1\}^{2n}$$

$$\boxed{\mathbf{s}^{(1)} \oplus \mathbf{s}^{(2)} = (\mathbf{x}, \mathbf{y})}$$

- Next, *both* parties proceed gate-by-gate evaluating $\mathcal{C}$ for $\mathcal{F}$ using inputs $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$, respectively.

GMW Protocol: Local Evaluation

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.

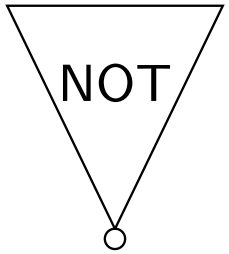
GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.

P_1's view

GMW Protocol: Local Evaluation

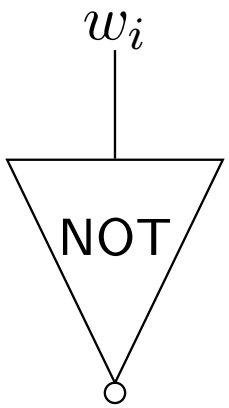
- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



P_1's view

GMW Protocol: Local Evaluation

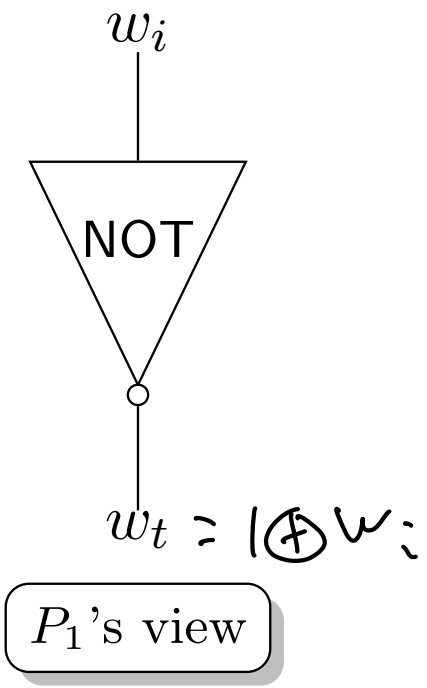
- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



P_1's view

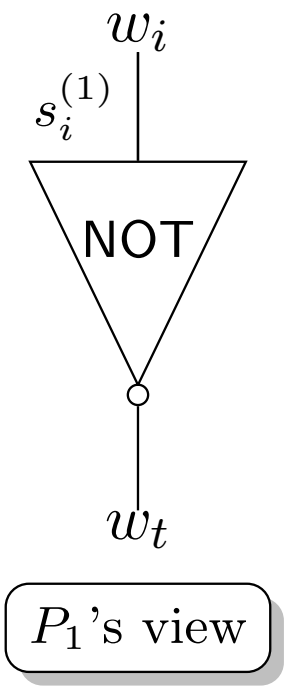
GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



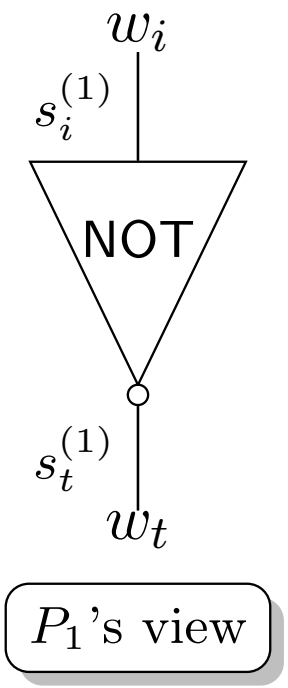
GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



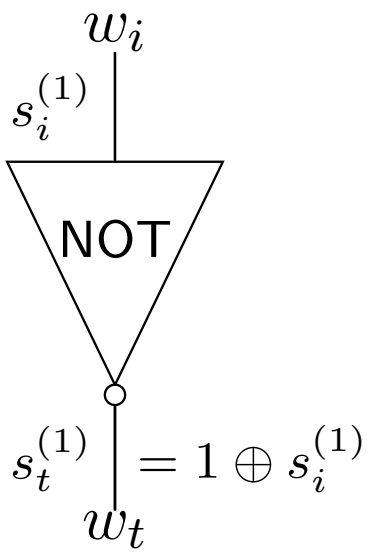
GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



GMW Protocol: Local Evaluation

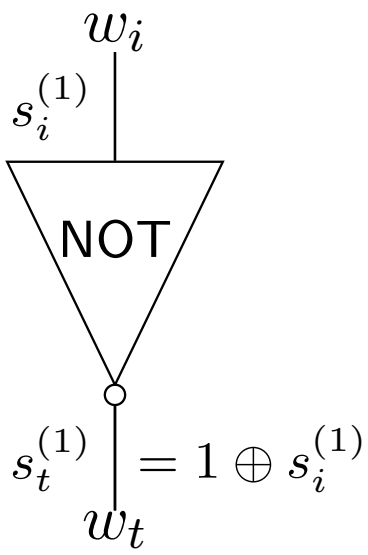
- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



P_1's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.

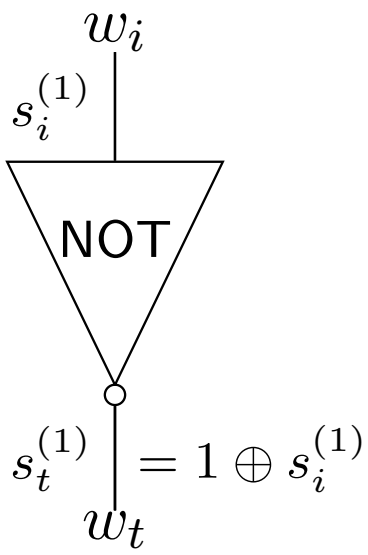


P_1's view

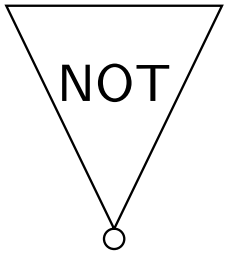
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



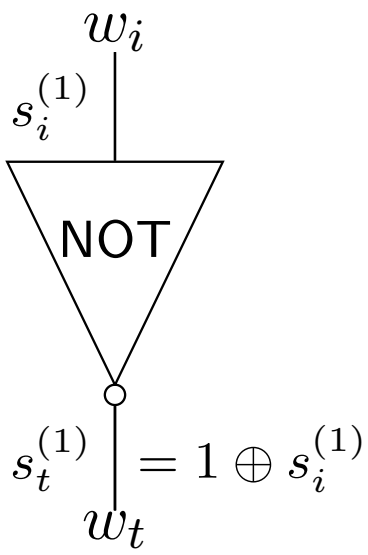
P_1's view



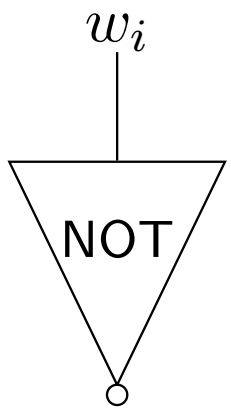
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



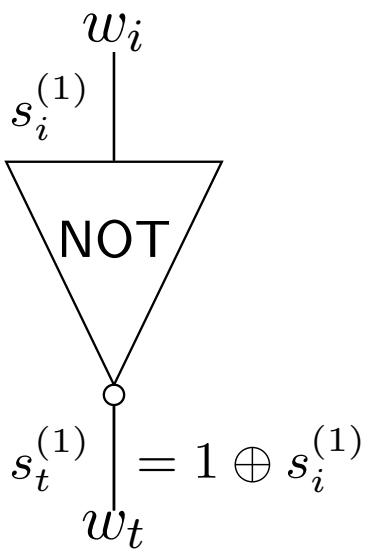
P_1's view



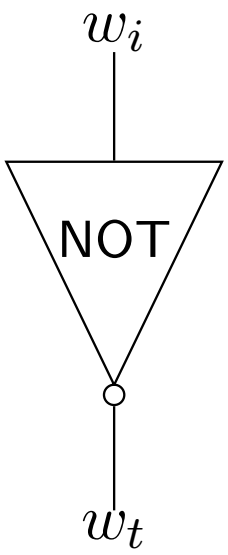
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



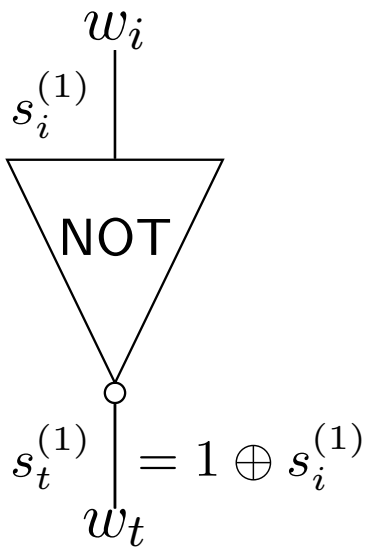
P_1's view



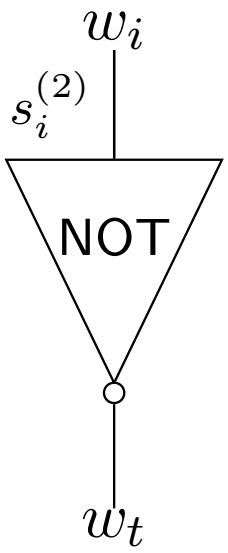
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



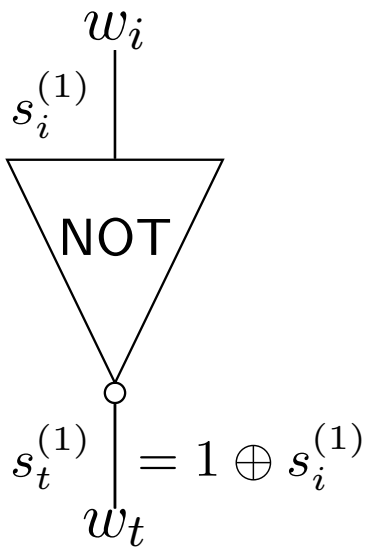
P_1's view



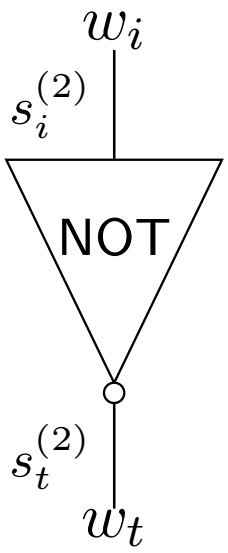
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



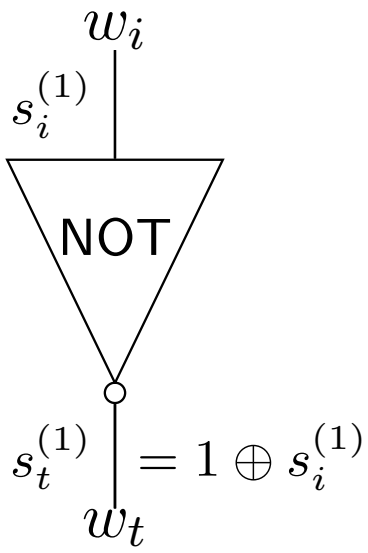
P_1's view



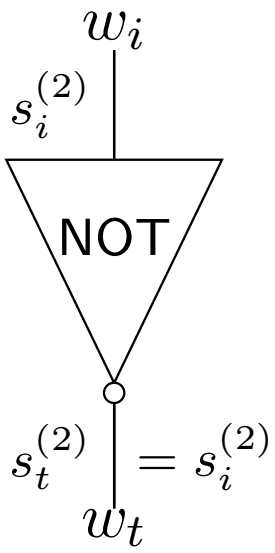
P_2's view

GMW Protocol: Local Evaluation

- Without loss of generality, we assume $\mathcal{C}$ only has NOT, XOR, and AND gates.
- For NOT and XOR gates, P_1 and P_2 can locally compute a secret share of the output wire so long as they have secret shares of the input wire(s).
  - For NOT gates, P_1 will flip the input share bit while P_2 doesn't change anything.



P_1's view



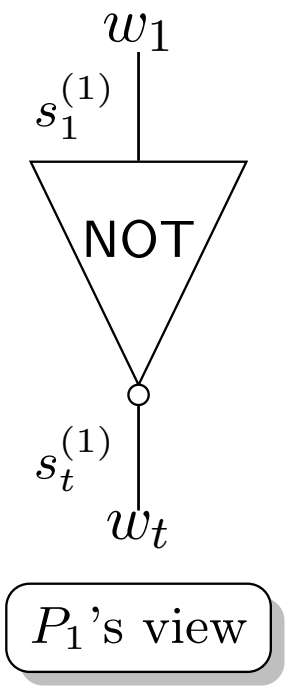
P_2's view

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.

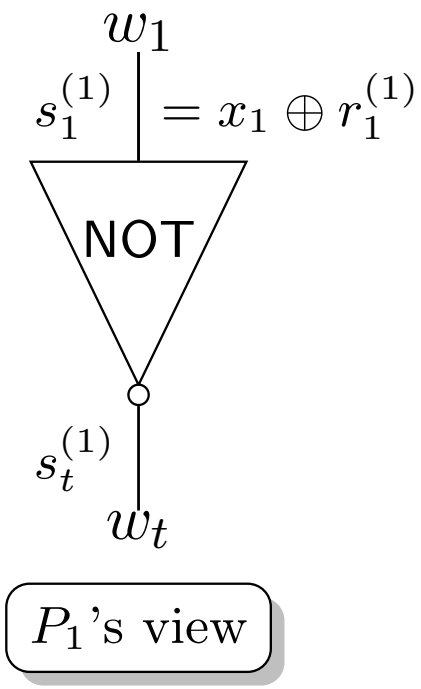
GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



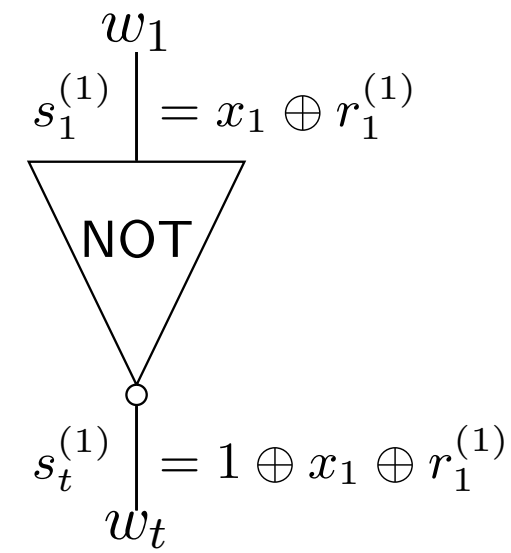
GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



GMW Protocol: Local Evaluation of NOT

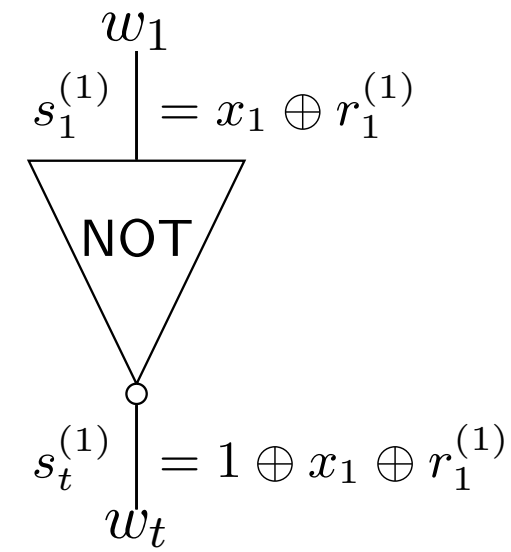
- As an example, suppose we have a NOT gate with input x_1.



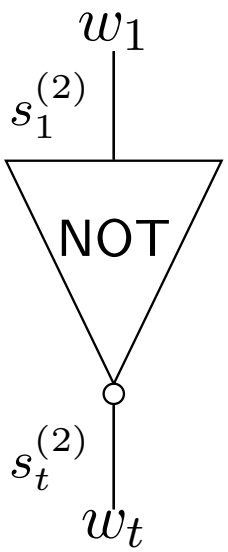
P_1's view

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



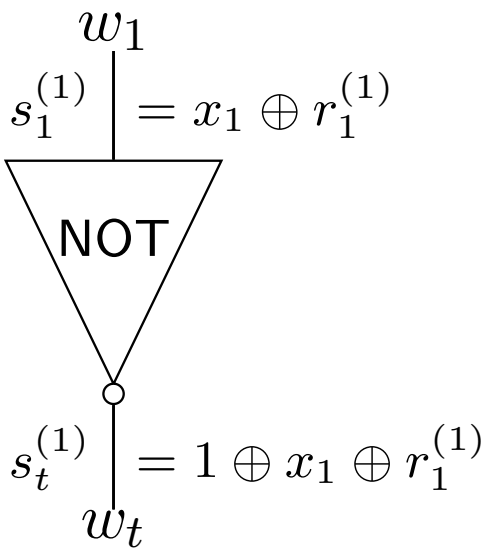
P_1's view



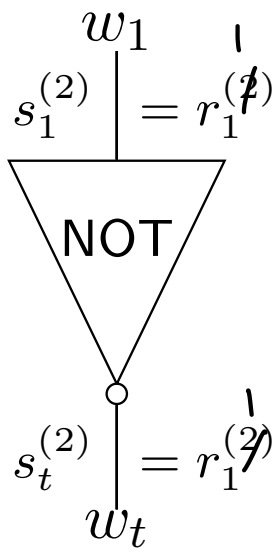
P_2's view

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



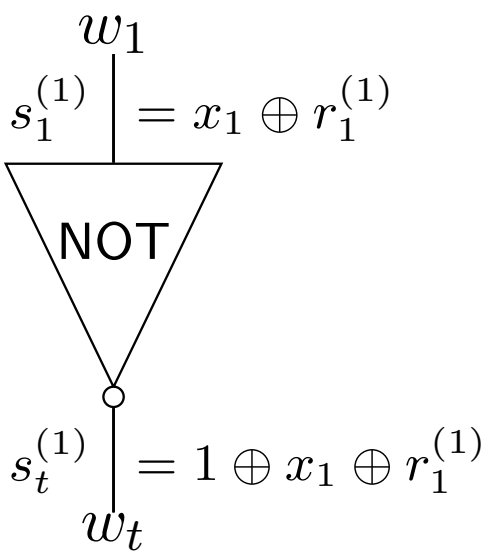
P_1's view



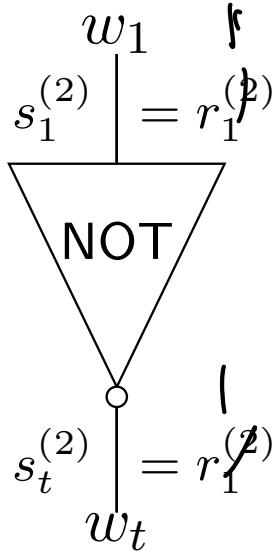
P_2's view

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



P_1's view

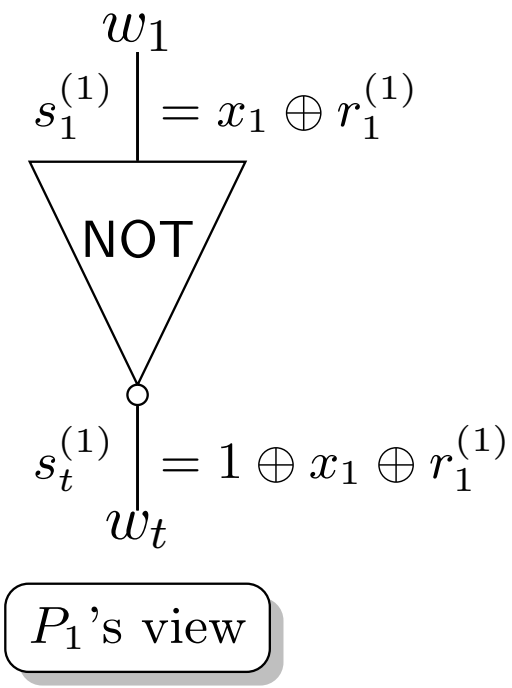


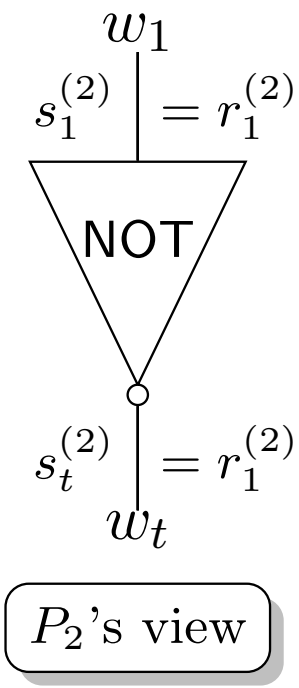
P_2's view

- Input is secret shared: $s_1^{(1)} \oplus s_1^{(2)} = (x_1 \oplus r_1^{(1)}) \oplus r_1^{(1)} = x_1$.

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



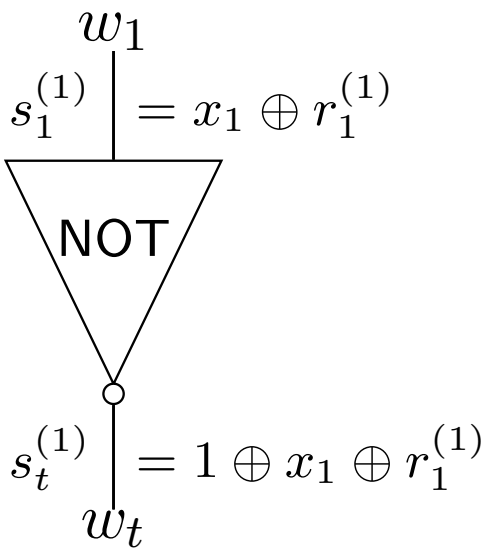


- Input is secret shared: $s_1^{(1)} \oplus s_1^{(2)} = (x_1 \oplus r_1^{(1)}) \oplus r_1^{(1)} = x_1$.

