CS4501 Cryptographic Protocols
Lecture 16: Pseudorandom Functions, Garbled Circuits

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

Recall: Preview of Garbled Circuits

AND ($\wedge$)

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Suppose that we were to pick a *random* string of length κ to encode each possible value on each of the wires.

AND ($\wedge$)

x	y	$x \wedge y$
k_0^x	k_0^y	$k_0^{x \wedge y}$
k_0^x	k_1^y	$k_0^{x \wedge y}$
k_1^x	k_0^y	$k_0^{x \wedge y}$
k_1^x	k_1^y	$k_1^{x \wedge y}$

Now we use the input wire encodings to encrypt the output wire encodings

"Garbled"

$x \wedge y$
$\mathsf{Enc}_{k_0^x}\left(\mathsf{Enc}_{k_0^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_0^x}\left(\mathsf{Enc}_{k_1^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_1^x}\left(\mathsf{Enc}_{k_0^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_1^x}\left(\mathsf{Enc}_{k_1^y}\left(k_1^{x \wedge y}\right)\right)$

Suppose Alice garbles a gate and sends the garbled table to Bob. What can Bob learn if Alice also sends him one of (k_0^x, k_1^x), and he uses OT to choose one of (k_0^y, k_1^y) for himself?

I claim that you should be unhappy about this slide! There are three problems. *What are they?*

1. What happens when Bob decrypts the wrong row in the table? Does it look any different from decrypting the right row?

2. If Bob *can* determine which row is the right one, what does he learn? Is this a problem?

Recall: Preview of Garbled Circuits

AND ($\wedge$)

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Suppose that we were to pick a *random* string of length κ to encode each possible value on each of the wires.

AND ($\wedge$)

x	y	$x \wedge y$
k_0^x	k_0^y	$k_0^{x \wedge y}$
k_0^x	k_1^y	$k_0^{x \wedge y}$
k_1^x	k_0^y	$k_0^{x \wedge y}$
k_1^x	k_1^y	$k_1^{x \wedge y}$

Now we use the input wire encodings to encrypt the output wire encodings

"Garbled"

$x \wedge y$
$\mathsf{Enc}_{k_0^x}\left(\mathsf{Enc}_{k_0^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_0^x}\left(\mathsf{Enc}_{k_1^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_1^x}\left(\mathsf{Enc}_{k_0^y}\left(k_0^{x \wedge y}\right)\right)$
$\mathsf{Enc}_{k_1^x}\left(\mathsf{Enc}_{k_1^y}\left(k_1^{x \wedge y}\right)\right)$

2. If Bob *can* determine which row is the right one, what does he learn? Is this a problem?

3. We're encrypting multiple messages with the same key, and all those messages are at least as long as the key.
   - Computational OTP scheme from last lecture doesn't allow key reuse.
   - ElGamal encryption (Lecture 14) only allows us to encrypt a few bits at a time.
   - So we need yet another kind of encryption that we haven't seen yet.

The Rest of this Lecture

We'll work our way through these three problems and show how to connect gates to one another in order to arrive at a secure garbled circuits protocol.

1. We'll start by introducing another pseudorandom object that will help us construct CPA-secure SKE with long messages.

2. We'll prove that our CPA-secure SKE scheme has a couple of extra properties that we need to help us resolve the issues with our garbled tables.

3. We'll show how to garble a whole circuit, and start toward a security proof.

4. Next time, we'll finish the security proof and talk about optimizations and generalizations.

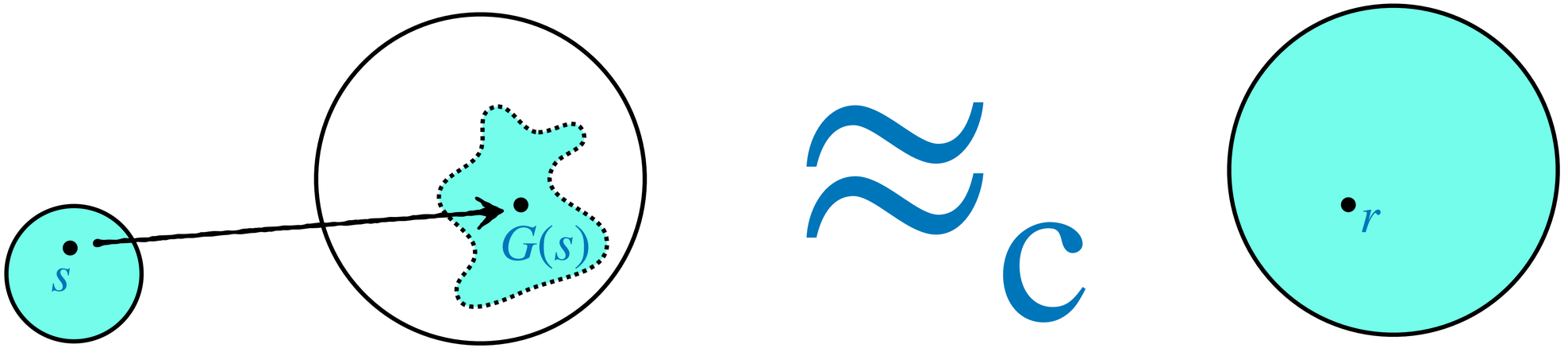
1. Pseudorandom Functions

Pseudorandomness

Definition 1 (Pseudorandom Generator). Let ℓ be a polynomial and let $G : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time function such that for any $\kappa \in \mathbb{N}$ and any $s \in \{0,1\}^\kappa$, $G(s) \in \{0,1\}^{\ell(\kappa)}$.

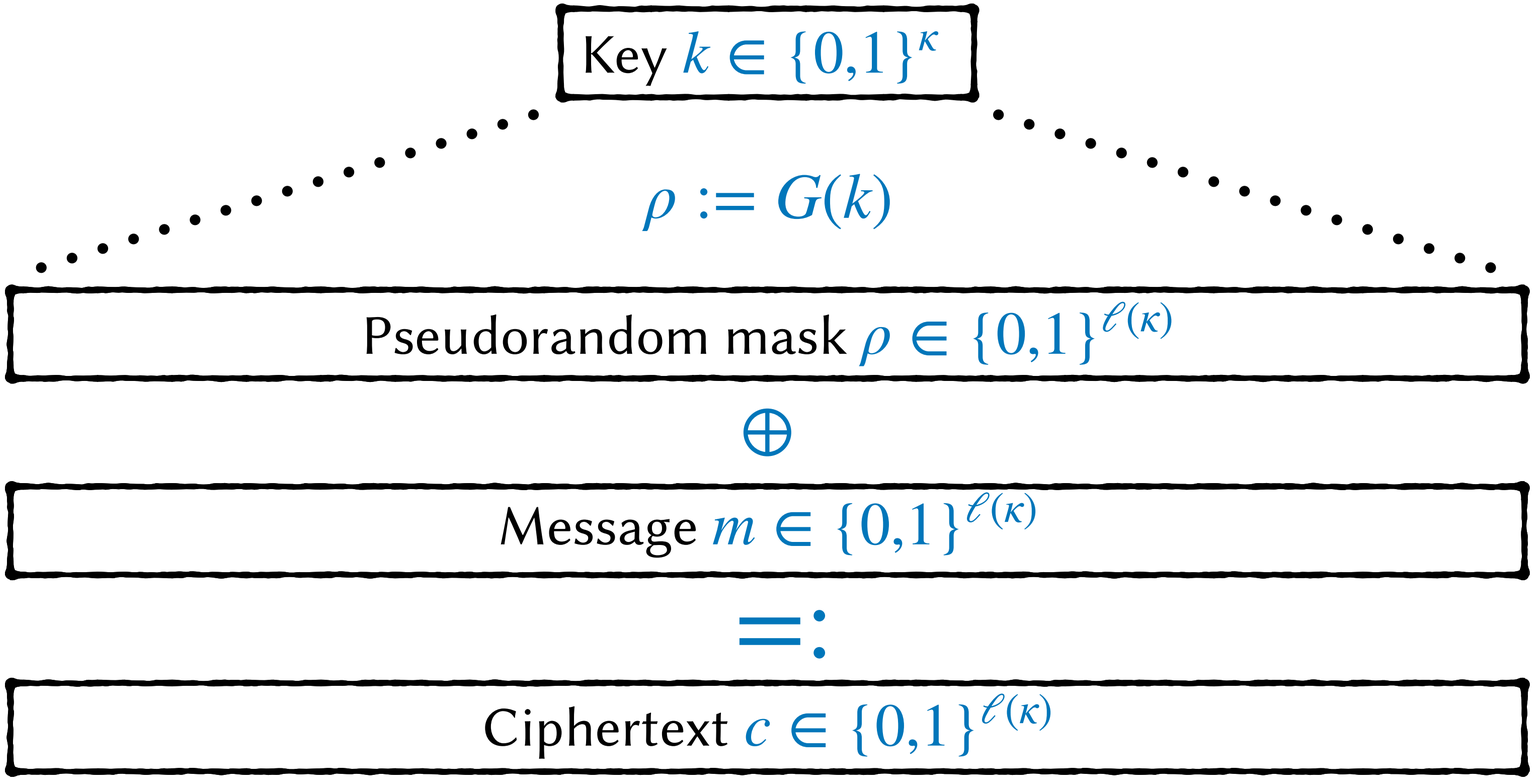
G is a *pseudorandom generator* if and only if $\ell(\kappa) > \kappa$ and $\forall$ PPT $\mathscr{D}$ $\exists$ negligible ε s.t.

