CS4501 Cryptographic Protocols Lecture 17: Proof for Yao's Garbled Circuits, Point and Permute, BMR

https://jackdoerner.net/teaching/#2026/Spring/CS4501

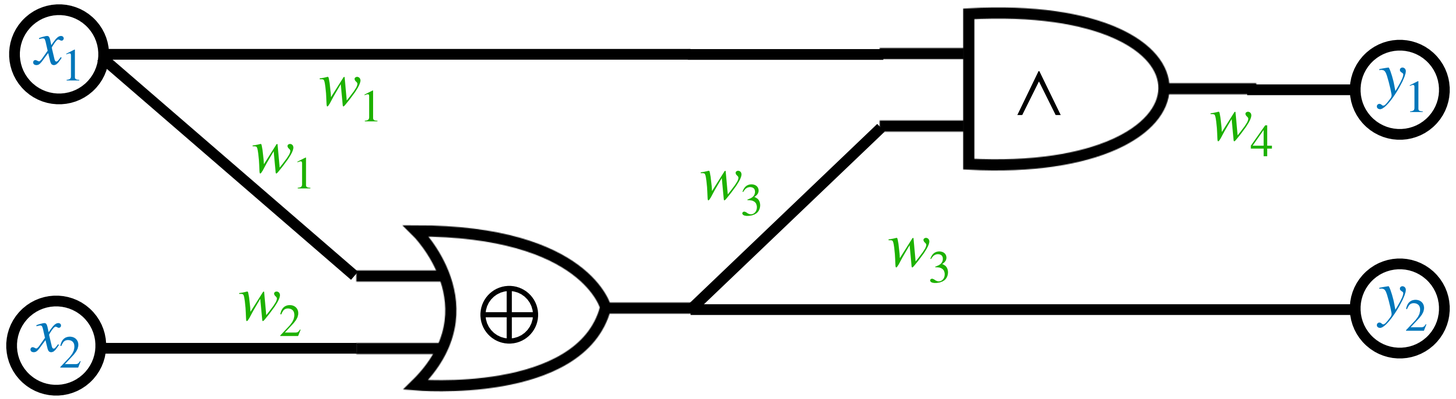
The Rest of this Lecture

1. Review of Yao's Garbled Circuits, plus the simulator for Bob.
2. Finish our proof of security for Yao's Garbled Circuits.
3. A brief history of GC optimizations.
4. The BMR protocol, or: how to Garble with many parties.

1a. GC Review (Protocol)

The Setting for Our Example

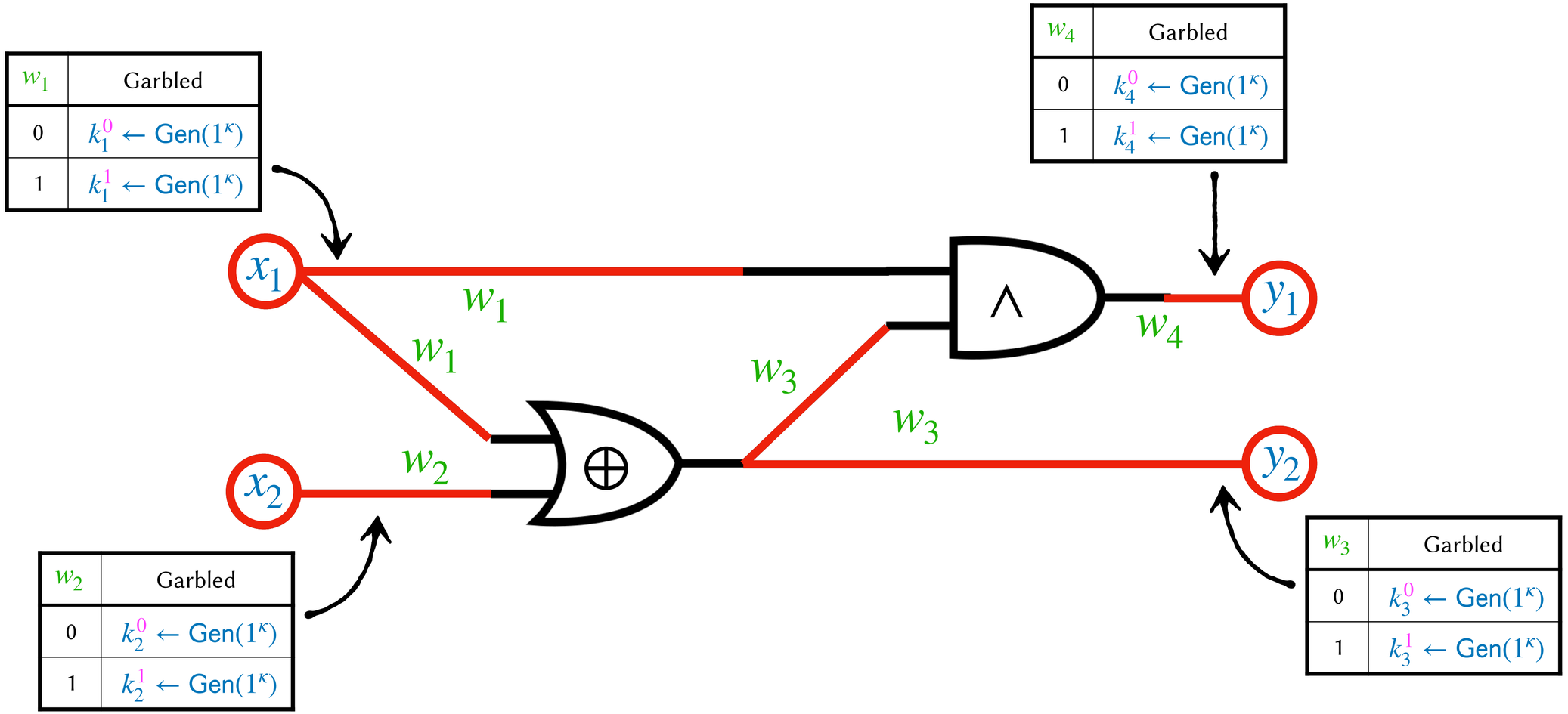
- Alice and Bob wish to compute $f(x_1, x_2) = ((x_1 \oplus x_2) \wedge x_1, x_1 \oplus x_2)$.
- The corresponding circuit is $C = \{(\mathsf{in},1,1), (\mathsf{in},2,2), (\oplus,1,2,3), (\wedge,1,3,4), (\mathsf{out},1,4)(\mathsf{out},2,3)\}$:



- We will fix $x_1 = 0$ and $x_2 = 1$. Notice that $f(0,1) = (0,1)$.
- Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure SKE with Evasive and Efficiently-Verifiable Range (Defs. 3+5 from Lecture 16).

Phase 1: Garbling

1. Alice samples a pair of encryption keys for each wire, using $\mathsf{Gen}(1^\kappa)$:



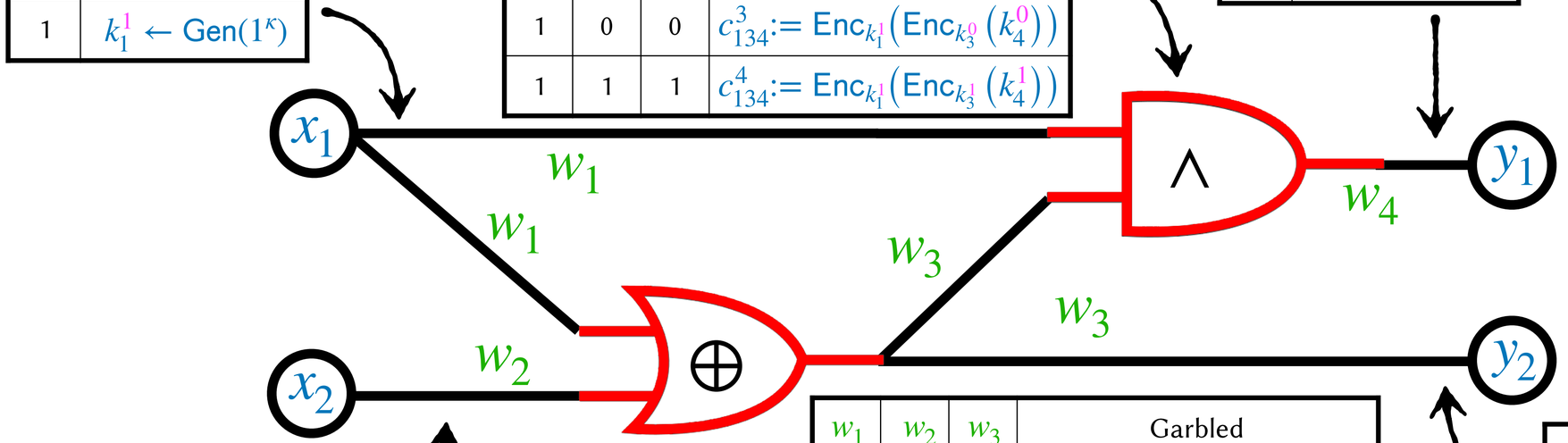
Phase 1: Garbling

2. Alice encrypts each gate's output keys under its input keys, per the truth table.

w_1	Garbled
0	$k_1^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	0	0	$c_{134}^1 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c_{134}^2 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c_{134}^3 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c_{134}^4 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_3	Garbled
0	0	0	$c_{123}^1 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$
0	1	1	$c_{123}^2 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
1	0	1	$c_{123}^3 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
1	1	0	$c_{123}^4 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

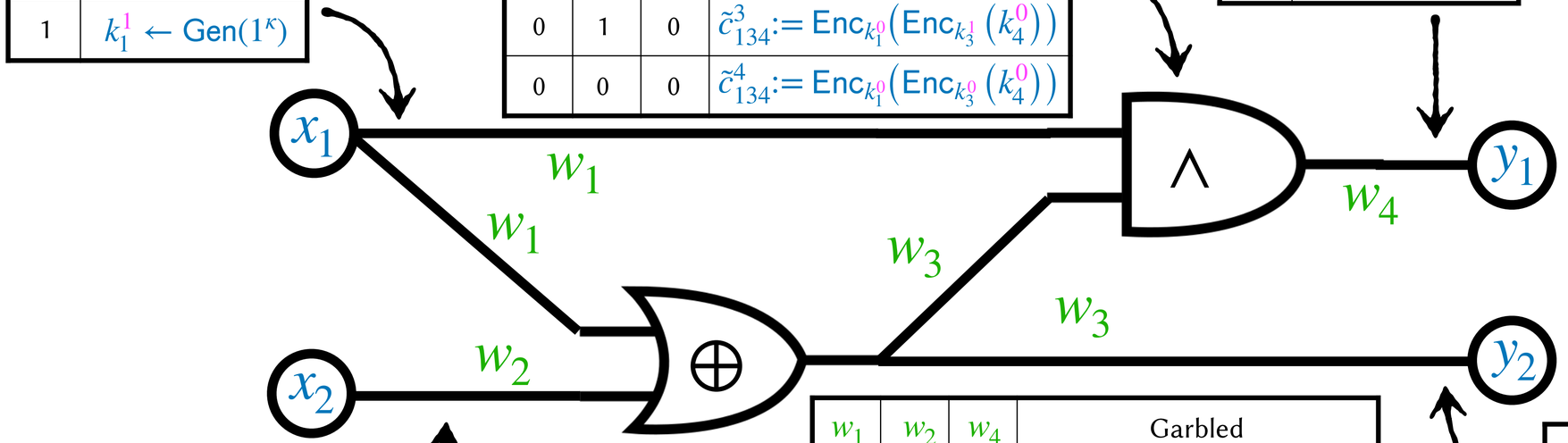
Phase 1: Garbling

3. Alice permutes the rows of each of the garbled tables randomly.

w_1	Garbled
0	$k_1^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

Phase 1: Garbling

4. Alice sends the garbled truth tables to Bob.

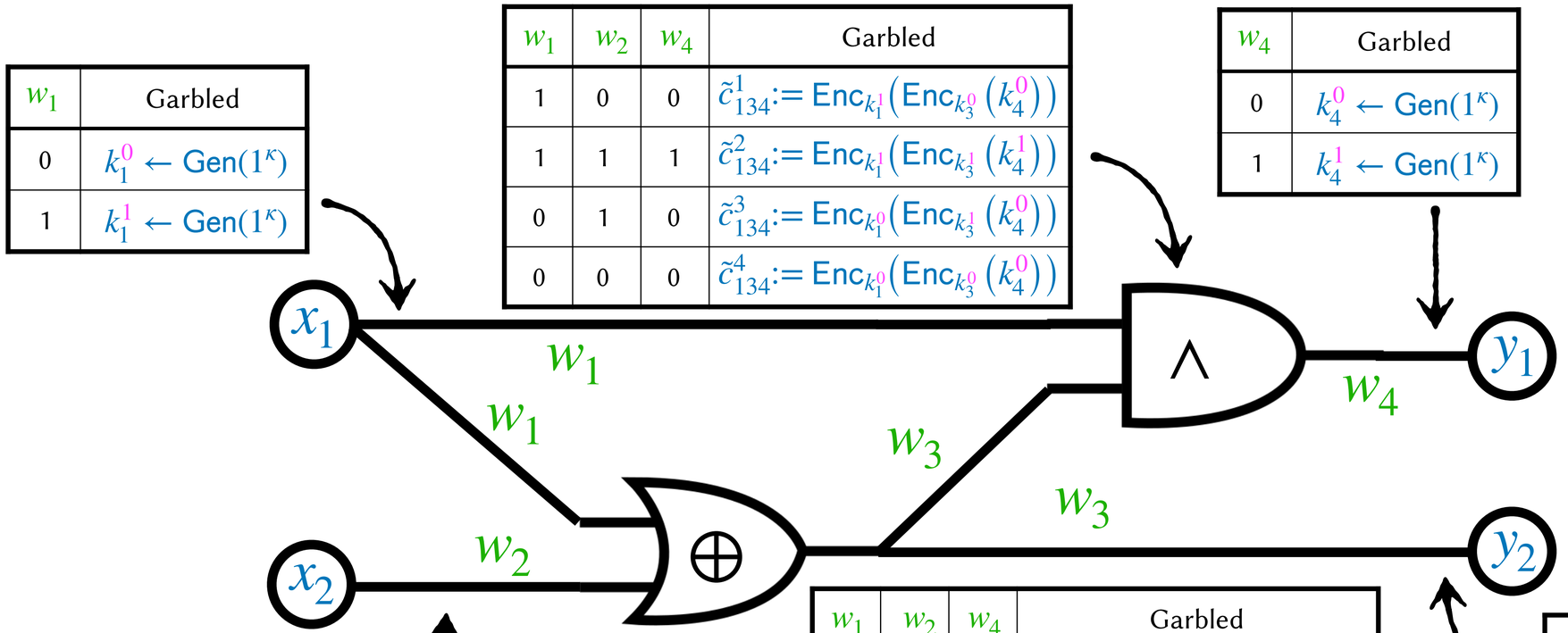
Garbled
$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_1	Garbled
0	$k_1^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