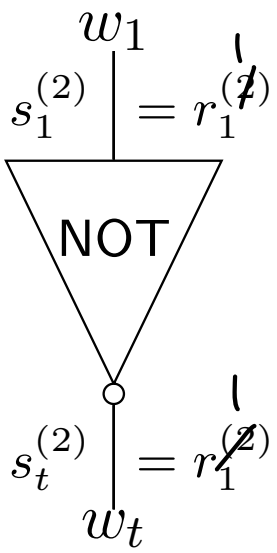
- Parties locally compute secret shared output:

GMW Protocol: Local Evaluation of NOT

- As an example, suppose we have a NOT gate with input x_1.



P_1's view



P_2's view

- Input is secret shared: $s_1^{(1)} \oplus s_1^{(2)} = (x_1 \oplus r_1^{(1)}) \oplus r_1^{(1)} = x_1$.

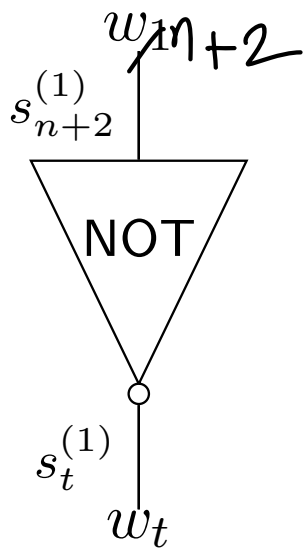
- Parties locally compute secret shared output:
  - $s_t^{(1)} \oplus s_t^{(2)} = (1 \oplus x_1 \oplus r_1^{(1)}) \oplus r_1^{(1)} = 1 \oplus x_1$.

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.

GMW Protocol: Local Evaluation of NOT

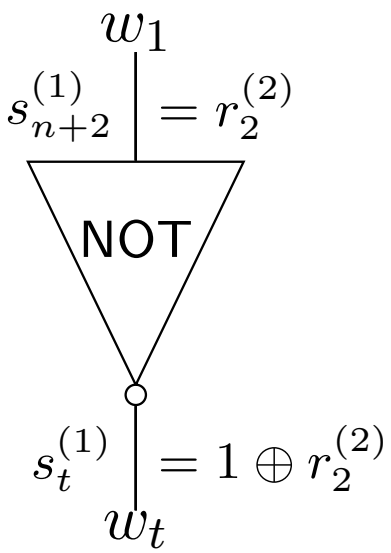
- As another example, suppose we have a NOT gate with input y_2.



P_1's view

GMW Protocol: Local Evaluation of NOT

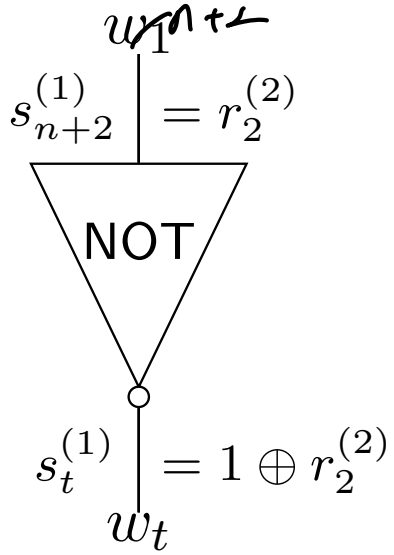
- As another example, suppose we have a NOT gate with input y_2.



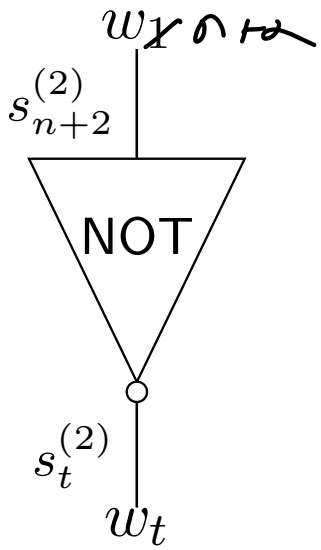
P_1's view

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.



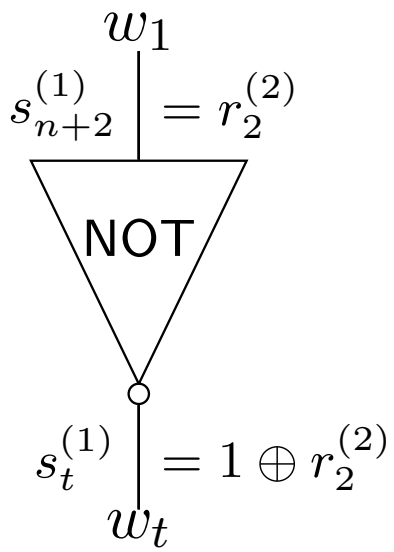
P_1's view



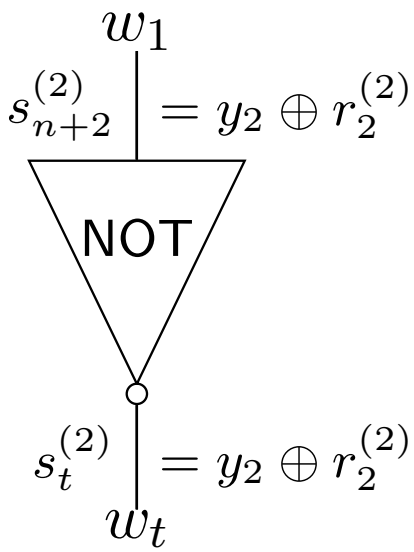
P_2's view

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.



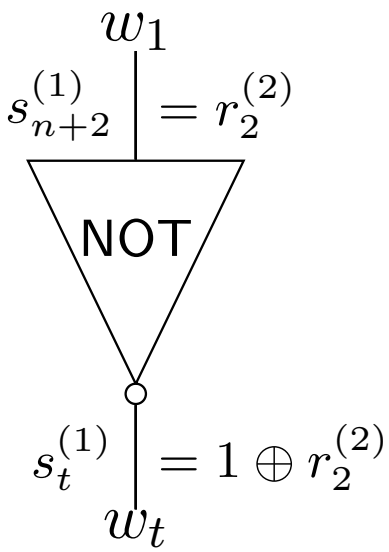
P_1's view



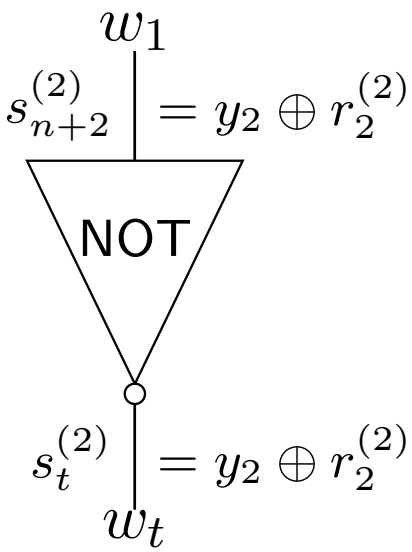
P_2's view

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.



P_1's view

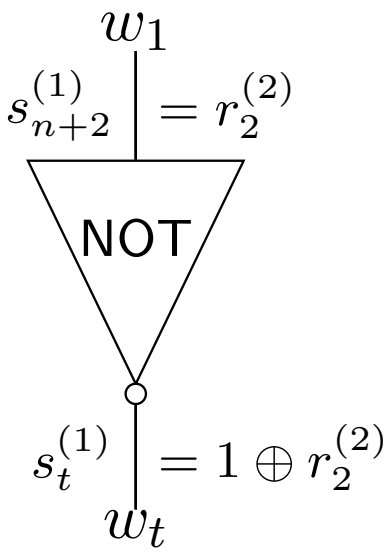


P_2's view

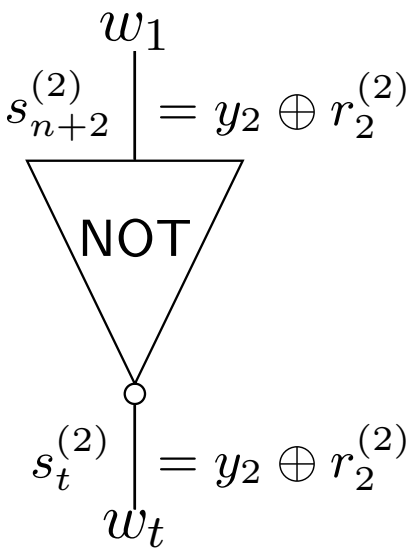
- Input is secret shared: $s_{n+2}^{(1)} \oplus s_{n+2}^{(2)} = r_2^{(2)} \oplus (y_2 \oplus r_2^{(2)}) = y_2$.

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.



P_1's view



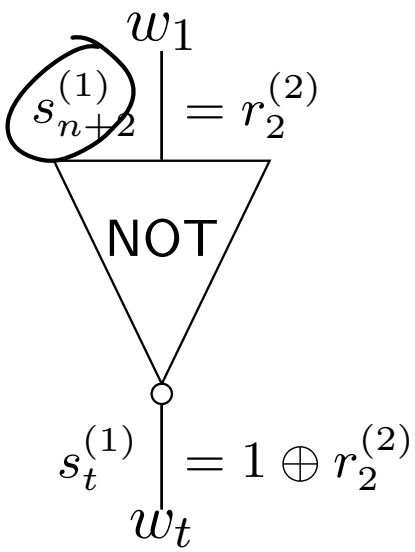
P_2's view

- Input is secret shared: $s_{n+2}^{(1)} \oplus s_{n+2}^{(2)} = r_2^{(2)} \oplus (y_2 \oplus r_2^{(2)}) = y_2$.

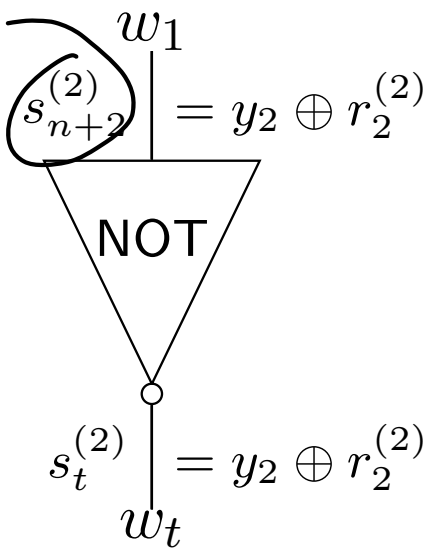
- Parties locally compute secret shared output:

GMW Protocol: Local Evaluation of NOT

- As another example, suppose we have a NOT gate with input y_2.



P_1's view



P_2's view

- Input is secret shared: $s_{n+2}^{(1)} \oplus s_{n+2}^{(2)} = r_2^{(2)} \oplus (y_2 \oplus r_2^{(2)}) = y_2$.

- Parties locally compute secret shared output:
  - $s_t^{(1)} \oplus s_t^{(2)} = (1 \oplus r_2^{(2)}) \oplus (y_2 \oplus r_2^{(2)}) = 1 \oplus y_2$. = w_t

GMW Protocol: Local Evaluation of XOR

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.

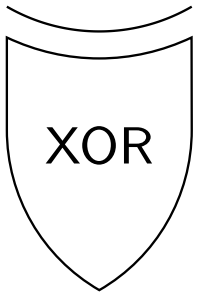
GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.

P_1's view

GMW Protocol: Local Evaluation of XOR

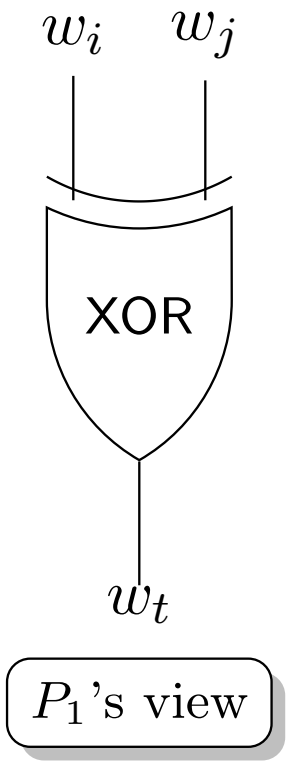
- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



P_1's view

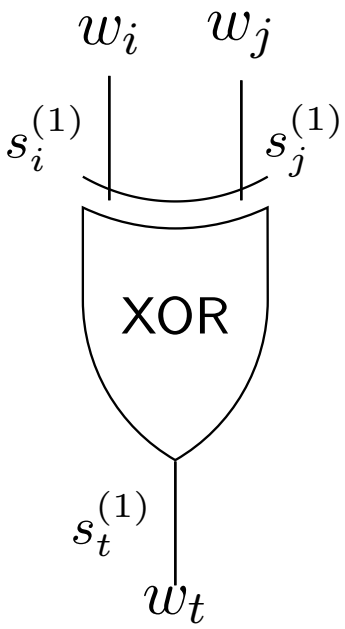
GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



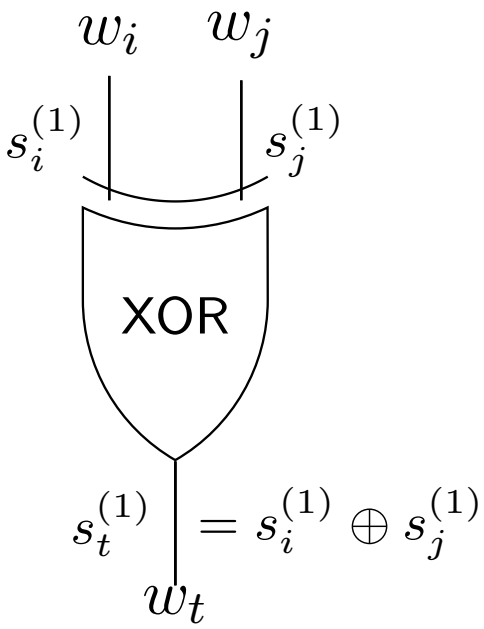
GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



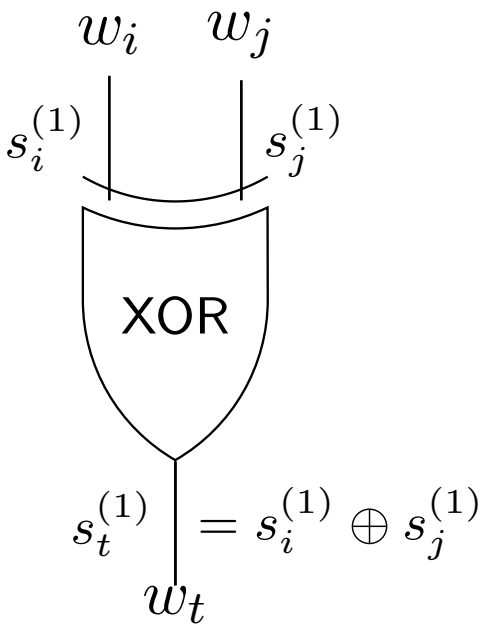
GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.