$$\left|\Pr[\mathscr{D}(1^\kappa, G(s)) = 1 : s \leftarrow \{0,1\}^\kappa] - \Pr[\mathscr{D}(1^\kappa, r) = 1 : r \leftarrow \{0,1\}^{\ell(\kappa)}]\right| \leq \varepsilon(\kappa)$$



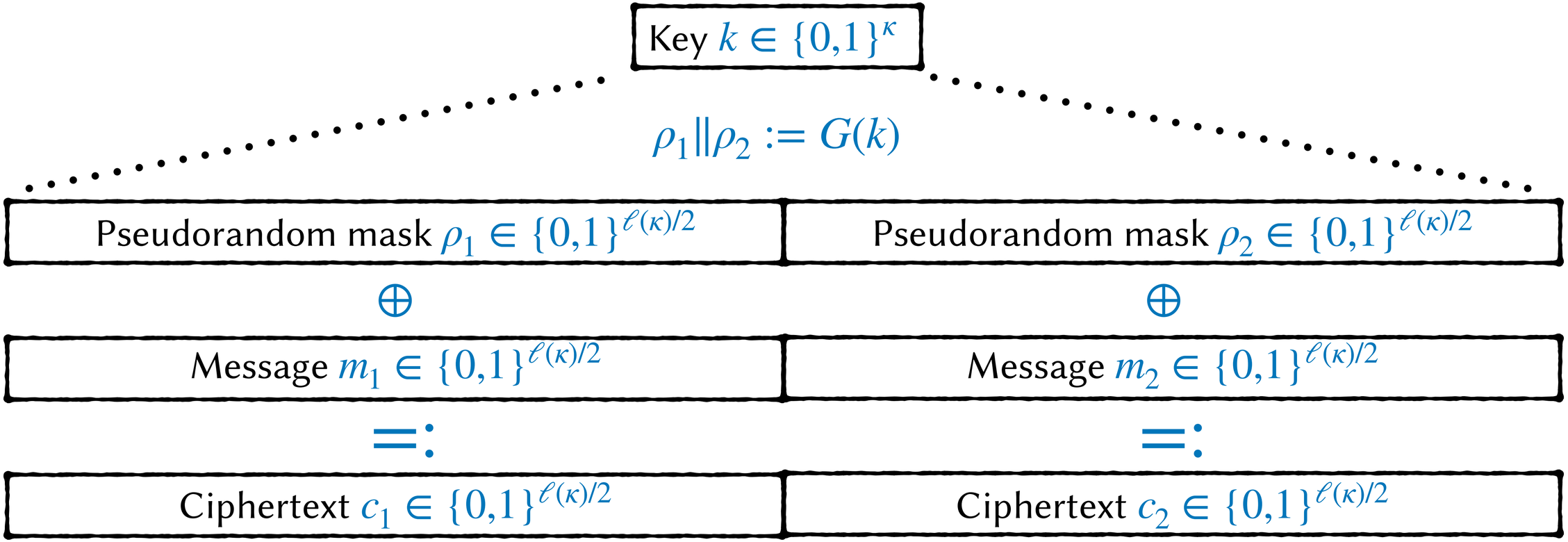
Uniform over $\{0,1\}^\kappa$ Image of G in $\{0,1\}^{\ell(\kappa)}$ Uniform distribution over $\{0,1\}^{\ell(\kappa)}$

Recall: Computational OTP



Encrypting one message was very intuitive. *What can we change to encrypt two messages?*

Computational *Two*-Time Pad



This is a good start. But if we want to encrypt *many* messages, we must keep state (which message is which in the sequence), and there's no guarantee we can compute just one part of the mask at a time; G might have to be computed all at once…

Idea:

Imagine that we could set $\ell(\kappa)$ to be *exponentially* large, and we had an easy way to compute short segments of it (say κ bits at a time) efficiently without computing the whole thing.

If we had $\ell(\kappa) = \kappa \cdot 2^\kappa$, then every time we need to encrypt, we can pick one of the 2^κ segments of κ bits *randomly*. The probability that we ever pick the same segment for two different encryptions is negligible in κ.

Although the key is secret and the bits in the segment we pick are secret, the location of the segment we use does *not* have to be secret. In fact, it had better not be, because you need this information in the clear in order to decrypt correctly.

We can use another PRG to expand each segment of κ bits into a mask that's the same length as the message is.

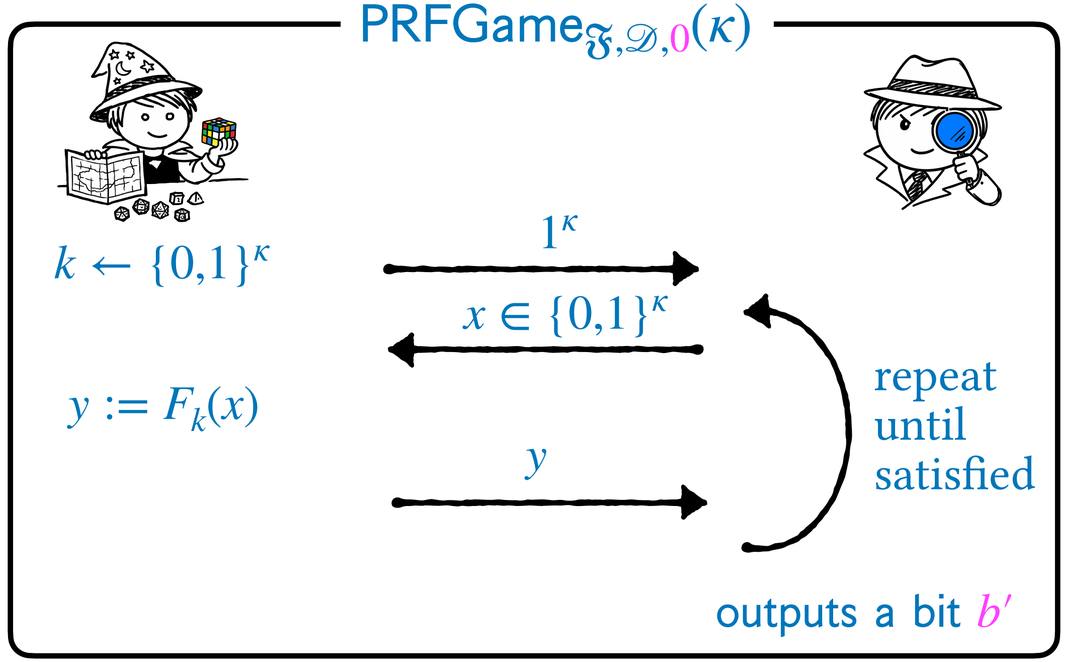
We will formalize this idea: we call our new primitive a *Pseudorandom Function*.