Phase 2: I/O Delivery

1. Alice sends her input key to Bob.

Garbled
$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

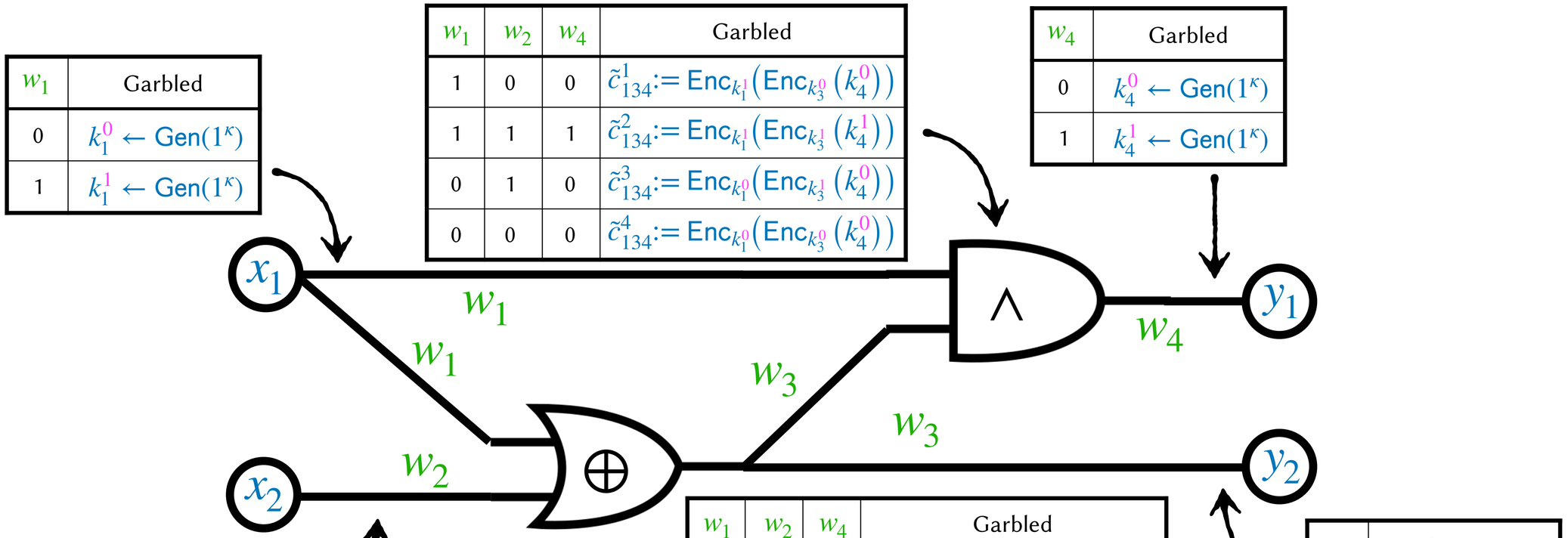
Garbled
$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_1	Garbled
0	$k_1^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

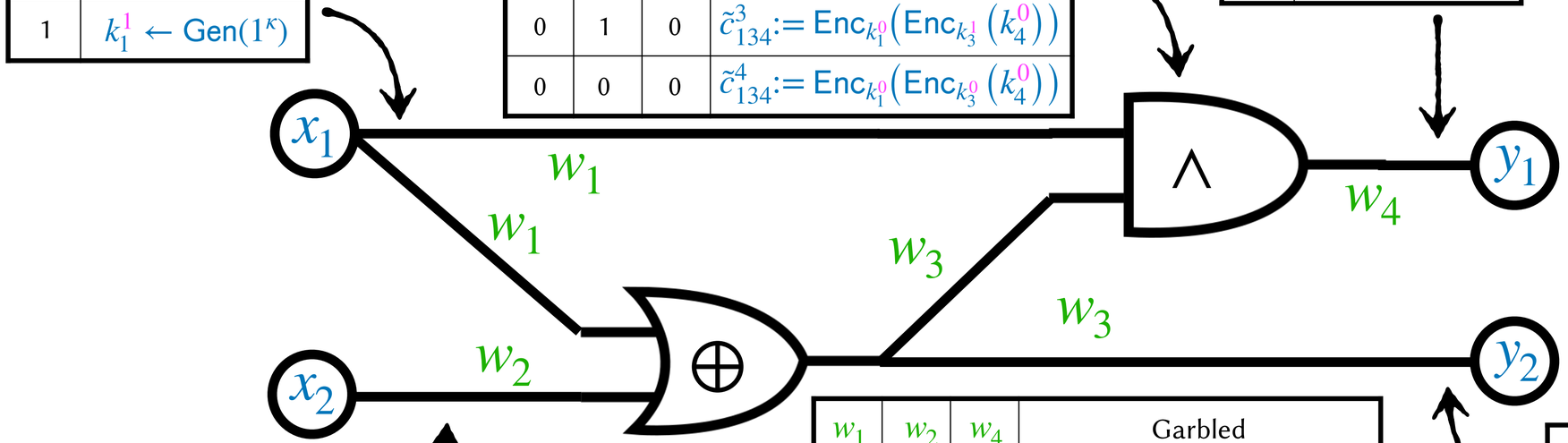
Phase 2: I/O Delivery

2. Alice sends Bob's input keys to $\mathcal{F}_{\mathsf{OT}}$, and Bob receives one.

w_1	Garbled
0	$k_1^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \mathsf{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \mathsf{Gen}(1^\kappa)$

Garbled
$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

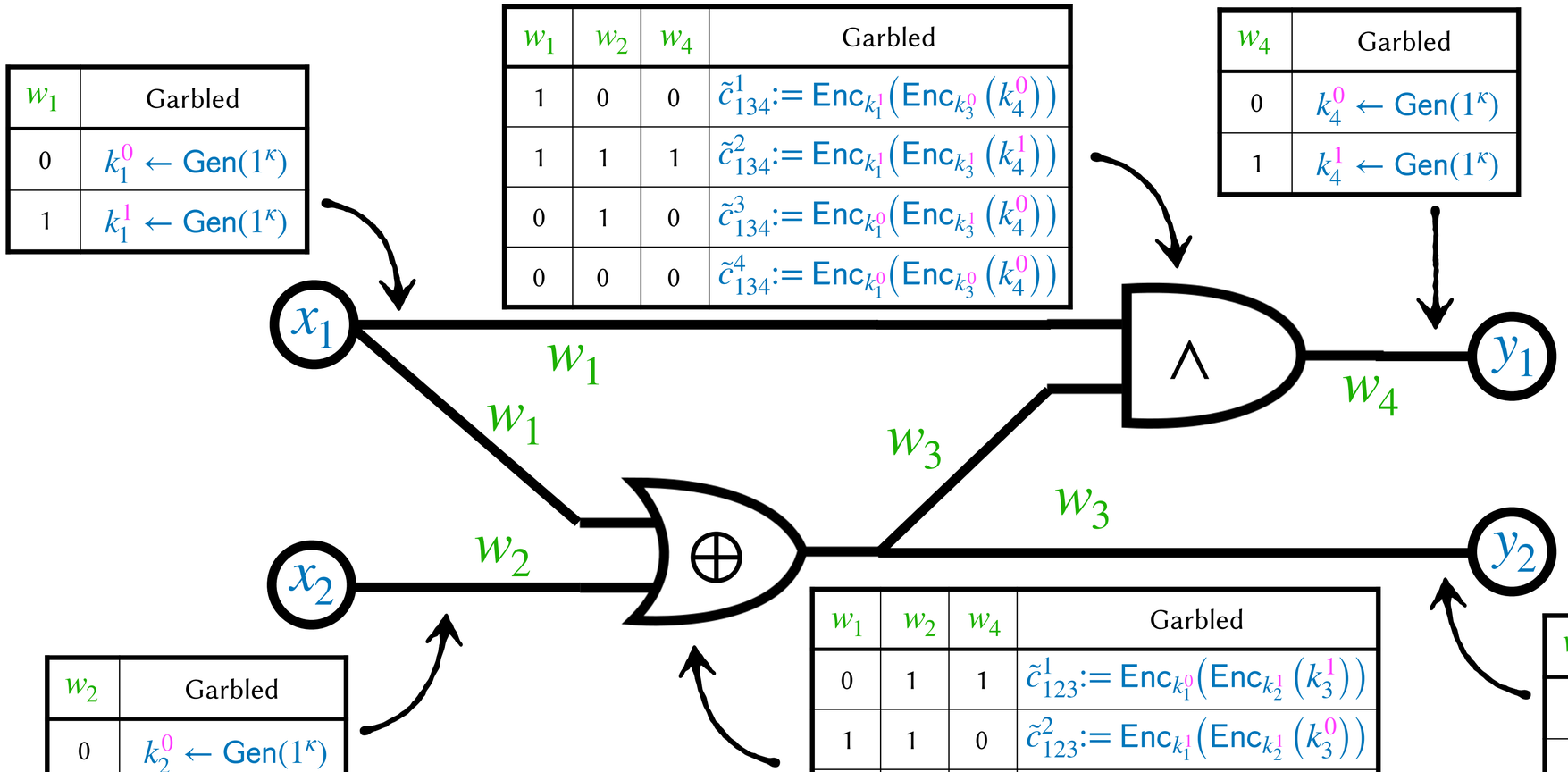
Phase 2: I/O Delivery

3. Alice sends Bob's output table to him.

w_1	Garbled
0	$k_1^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_1^1 \leftarrow \text{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

w_4	Garbled
0	$k_4^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \text{Gen}(1^\kappa)$



w_2	Garbled
0	$k_2^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_2^1 \leftarrow \text{Gen}(1^\kappa)$

w_1	w_2	w_4	Garbled
0	1	1	$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
1	1	0	$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
1	0	1	$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
0	0	0	$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	$k_3^0 \leftarrow \text{Gen}(1^\kappa)$
1	$k_3^1 \leftarrow \text{Gen}(1^\kappa)$

Garbled
$\tilde{c}^1_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \text{Enc}_{k_1^1}(\text{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \text{Enc}_{k_1^0}(\text{Enc}_{k_3^0}(k_4^0))$