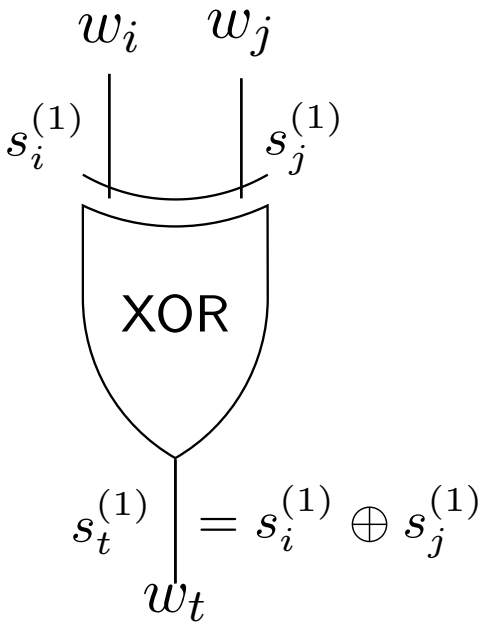


P_1's view

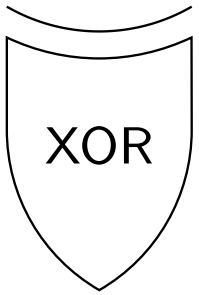
P_2's view

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



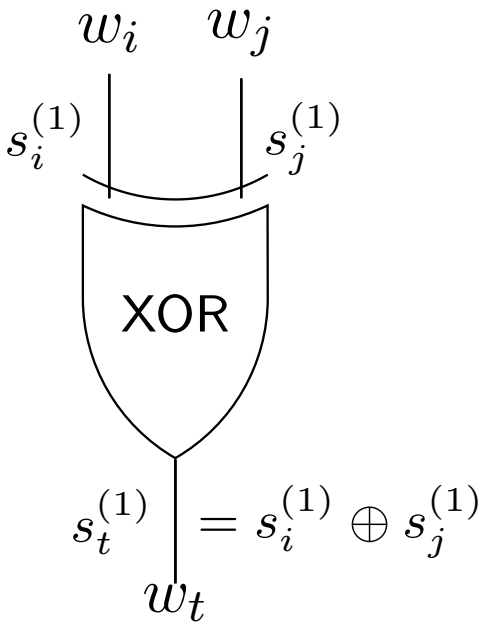
P_1's view



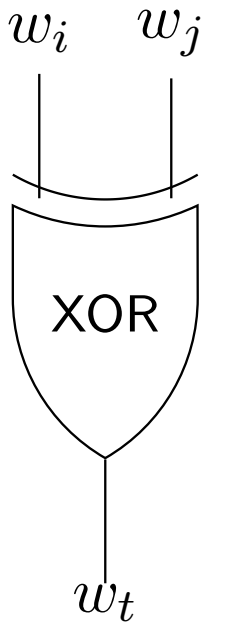
P_2's view

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



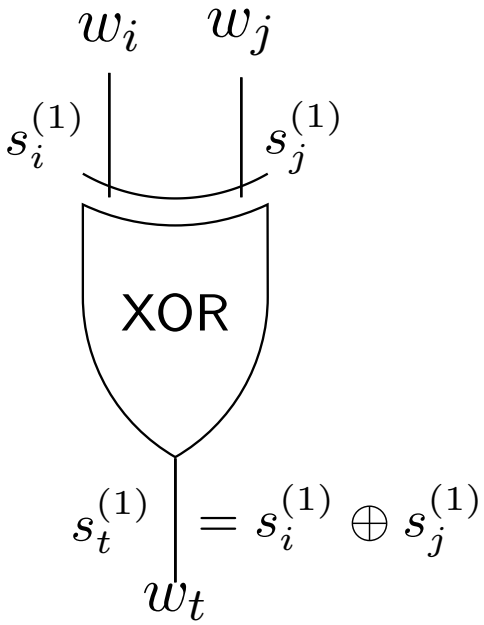
P_1's view



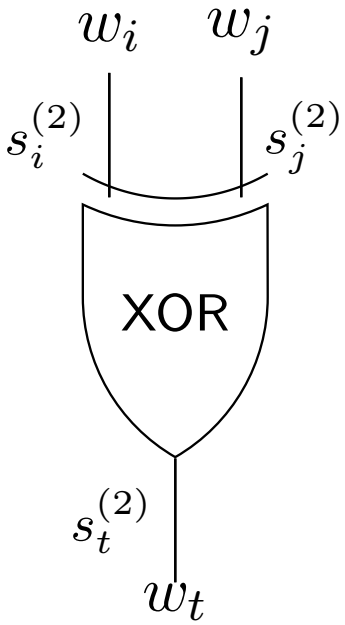
P_2's view

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



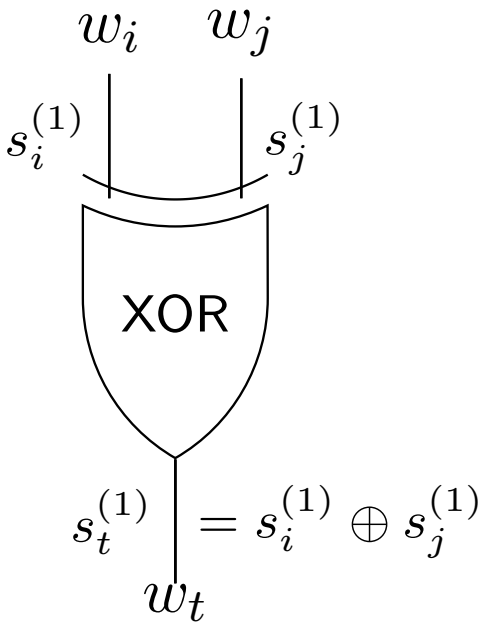
P_1's view



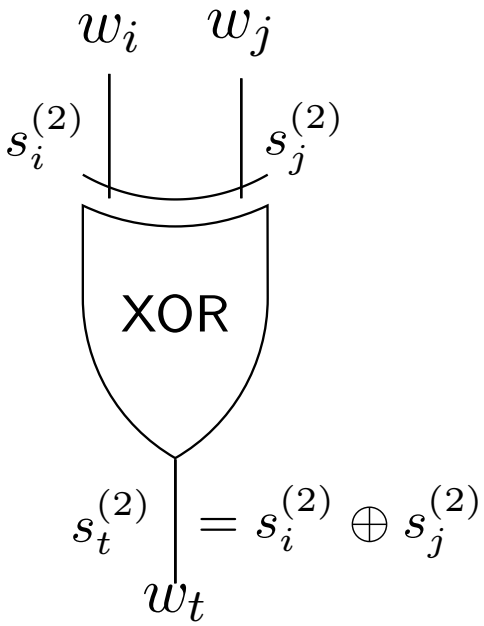
P_2's view

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



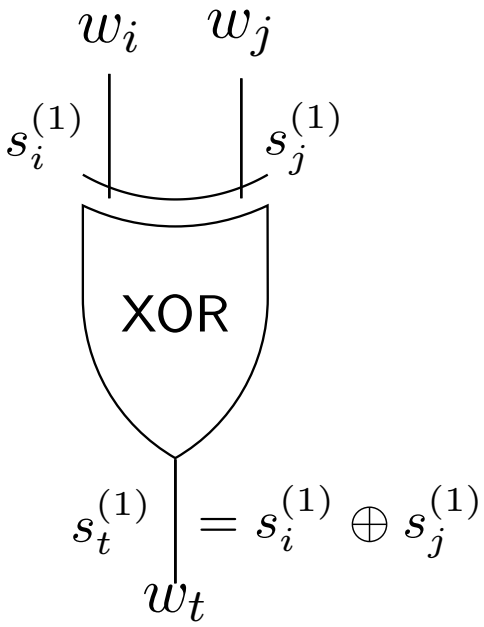
P_1's view



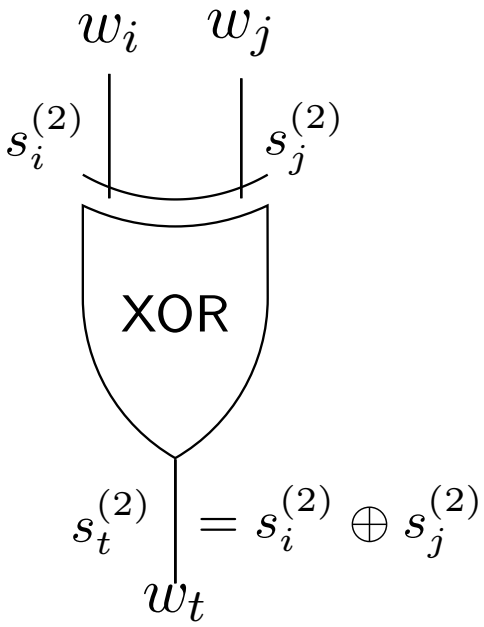
P_2's view

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



P_1's view

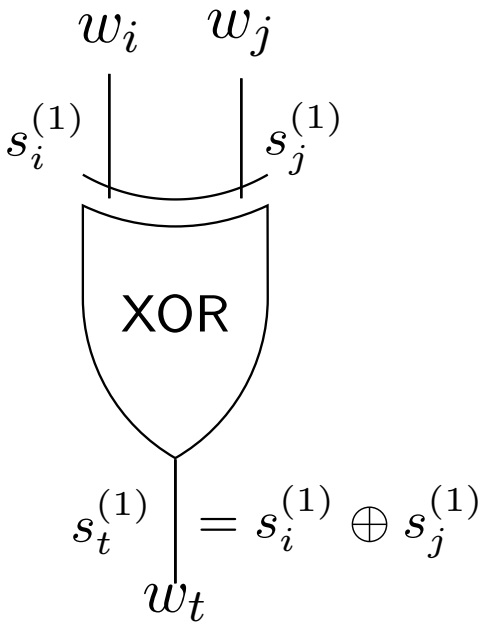


P_2's view

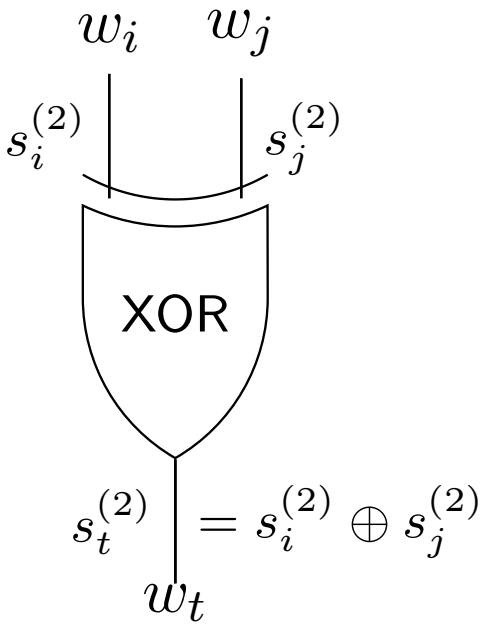
- Here, $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.

GMW Protocol: Local Evaluation of XOR

- For XOR gates, P_1 and P_2 will simply compute the XOR of the secret shares it holds for each wire.



P_1's view



P_2's view

- Here, $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.
- Then: $s_t^{(1)} \oplus s_t^{(2)} = (s_i^{(1)} \oplus s_j^{(1)}) \oplus (s_i^{(2)} \oplus s_j^{(2)}) = w_i \oplus w_j$.

GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.

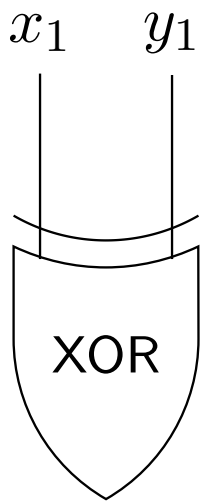
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.

P_1's view

GMW Protocol: Local Evaluation of XOR

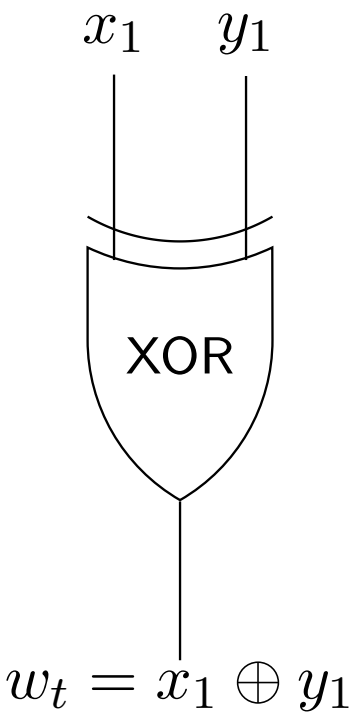
- Example: XOR gate with inputs x_1 and y_1.



P_1's view

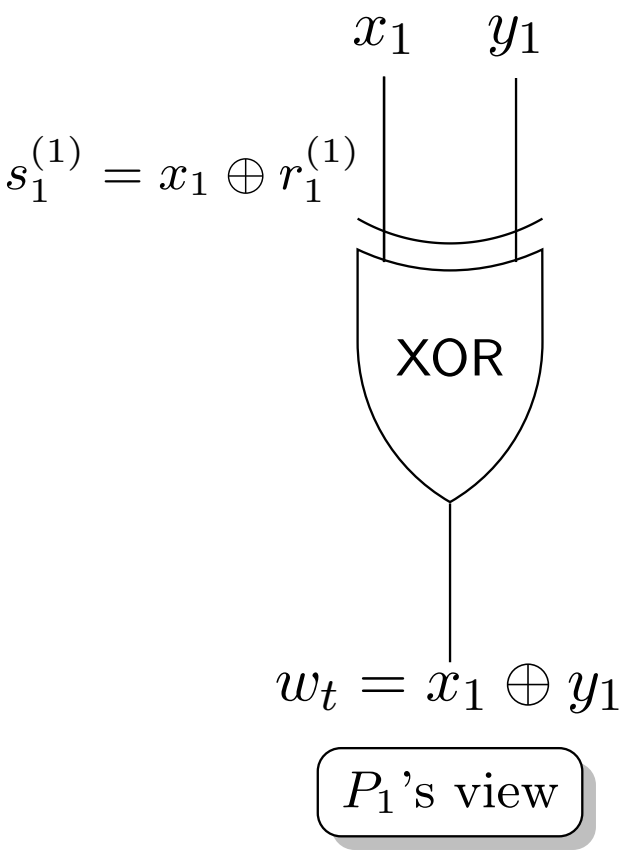
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



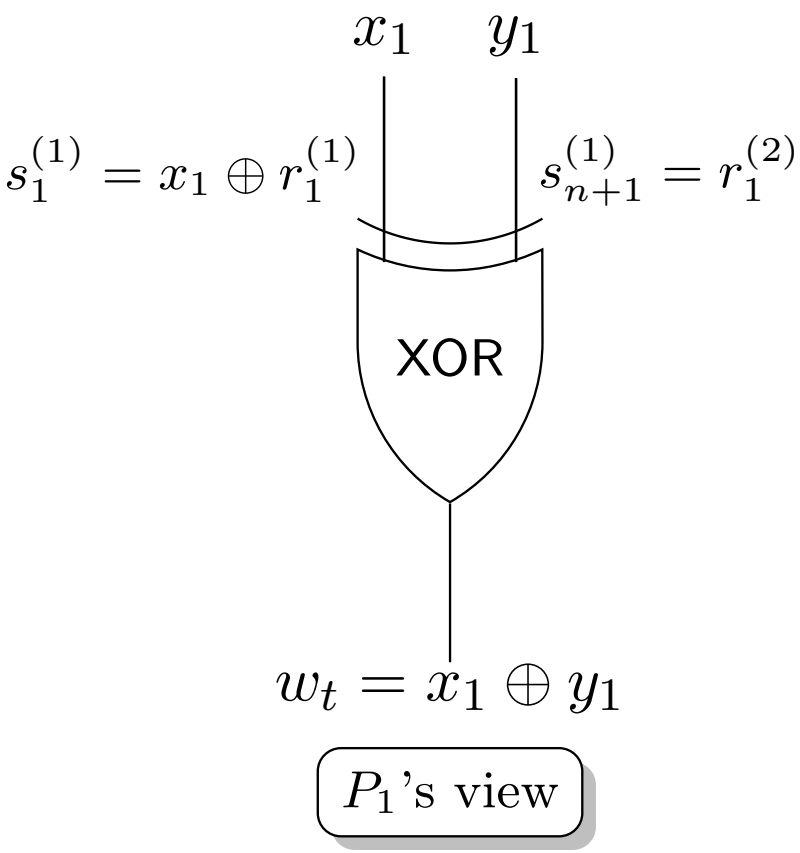
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



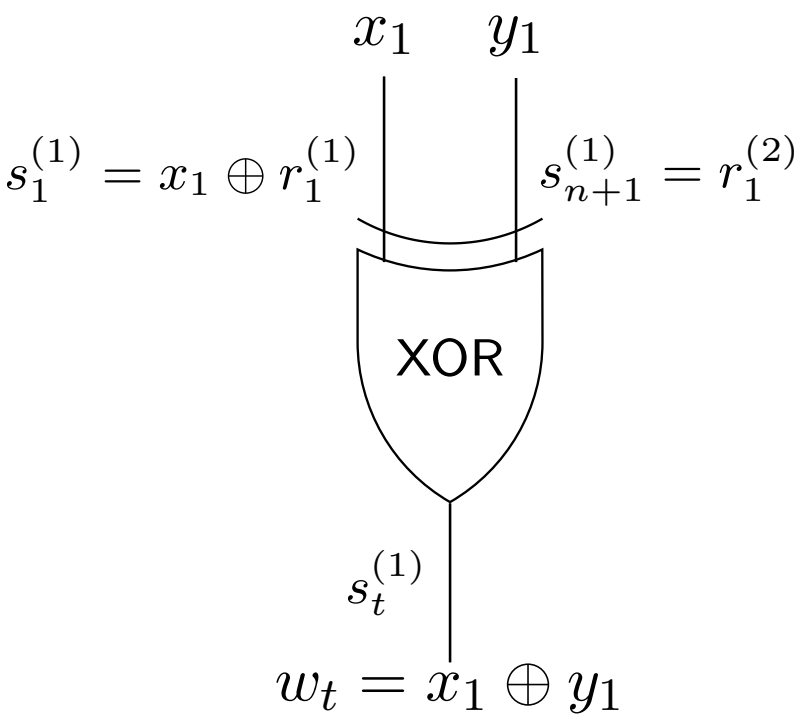
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



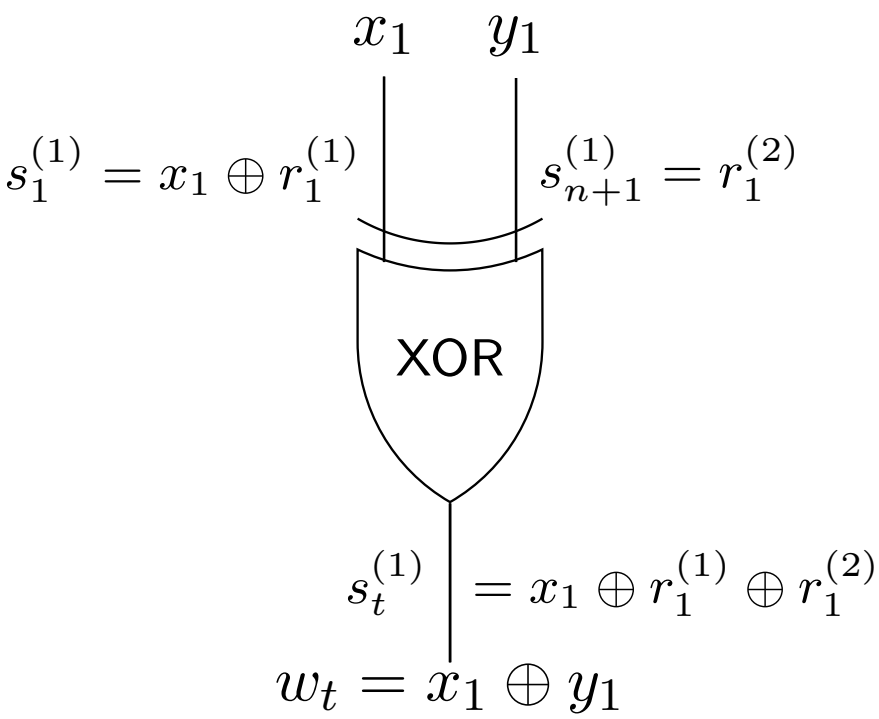
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



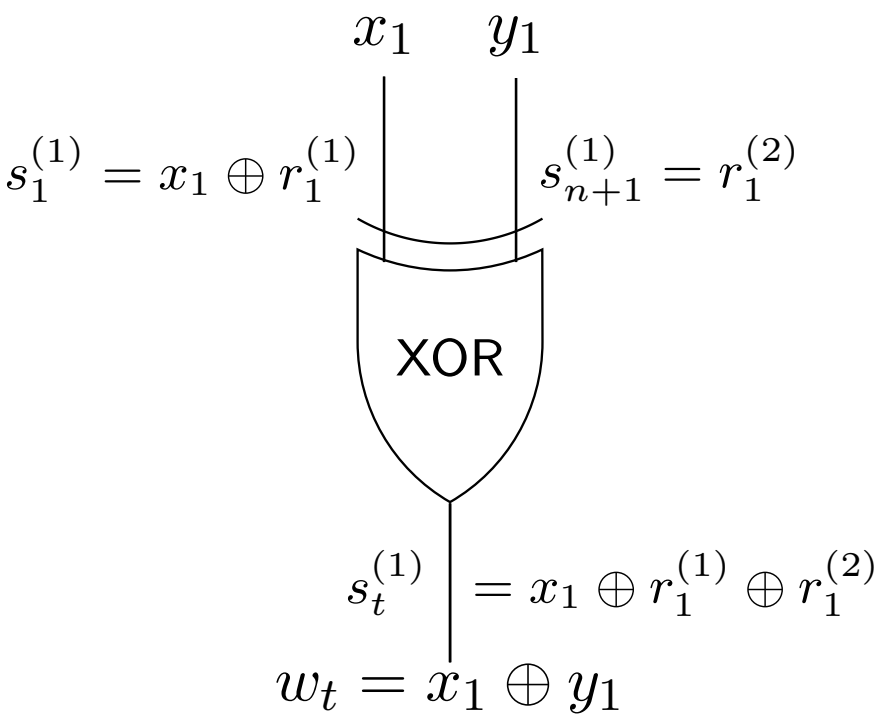
GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.

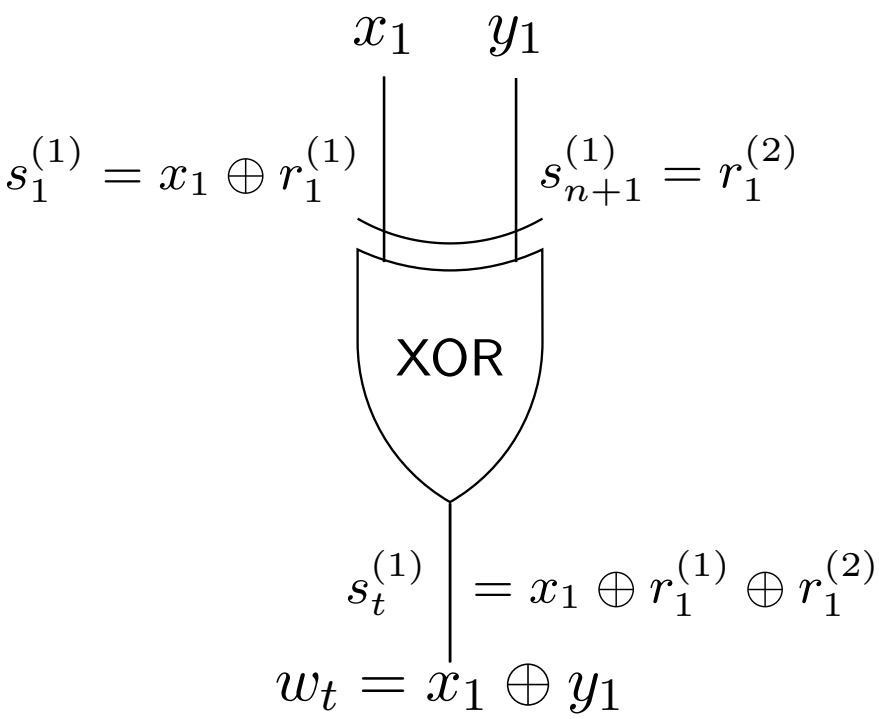


P_1's view

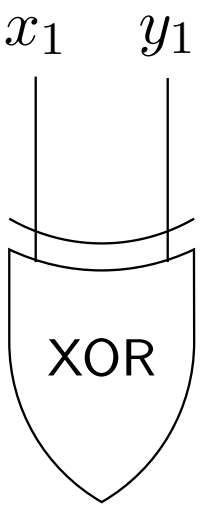
P_2's view

GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.



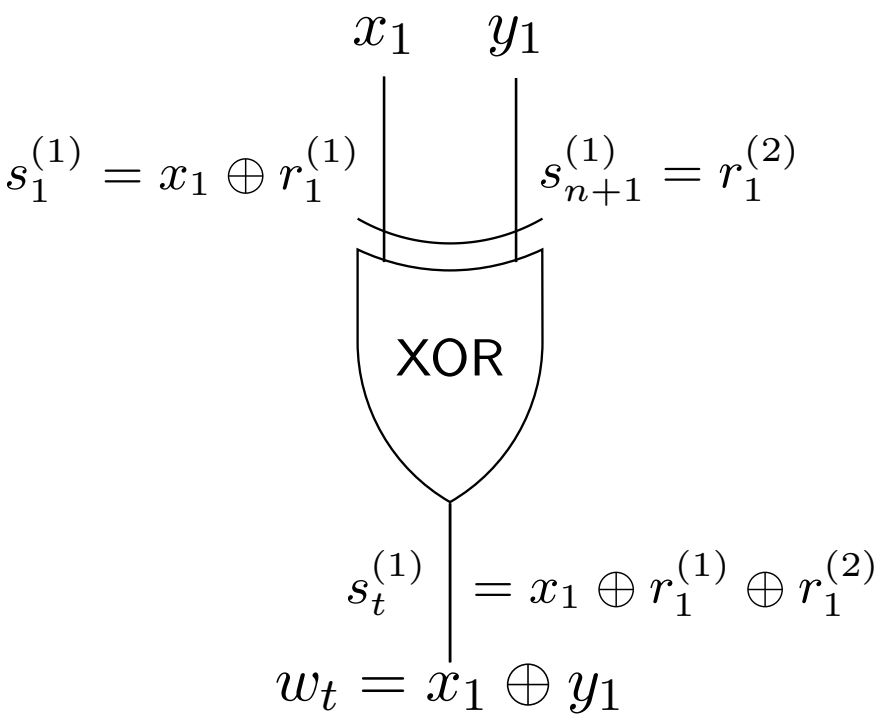
P_1's view

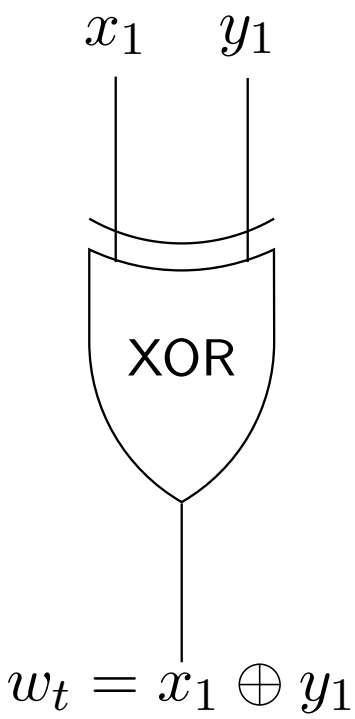


P_2's view

GMW Protocol: Local Evaluation of XOR

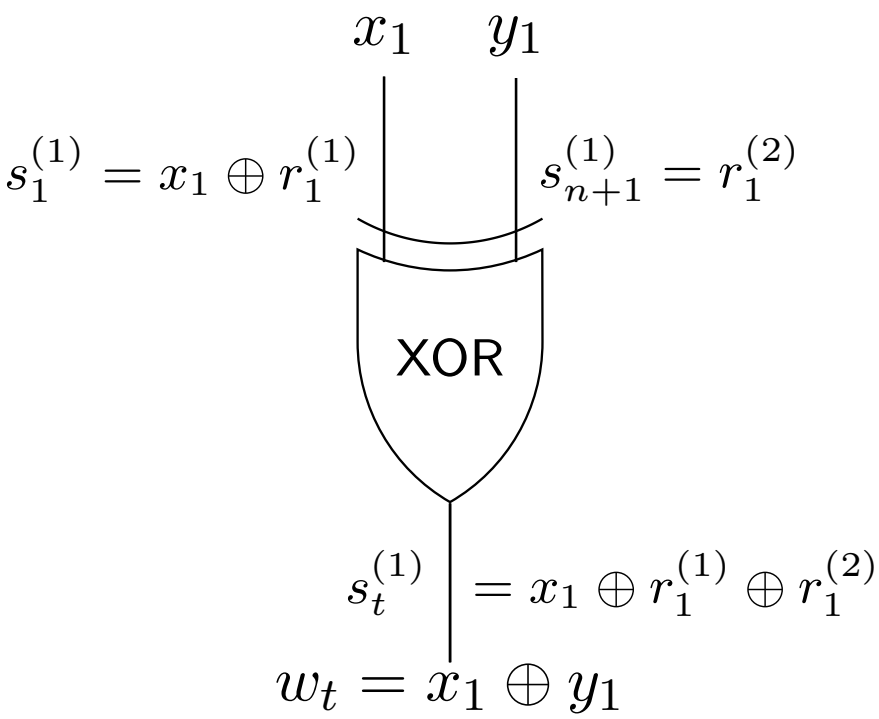
- Example: XOR gate with inputs x_1 and y_1.