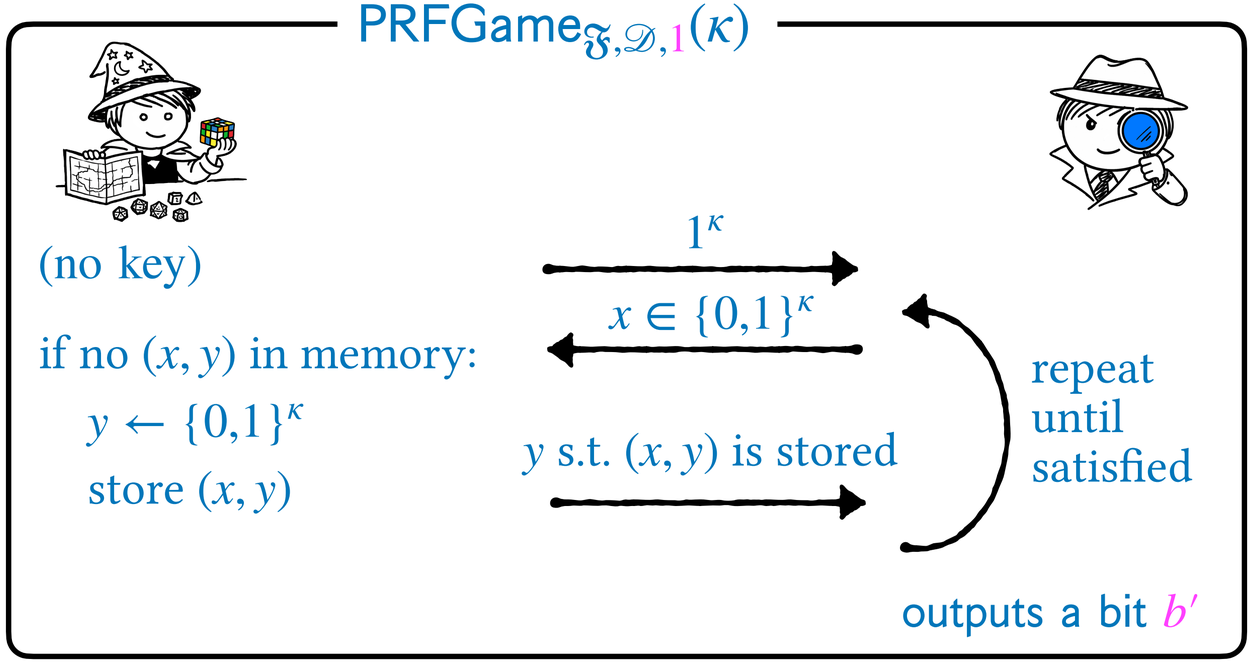
Pseudorandom *Functions*

Definition 2 (Pseudorandom Function). A family of length-preserving functions $\mathfrak{F} = \left\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\right\}_{k \in \{0,1\}^*}$ is a pseudorandom function family if:

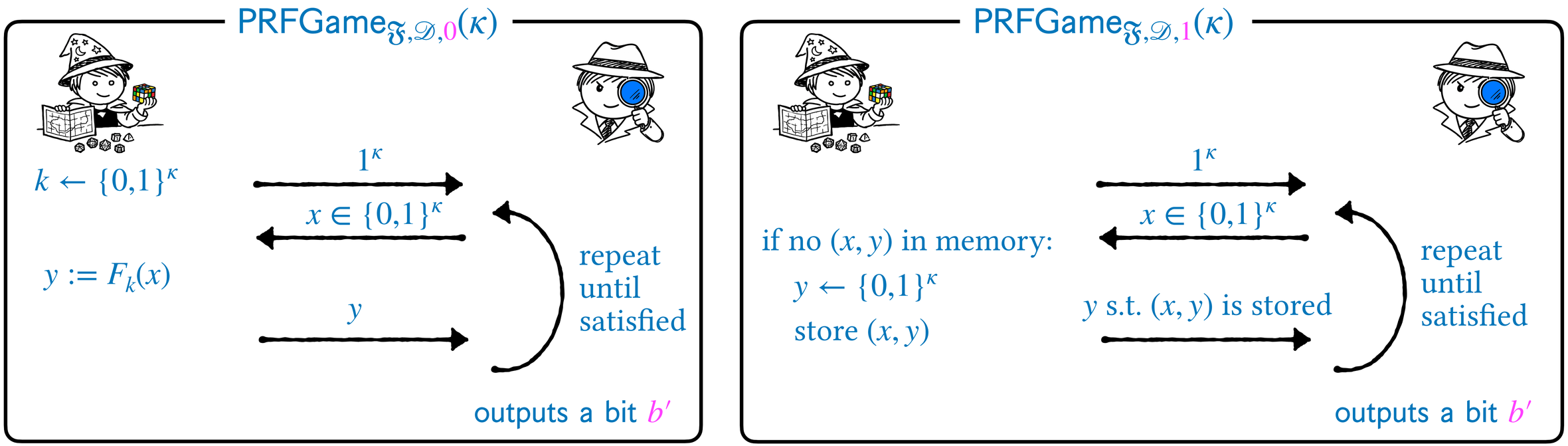
1. For every $k \in \{0,1\}^*$, F_k is poly-time.
2. $\forall$ PPT $\mathscr{D}$ $\exists$ a negligible function ε such that

$$\left| \Pr\left[\mathsf{PRFGame}_{\mathfrak{F},\mathscr{D},0}(\kappa) = 1\right] - \Pr\left[\mathsf{PRFGame}_{\mathfrak{F},\mathscr{D},1}(\kappa) = 1\right] \right| \leq \varepsilon(\kappa)$$





Pseudorandom *Functions*



These two games are very different! One of the challengers is stateful, and one isn't!

Notice: k is secret, like the secret seed s for a PRG. If the distinguisher learns k (or s) then distinguishing is very easy! *How?*

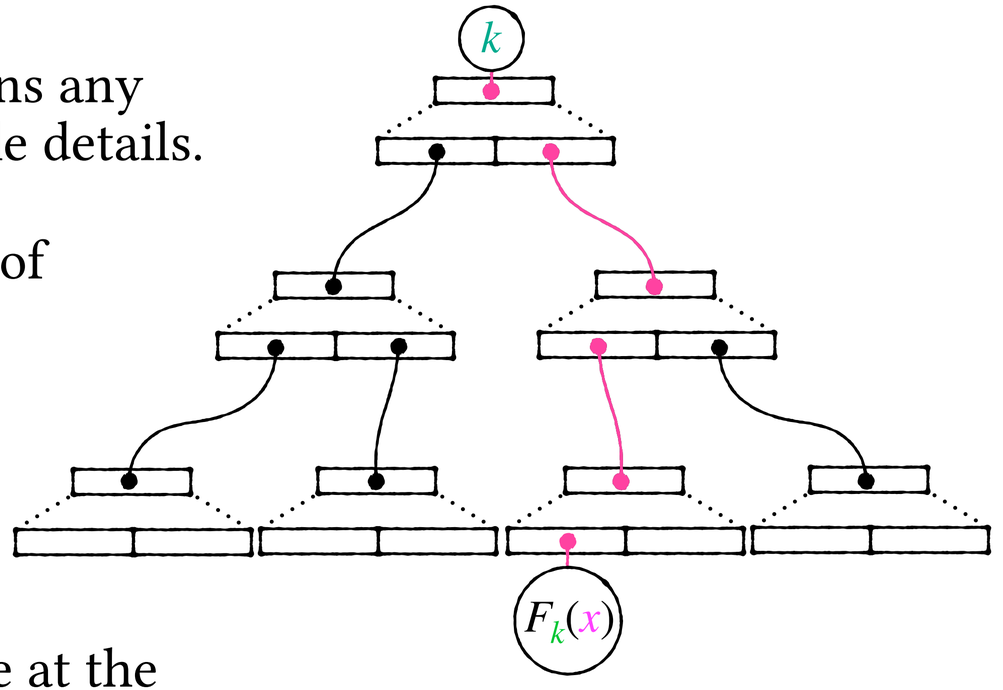
On the other hand the *input* x is public and *adaptively* chosen by the distinguisher.