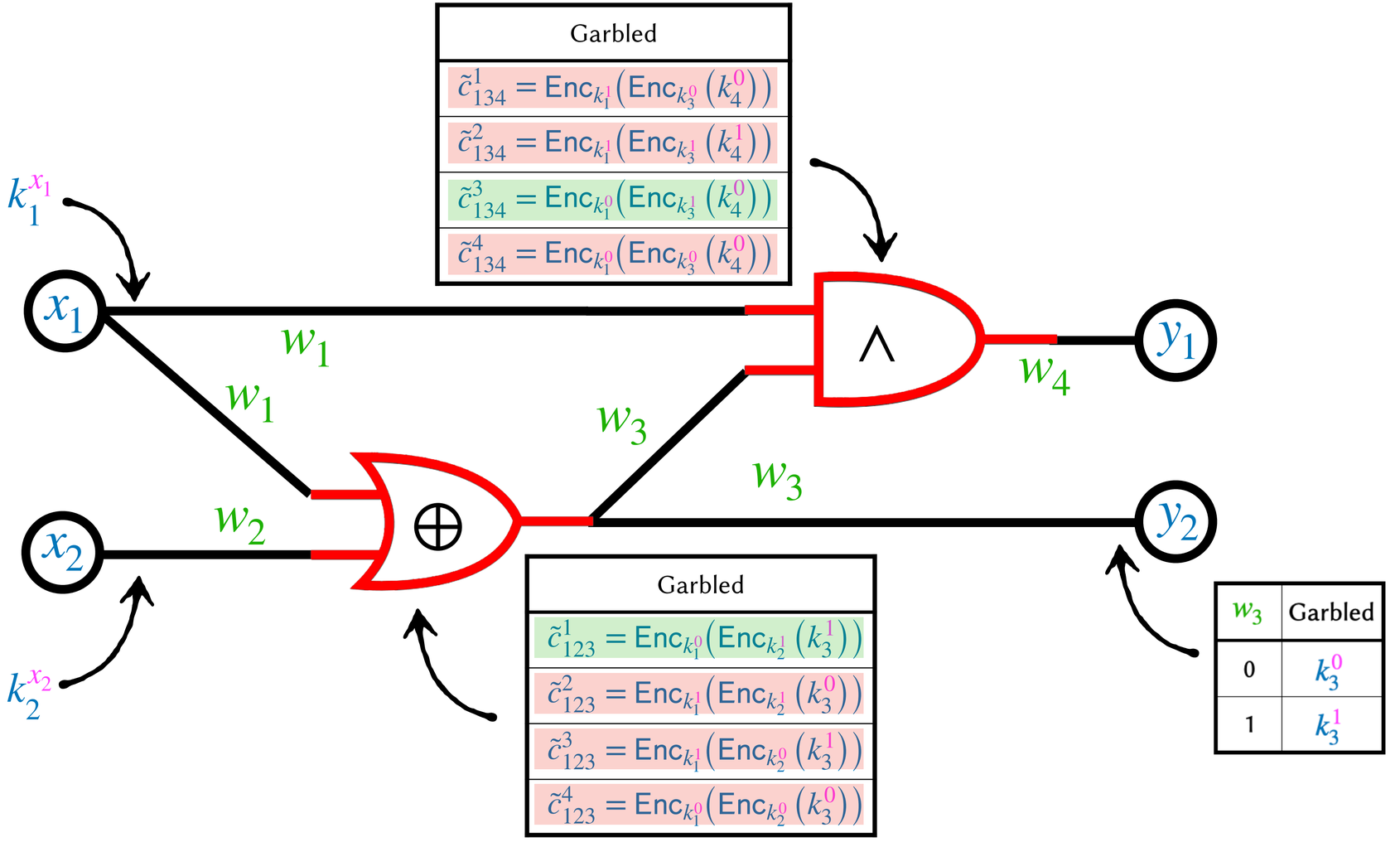
Garbled
$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

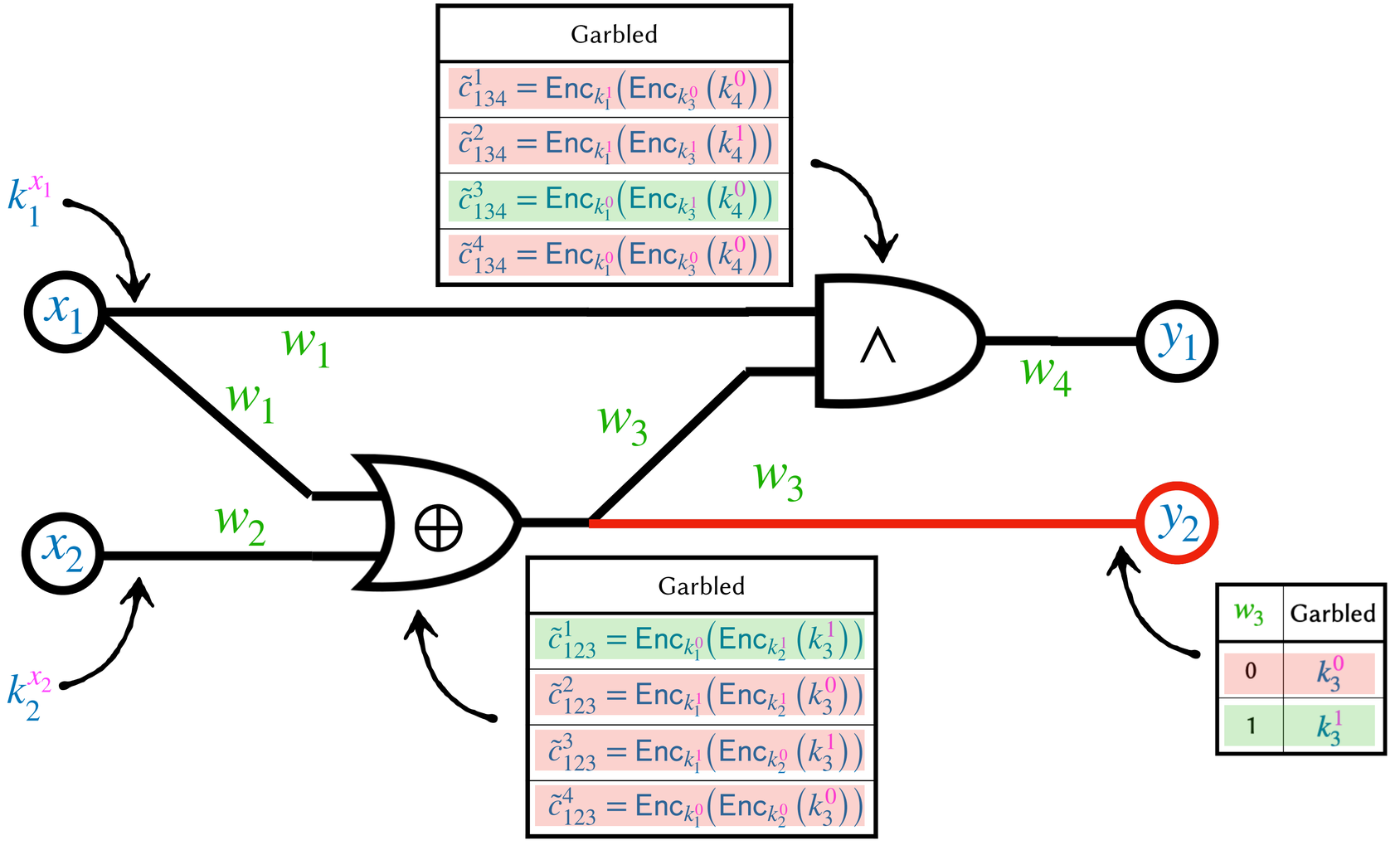
Phase 3: Evaluation

1. Bob traverses in topological order, and tries decrypting until he finds the right row.



Phase 3: Evaluation

2. Bob translates his output key into a logical output bit, using the output table.



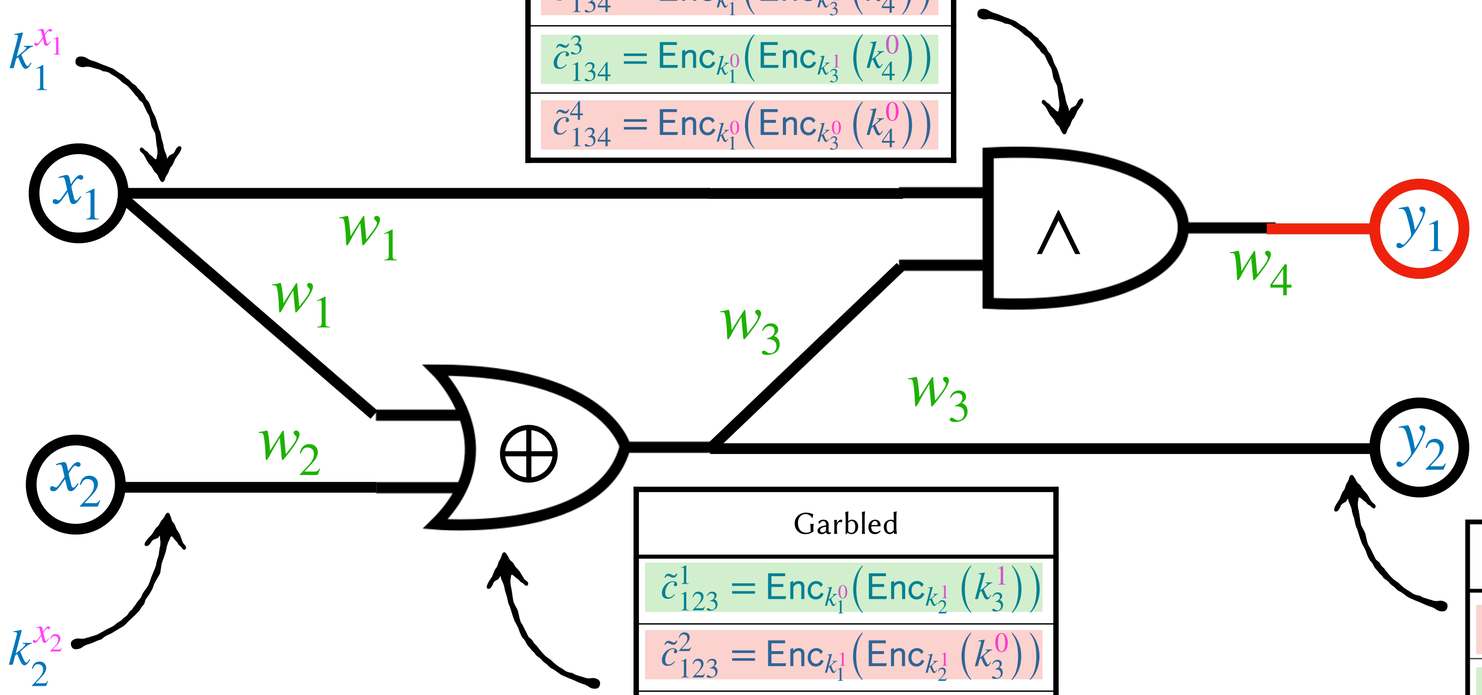
Phase 3: Evaluation

3. Bob sends Alice's output key back to her, and she translates it.

w_4	Garbled
0	$k_4^0 \leftarrow \mathsf{Gen}(1^\kappa)$
1	$k_4^1 \leftarrow \mathsf{Gen}(1^\kappa)$

$k_4^{w_4}$

Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$



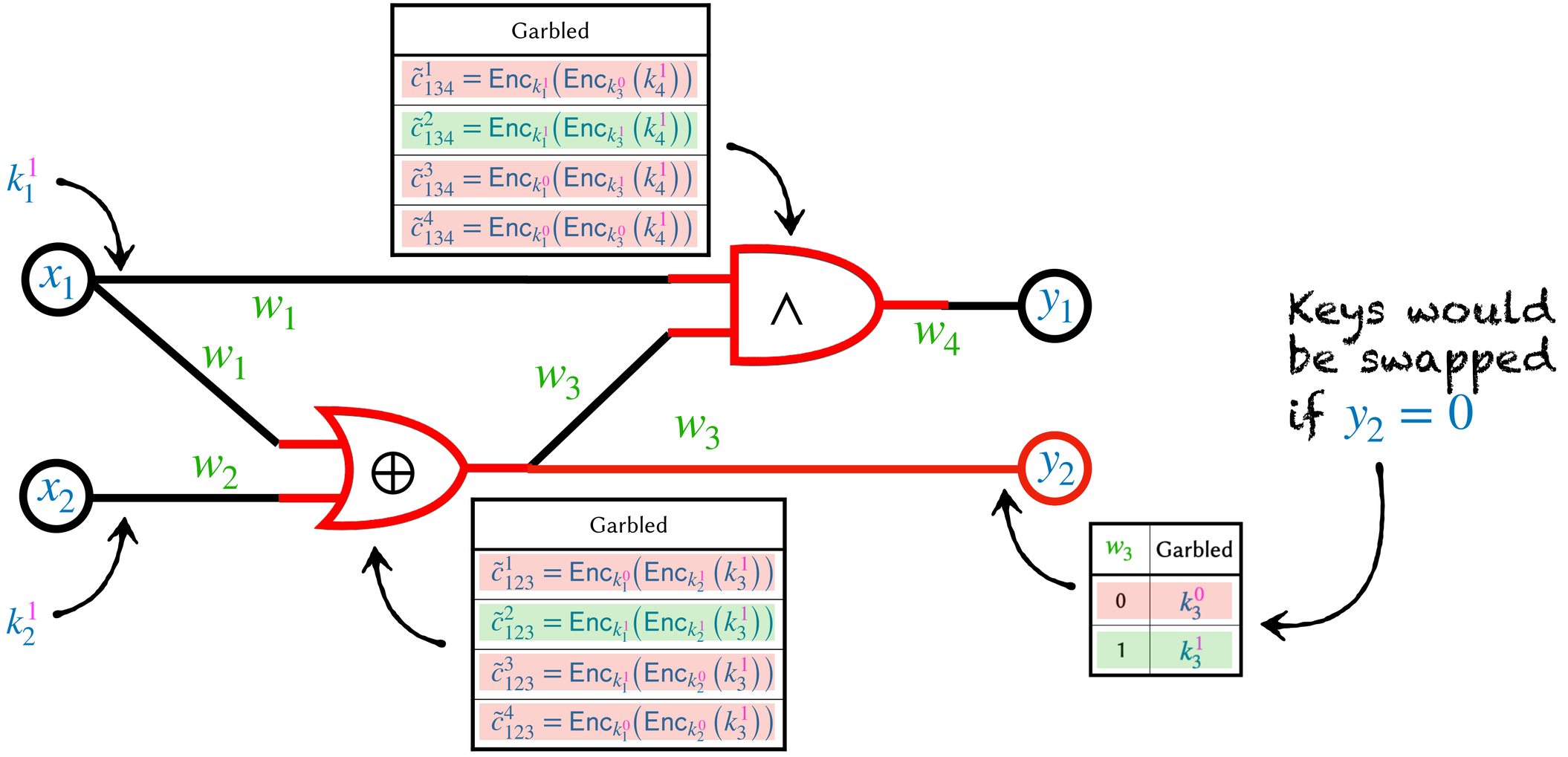
Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	k_3^0
1	k_3^1

1b. GC Review (Bob's Simulator)

Bob's Simulated View

Sim_B constructs gates that *always* output 1, and adjusts Bob's output table if needed.