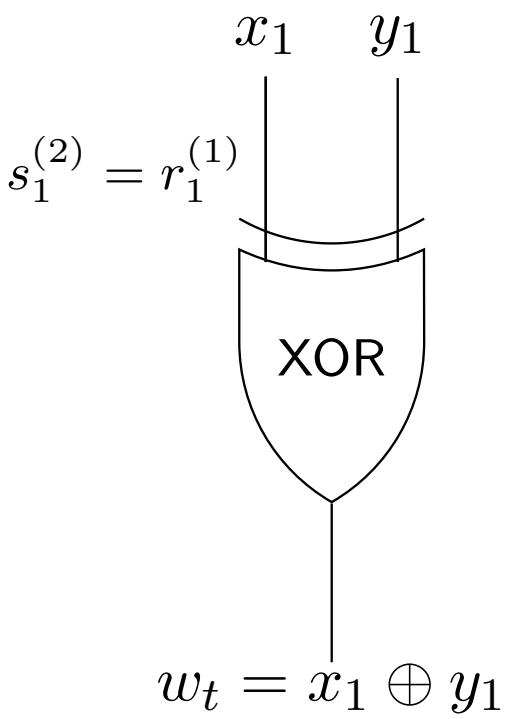




GMW Protocol: Local Evaluation of XOR

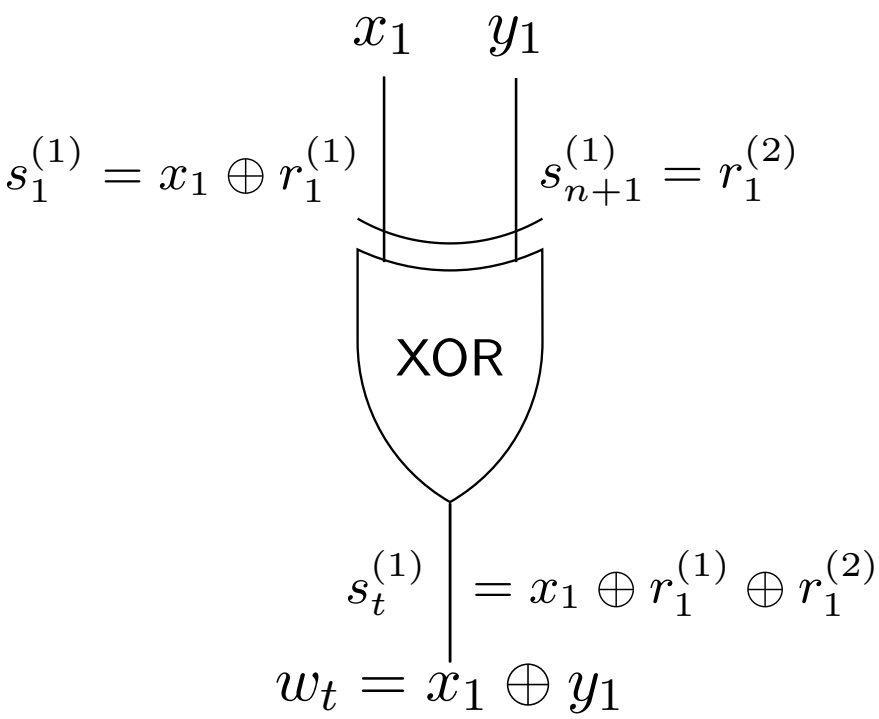
- Example: XOR gate with inputs x_1 and y_1.

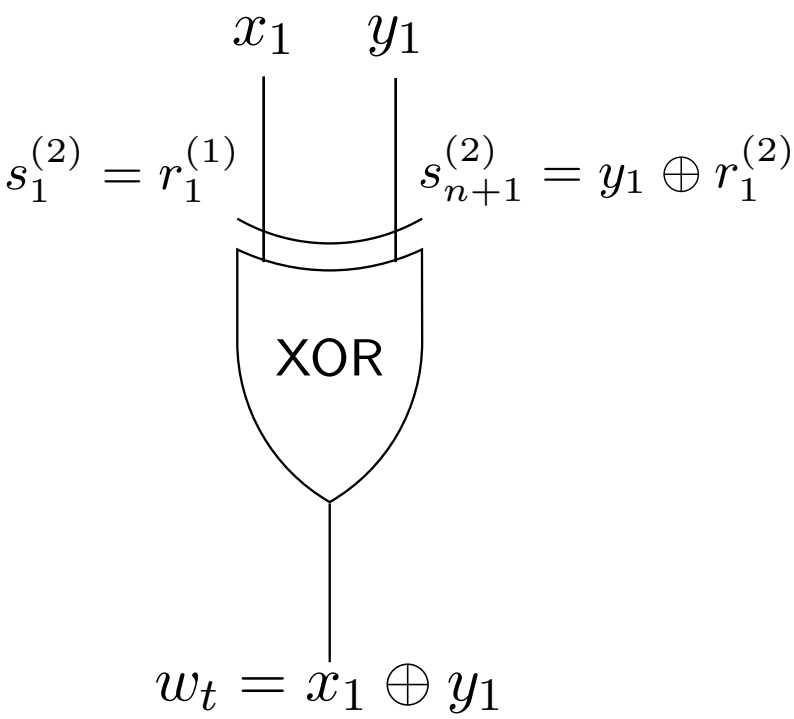




GMW Protocol: Local Evaluation of XOR

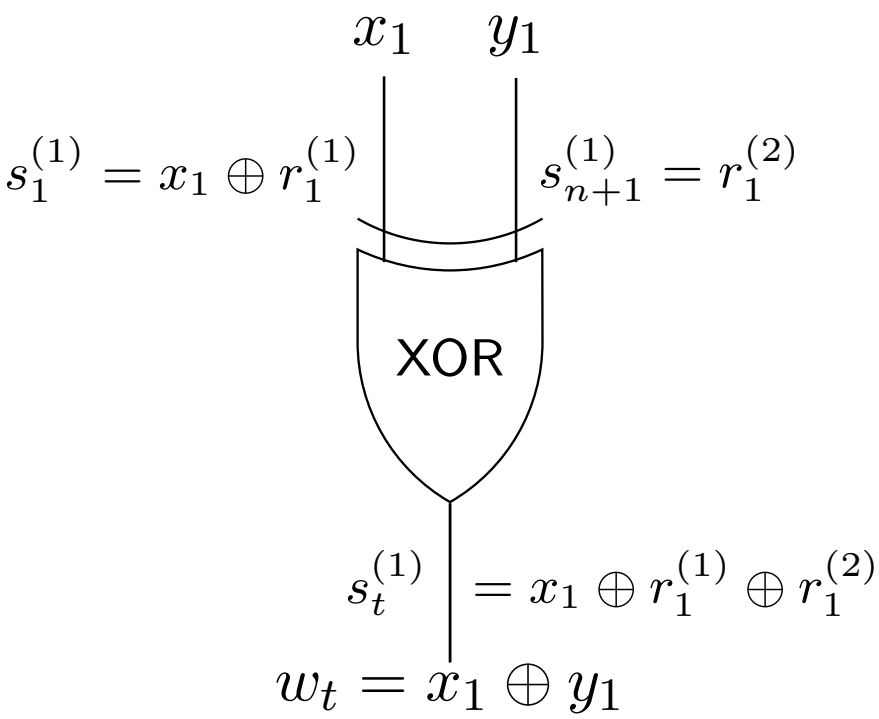
- Example: XOR gate with inputs x_1 and y_1.

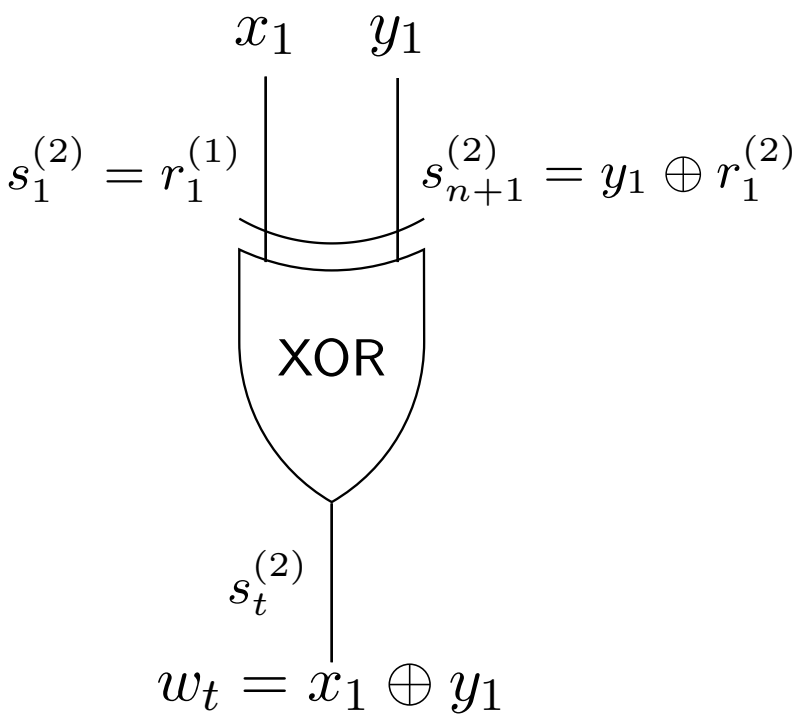




GMW Protocol: Local Evaluation of XOR

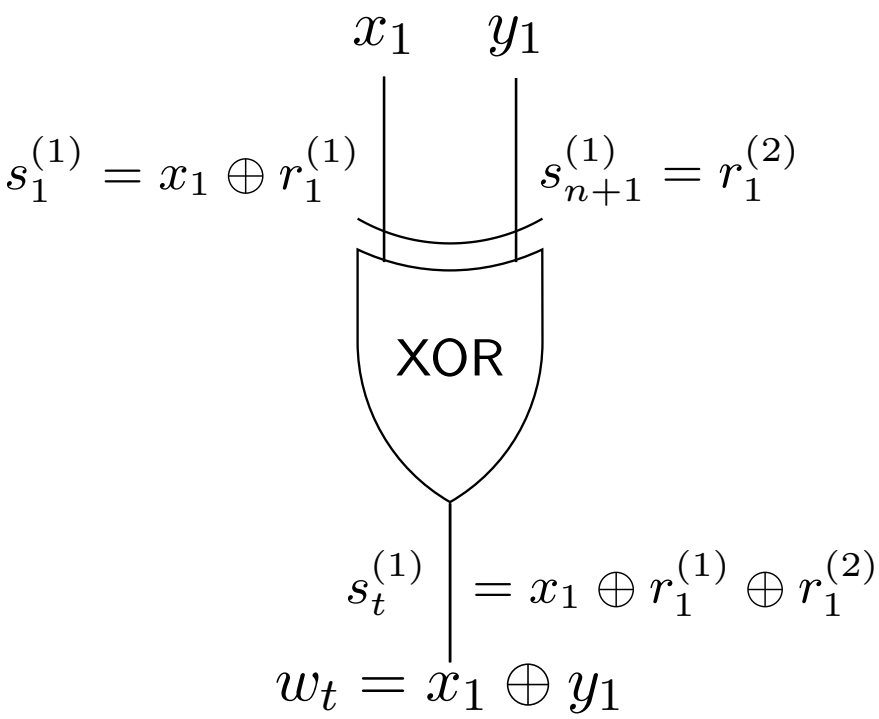
- Example: XOR gate with inputs x_1 and y_1.

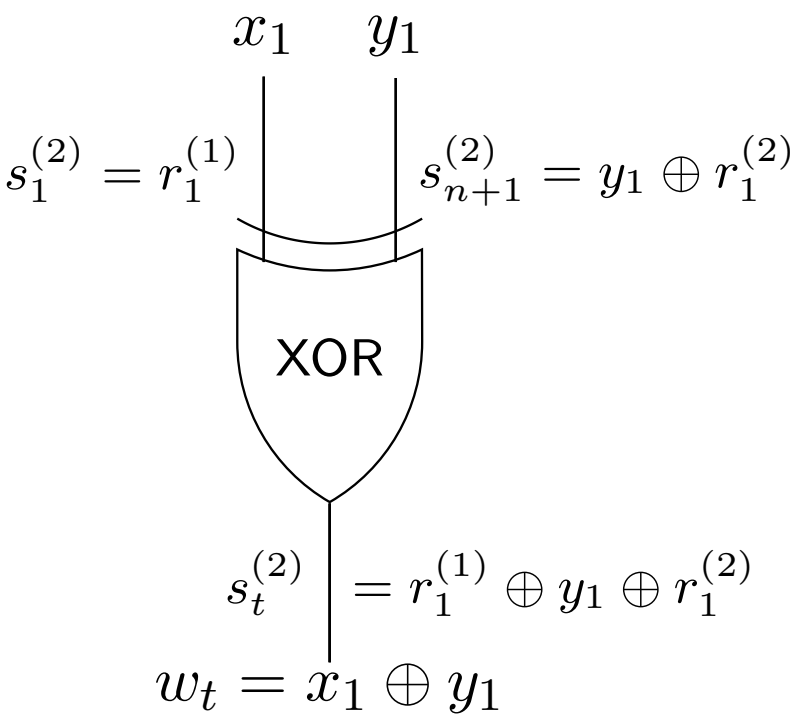




GMW Protocol: Local Evaluation of XOR

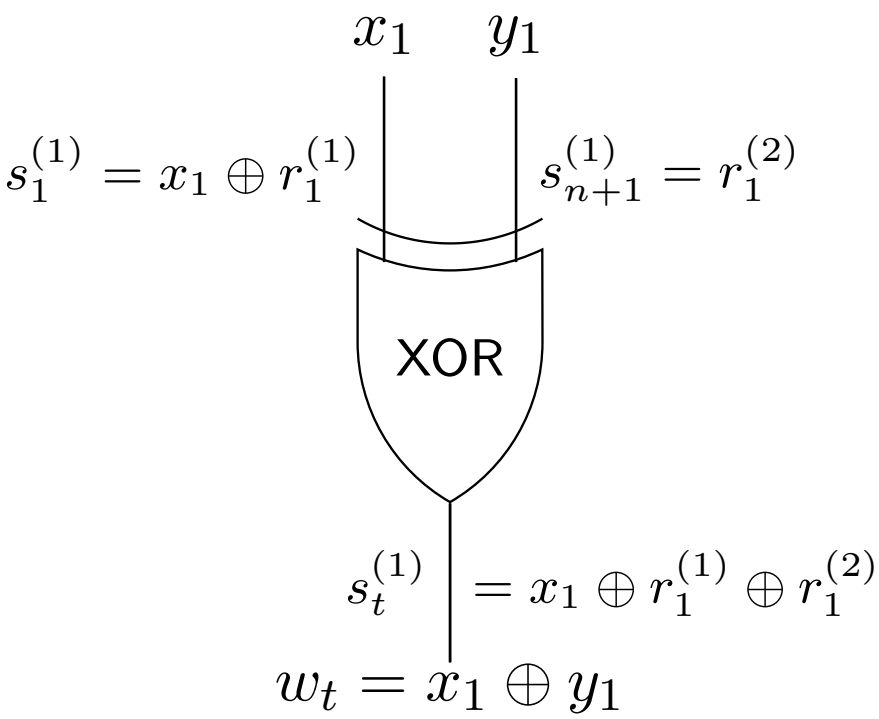
- Example: XOR gate with inputs x_1 and y_1.

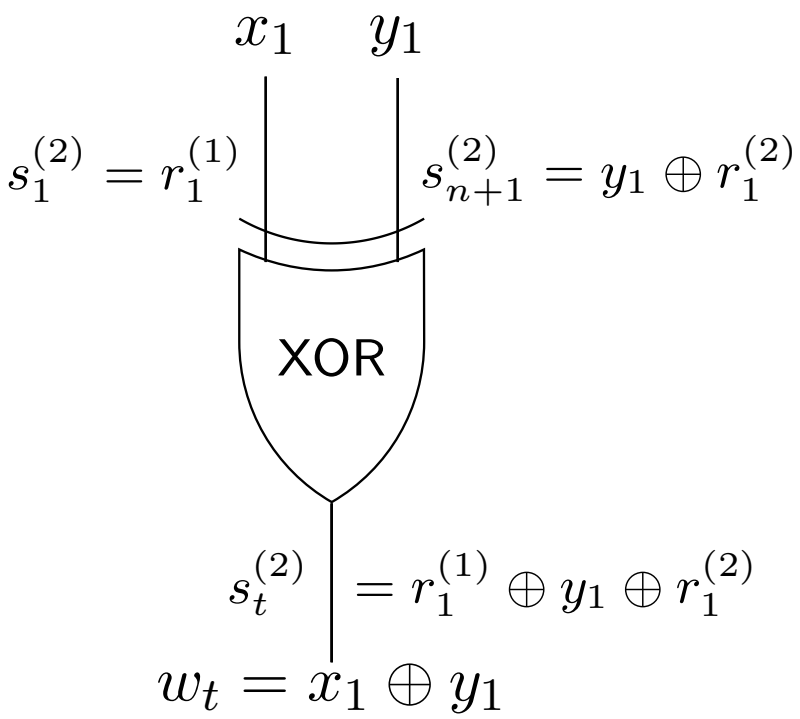




GMW Protocol: Local Evaluation of XOR

- Example: XOR gate with inputs x_1 and y_1.





- Notice: $s_t^{(1)} \oplus s_t^{(2)} = (x_1 \oplus r_1^{(1)} \oplus r_1^{(2)}) \oplus (r_1^{(1)} \oplus y_1 \oplus r_1^{(2)}) = x_1 \oplus y_1.$

$\| \; w_t$

GMW Protocol: Evaluation of AND

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

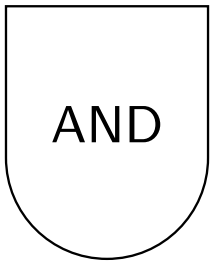
GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

P_1's view

GMW Protocol: Evaluation of AND

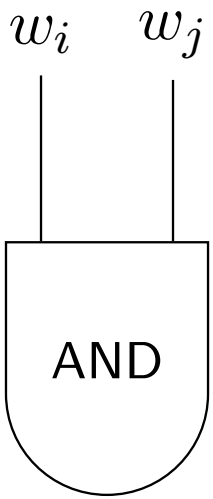
- AND gates require interaction between P_1 and P_2.



P_1's view

GMW Protocol: Evaluation of AND

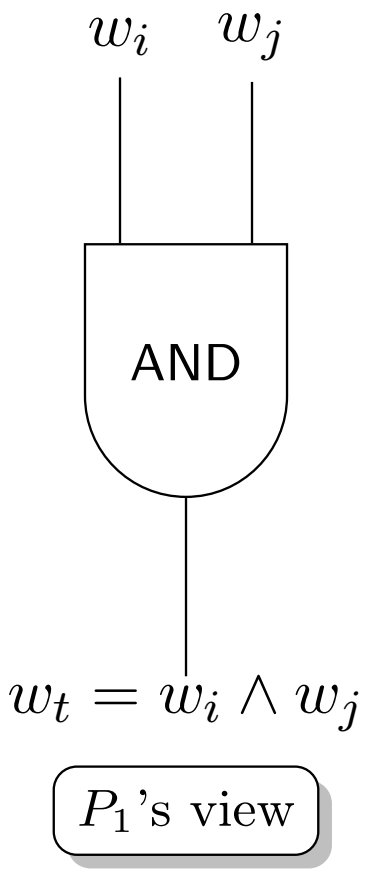
- AND gates require interaction between P_1 and P_2.