Functions vs Generators

It's easy to see that a PRF implies a PRG with *dynamic* stretch. *How?*

We can also prove that any PRG implies a PRF (which means any OWF implies a PRF). It's not hard, but there are a few subtle details.

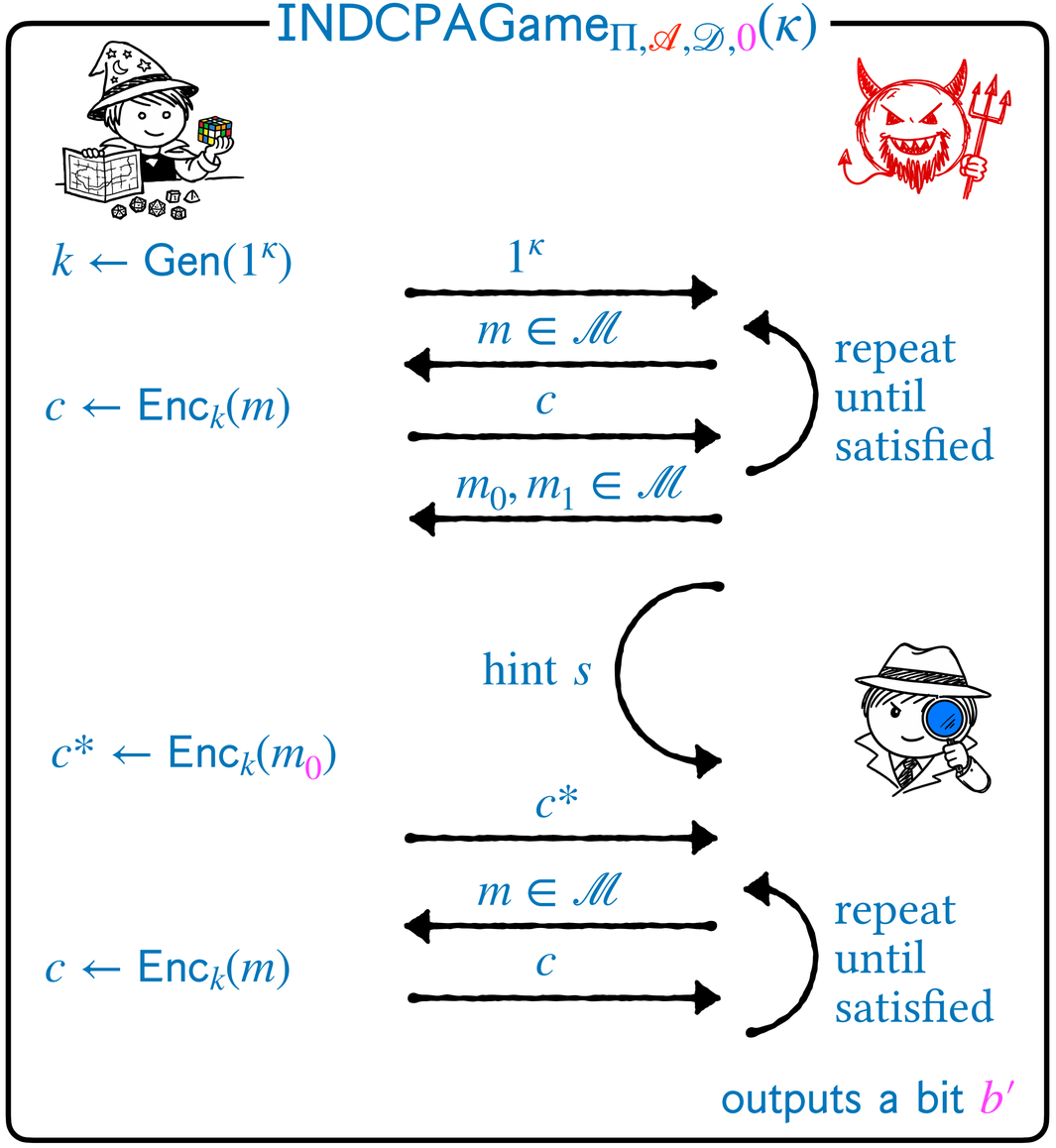
Imagine you have a PRG that expands its input by a factor of two. You can use it recursively, arranged in a tree (so each left child of a node uses the left side of the output at that node as a new seed, and likewise on the right). After κ layers, you have 2^κ leaves, each of size κ.