1c. GC Review (Double Encryption)

IND-CPA implies Double-IND-CPA

Lemma 1: Let ℓ be a polynomial and let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA-secure SKE scheme for message space $\mathcal{M}$. For any $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1) \in \mathcal{M}^{\ell(\kappa)+2}$ and any $k_2 \in \mathrm{image}(\mathsf{Gen})$, if we let $k_1 \leftarrow \mathsf{Gen}(1^\kappa)$ and

$$c'_i \leftarrow \mathsf{Enc}_{k_1}\left(\mathsf{Enc}_{k_2}\left(m'_i\right)\right) \forall i \in [\ell(\kappa)] \quad \text{and} \quad c^*_b \leftarrow \mathsf{Enc}_{k_1}\left(\mathsf{Enc}_{k_2}\left(m^*_b\right)\right) \forall b \in \{0,1\}$$

then $\left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_0\right)\right\}_{\kappa\in\mathbb{N}} \approx_c \left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_1\right)\right\}_{\kappa\in\mathbb{N}}$.

Lemma 2: Let ℓ be a polynomial and let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA-secure SKE scheme for message space $\mathcal{M}$. For any $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1) \in \mathcal{M}^{\ell(\kappa)+2}$ and any $k_1 \in \mathrm{image}(\mathsf{Gen})$, if we let $k_2 \leftarrow \mathsf{Gen}(1^\kappa)$ and

$$c'_i \leftarrow \mathsf{Enc}_{k_1}\left(\mathsf{Enc}_{k_2}\left(m'_i\right)\right) \forall i \in [\ell(\kappa)] \quad \text{and} \quad c^*_b \leftarrow \mathsf{Enc}_{k_1}\left(\mathsf{Enc}_{k_2}\left(m^*_b\right)\right) \forall b \in \{0,1\}$$

then $\left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_0\right)\right\}_{\kappa\in\mathbb{N}} \approx_c \left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_1\right)\right\}_{\kappa\in\mathbb{N}}$.

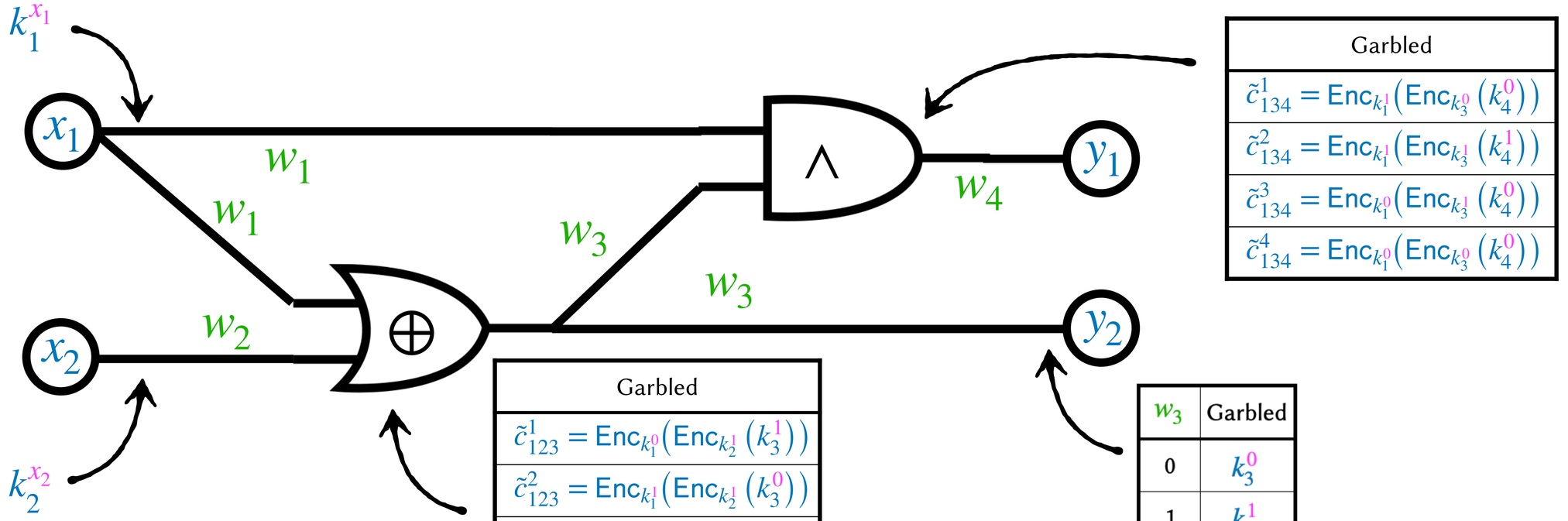
2. Proving Bob’s Simulation is Indistinguishable from his Real View

A Hybrid Argument for Simulating Corrupt Bob

Now we are ready to construct a sequence of hybrid experiment that blend the features of the real and simulated (i.e. ideal) worlds for Yao's Garbled Circuits.

We consider an arbitrary, fixed circuit, and the number of hybrids will be proportionate to the number of gates. We'll start with the real world.

$\mathsf{VIEW}_B =$

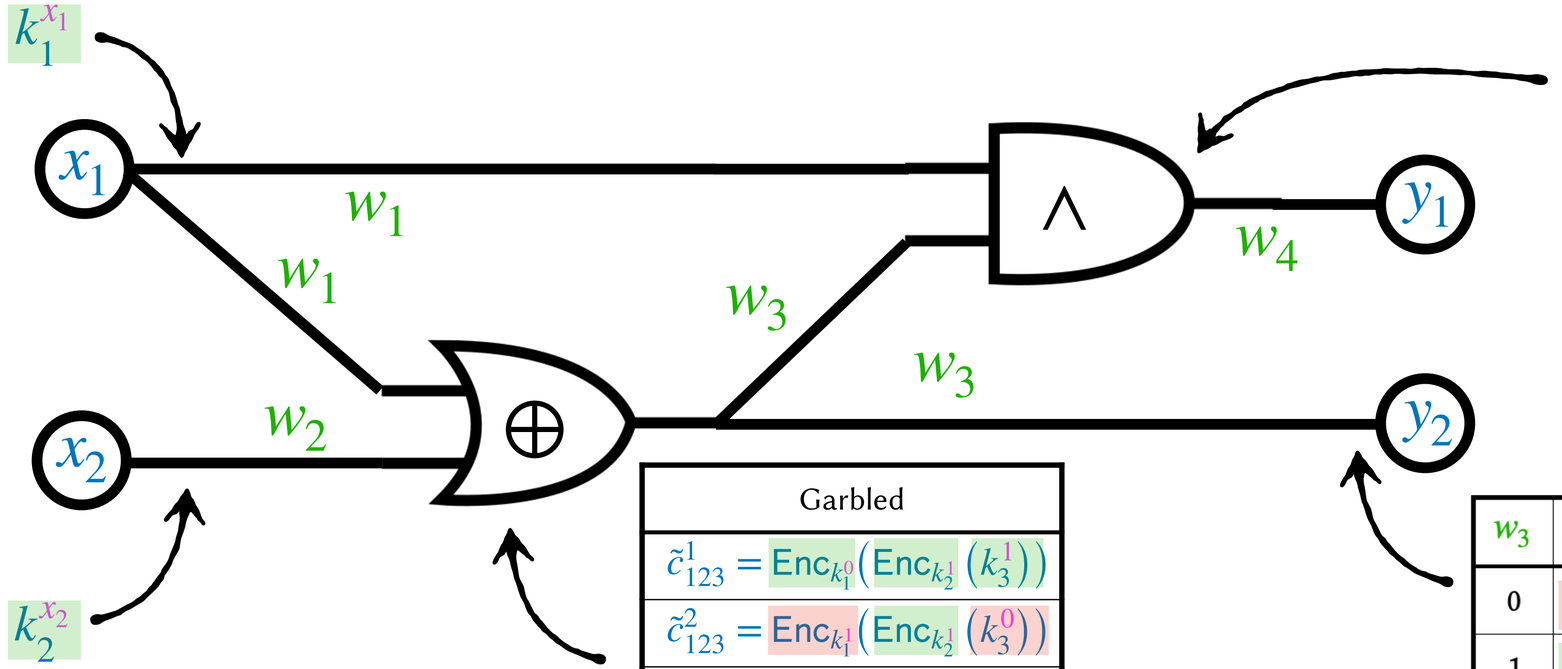


Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	k_3^0
1	k_3^1

$\mathsf{VIEW}_B =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

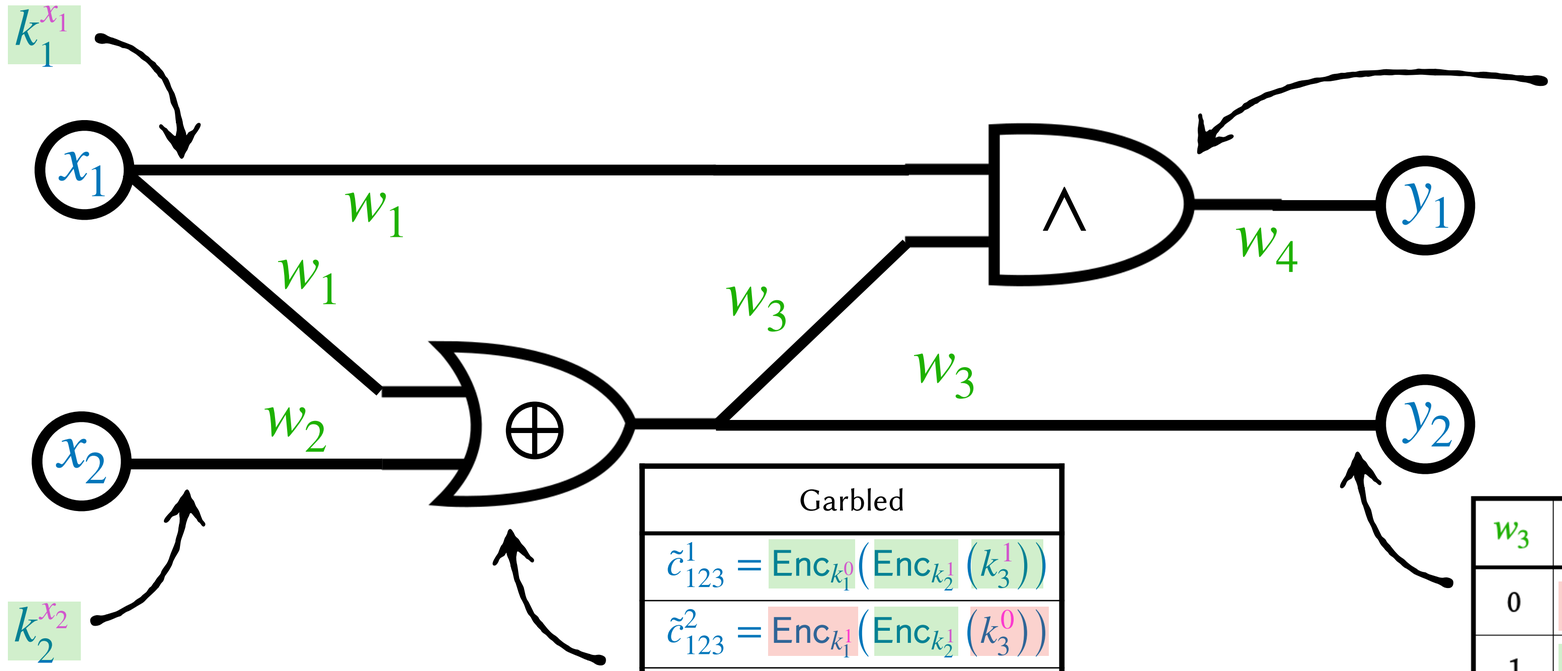
w_3	Garbled
0	k_3^0
1	k_3^1

We will call any key that is learned by Bob in the real world an *active key,* and the set of all keys that he learns is the *active path* through the circuit.