P_1's view

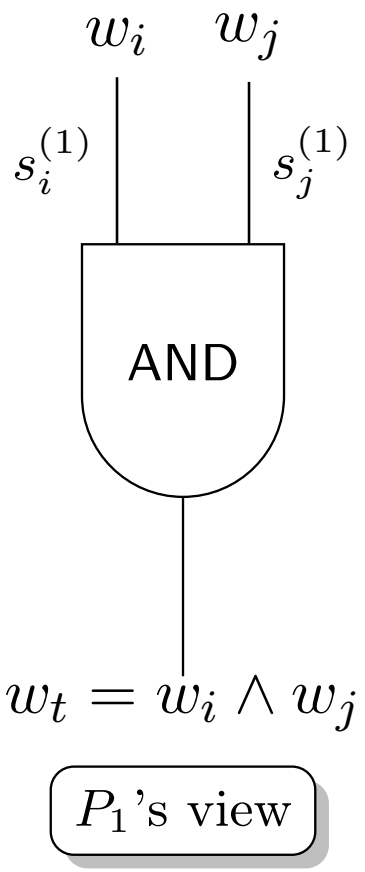
GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.



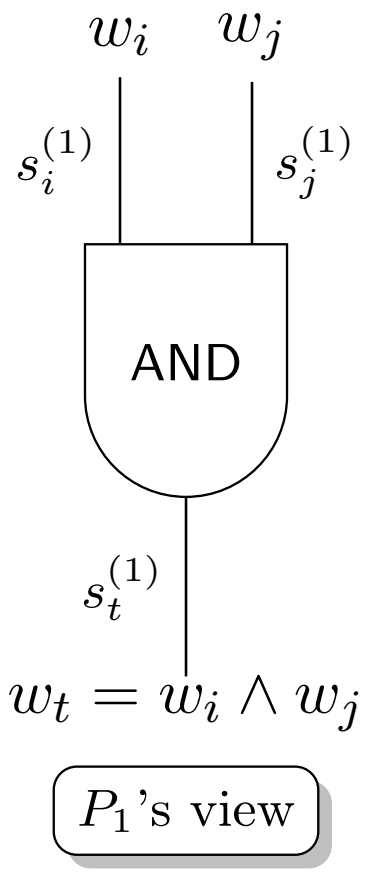
GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.



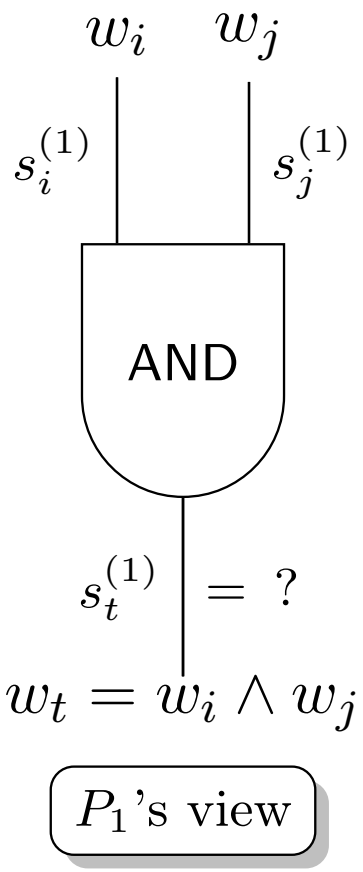
GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.



GMW Protocol: Evaluation of AND

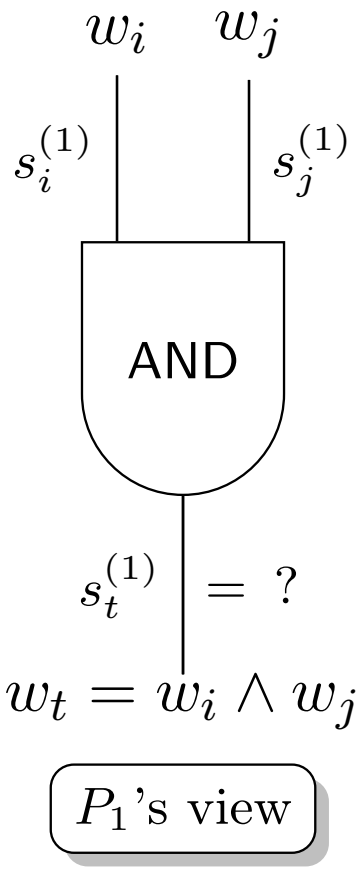
- AND gates require interaction between P_1 and P_2.



GMW Protocol: Evaluation of AND

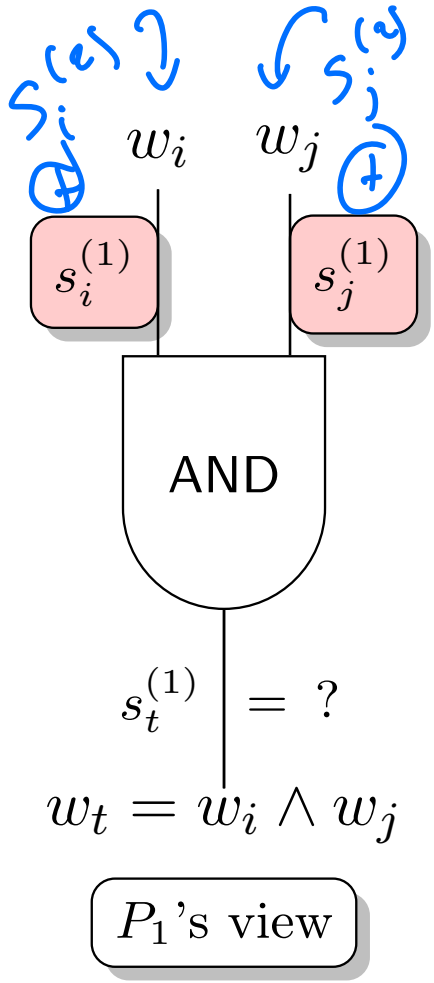
- AND gates require interaction between P_1 and P_2.

P_1's View:



GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

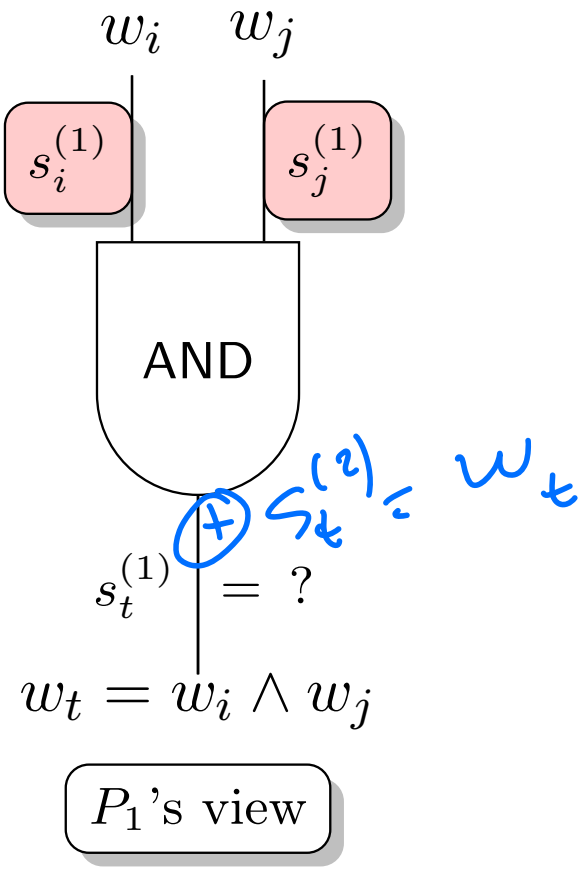


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

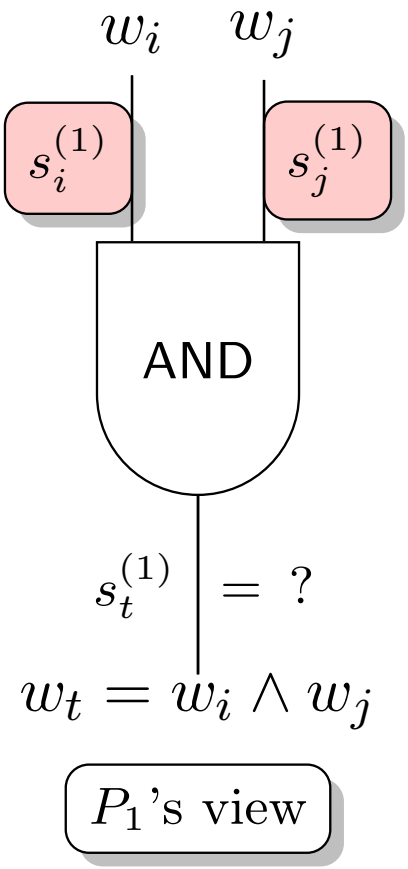


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

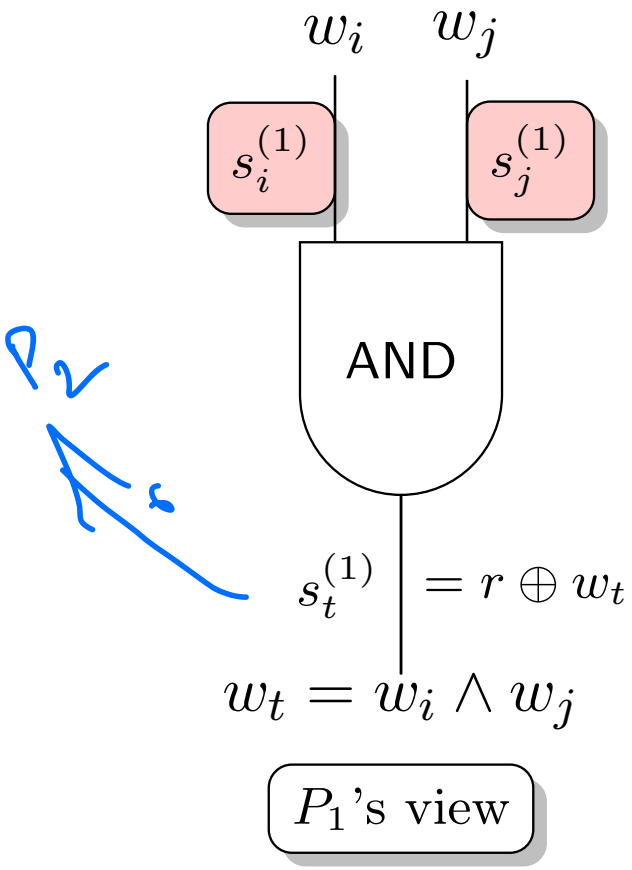


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

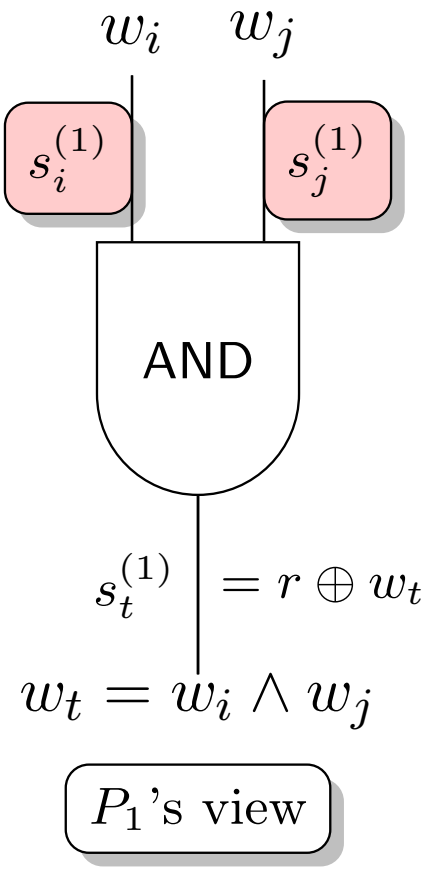


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

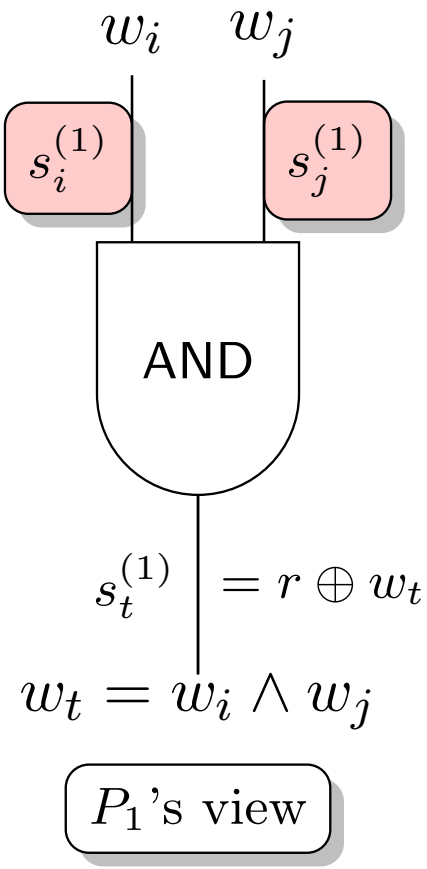


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.
  - **Why can't we do this?**

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

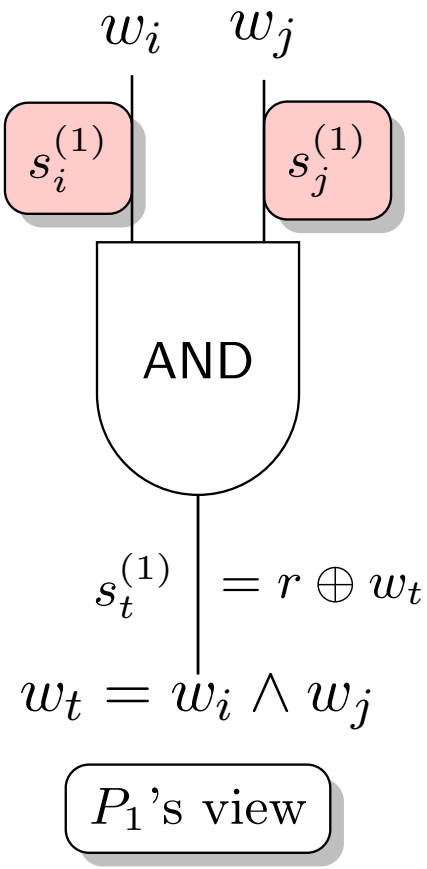


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.
  - **Why can't we do this?**
- Solution:

GMW Protocol: Evaluation of AND

- AND gates require interaction between P_1 and P_2.

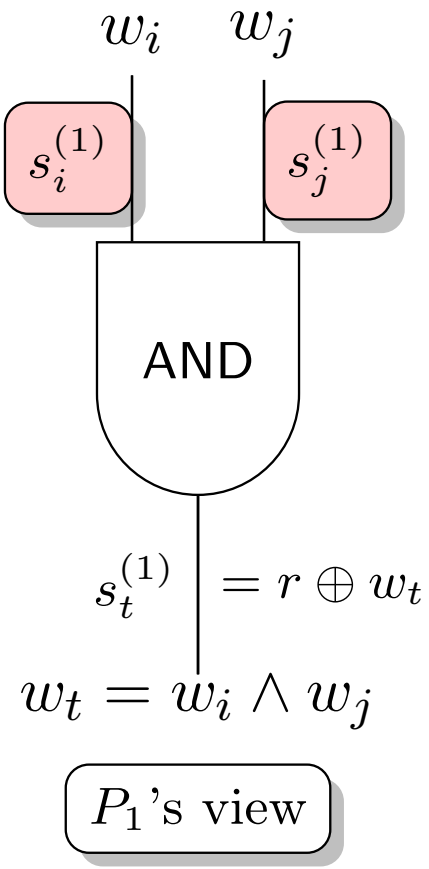


P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.
  - **Why can't we do this?**
- Solution:
  - P_1 prepares a secret share for each of P_2's possible inputs.

GMW Protocol: Evaluation of AND

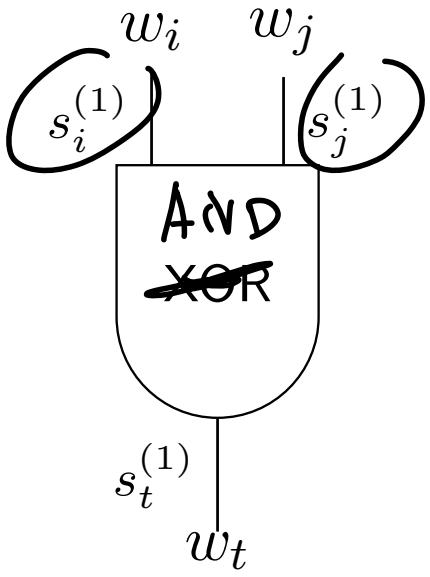
- AND gates require interaction between P_1 and P_2.



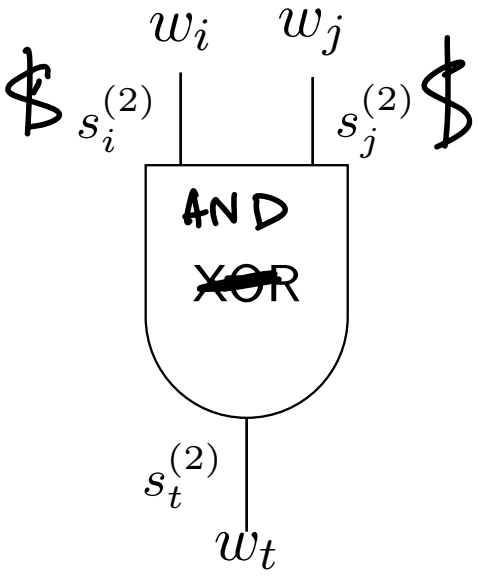
P_1's View:

- Shares $s_i^{(1)}$ and $s_j^{(1)}$ are fixed.
- There are 4 possible values for P_2's shares $(s_i^{(2)}, s_j^{(2)}) \in \{0,1\}^2$.
- If P_1 knew P_2's shares, it could compute w_t by reconstructing w_i and w_j, then re-share w_t with P_2 using $r \xleftarrow{\$} \{0,1\}$.
  - **Why can't we do this?**
- Solution:
  - P_1 prepares a secret share for each of P_2's possible inputs.
  - P_1 and P_2 then run a 1-out-of-4 OT to give P_2 the correct shares.

GMW Protocol: Evaluation of AND

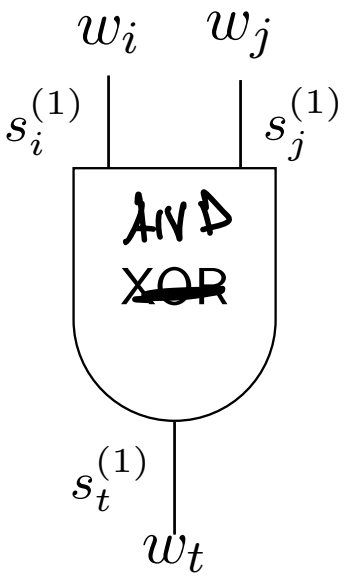


P_1's view

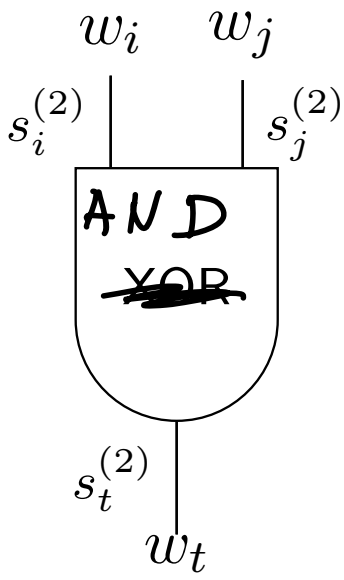


P_2's view

GMW Protocol: Evaluation of AND



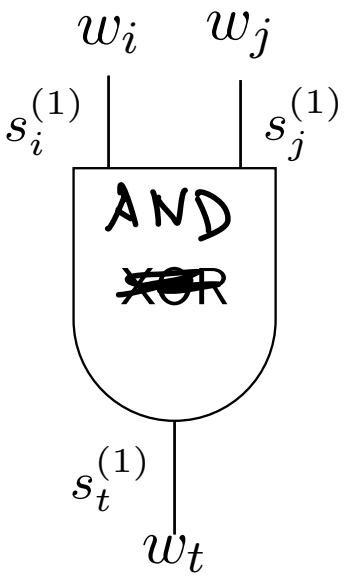
P_1's view



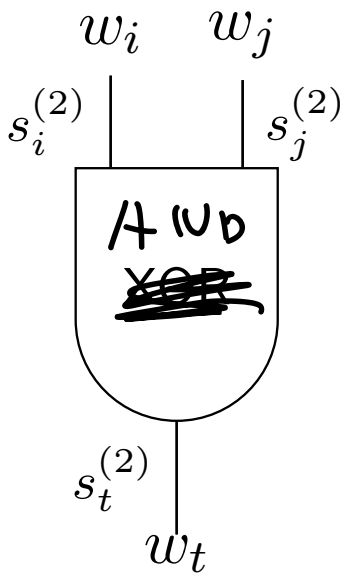
P_2's view

- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$

GMW Protocol: Evaluation of AND



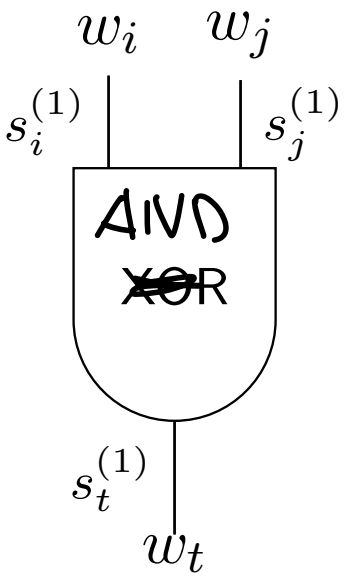
P_1's view



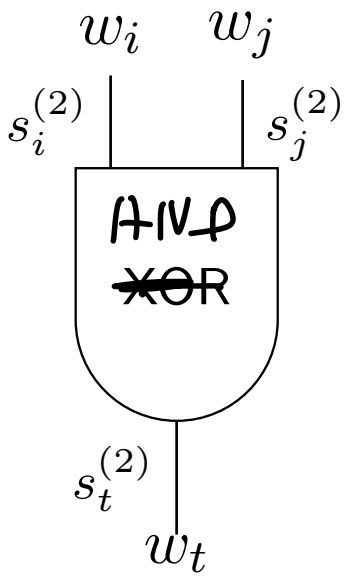
P_2's view

- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.
  - Know that: $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.