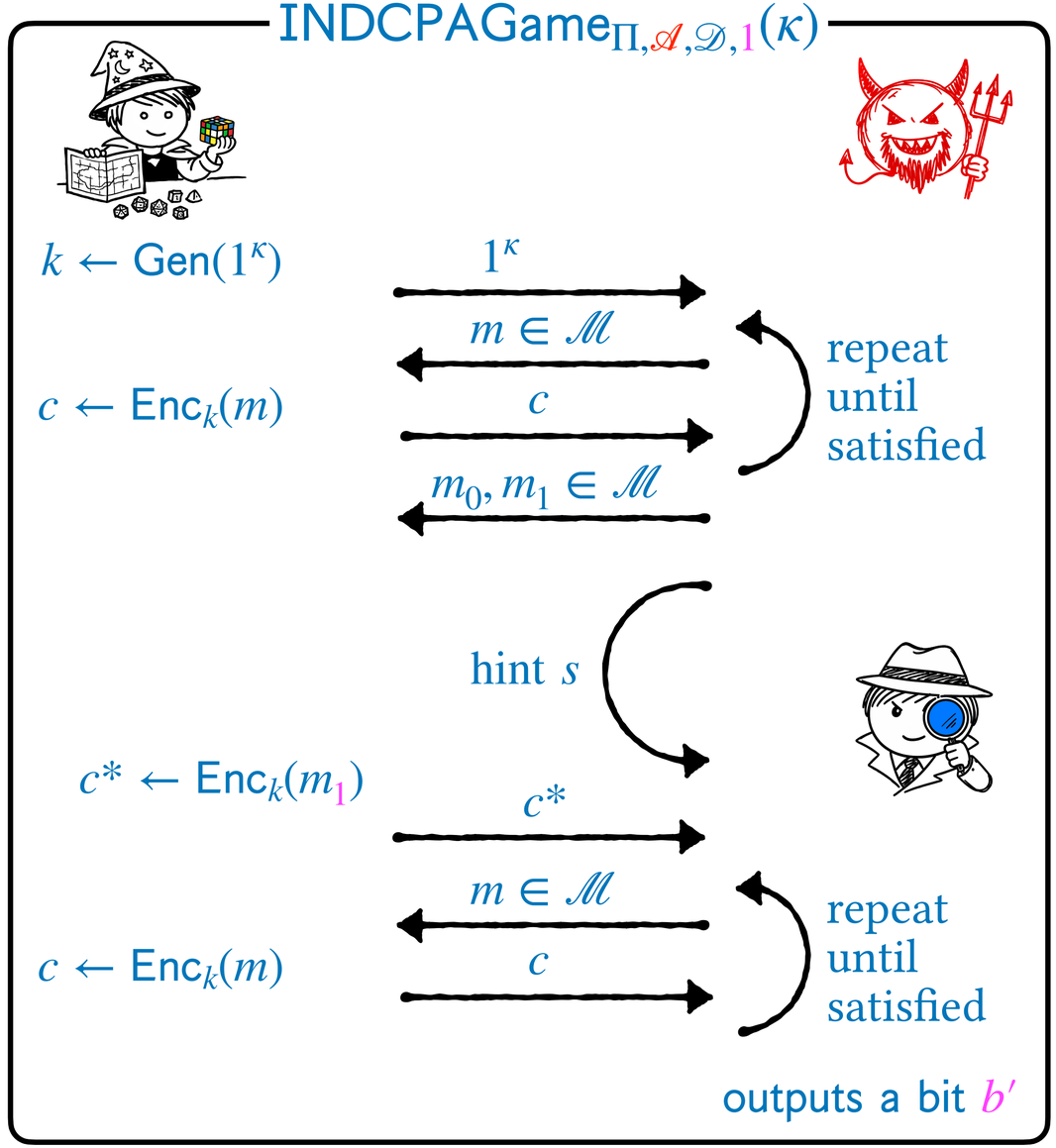


The secret PRF key k is the secret seed for the PRG instance at the root of the tree. Notice: k fixes all the outputs! The public PRF input x determines the path (i.e. it determines which leaf to output).

Can you prove that this is secure (and if so, do you want to guest lecture in CS6222 next semester)?

Recall **Def 3**: The IND-CPA Game (for SKE)





IND-CPA Secure SKE from a PRF

Definition 4 (the canonical SKE). For polynomial ℓ, let $\mathcal{M} = \{0,1\}^{\ell(\kappa)}$. Let $G : \{0,1\}^* \to \{0,1\}^*$ be PRG with output length $\ell(\kappa)$, and let $\mathfrak{F} = \left\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\right\}_{k \in \{0,1\}^*}$ be a PRF.

- $\mathsf{Gen}(1^\kappa)$ outputs $k \leftarrow \{0,1\}^\kappa$.
- $\mathsf{Enc}_k(m)$ outputs $c := (r, m \oplus G(F_k(r)))$ where $r \leftarrow \{0,1\}^\kappa$.
- $\mathsf{Dec}_k(c)$ outputs $m' := c' \oplus G(F_k(r))$ where $(r, c') = c$.

Theorem 1: The above scheme is IND-CPA secure.

Proof Idea: Consider a first hybrid game that uses a stateful, truly random function instead of F_k (like version 1 of the PRF game). By reduction to the pseudorandomness of F_k, this hybrid game is indistinguishable from the IND-CPA game. Notice that in the first hybrid game the input to G is truly random and sampled fresh for each additional encryption. Our second hybrid game uses OTP, with a freshly sampled key for each encryption. By reduction to the pseudorandomness of G, the second game is indistinguishable from the first. Perfect secrecy of OTP completes the proof. ∎

SKE in Practice

Definition 4 (the canonical SKE). For polynomial ℓ, let $\mathcal{M} = \{0,1\}^{\ell(\kappa)}$. Let $G : \{0,1\}^* \to \{0,1\}^*$ be PRG with output length $\ell(\kappa)$, and let $\mathfrak{F} = \left\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\right\}_{k \in \{0,1\}^*}$ be a PRF.

- $\mathsf{Gen}(1^\kappa)$ outputs $k \leftarrow \{0,1\}^\kappa$.
- $\mathsf{Enc}_k(m)$ outputs $c := (r, m \oplus G(F_k(r)))$ where $r \leftarrow \{0,1\}^\kappa$.
- $\mathsf{Dec}_k(c)$ outputs $m' := c' \oplus G(F_k(r))$ where $(r, c') = c$.

Apart from implementation details, this is really how your computer does encryption when you visit a website or send a message. But the devil is in the details, and there are a lot of them!

Similarly, DHKE is really how the key k is generated (though you need to specify a group and turn the agreed-upon group element into a string of bits).

The "details" are specified in the TLS standard, originally developed by Taher ElGamal.

2. Evasive-Range Encryption

Another Look at the Problems

1. Decrypting the wrong row of the table gives Bob a random-looking string. Decrypting the right row of the table also gives Bob a random-looking string. **So we need an encryption scheme that lets us know if we've decrypted correctly or not…** ***any ideas?***

2. If Bob *can* determine which row is the right one, the position of that row in the table leaks the output of the gate, which is a problem for internal wires. **Solved by shuffling the rows of the table randomly before sending it.**

3. We're encrypting multiple messages with the same key, and all those messages are at least as long as the key. **Solved using IND-CPA secure SKE from PRFs.**

A Simple Adjustment to our SKE

Definition 5 (SKE with Elusive, Efficiently Verifiable Range). Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an SKE scheme for $\mathcal{M} = \{0,1\}^{\ell(\kappa)}$, where ℓ is some polynomial.

1. We say Π has an *elusive range* if $\forall$ PPT $\mathcal{A}$ $\exists$ negligible ε such that $\Pr[\mathcal{A}(1^\kappa) \in \mathrm{range}(\mathsf{Enc}_k) : k \leftarrow \mathsf{Gen}(1^\kappa)] \leq \varepsilon(\kappa)$.

2. We say Π has an *efficiently verifiable range* if $\forall k \in \mathrm{range}(\mathsf{Gen})$, $\forall c \notin \mathrm{range}(\mathsf{Enc}_k)$, $\mathsf{Dec}_k(c) = \perp$

How to construct such an encryption scheme? *Any ideas?*

Let $G : \{0,1\}^* \to \{0,1\}^*$ be PRG with output length $\ell(\kappa) + \kappa$.

$\mathsf{Enc}_k(m)$ outputs $c := (r, (m \| 0^\kappa) \oplus G(F_k(r)))$ where $r \leftarrow \{0,1\}^\kappa$.

Dec_k verifies that c is in range by checking that its last κ bits are the same as those of $G(F_k(r))$.

By pseudorandomness of F_k and G, finding r and the κ bits of $G(F_k(r))$ is *hard* without k.

3a. Yao’s Garbled Circuits

Yao's Garbled Circuits Protocol $\pi_{\mathsf{Yao}}(C)$

There are two parties, Alice (P_1) and Bob (P_2) with inputs $x_1, x_2 \in \mathbb{F}_2$ respectively. π_{Yao} computes a well-formed 2-ary *boolean* circuit $C : \mathbb{F}_2^2 \to \mathbb{F}_2^2$. The parties output $y_1, y_2 \in \mathbb{F}_2$ respectively. Generalization to many input and output bits is easy.

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure SKE scheme for κ-bit messages, with an evasive and efficiently-verifiable range. For simplicity assume that random gates are rewritten as usual, in terms of $\oplus$ and in.