On the other hand, we will call keys that he doesn't learn *inactive keys.*

Our goal is to define each successive hybrid experiment to replace exactly encryption of an inactive key with an encryption on the corresponding active key, using Lemmas 1 and 2. We care only about the *encrypted* keys, not the *encrypting* keys.

$\mathsf{VIEW}_B =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	k_3^0
1	k_3^1

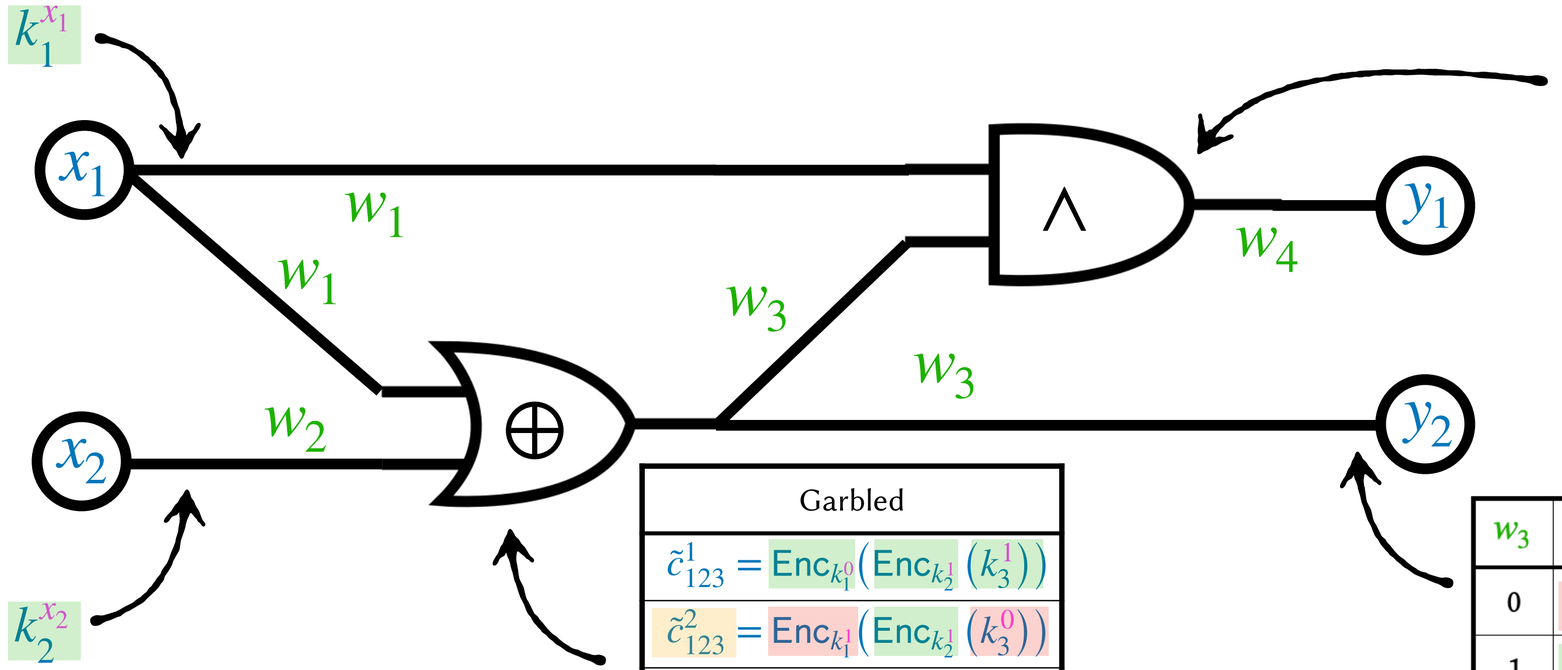
The following invariant is critical: any time an *inactive* key is *encrypted*, at least one of the *encrypting keys* is also by definition *inactive* (i.e. unknown to Bob).

This is what allows us to apply Lemma 1 or Lemma 2 (both of which assume that one of the keys is unknown to the adversary) to encryptions of *inactive* keys.

Note that the sets of active and inactive keys are both defined by both Bob's input and Alice's. This means that the sequence of key replacements must be a (deterministic) function of the inputs!

Nevertheless, the total number of hybrids is determined by the total number table rows.

$\mathsf{VIEW}_B =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	k_3^0
1	k_3^1

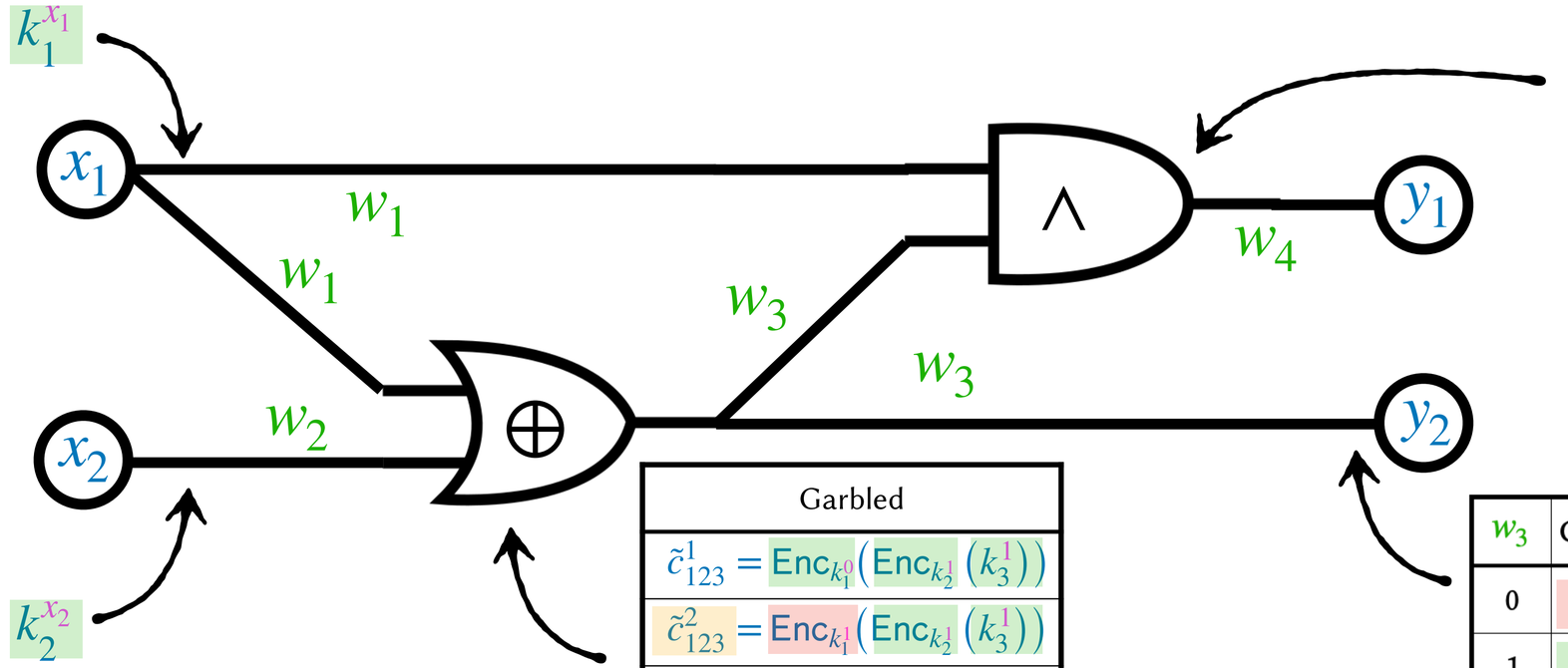
Recall, in the (double) IND-CPA game we do not give $(\mathscr{A}, \mathscr{D})$ an encryption of the secret key!

This means that we can *only* apply Lemmas 1 and 2 if Bob does not know any encryptions of the *inactive* key with respect to which we are applying them.

Therefore, we have to be very careful about the order of our hybrids. Above, we cannot yet apply Lemma 1 to replace $\tilde{c}^4_{134}$ because Bob knows an encryption of k_3^0 (specifically, $\tilde{c}^2_{123}$ and $\tilde{c}^4_{123}$).

Finding the sequence of hybrids that replaces interrelated parts of a protocol while carefully respecting some set of constraints is related to the mathematical idea of *pebbling games.*

$H_1(1^\kappa, x_1, x_2) =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

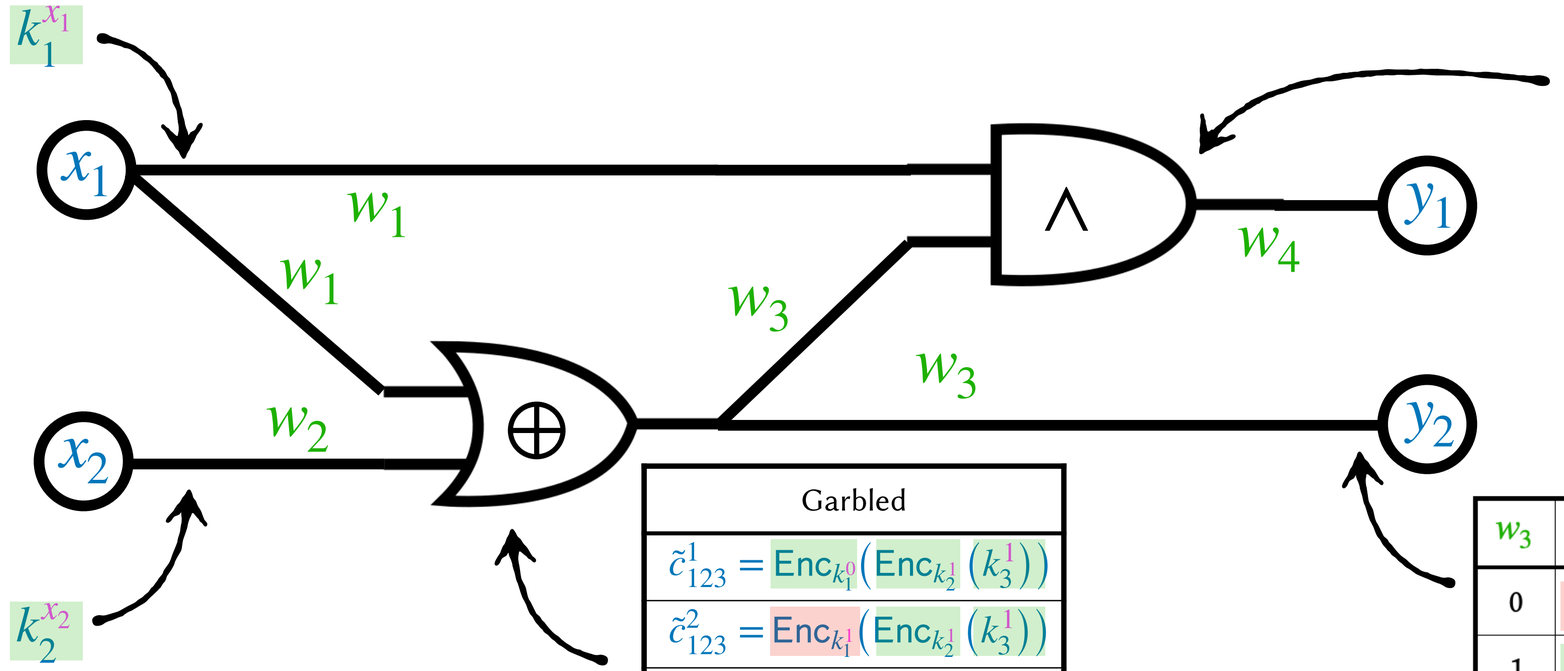
Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

w_3	Garbled
0	k_3^0
1	k_3^1

We will visit the gates in topological order. Observe that in the first gate, Bob does not possess any encryptions of the inactive *encrypting* keys (i.e. keys for w_1, w_2), so Lemmas 1 and 2 can be applied!

Our first hybrid, $H_1(1^\kappa, x_1, x_2)$, replaces $\tilde{c}^2_{123}$ with an encryption of the *active* key for w_3, which is k_3^1. By Lemma 1, we have that $\{\mathsf{VIEW}_\mathsf{B}\}_{\kappa\in\mathbb{N}} \approx_c \{H_1(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}}$.