GMW Protocol: Evaluation of AND



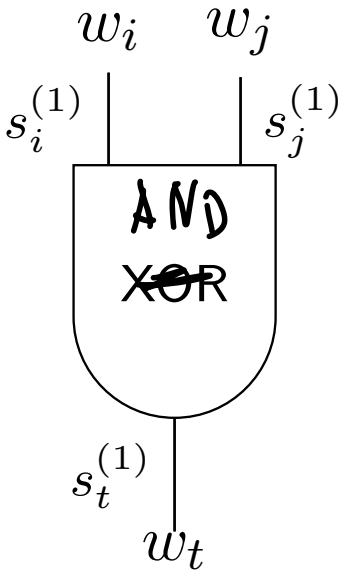
P_1's view



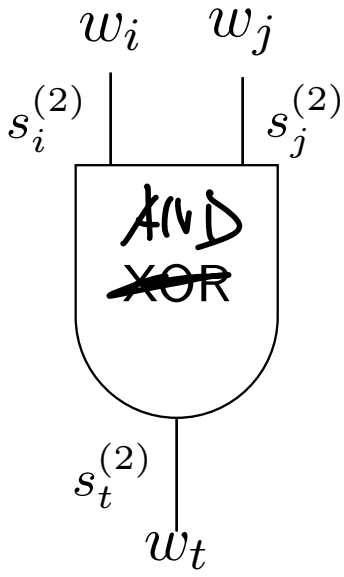
P_2's view

- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.
  - Know that: $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.
- Idea: need to construct new secret sharing of w_t without knowing its value.

GMW Protocol: Evaluation of AND



P_1's view

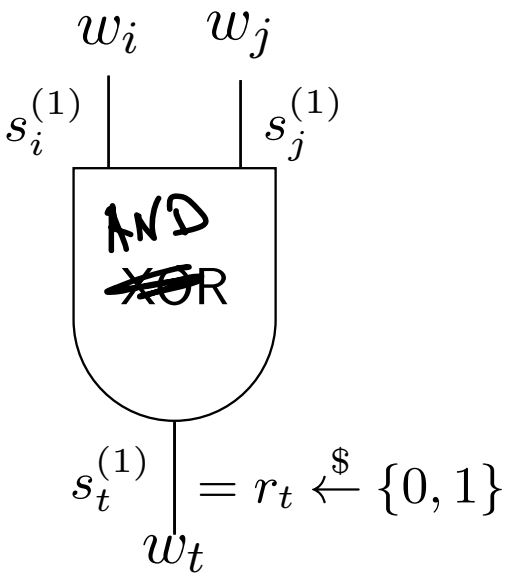


P_2's view

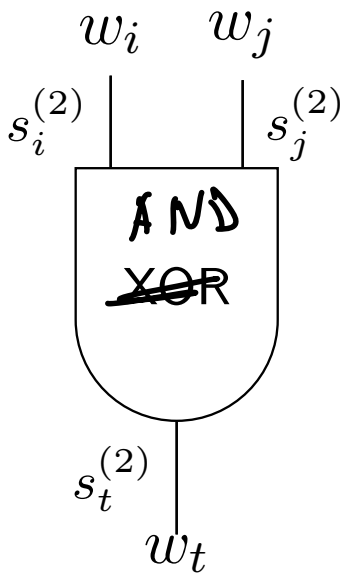
- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.
  - Know that: $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.

- Idea: need to construct new secret sharing of w_t without knowing its value.
  - P_1 samples $r_t \xleftarrow{\$} \{0,1\}$, sets this as share for $s_t^{(1)}$.

GMW Protocol: Evaluation of AND



P_1's view

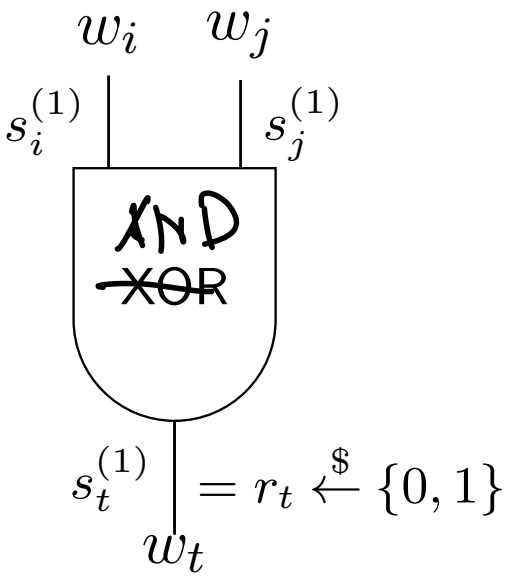


P_2's view

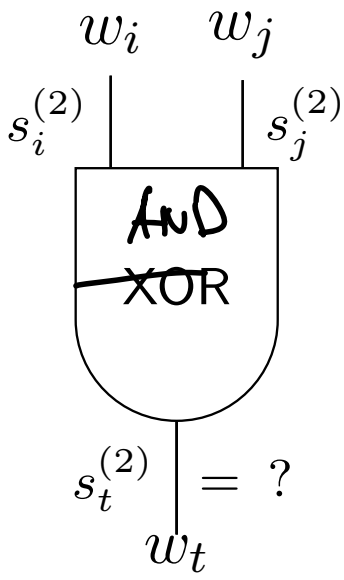
- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.
  - Know that: $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.

- Idea: need to construct new secret sharing of w_t without knowing its value.
  - P_1 samples $r_t \xleftarrow{\$} \{0,1\}$, sets this as share for $s_t^{(1)}$.

GMW Protocol: Evaluation of AND



P_1's view



P_2's view

- Need $s_t^{(1)}$ and $s_t^{(2)}$ such that $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.
  - Know that: $s_i^{(1)} \oplus s_i^{(2)} = w_i$ and $s_j^{(1)} \oplus s_j^{(2)} = w_j$.

- Idea: need to construct new secret sharing of w_t without knowing its value.
  - P_1 samples $r_t \xleftarrow{\$} \{0,1\}$, sets this as share for $s_t^{(1)}$.

GMW Protocol: Evaluation of AND

GMW Protocol: Evaluation of AND

P_1 Defines this function

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.

T — fixed for P_1

variables

$S : \{0,1\}^2 \to \{0,1\}$

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

$\mathcal{F}_{\mathsf{OT}}^{(1,4)}$

GMW Protocol: Evaluation of AND

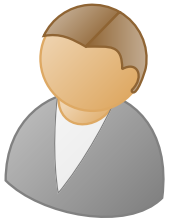
- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

P_1

$\mathcal{F}_{\mathsf{OT}}^{(1,4)}$



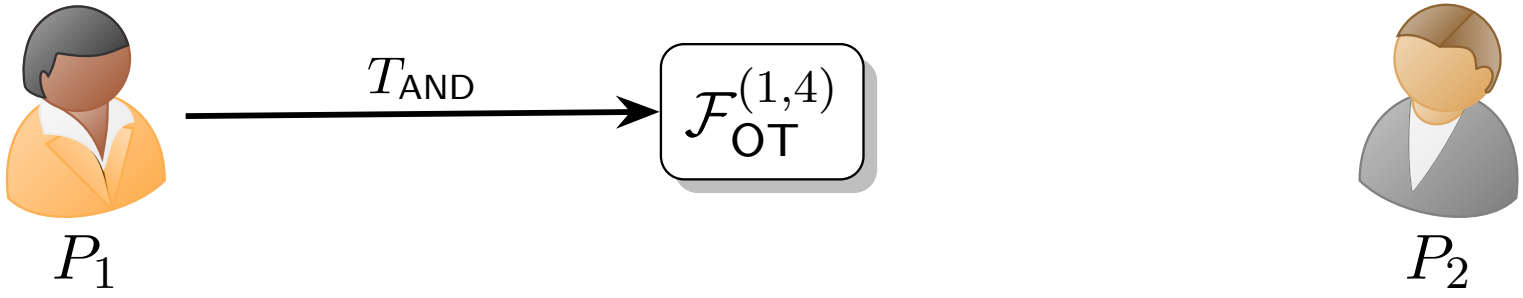
P_2

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

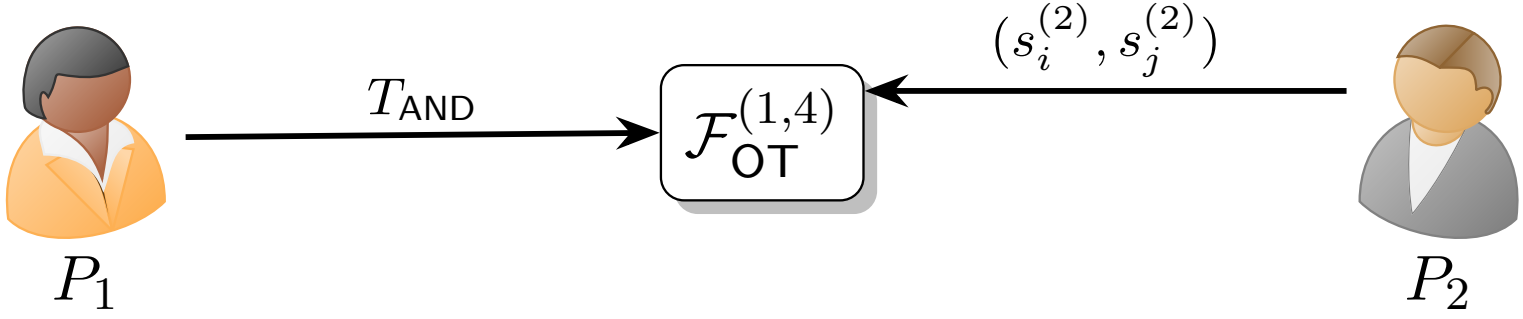


GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

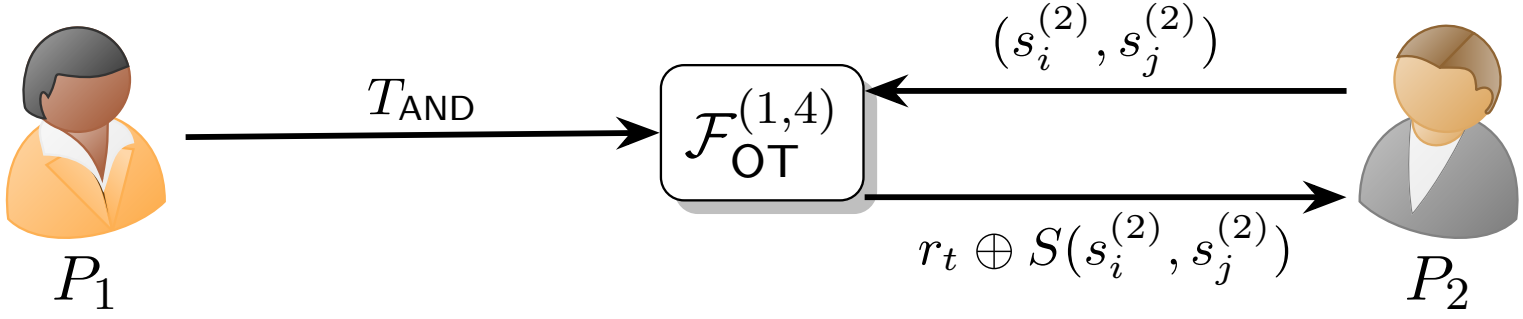


GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

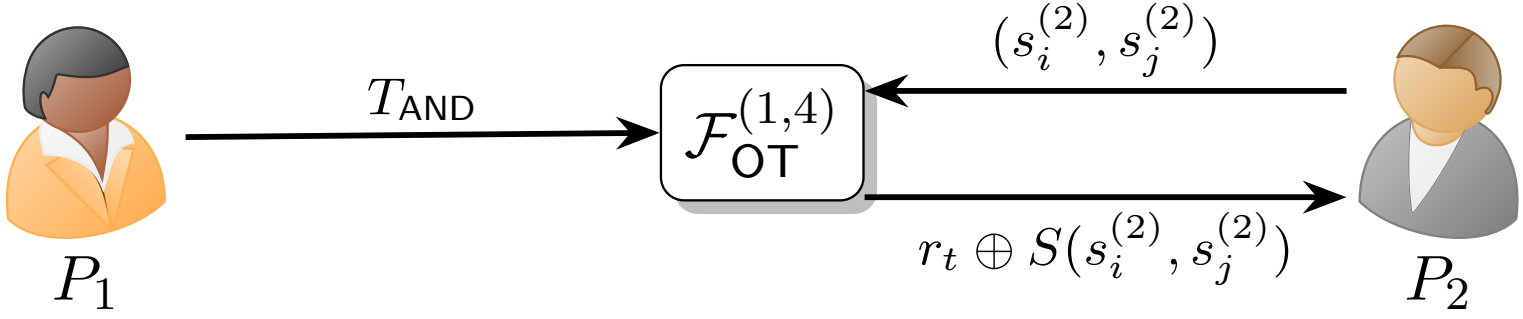


GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.



- P_2 sets $s_t^{(2)} = r_t \oplus S(s_i^{(2)}, s_j^{(2)})$.

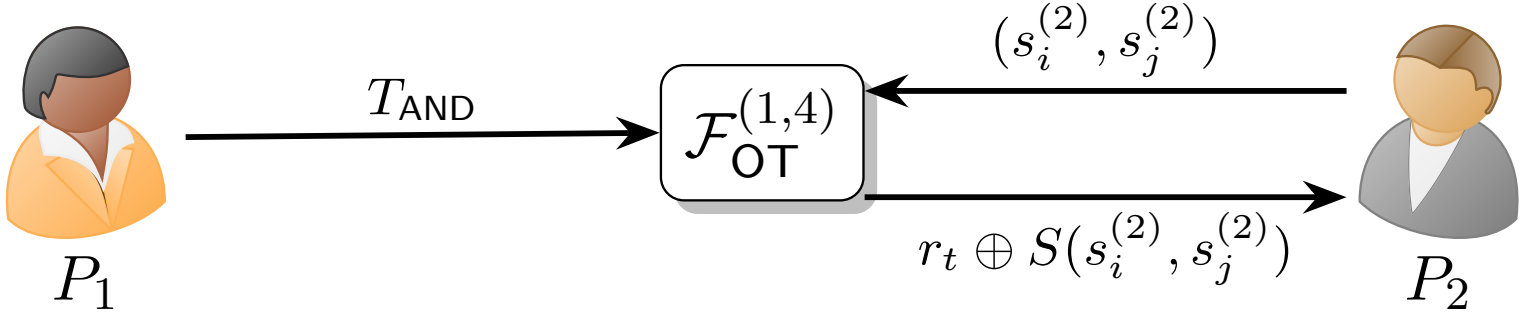
$s_t^{(1)} = r_t$

GMW Protocol: Evaluation of AND

- Let $S := S_{s_i^{(1)}, s_j^{(1)}}(s_i^{(2)}, s_j^{(2)}) = (s_i^{(1)} \oplus s_i^{(2)}) \wedge (s_j^{(1)} \oplus s_j^{(2)})$.
- P_1 prepares the table

$$T_{\mathsf{AND}} = \begin{bmatrix} r_t \oplus S(0,0) \\ r_t \oplus S(0,1) \\ r_t \oplus S(1,0) \\ r_t \oplus S(1,1) \end{bmatrix}.$$

- P_1 and P_2 run $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.



- P_2 sets $s_t^{(2)} = r_t \oplus S(s_i^{(2)}, s_j^{(2)})$.
- P_1 has $s_t^{(1)} = r_t$, implying $s_t^{(1)} \oplus s_t^{(2)} = w_i \wedge w_j$.

GMW Protocol: Output Wires

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.

- Finally, for every output wire w_ℓ:

P_1 : $s_\ell^{(1)}$

P_2 : $s_\ell^{(2)}$

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.

- Finally, for every output wire w_ℓ:
  - P_1 sends $s_\ell^{(1)}$ to P_2; and

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.

- Finally, for every output wire w_ℓ:
  - P_1 sends $s_\ell^{(1)}$ to P_2; and
  - P_2 sends $s_\ell^{(2)}$ to P_1.

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.

- Finally, for every output wire w_ℓ:
  - P_1 sends $s_\ell^{(1)}$ to P_2; and
  - P_2 sends $s_\ell^{(2)}$ to P_1.

- Parties locally compute $w_\ell = s_\ell^{(1)} \oplus s_\ell^{(2)}$.

GMW Protocol: Output Wires

- P_1 and P_2 evaluate all wires/gates in the circuit $\mathcal{C}$ as described in the previous sections.
  - I.e., locally compute NOT and XOR gates, communicate using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ for every AND gate.
- Finally, for every output wire w_ℓ:
  - P_1 sends $s_\ell^{(1)}$ to P_2; and
  - P_2 sends $s_\ell^{(2)}$ to P_1.
- Parties locally compute $w_\ell = s_\ell^{(1)} \oplus s_\ell^{(2)}$.
- Parties have learned $\mathcal{F}(\mathbf{x}, \mathbf{y})$!

GMW Protocol: Communication Costs

GMW Protocol: Communication Costs

- Setup Phase.

GMW Protocol: Communication Costs

- Setup Phase.

- Output Phase.

GMW Protocol: Communication Costs

- Setup Phase.
- Output Phase.
- Evaluation Phase.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
- Output Phase.
- Evaluation Phase.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
- Evaluation Phase.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
- Evaluation Phase.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - Each AND gate requires executing $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - Each AND gate requires executing $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - Each AND gate requires executing $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.
  - The protocol requires $s_{\mathsf{AND}} \cdot \mathsf{rounds}(\mathcal{F}_{\mathsf{OT}}^{(1,4)}) + 2$ rounds of communication.

GMW Protocol: Communication Costs

- Setup Phase.
  - P_1 samples and sends n random bits $\mathbf{r}^{(1)}$.
  - Likewise, P_2 samples and sends n random bits $\mathbf{r}^{(2)}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - P_1 and P_2 then exchange s_{out} bits with each other.
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - Each AND gate requires executing $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.
  - The protocol requires $s_{\mathsf{AND}} \cdot \mathsf{rounds}(\mathcal{F}_{\mathsf{OT}}^{(1,4)}) + 2$ rounds of communication.
  - Parties exchange $n + s_{\mathsf{out}} + s_{\mathsf{AND}} \cdot \mathsf{bits}(\mathcal{F}_{\mathsf{OT}}^{(1,4)})$ bits.

GMW Protocol: Generalization to m Parties

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$: $i \mapsto j$
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.

j ↦ i

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.
  - P_i computes $\mathbf{s}^{(i,i)} = \mathbf{x}_i^{(i)} \oplus \bigoplus_{j \neq i} \mathbf{r}^{(i,j)}$.

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.
  - P_i computes $\mathbf{s}^{(i,i)} = \mathbf{x}_i \oplus \bigoplus_{j \neq i} \mathbf{r}^{(i,j)}$.
  - P_i uses $\mathbf{s}^{(i)} = (\mathbf{s}^{(i,1)}, \ldots, \mathbf{s}^{(i,m)})$ as its input to the circuit $\mathcal{C}$.

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.
  - P_i computes $\mathbf{s}^{(i,i)} = \mathbf{x}_i \oplus \bigoplus_{j \neq i} \mathbf{r}^{(i,j)}$.
  - P_i uses $\mathbf{s}^{(i)} = (\mathbf{s}^{(i,1)}, \ldots, \mathbf{s}^{(i,m)})$ as its input to the circuit $\mathcal{C}$.

- For evaluating NOT gates: P_1 flips its input share bit, all other players do nothing (output share is equal to input share).

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.
  - P_i computes $\mathbf{s}^{(i,i)} = \mathbf{x}_i \oplus \bigoplus_{j \neq i} \mathbf{r}^{(i,j)}$.
  - P_i uses $\mathbf{s}^{(i)} = (\mathbf{s}^{(i,1)}, \ldots, \mathbf{s}^{(i,m)})$ as its input to the circuit $\mathcal{C}$.
- For evaluating NOT gates: P_1 flips its input share bit, all other players do nothing (output share is equal to input share).
- For evaluating XOR gates: each P_i simply computes the XOR of their shares locally.