There are Three Phases:

1. **Circuit Garbling:** For the output wire of every in, $\oplus$, and $\wedge$ gate in the circuit, Alice (the garbler) samples two keys $k_o^0, k_o^1 \leftarrow \mathsf{Gen}(1^\kappa)$. For every $(\neg, i, o)$ gate, she sets $k_o^0 := k_i^1$ and $k_o^1 := k_i^0$. She iterates though the $\oplus$, $\wedge$, and out gates in any order:

Yao's Garbled Circuits Protocol $\pi_{\mathsf{Yao}}(C)$

1. **Circuit Garbling:** For the output wire of every in, $\oplus$, and $\wedge$ gate in the circuit, Alice (the garbler) samples two keys $k_o^0, k_o^1 \leftarrow \mathsf{Gen}(1^\kappa)$. For every $(\neg, i, o)$ gate, she sets $k_o^0 := k_i^1$ and $k_o^1 := k_i^0$. She iterates though the $\oplus$, $\wedge$, and out gates in any order:

   - Suppose Alice arrives at the gate $(\oplus, i, j, o) \in C$. She computes the garbled table:

$$c_{ijo}^1 := \mathsf{Enc}_{k_i^0}\left(\mathsf{Enc}_{k_j^0}\left(k_o^0\right)\right)$$

$$c_{ijo}^2 := \mathsf{Enc}_{k_i^0}\left(\mathsf{Enc}_{k_j^1}\left(k_o^1\right)\right)$$

$$c_{ijo}^3 := \mathsf{Enc}_{k_i^1}\left(\mathsf{Enc}_{k_j^0}\left(k_o^1\right)\right)$$

$$c_{ijo}^4 := \mathsf{Enc}_{k_i^1}\left(\mathsf{Enc}_{k_j^1}\left(k_o^0\right)\right)$$

and sets $g_{ijo} = \left(\tilde{c}_{ijo}^1, \tilde{c}_{ijo}^2, \tilde{c}_{ijo}^3, \tilde{c}_{ijo}^4\right)$ to be a random shuffling of $\left(c_{ijo}^1, c_{ijo}^2, c_{ijo}^3, c_{ijo}^4\right)$.

Then she sends (j, k, o, g_{ijo}) to Bob.

Yao's Garbled Circuits Protocol $\pi_{\mathsf{Yao}}(C)$

- Suppose Alice arrives at the gate $(\wedge, i, j, o) \in C$. She computes the garbled table:

$$c^1_{ijo} := \mathsf{Enc}_{k^0_i}\left(\mathsf{Enc}_{k^0_j}\left(k^0_o\right)\right)$$

$$c^2_{ijo} := \mathsf{Enc}_{k^0_i}\left(\mathsf{Enc}_{k^1_j}\left(k^0_o\right)\right)$$

$$c^3_{ijo} := \mathsf{Enc}_{k^1_i}\left(\mathsf{Enc}_{k^0_j}\left(k^0_o\right)\right)$$

$$c^4_{ijo} := \mathsf{Enc}_{k^1_i}\left(\mathsf{Enc}_{k^1_j}\left(k^1_o\right)\right)$$

and sets $g_{ijo} = \left(\tilde{c}^1_{ijo}, \tilde{c}^2_{ijo}, \tilde{c}^3_{ijo}, \tilde{c}^4_{ijo}\right)$ to be a random shuffling of $\left(c^1_{ijo}, c^2_{ijo}, c^3_{ijo}, c^4_{ijo}\right)$.

Then she sends (j, k, o, g_{ijo}) to Bob.

Notice: Since all communication so far is unidirectional and all computation is independent of the inputs, it is a single message that can be preprocessed.

Yao's Garbled Circuits Protocol $\pi_{\mathsf{Yao}}(C)$

2. **Input/Output Delivery:**

   - Alice finds the gate $(\mathsf{in},1,o) \in C$, if it exists, and sends $\left(o, k_o^{x_1}\right)$ to Bob.
   - Alice finds the gate $(\mathsf{in},2,o) \in C$, if it exists, and sends $\left(\left(o, k_o^0\right), \left(o, k_o^1\right)\right)$ to $\mathcal{F}_{\mathsf{OT}}(2,1)$. Bob sends x_2 to $\mathcal{F}_{\mathsf{OT}}(2,1)$ and receives $\left(o, k_o^{x_2}\right)$in reply.
   - Alice finds the gate $(\mathsf{out},2,i) \in C$ and sends $\left(i, k_i^0, k_i^1\right)$ to Bob.

3. **Circuit Evaluation:** Bob (the evaluator) iterates though the $\oplus$, $\wedge$, and out gates in topological order. If wires i and j are the input wires of Alice and Bob, then he begins with knowledge of $k_i^{w_i} = k_i^{x_1}$ and $k_j^{w_j} = k_j^{x_2}$.

   - Suppose he arrives at gate $(\wedge, i, j, o)$ or $(\oplus, i, j, o)$. He finds the corresponding $g_{ijo} = \left(\tilde{c}_{ijo}^1, \tilde{c}_{ijo}^2, \tilde{c}_{ijo}^3, \tilde{c}_{ijo}^4\right)$ that was sent by Alice, and he knows $k_i^{w_i}$ and $k_j^{w_j}$ already.

Yao's Garbled Circuits Protocol $\pi_{\mathsf{Yao}}(C)$

3. **Circuit Evaluation:** Bob (the evaluator) iterates though the $\oplus$, $\wedge$, and out gates in topological order. If wires i and j are the input wires of Alice and Bob, then he begins with knowledge of $k_i^{w_i} = k_i^{x_1}$ and $k_j^{w_j} = k_j^{x_2}$.

   - Suppose he arrives at gate $(\wedge, i, j, o)$ or $(\oplus, i, j, o)$. He finds the corresponding $g_{ijo} = \left(\tilde{c}_{ijo}^1, \tilde{c}_{ijo}^2, \tilde{c}_{ijo}^3, \tilde{c}_{ijo}^4\right)$ that was sent by Alice, and he knows $k_i^{w_i}$ and $k_j^{w_j}$ already.

     For $\ell \in [4]$:

     If $\mathsf{Dec}_{k_i^{w_i}}\left(\mathsf{Dec}_{k_j^{w_j}}\left(\tilde{c}_{ijo}^{\ell}\right)\right) \neq \perp$ then he sets $k_o^{w_o} := \mathsf{Dec}_{k_i^{w_i}}\left(\mathsf{Dec}_{k_j^{w_j}}\left(\tilde{c}_{ijo}^{\ell}\right)\right)$.

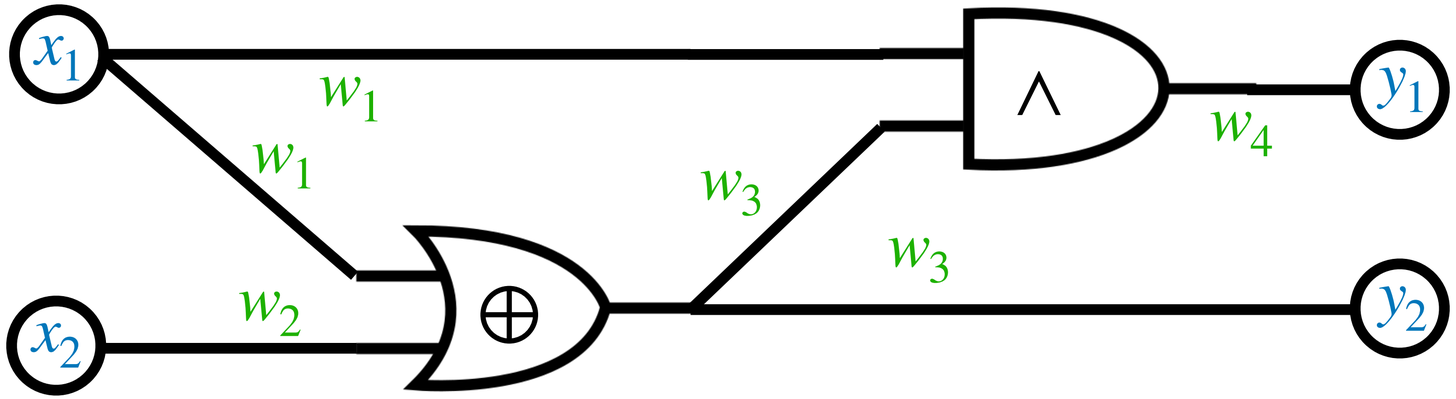
   - Suppose he arrives at gate $(\mathsf{out}, 1, i)$. He sends $k_i^{w_i}$ to Alice, who outputs $y_1 := 0$ if $k_i^{w_i} = k_i^0$, or $y_1 := 1$ if $k_i^{w_i} = k_i^1$.