$H_2(1^\kappa, x_1, x_2) =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^1))$

w_3	Garbled
0	k_3^0
1	k_3^1

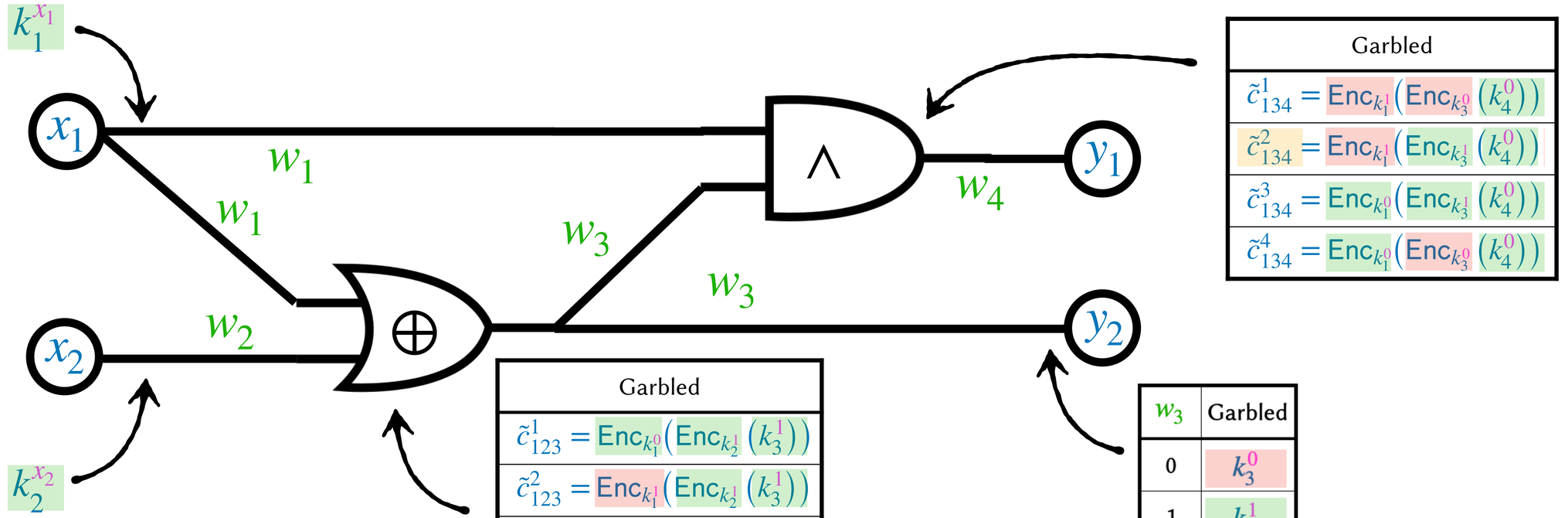
We will visit the gates in topological order. Observe that in the first gate, Bob does not possess any encryptions of the inactive *encrypting* keys (i.e. keys for w_1, w_2), so Lemmas 1 and 2 can be applied!

Our first hybrid, $H_1(1^\kappa, x_1, x_2)$, replaces $\tilde{c}^2_{123}$ with an encryption of the *active* key for w_3, which is k_3^1. By Lemma 1, we have that $\{\mathsf{VIEW}_\mathsf{B}\}_{\kappa\in\mathbb{N}} \approx_c \{H_1(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}}$.

Similarly, our second hybrid, $H_2(1^\kappa, x_1, x_2)$, replaces $\tilde{c}^4_{123}$ with an encryption of k_3^1. By Lemma 2, we have that $\{H_1(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}} \approx_c \{H_2(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}}$.

In both cases, given a distinguisher, we can write a reduction to contradict the lemma.

$H_3(1^\kappa, x_1, x_2) =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^1))$

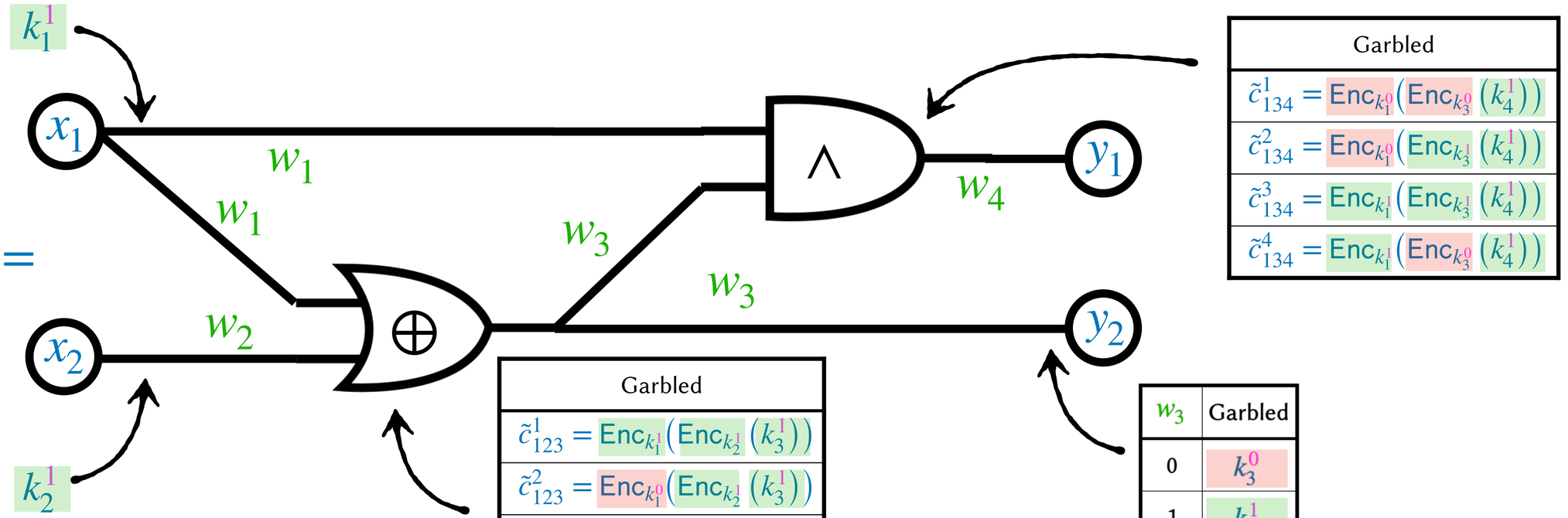
w_3	Garbled
0	k_3^0
1	k_3^1

Now we can move to the next gate, topologically. Notice that Bob no longer has any ciphertexts that contain the inactive encrypting keys (k_1^1 or k_3^0) for the $\wedge$ gate! So we can apply our Lemmas.

Our third hybrid, $H_3(1^\kappa, x_1, x_2)$, replaces $\tilde{c}^2_{134}$ with an encryption of the *active* key for w_4, which is k_4^0. By Lemma 1, we have that $\{H_2(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}} \approx_c \{H_3(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}}$.

Notice that in $H_3(1^\kappa, x_1, x_2)$, the garbled tables only contain encryptions of active keys! This is starting to look a lot like the ideal world.

$\mathsf{Sim}_{\mathsf{B}}(1^\kappa, x_2, y_2) =$



Garbled
$\tilde{c}^1_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^1))$
$\tilde{c}^2_{134} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^4_{134} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^1))$

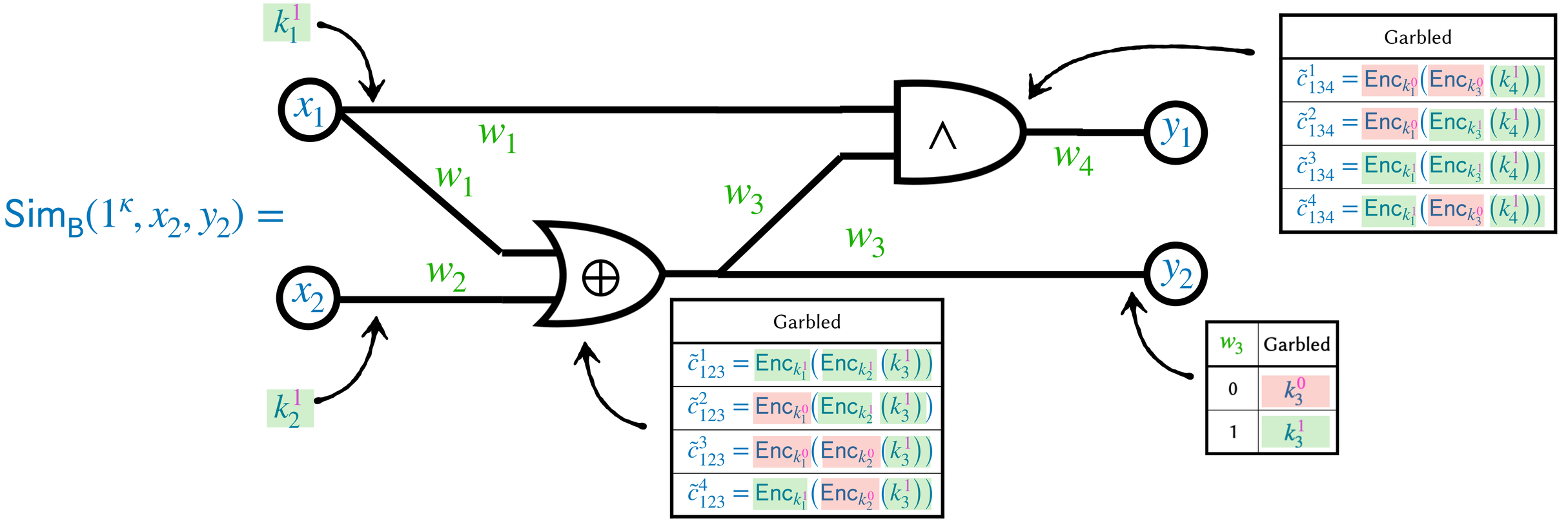
Garbled
$\tilde{c}^1_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^3_{123} = \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} = \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$

w_3	Garbled
0	k_3^0
1	k_3^1

In fact, the only difference between $H_3(1^\kappa, x_1, x_2)$ and the simulation we designed earlier is the *names* of some of the keys.

Observe that all keys that correspond between $H_3(1^\kappa, x_1, x_2)$ and the simulated view have identical distributions, and all tables are *shuffled* according to identical distributions.

We can simply swap the names of k_1^0 and k_1^1, and the names of k_4^0 and k_4^1, and we have arrived at exactly the distribution that is produced by $\mathsf{Sim}_{\mathsf{B}}(1^\kappa, x_2, y_2)$.

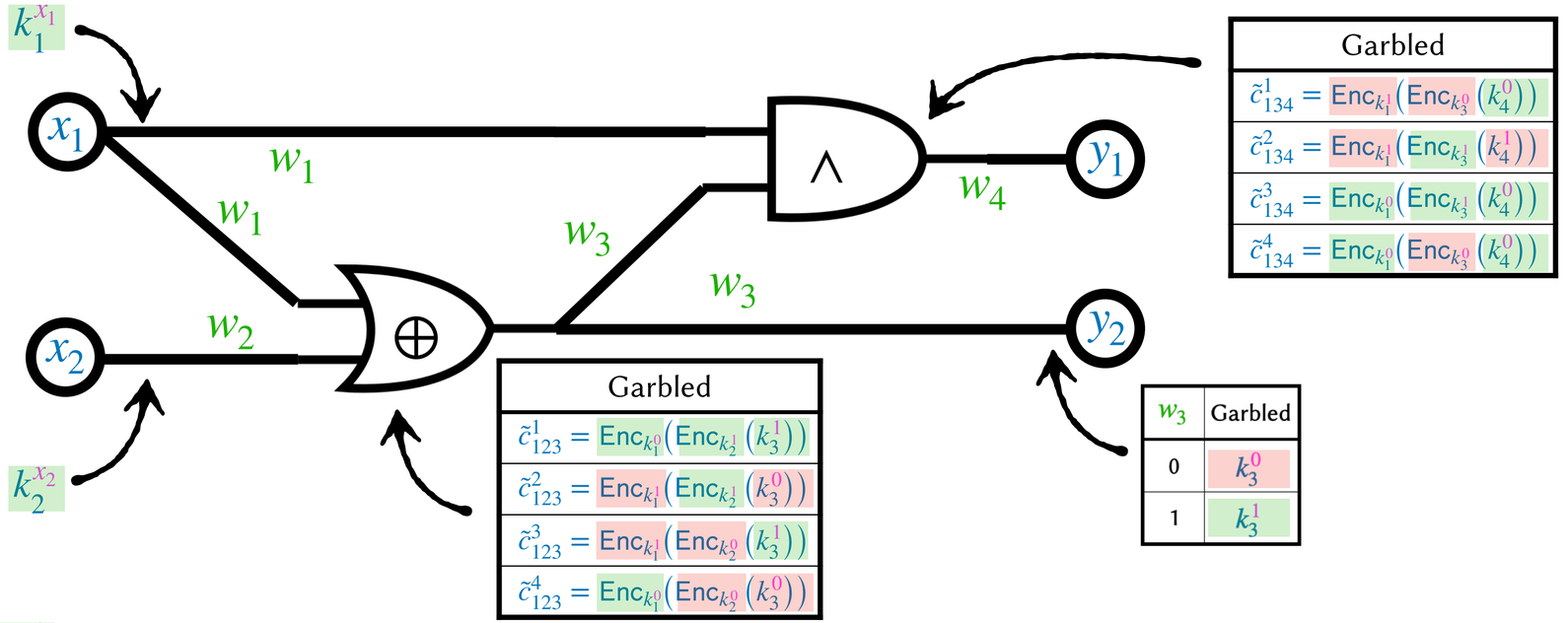


The sequence of hybrids we just constructed could be generalized to work for any boolean circuit. The number of hybrids in the sequence would always be polynomial because there are at most four rows per gate, and at most polynomially many gates (since the protocol is PPT).