GMW Protocol: Generalization to m Parties

- Suppose each party P_i has input $\mathbf{x}^{(i)} \in \{0,1\}^n$.
- For all $i \in [m]$:
  - P_i samples $\mathbf{r}^{(i,j)} \xleftarrow{\$} \{0,1\}^n$ for all $j \neq i$ and sends $\mathbf{r}^{(i,j)}$ to party j.
  - P_i computes $\mathbf{s}^{(i,j)} = \mathbf{r}^{(j,i)}$ for all $j \neq i$.
  - P_i computes $\mathbf{s}^{(i,i)} = \mathbf{x}_i \oplus \bigoplus_{j \neq i} \mathbf{r}^{(i,j)}$.
  - P_i uses $\mathbf{s}^{(i)} = (\mathbf{s}^{(i,1)}, \ldots, \mathbf{s}^{(i,m)})$ as its input to the circuit $\mathcal{C}$.
- For evaluating NOT gates: P_1 flips its input share bit, all other players do nothing (output share is equal to input share).
- For evaluating XOR gates: each P_i simply computes the XOR of their shares locally.
- Evaluating AND gates is more complicated; we'll come back to this.

GMW Protocol: 3-Player Example

GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.

GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.

GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.

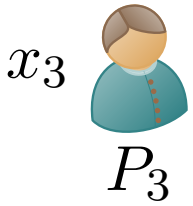
P_1

P_2

P_3

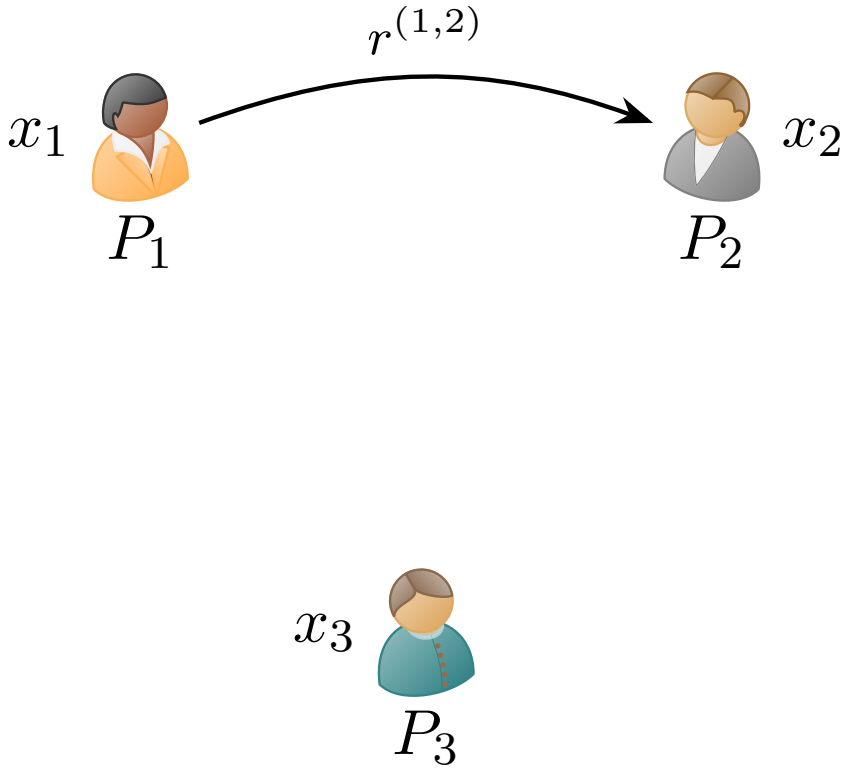
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



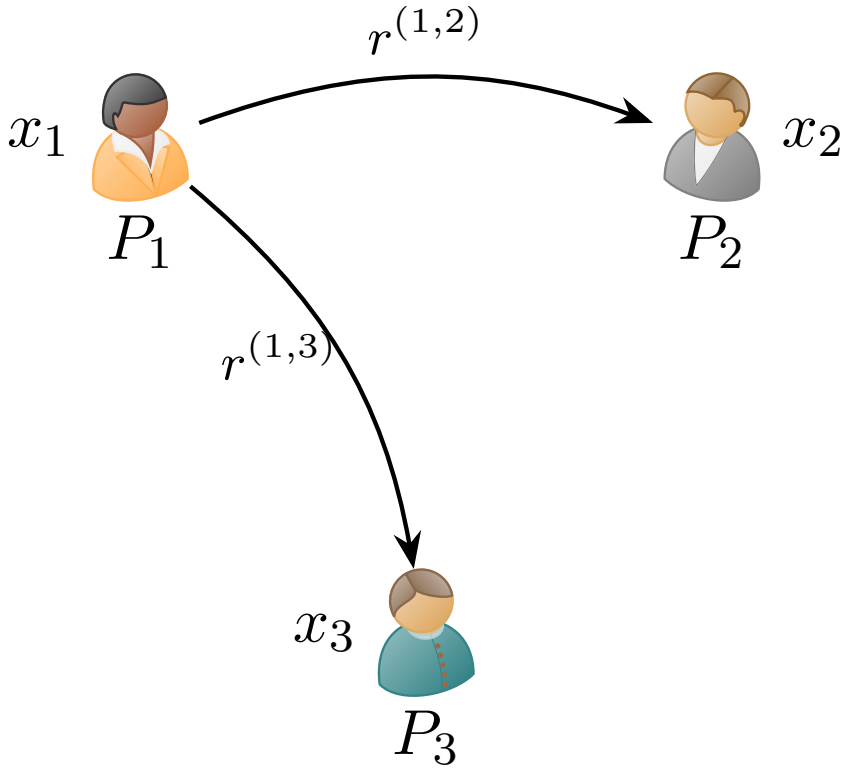
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



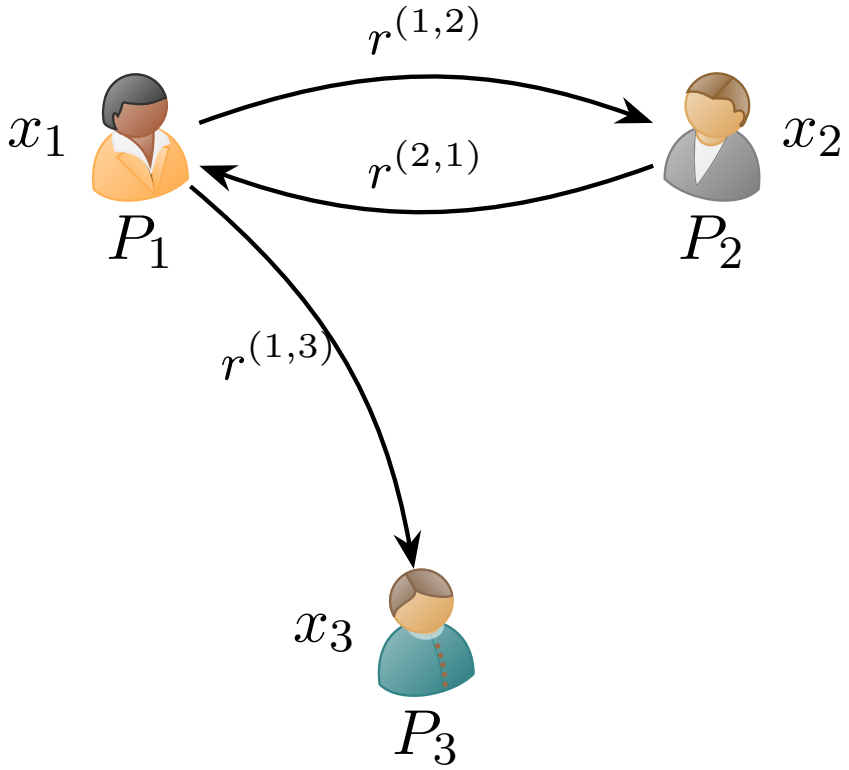
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



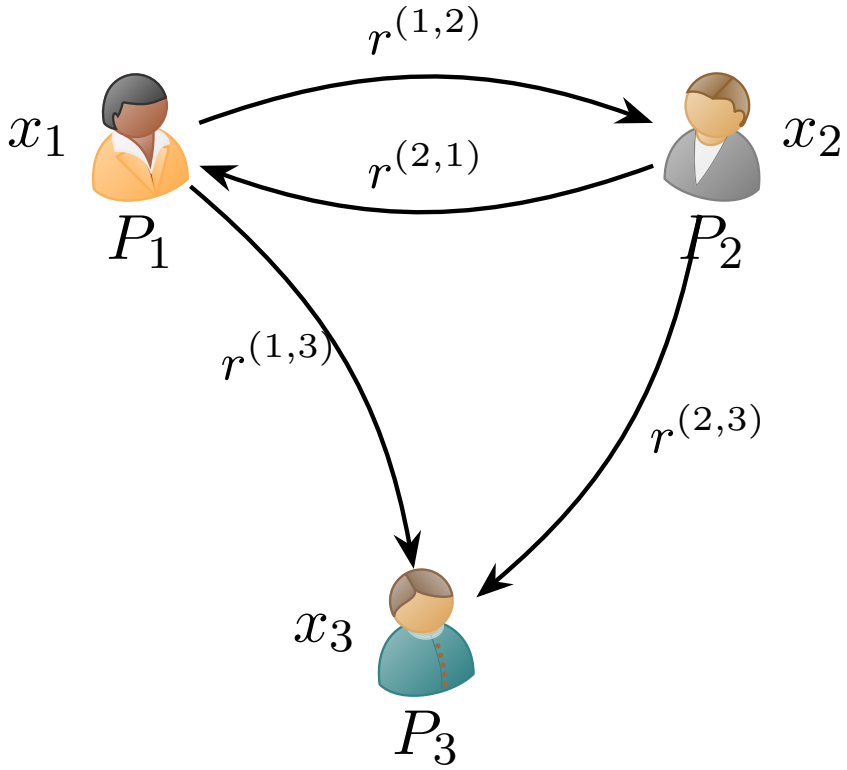
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



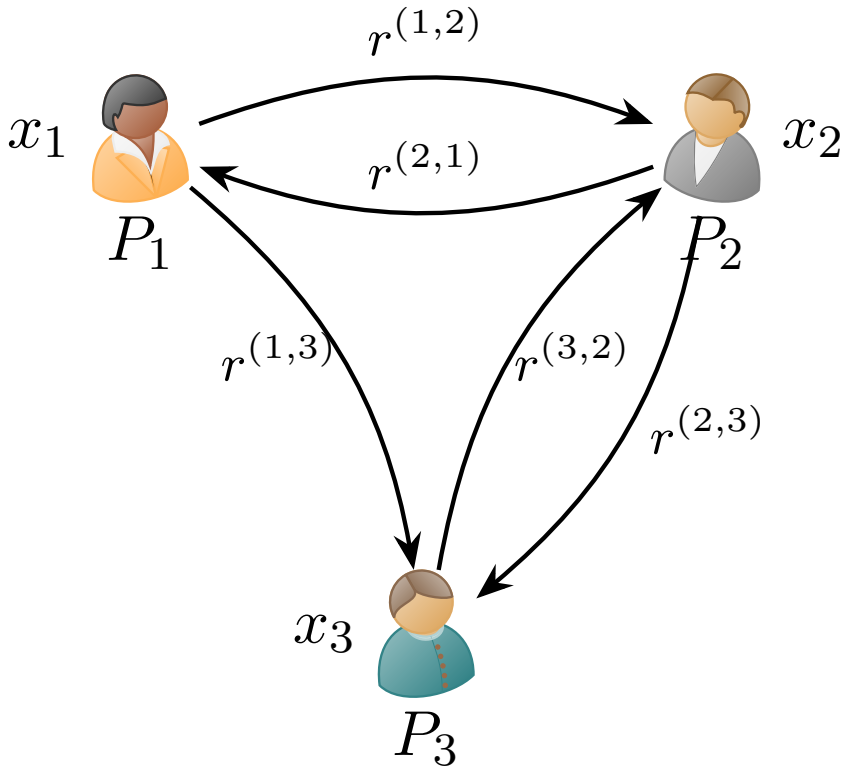
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



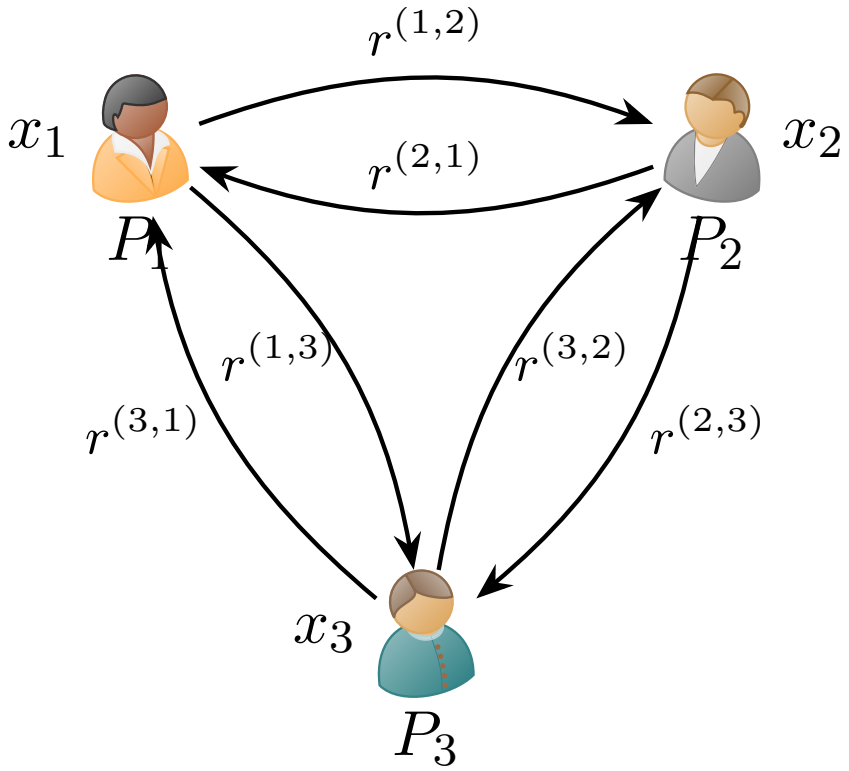
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



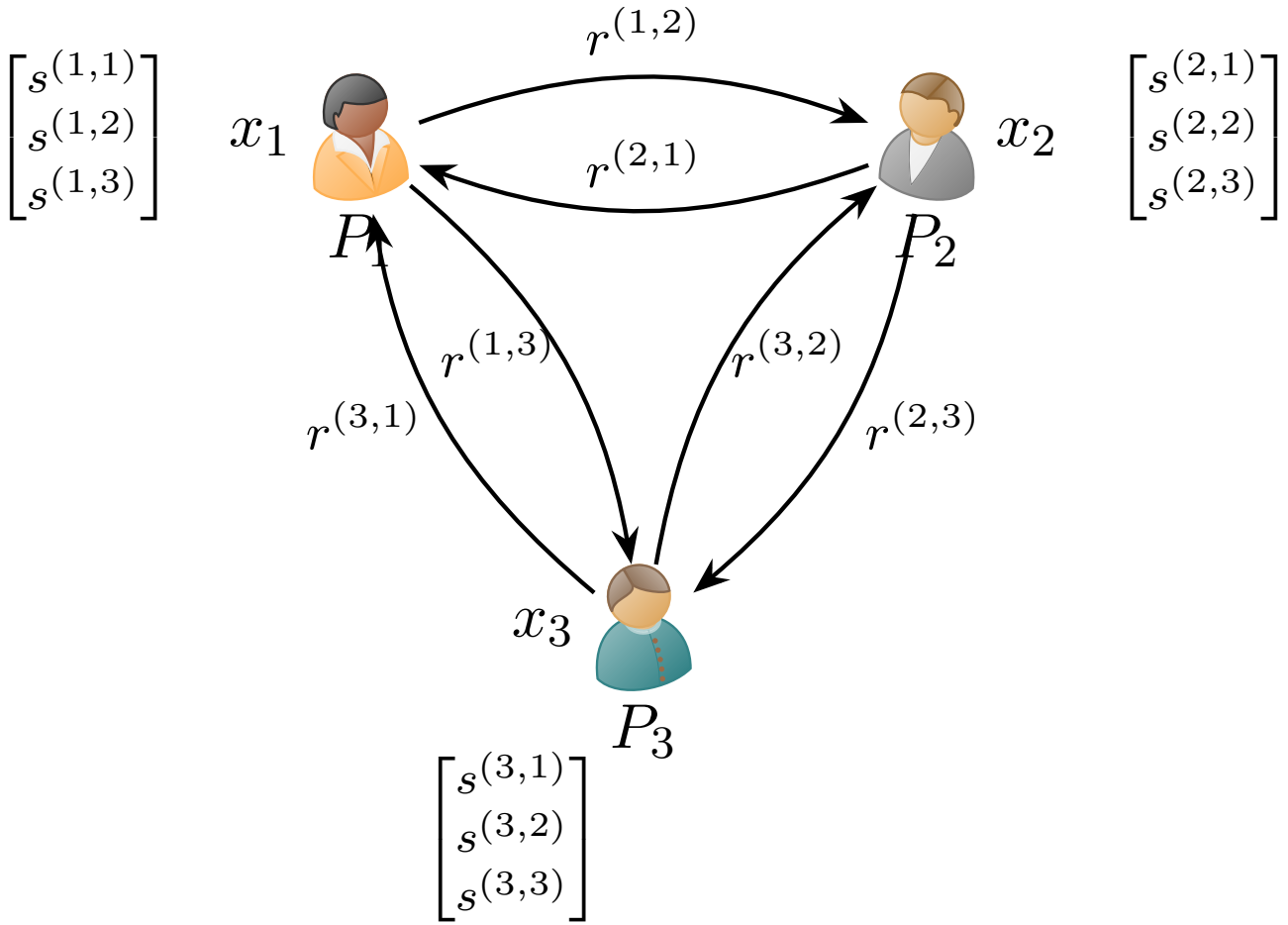
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\mathsf{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



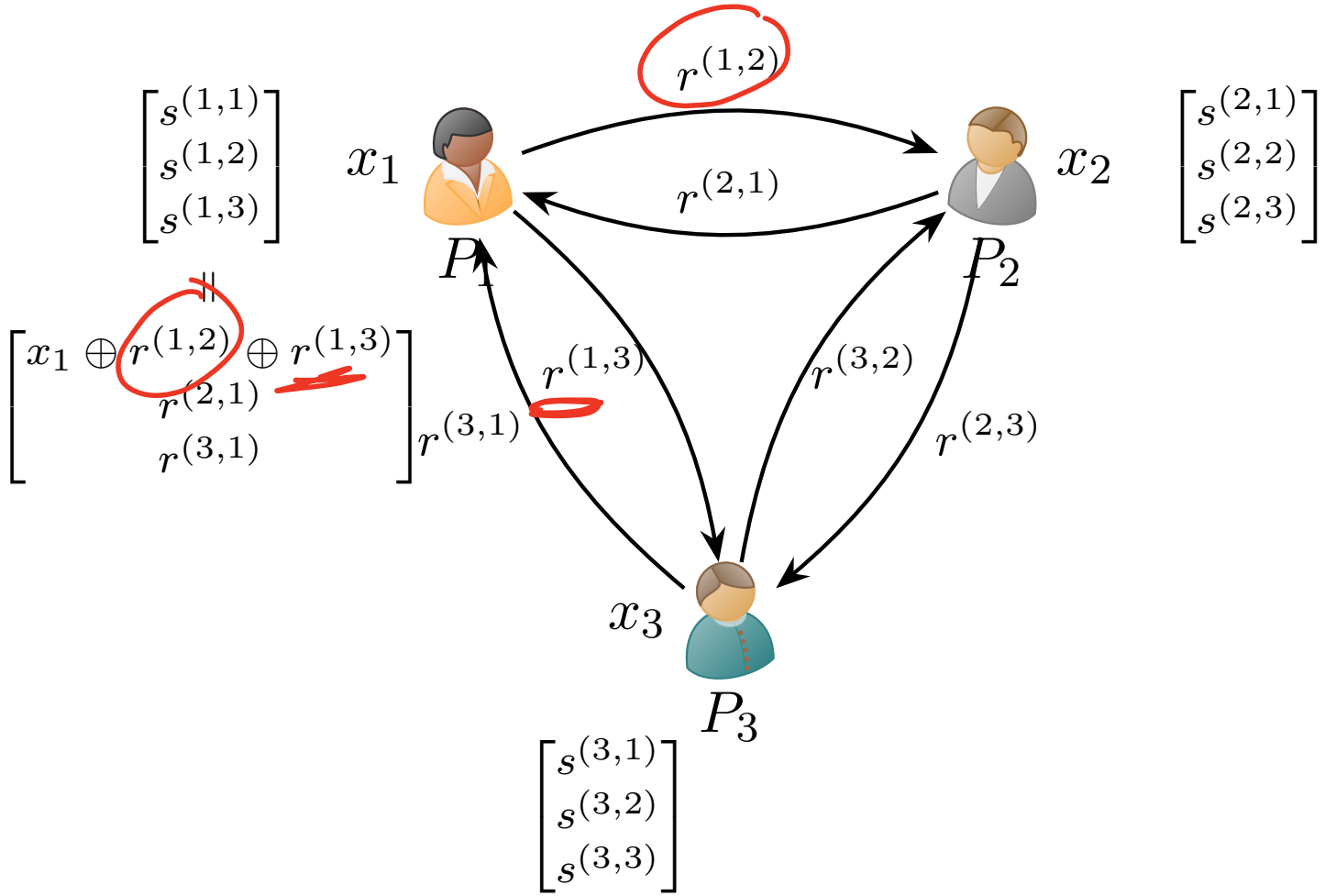
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\text{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