   - Suppose he arrives at gate $(\mathsf{out}, 2, i)$. He finds the corresponding $\left(i, k_i^0, k_i^1\right)$ that was sent by Alice, and outputs $y_2 := 0$ if $k_i^{w_i} = k_i^0$, or $y_2 := 1$ if $k_i^{w_i} = k_i^1$.

3b. A Worked Example

The Setting for Our Example

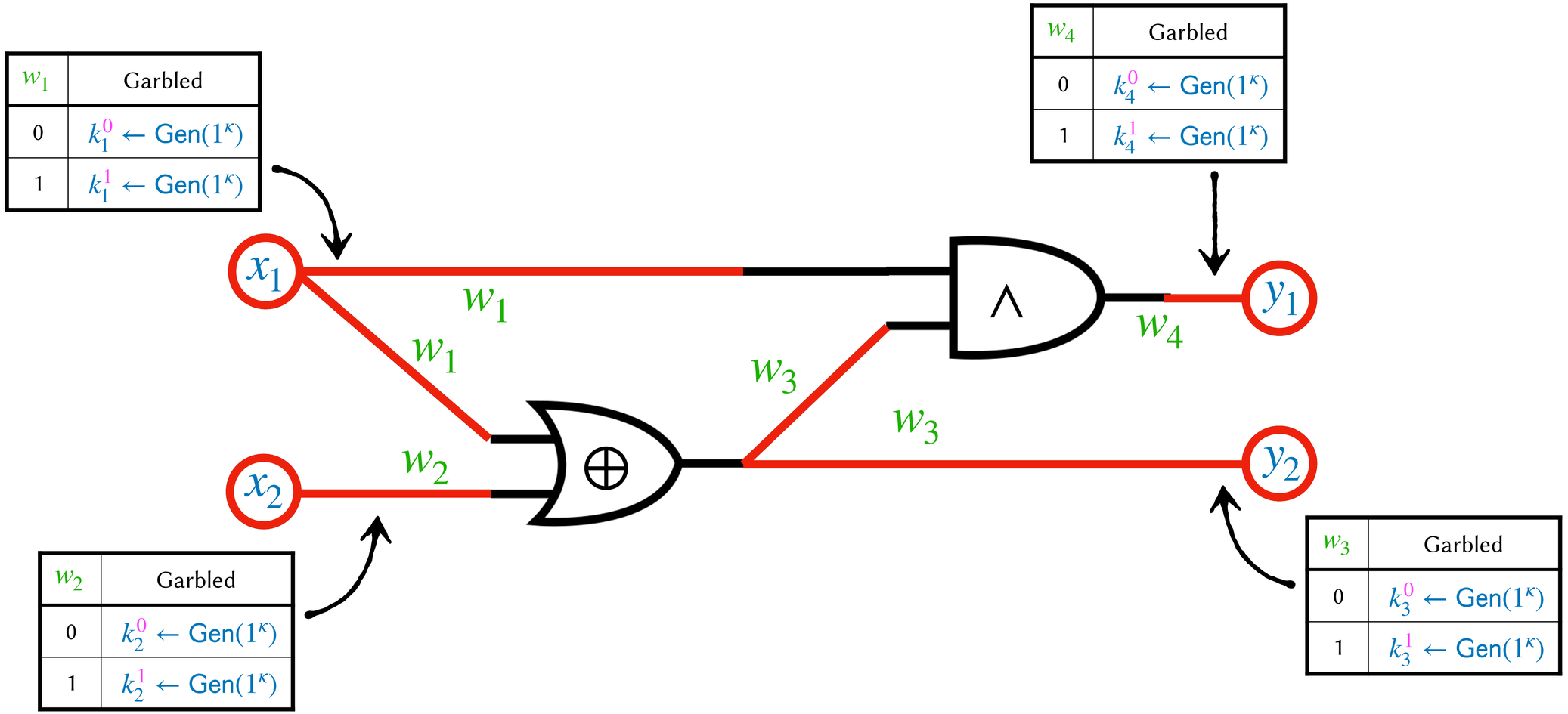
- The parties wish to compute $f(x_1, x_2) = ((x_1 \oplus x_2) \wedge x_1, x_1 \oplus x_2)$.
- The corresponding circuit is $C = \{(\text{in},1,1), (\text{in},2,2), (\oplus,1,2,3), (\wedge,1,3,4), (\text{out},1,4)(\text{out},2,3)\}$:



- They will compute the gates in the order listed above.
- We will fix $x_1 = 0$ and $x_2 = 1$. Notice that $f(0,1) = (0,1)$.