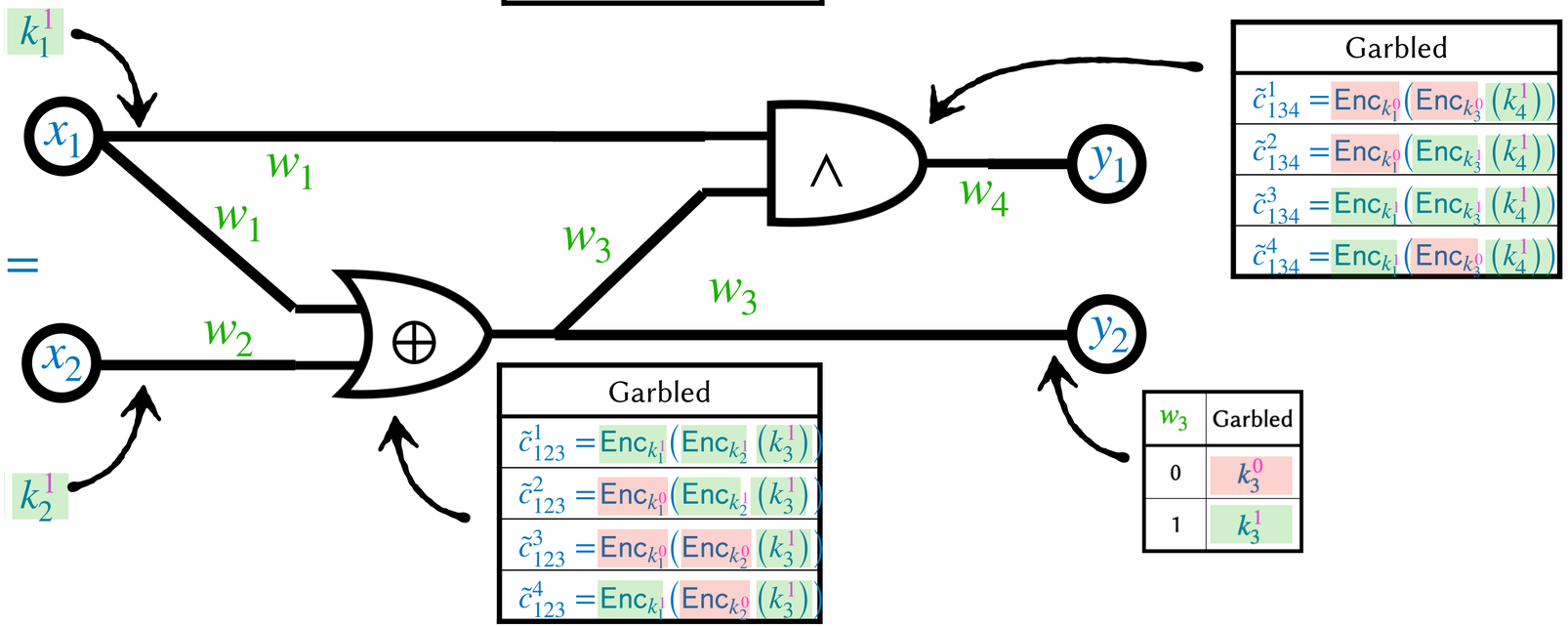
In this example, we have

$$\{\mathsf{VIEW}_{\mathsf{B}}\}_{\kappa\in\mathbb{N}} \approx_c \{H_1(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}} \approx_c \{H_2(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}} \approx_c \{H_3(1^\kappa, x_1, x_2)\}_{\kappa\in\mathbb{N}} \equiv \{\mathsf{Sim}_{\mathsf{B}}(1^\kappa, x_2, y_2)\}_{\kappa\in\mathbb{N}}$$

$\mathsf{VIEW}_\mathsf{B} =$



$\mathsf{Sim}_\mathsf{B}(1^\kappa, x_2, y_2) =$



Because Sim_B is PPT and computational indistinguishability is transitive, this finally implies $\{\mathsf{VIEW}_\mathsf{B}\}_{\kappa\in\mathbb{N}} \approx_c \{\mathsf{Sim}_\mathsf{B}(1^\kappa, x_2, y_2)\}_{\kappa\in\mathbb{N}}$.

Security for Yao: Conclusion

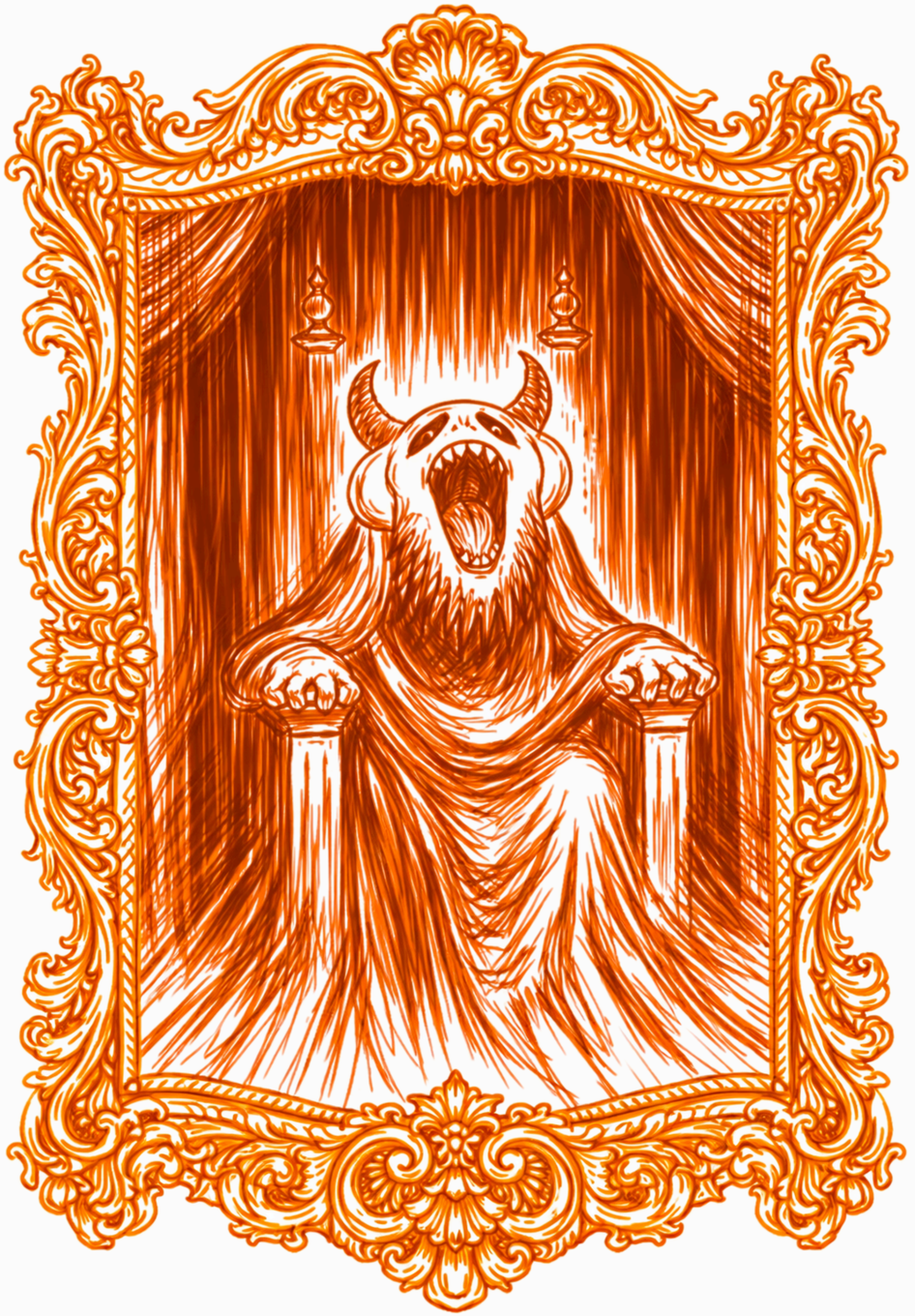
Because Sim_B is PPT and computational indistinguishability is transitive, this finally implies $\{\mathsf{VIEW}_\mathsf{B}\}_{\kappa\in\mathbb{N}} \approx_c \{\mathsf{Sim}_\mathsf{B}(1^\kappa, x_2, y_2)\}_{\kappa\in\mathbb{N}}$.

Together with all-but-negligible correctness and efficient, perfect simulation of a corrupt Alice (which we proved in Lecture 16), this finally implies:

Theorem 1: Let $f : \mathbb{F}_2^2 \to \mathbb{F}_2^2$ be any randomized 2-ary polynomial-time function, and let C be a polynomial-sized boolean circuit that computes f. Assuming synchronicity, secure channels, and the existence of Pseudorandom Functions, there exists a two-party protocol with a constant number of rounds that realizes $\mathscr{F}_{\mathsf{SFE}}(2, f, \mathbb{F}_2, \mathbb{F}_2)$ in the presence of a semi-honest $\mathscr{A}$ that statically corrupts either party in the $\mathscr{F}_{\mathsf{OT}}(2,1)$-hybrid model.

From here, the composition theorem can be used to achieve the same in the plain model if the Decisional Diffie-Hellman Assumption is true. Note that DDH also implies the existence of PRFs.

(I didn't prove to you that there's a polynomial-size circuit to compute any poly-time function, but it's true. The proof should is taught in any introductory theory of computation course!)



3. Costs and Optimizations

Costs for Basic Yao

We introduced Garbled Circuits as a means to achieve constant-round secure computations. Our protocol *does* achieve this property:

- 1 invocation of $\mathscr{F}_{\mathsf{OT}}$ for each of Bob's input bits (remember: our OT protocol required 3 rounds).
- 1 round to communicate the circuit, Alice's input keys, and Bob's output keys.
- 1 round to communicate Alice's output (if necessary).

However, our *bandwidth* cost is very high:

- Our evasive-range encryption produces ciphertexts that are κ bits longer than the plaintext.
- Each garbled table requires us to double-encrypt four κ-bit keys.
- So representing a single boolean gate requires 12κ bits to be sent!
- In practice, we usually set $\kappa \geq 128$, so this is about 0.2KB per gate.
- For comparison, if we evaluated 3-party BGW over $\mathbb{F}_5$, using GRR multiplication, each party would send 10 bits per multiplication gate.

Costs for Basic Yao

What does this add up to in practice?

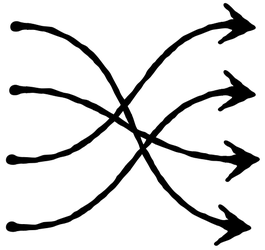
- A schoolbook multiplication circuit for 32-bit inputs has a little under 6000 boolean AND and XOR gates.
- Multiplying two 32-bit numbers using the above circuit requires us to transmit 1.13MB of data. Think about that next time you type "*" into your text editor and assume it will be fast.
- This motivates us to design optimized circuits! The optimization criteria are different from hardware, so new research is required.
- Sometimes it makes sense to take completely new approaches to solving certain problems and consider the constraints of multi-party computation from the outset.
  - For example, the AES blockcipher was designed to yield fast hardware implementations, whereas the LowMC blockcipher was designed to yield low-bandwidth protocols.
  - There are many works on low-bandwidth activation functions for neural networks.
- We should start by optimizing the protocol itself, though. Can we reduce that 12κ?

Optimizing GC: Point and Permute

Recall: we added κ bits of overhead per encryption operation (8κ per gate) so that Bob could determine which row in a garbled table was the correct one.

Then we shuffled the tables by permuting their rows completely randomly, to hide which (logical) row was decrypted from Bob.

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c^1_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c^2_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c^3_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c^4_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$



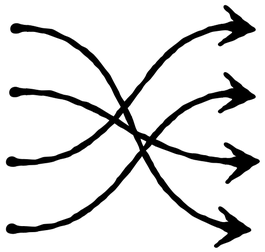
w_1	w_2	w_4	Garbled
			$\tilde{c}^1_{134} :=$
			$\tilde{c}^2_{134} :=$
			$\tilde{c}^3_{134} :=$
			$\tilde{c}^4_{134} :=$

Optimizing GC: Point and Permute

Recall: we added κ bits of overhead per encryption operation (8κ per gate) so that Bob could determine which row in a garbled table was the correct one.

Then we shuffled the tables by permuting their rows completely randomly, to hide which (logical) row was decrypted from Bob.

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c^1_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c^2_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c^3_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c^4_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$



w_1	w_2	w_4	Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	1	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

This approach also introduced a possibility of error.

What we actually need to hide is the values of the *input* wires. The value of the *output* wire is already hidden by the fact that the output keys are identically distributed.

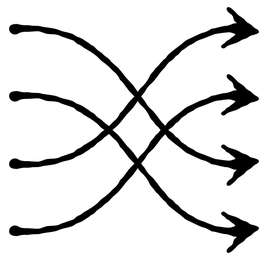
Optimizing GC: Point and Permute

An alternative approach: Let's use *regular* (non-evasive-range) IND-CPA SKE with no overhead.

For each *individual* layer of encryption, we will apply an *individual* permutation that hides the logical value of the input bit associated with that encryption only.