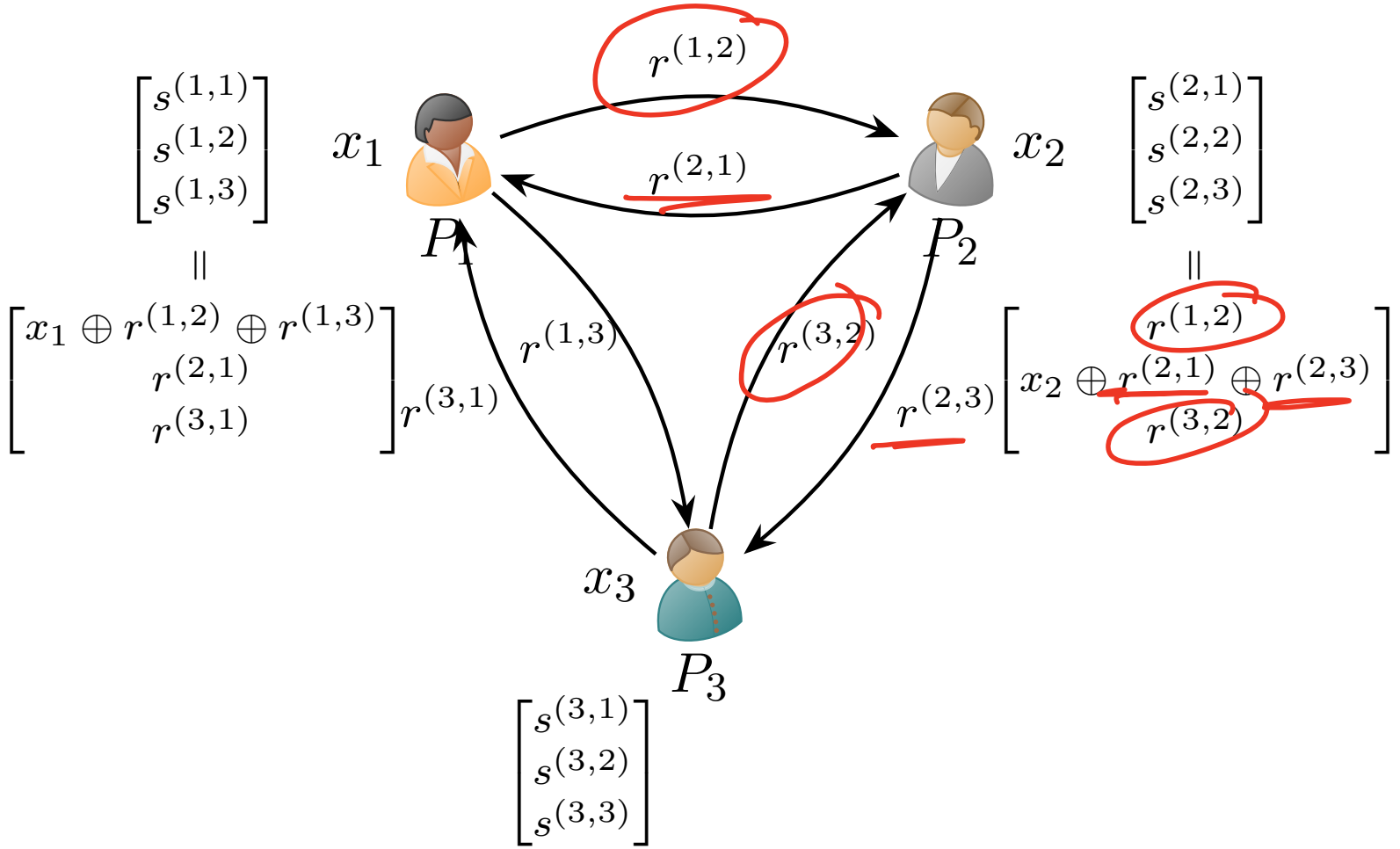
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\text{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



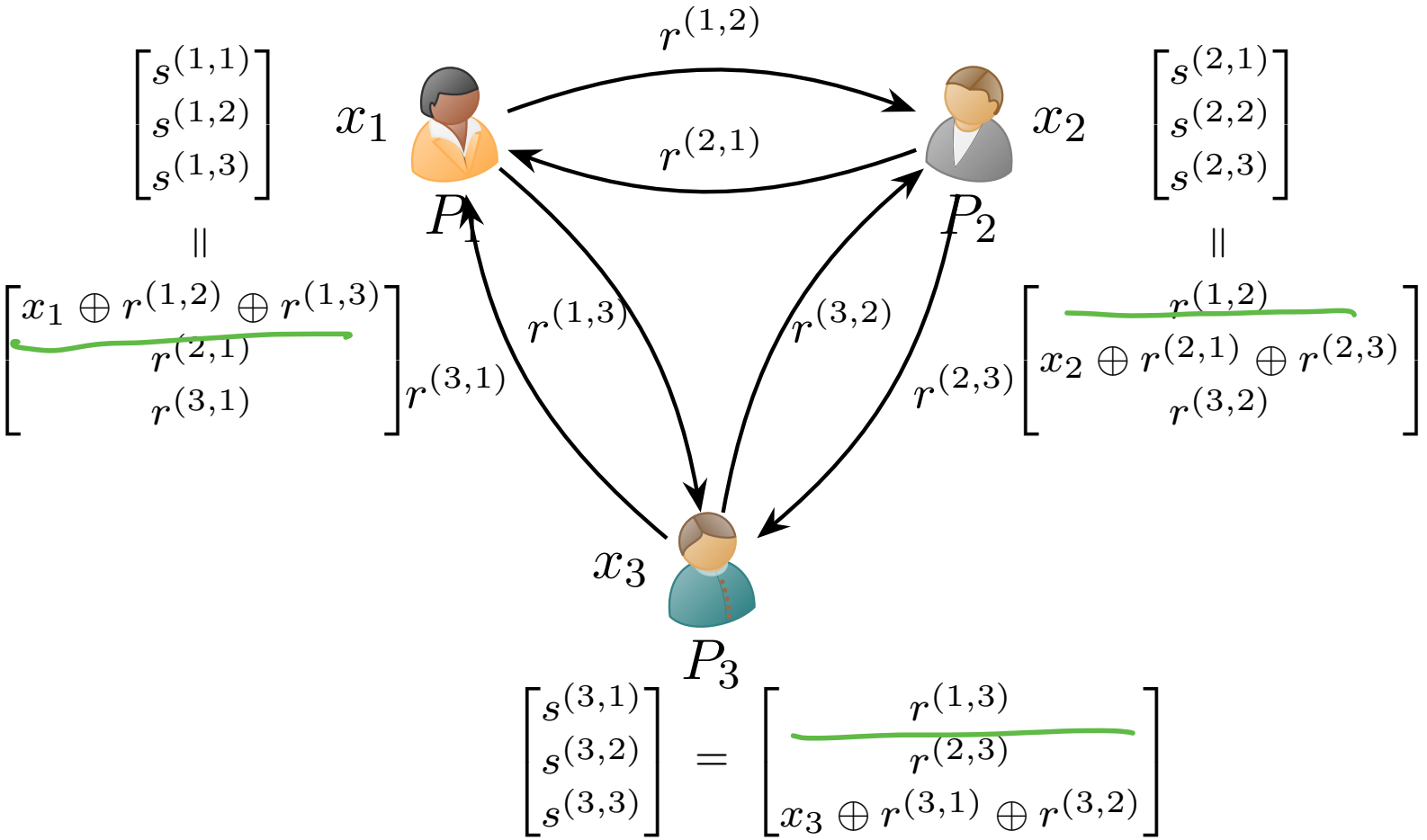
GMW Protocol: 3-Player Example

- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\text{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.

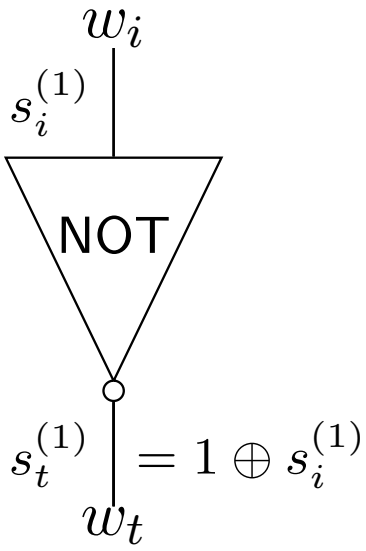


GMW Protocol: 3-Player Example

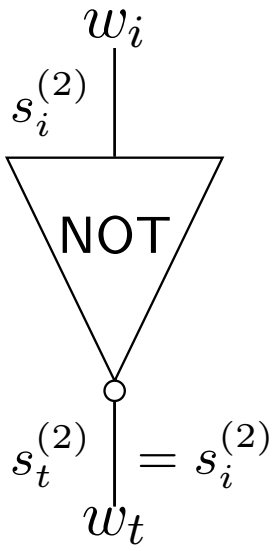
- Suppose that $\mathcal{C}\colon \{0,1\}^3 \to \{0,1\}^{s_{\text{out}}}$.
- Consider a 3-Player version of the GMW Protocol where each party gets a single bit as input.



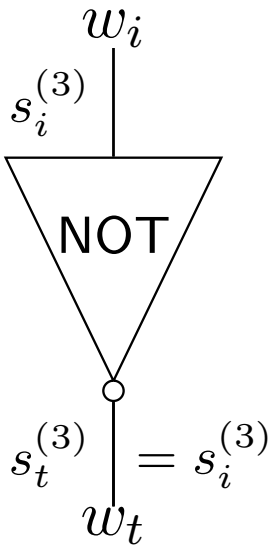
GMW Protocol: 3-Player NOT Gate



P_1's view



P_2's view

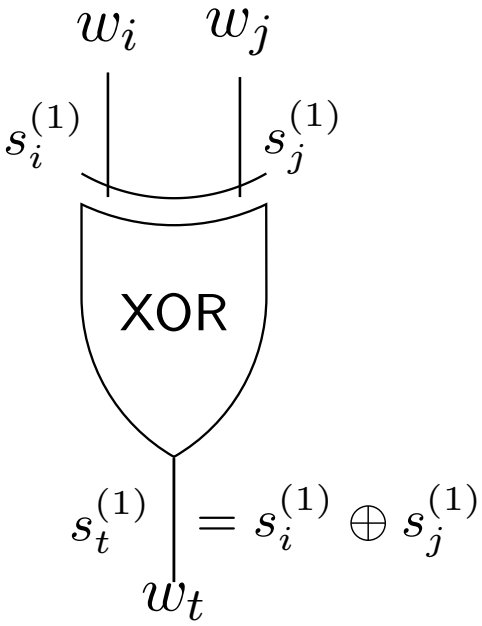


P_3's view

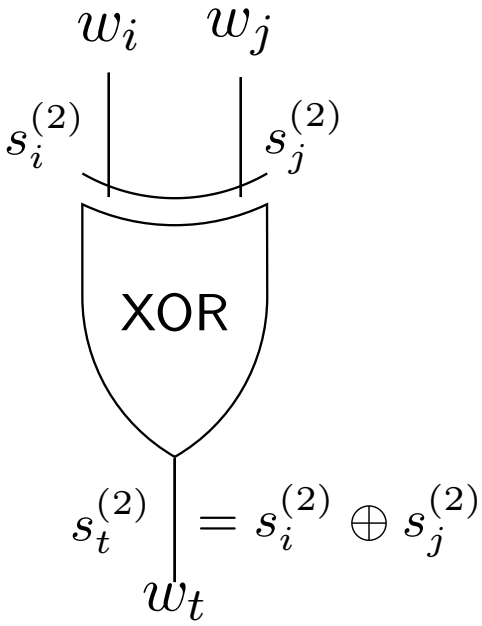
$$s_i^{(1)} \oplus s_i^{(2)} \oplus s_i^{(3)} = w_i$$

$$s_t^{(1)} \oplus s_t^{(2)} \oplus s_t^{(3)} = 1 \oplus (s_i^{(1)} \oplus s_i^{(2)} \oplus s_i^{(3)}) = 1 \oplus w_i$$

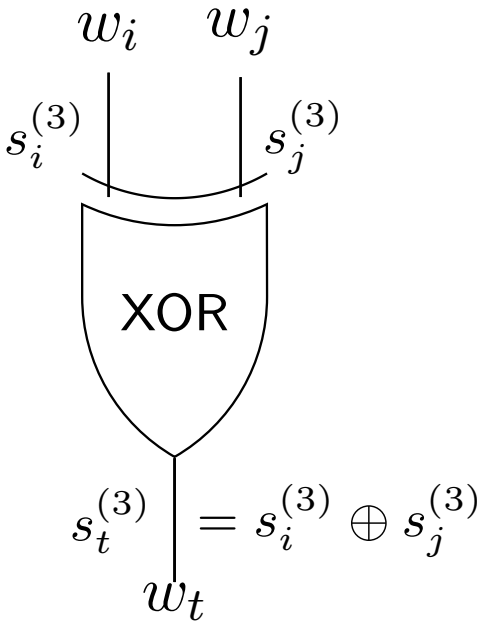
GMW Protocol: 3-Player XOR Gate



P_1's view



P_2's view



P_3's view

$s_i^{(1)} \oplus s_i^{(2)} \oplus s_i^{(3)} = w_i \quad s_j^{(1)} \oplus s_j^{(2)} \oplus s_j^{(3)} = w_j$

GMW Protocol: m-Player AND Gate

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.
- Suppose we evaluate an AND Gate $w_t = w_i \wedge w_j$.

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.
- Suppose we evaluate an AND Gate $w_t = w_i \wedge w_j$.
  - Let P_k have shares $s_i^{(k)}$ and $s_j^{(k)}$ for input wires w_i and w_j, for all $k \in [m]$.

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.
- Suppose we evaluate an AND Gate $w_t = w_i \wedge w_j$.
  - Let P_k have shares $s_i^{(k)}$ and $s_j^{(k)}$ for input wires w_i and w_j, for all $k \in [m]$.
- We use the following identity:

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.
- Suppose we evaluate an AND Gate $w_t = w_i \wedge w_j$.
  - Let P_k have shares $s_i^{(k)}$ and $s_j^{(k)}$ for input wires w_i and w_j, for all $k \in [m]$.

- We use the following identity:

$$w_t = w_i \wedge w_j = \left(s_i^{(1)} \oplus \cdots \oplus s_i^{(m)}\right) \wedge \left(s_j^{(1)} \oplus \cdots \oplus s_j^{(m)}\right)$$

GMW Protocol: m-Player AND Gate

- Evaluating AND Gates is more complicated.
- Suppose we evaluate an AND Gate $w_t = w_i \wedge w_j$.
  - Let P_k have shares $s_i^{(k)}$ and $s_j^{(k)}$ for input wires w_i and w_j, for all $k \in [m]$.

- We use the following identity:

$$w_t = w_i \wedge w_j = \left(s_i^{(1)} \oplus \cdots \oplus s_i^{(m)}\right) \wedge \left(s_j^{(1)} \oplus \cdots \oplus s_j^{(m)}\right)$$

$$= \left(\bigoplus_{k=1}^{m} s_i^{(k)} \wedge s_j^{(k)}\right) \oplus \left(\bigoplus_{\substack{k,k' \in [m] \\ k \neq k'}} s_i^{(k)} \wedge s_j^{(k')}\right).$$

GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

- Each party P_k locally computes $s_i^{(k)} \wedge s_j^{(k)}$.

GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

- Each party P_k locally computes $s_i^{(k)} \wedge s_j^{(k)}$.
  - This is a secret share of $\bigoplus_{k \in [m]} s_i^{(k)} \wedge s_j^{(k)}$.

GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

- Each party P_k locally computes $s_i^{(k)} \wedge s_j^{(k)}$.
  - This is a secret share of $\bigoplus_{k \in [m]} s_i^{(k)} \wedge s_j^{(k)}$.

- Each pair of parties P_k and $P_{k'}$ then jointly compute secret shares of $s_i^{(k)} \wedge s_j^{(k')}$.

GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

- Each party P_k locally computes $s_i^{(k)} \wedge s_j^{(k)}$.
  - This is a secret share of $\bigoplus_{k \in [m]} s_i^{(k)} \wedge s_j^{(k)}$.

- Each pair of parties P_k and $P_{k'}$ then jointly compute secret shares of $s_i^{(k)} \wedge s_j^{(k')}$.
  - This is done as in the 2-Player version, using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

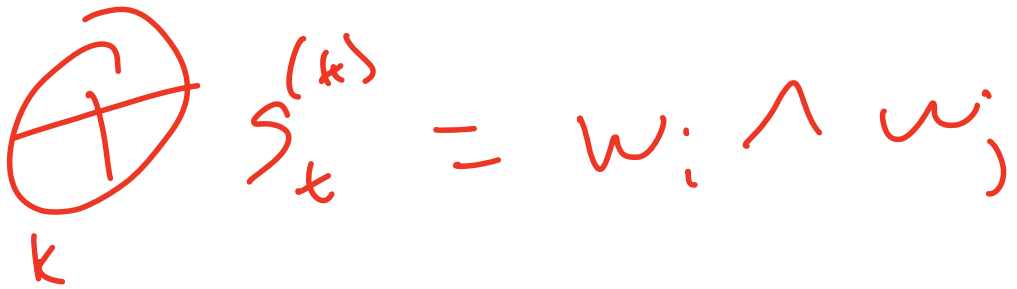
GMW Protocol: m-Player AND Gate

- Given the previous identity, parties evaluate the AND gate as follows.

- Each party P_k locally computes $s_i^{(k)} \wedge s_j^{(k)}$.
  - This is a secret share of $\bigoplus_{k \in [m]} s_i^{(k)} \wedge s_j^{(k)}$.

- Each pair of parties P_k and $P_{k'}$ then jointly compute secret shares of $s_i^{(k)} \wedge s_j^{(k')}$.
  - This is done as in the 2-Player version, using $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$.

- Then for all $k \in [m]$, P_k sets $s_t^{(k)} = (s_i^{(k)} \wedge s_j^{(k)}) \oplus \bigoplus_{k' \neq k} s_i^{(k)} \wedge s_j^{(k')}$.



GMW Protocol: m-Party Output wires

GMW Protocol: m-Party Output wires

- For $k \in [m]$, P_k evaluates all gates in $\mathcal{C}$ as described previously.

GMW Protocol: m-Party Output wires

- For $k \in [m]$, P_k evaluates all gates in $\mathcal{C}$ as described previously.
- For every output wire w_ℓ,

GMW Protocol: m-Party Output wires

- For $k \in [m]$, P_k evaluates all gates in $\mathcal{C}$ as described previously.
- For every output wire w_ℓ,
  - For $k \in [m]$, P_k sends $s_\ell^{(k)}$ to all other $P_{k'}$.

GMW Protocol: m-Party Output wires

- For $k \in [m]$, P_k evaluates all gates in $\mathcal{C}$ as described previously.
- For every output wire w_ℓ,
  - For $k \in [m]$, P_k sends $s_\ell^{(k)}$ to all other $P_{k'}$.
- Parties locally compute $w_\ell = \bigoplus_{k \in [m]} s_\ell^{(k)}$.

GMW Protocol: m-Party Output wires

- For $k \in [m]$, P_k evaluates all gates in $\mathcal{C}$ as described previously.
- For every output wire w_ℓ,
  - For $k \in [m]$, P_k sends $s_\ell^{(k)}$ to all other $P_{k'}$.
- Parties locally compute $w_\ell = \bigoplus_{k \in [m]} s_\ell^{(k)}$.
- Parties have learned $\mathcal{F}(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)})$!

GMW Protocol: m-Party Costs

GMW Protocol: m-Party Costs

- Setup Phase.

GMW Protocol: m-Party Costs

- Setup Phase.

- Output Phase.

GMW Protocol: m-Party Costs

- Setup Phase.
- Output Phase.
- Evaluation Phase.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$ $O(m^2 \cdot n)$ bits
- Output Phase.
- Evaluation Phase.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
- Evaluation Phase.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - For each AND gate, every P_k for $k \in [m]$ executes $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ with every other $P_{k'}$.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - For each AND gate, every P_k for $k \in [m]$ executes $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ with every other $P_{k'}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - For each AND gate, every P_k for $k \in [m]$ executes $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ with every other $P_{k'}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.
  - The protocol requires $s_{\mathsf{AND}} \cdot \mathsf{rounds}(\mathcal{F}_{\mathsf{OT}}^{(1,4)}) + 2$ rounds of communication (assuming all $\mathcal{F}_{\mathsf{OT}}$ are executed in parallel).

GMW Protocol: m-Party Costs

- Setup Phase.
  - For $k \in [m]$, P_k samples and sends n random bits $\mathbf{r}^{(k)}$ to each other $P_{k'}$.
- Output Phase.
  - Let s_{out} be the number of output gates.
  - For every $k \in [m]$, P_k sends s_{out} bits to every other $P_{k'}$
- Evaluation Phase.
  - XOR and NOT gates require *no communication* between parties.
  - For each AND gate, every P_k for $k \in [m]$ executes $\mathcal{F}_{\mathsf{OT}}^{(1,4)}$ with every other $P_{k'}$.
- Communication complexity: let s_{AND} denote the number of AND gates in the circuit.
  - The protocol requires $s_{\mathsf{AND}} \cdot \mathsf{rounds}(\mathcal{F}_{\mathsf{OT}}^{(1,4)}) + 2$ rounds of communication (assuming all $\mathcal{F}_{\mathsf{OT}}$ are executed in parallel).
  - Parties exchange $n + s_{\mathsf{out}} + s_{\mathsf{AND}} \cdot O(m^2 \cdot \mathsf{bits}(\mathcal{F}_{\mathsf{OT}}^{(1,4)}))$ bits.

Next Time: MPC Wrap-up