So for the first input, we randomly swap the top and bottom halves of the table. Note that there are only two options for this permutation!

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c^1_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c^2_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c^3_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c^4_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$



w_1	w_2	w_4	Half-Permuted Garbled
			$\tilde{c}^1_{134} :=$
			$\tilde{c}^2_{134} :=$
			$\tilde{c}^3_{134} :=$
			$\tilde{c}^4_{134} :=$

Optimizing GC: Point and Permute

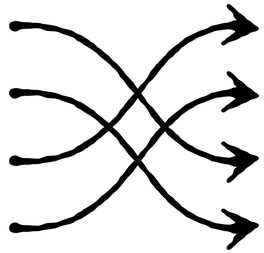
An alternative approach: Let's use *regular* (non-evasive-range) IND-CPA SKE with no overhead.

For each *individual* layer of encryption, we will apply an *individual* permutation that hides the logical value of the input bit associated with that encryption only.

So for the first input, we randomly swap the top and bottom halves of the table. Note that there are only two options for this permutation!

Now, anywhere that we encrypt k_1^b we can also encrypt an extra pointer p_1^b that tells Bob whether k_1^b corresponds to the top or bottom halves of the garbled table! In this example, $p_1^0 = 1$ and $p_1^1 = 0$.

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c^1_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c^2_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c^3_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c^4_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$

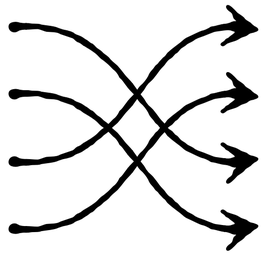


w_1	w_2	w_4	Half-Permuted Garbled
1	0	0	$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	0	0	$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$

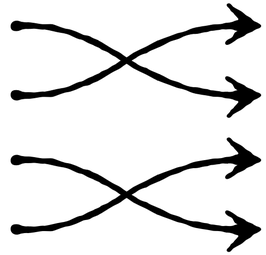
Optimizing GC: Point and Permute

Now, anywhere that we encrypt k_1^b we can also encrypt an extra pointer p_1^b that tells Bob whether k_1^b corresponds to the top or bottom halves of the garbled table! In this example, $p_1^0 = 1$ and $p_1^1 = 0$.

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c_{134}^1 := \mathsf{Enc}_{k_1^0}\left(\mathsf{Enc}_{k_3^0}\left(k_4^0\right)\right)$
0	1	0	$c_{134}^2 := \mathsf{Enc}_{k_1^0}\left(\mathsf{Enc}_{k_3^1}\left(k_4^0\right)\right)$
1	0	0	$c_{134}^3 := \mathsf{Enc}_{k_1^1}\left(\mathsf{Enc}_{k_3^0}\left(k_4^0\right)\right)$
1	1	1	$c_{134}^4 := \mathsf{Enc}_{k_1^1}\left(\mathsf{Enc}_{k_3^1}\left(k_4^1\right)\right)$



w_1	w_2	w_4	Half-Permuted Garbled
1	0	0	$\tilde{c}_{134}^1 := \mathsf{Enc}_{k_1^1}\left(\mathsf{Enc}_{k_3^0}\left(k_4^0\right)\right)$
1	1	1	$\tilde{c}_{134}^2 := \mathsf{Enc}_{k_1^1}\left(\mathsf{Enc}_{k_3^1}\left(k_4^1\right)\right)$
0	0	0	$\tilde{c}_{134}^3 := \mathsf{Enc}_{k_1^0}\left(\mathsf{Enc}_{k_3^0}\left(k_4^0\right)\right)$
0	1	0	$\tilde{c}_{134}^4 := \mathsf{Enc}_{k_1^0}\left(\mathsf{Enc}_{k_3^1}\left(k_4^0\right)\right)$



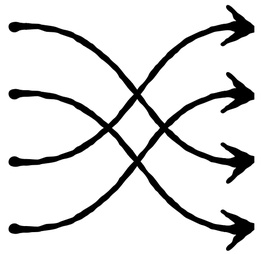
w_1	w_2	w_4	Garbled
			$\tilde{c}_{134}^1 :=$
			$\tilde{c}_{134}^2 :=$
			$\tilde{c}_{134}^3 :=$
			$\tilde{c}_{134}^4 :=$

For the second input, we randomly swap the first and second rows, and swap the third and fourth rows to match (i.e. we either swap both pairs, or swap neither pair).

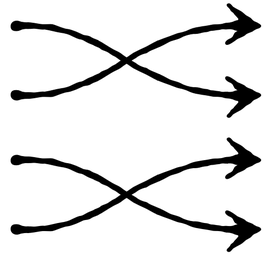
Optimizing GC: Point and Permute

Now, anywhere that we encrypt k_1^b we can also encrypt an extra pointer p_1^b that tells Bob whether k_1^b corresponds to the top or bottom halves of the garbled table! In this example, $p_1^0 = 1$ and $p_1^1 = 0$.

w_1	w_2	w_4	Unpermuted Garbled
0	0	0	$c_{134}^1 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$c_{134}^2 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
1	0	0	$c_{134}^3 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$c_{134}^4 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$



w_1	w_2	w_4	Half-Permuted Garbled
1	0	0	$\tilde{c}_{134}^1 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
1	1	1	$\tilde{c}_{134}^2 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
0	0	0	$\tilde{c}_{134}^3 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$\tilde{c}_{134}^4 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$



w_1	w_2	w_4	Garbled
1	1	1	$\tilde{c}_{134}^1 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
1	0	0	$\tilde{c}_{134}^2 := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
0	1	0	$\tilde{c}_{134}^3 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
0	0	0	$\tilde{c}_{134}^4 := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

For the second input, we randomly swap the first and second rows, and swap the third and fourth rows to match (i.e. we either swap both pairs, or swap neither pair).

Anywhere that we encrypt or transmit k_1^b we can also encrypt or transmit an extra pointer p_2^b that tells Bob whether k_1^b corresponds to rows $\{1,3\}$ or $\{2,4\}$.

To make sure we only need one pointer bit per wire, swaps must be *consistent* everywhere k_i^b for $b \in \{0,1\}$ is used (i.e. every time w_i is the input wire for a gate). This correlation among gate permutations doesn't hurt security: we're only attempting to hide the value of b.

Optimizing GC: Point and Permute

To make sure we only need one pointer bit per wire, swaps must be *consistent* everywhere k_i^b for $b \in \{0,1\}$ is used (i.e. every time w_i is the input wire for a gate). This correlation among gate permutations doesn't hurt security: we're only attempting to hide the value of b.

Notice that since Bob always knows *exactly* which row he can correctly decrypt, we do not need to use evasive-range encryption. Now a garbled table has only $4\kappa + 4$ bits (the extra 4 are for encrypting the pointers), and our scheme is perfectly correct!

Furthermore Bob doesn't waste computation time "testing" rows he can't decrypt correctly.

This is known as the *Point and Permute* technique of Beaver, Micali, and Rogaway (but the "BMR protocol" refers to something else they introduced in the same paper!)



Garbling Optimization Scorecard

Name	Year	Authors	Bits per AND	Bits per XOR
Yao's Garbled Circuits	1986	Yao	12κ	12κ
Point and Permute	1990	Beaver, Micali, Rogaway	4κ	4κ
Garbled Row Reduction 3	1999	Naor, Pinkas, Sumner	3κ	3κ
Free XOR	2008	Kolesnikov, Schneider	3κ	0
Garbled Row Reduction 2	2009	Pinkas, Schneider, Smart, Williams	2κ	2κ
Half Gates	2015	Zahur, Rosulek, Evans	2κ	0
Slice and Dice	2022	Rosulek, Roy	1.5κ	0
???	???	*Your Name Here*	???	0

UVa Grad Student

UVa Professor

4. Generalizing to many parties: the BMR Protocol.



Toward Constant Rounds for Many Parties

- We want *any* number of parties to be able to perform secure computation in a constant number of rounds.
- The protocol we have just seen has asymmetric roles for Alice and Bob, so it's not obvious how we would generalize it for more parties. *Any ideas?*
- Recall: the round count of GMW depends upon the *multiplicative depth* of the circuit that it computes.
- Observation: When Alice is garbling, she can sample the wire keys in parallel, and then garble all gates in parallel.
- Suppose she wants to garble a circuit C. If we think of her process for garbling C as an instance of a randomized algorithm Garble_C that takes no inputs, and outputs the garbled tables plus all of the input and output keys, then the circuit C' that computes Garble_C can have multiplicative depth *independent of* C.
- Notice that the same isn't true of Bob - he must evaluate the circuit in order.

The BMR Protocol (Intuition)

For some $n \in \mathbb{N}$ and some n-ary function $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$, let C_f be a boolean circuit that computes f.

Let g be an n-ary function that takes n inputs from the same domains as f, and runs Garble_{C_f}. The i^{th} output of g comprises:

- The garbling of C_f.
- The input wire key(s) corresponding to the input x_j for every $j \in [n]$.
- The output wire key(s) corresponding to all possible values of the output y_i.

Suppose that parties $P_1, \ldots, P_n$ wish to securely compute f. They can do this via the BMR protocol, as follows:

1. The parties invoke $\mathcal{F}_{\mathsf{SFE}}(n, g, \mathcal{X}_1, \ldots, \mathcal{X}_n)$. Each party P_i sends its input x_i.
2. The parties all receive identical garblings of C_f, and identical sets of input keys. Each party evaluates the circuit as Bob would to arrive at a set of output keys.

The BMR Protocol (Intuition)

Suppose that parties $P_1, \ldots, P_n$ wish to securely compute f. They can do this via the BMR protocol, as follows:

1. The parties invoke $\mathcal{F}_{\mathsf{SFE}}(n, g, \mathcal{X}_1, \ldots, \mathcal{X}_n)$. Each party P_i sends its input x_i.
2. The parties all receive *identical* garblings of C_f, and identical sets of input keys. Each party evaluates the circuit as Bob would to arrive at a set of output keys.
3. Each party uses the individual output table that it received to decode its own output from the circuit evaluation in the last step.

Next, we construct a circuit C_g that computes g with multiplicative depth independent of the structure of C_f.

Finally, we can use the composition theorem to replace $\mathcal{F}_{\mathsf{SFE}}(n, g, \mathcal{X}_1, \ldots, \mathcal{X}_n)$ with $\pi_{\mathsf{GMW}}(n, t, C_g)$. Since C_g has multiplicative depth independent of C_f, the round count of the GMW protocol is also independent of C_f (and f, by proxy).

The BMR Protocol (Intuition)

Finally, we can use the composition theorem to replace $\mathcal{F}_{\mathsf{SFE}}(n, g, \mathcal{X}_1, \ldots, \mathcal{X}_n)$ with $\pi_{\mathsf{GMW}}(n, t, C_g)$. Since C_g has multiplicative depth independent of C_f, the round count of the GMW protocol is also independent of C_f (and f, by proxy).

Thus we have:

Theorem 2: Let $t < n \in \mathbb{N}$, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be any randomized poly-time n-ary function, and let C_f be a poly-size boolean circuit that computes f. Assuming synchronicity, secure channels, and the existence of Pseudorandom Functions, there exists an n-party protocol in the $\mathcal{F}_{\mathsf{OT}}(2,1)$-hybrid model that realizes $\mathcal{F}_{\mathsf{SFE}}(n, f, \mathcal{X}_1, \ldots, \mathcal{X}_n)$ in the presence of a semi-honest $\mathcal{A}$ that statically corrupts up to t parties. Moreover, the round count of this protocol *does not* depend upon f.

The BMR Protocol (Intuition)

Theorem 2: Let $t < n \in \mathbb{N}$, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be any randomized poly-time n-ary function, and let C_f be a poly-size boolean circuit that computes f. Assuming synchronicity, secure channels, and the existence of Pseudorandom Functions, there exists an n-party protocol in the $\mathcal{F}_{\mathsf{OT}}(2,1)$-hybrid model that realizes $\mathcal{F}_{\mathsf{SFE}}(n, f, \mathcal{X}_1, \ldots, \mathcal{X}_n)$ in the presence of a semi-honest $\mathcal{A}$ that statically corrupts up to t parties. Moreover, the round count of this protocol *does not* depend upon f.

- Note that in this simplified description, C_g involves evaluating a PRF many times in parallel, so its depth might depend upon κ (the security parameter).
- BMR showed that there is a way for the parties to *individually* and *locally* compute all of the PRF evaluations, before invoking $\mathcal{F}_{\mathsf{SFE}}(n, g', \mathcal{X}_1, \ldots, \mathcal{X}_n)$ with a modified g' that garbles C_f by combining the outputs of those evaluations.
- They also showed that there is a *constant-depth* $C_{g'}$ that can be used to realize $\mathcal{F}_{\mathsf{SFE}}(n, g', \mathcal{X}_1, \ldots, \mathcal{X}_n)$, yielding a constant-round protocol overall.

CS4501 Cryptographic Protocols Lecture 17: Proof for Yao's Garbled Circuits, Point and Permute, BMR

https://jackdoerner.net/teaching/#2026/Spring/CS4501