Phase 1: Garbling

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



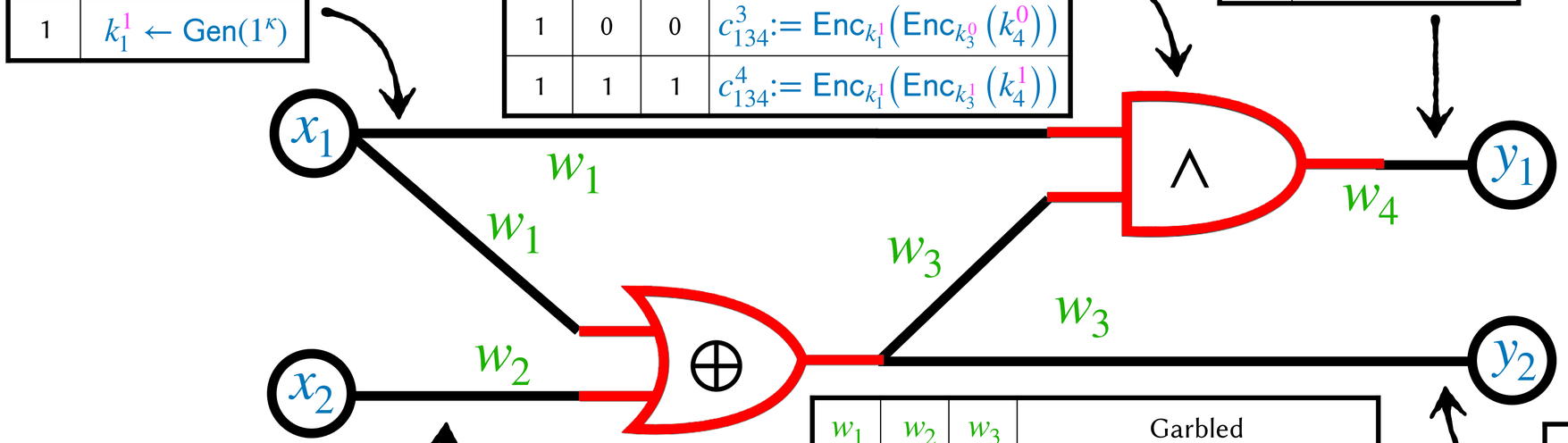
Phase 1: Garbling

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

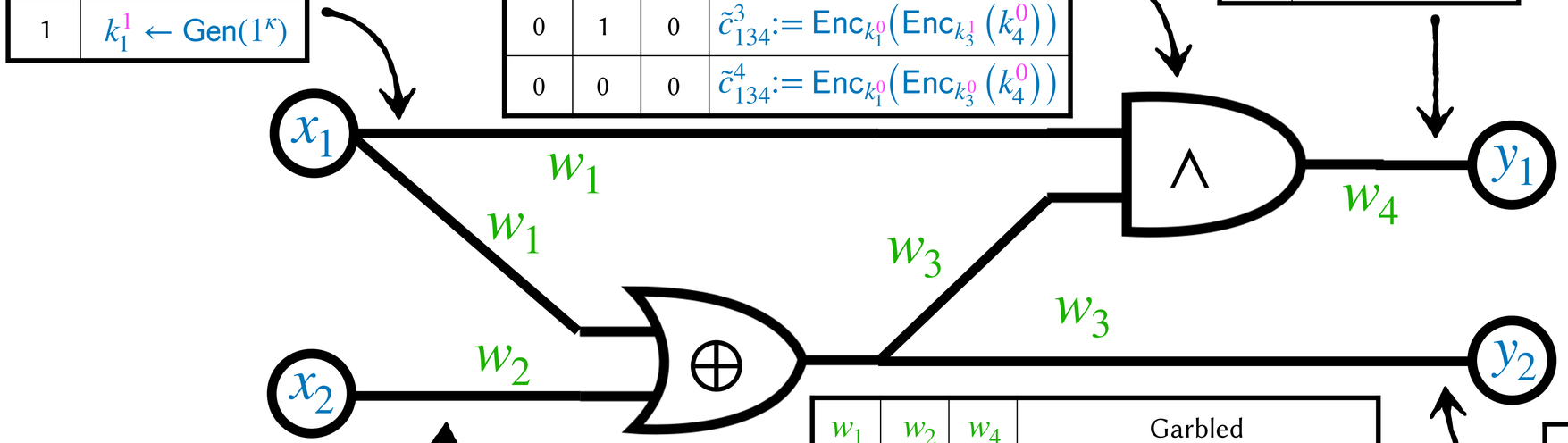
Phase 1: Garbling

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.

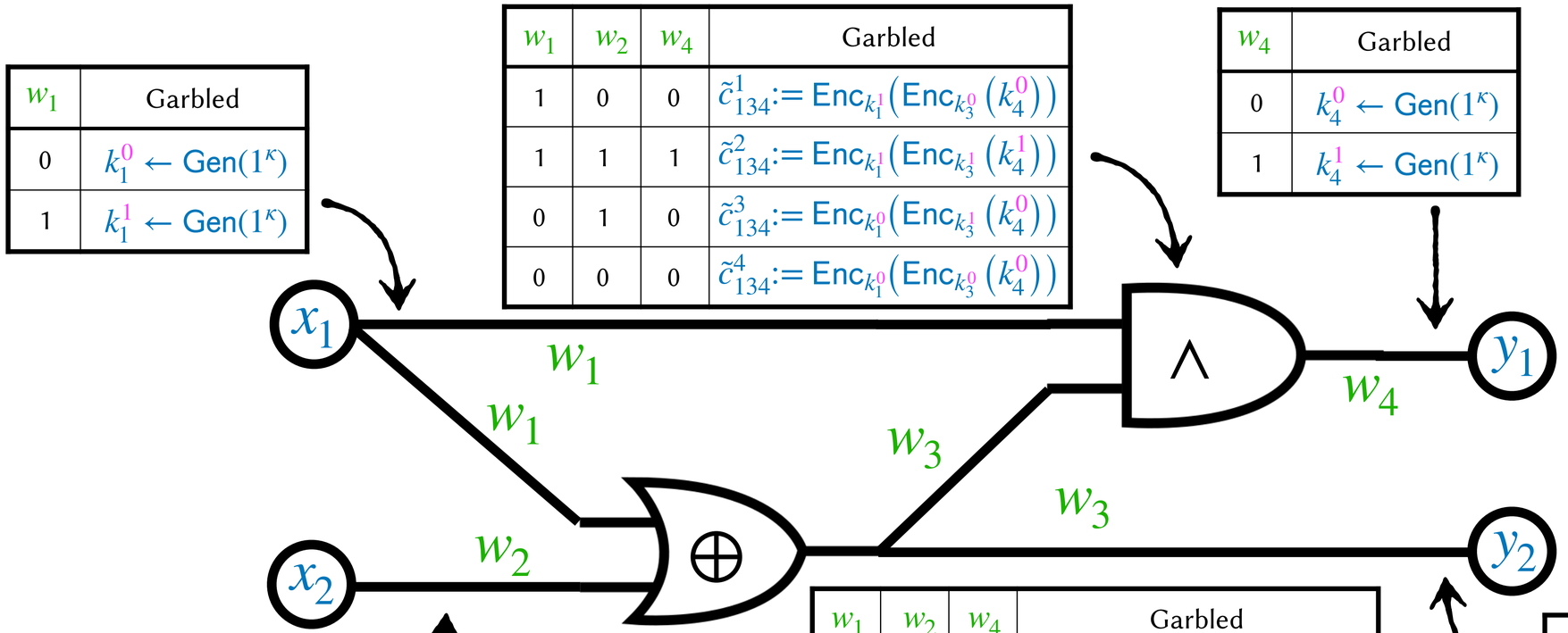
Garbled
$\tilde{c}^1_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^0}(k_4^0))$
$\tilde{c}^2_{134} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_3^1}(k_4^1))$
$\tilde{c}^3_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^1}(k_4^0))$
$\tilde{c}^4_{134} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_3^0}(k_4^0))$

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

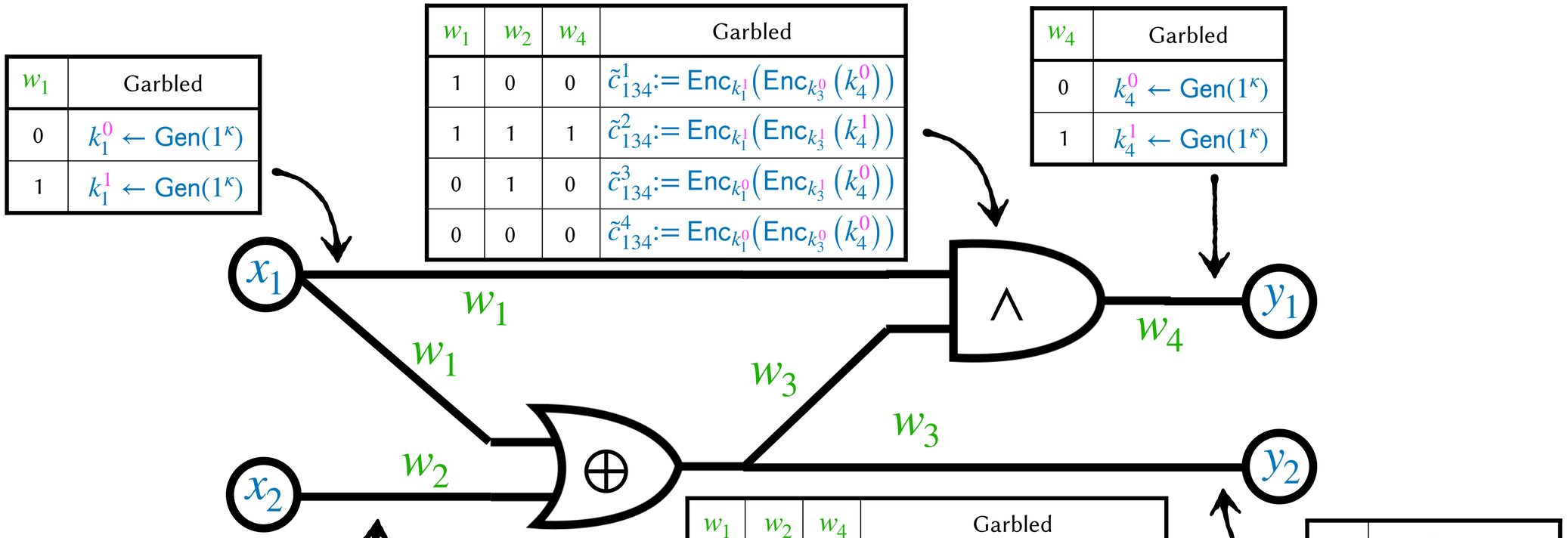
Garbled
$\tilde{c}^1_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

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

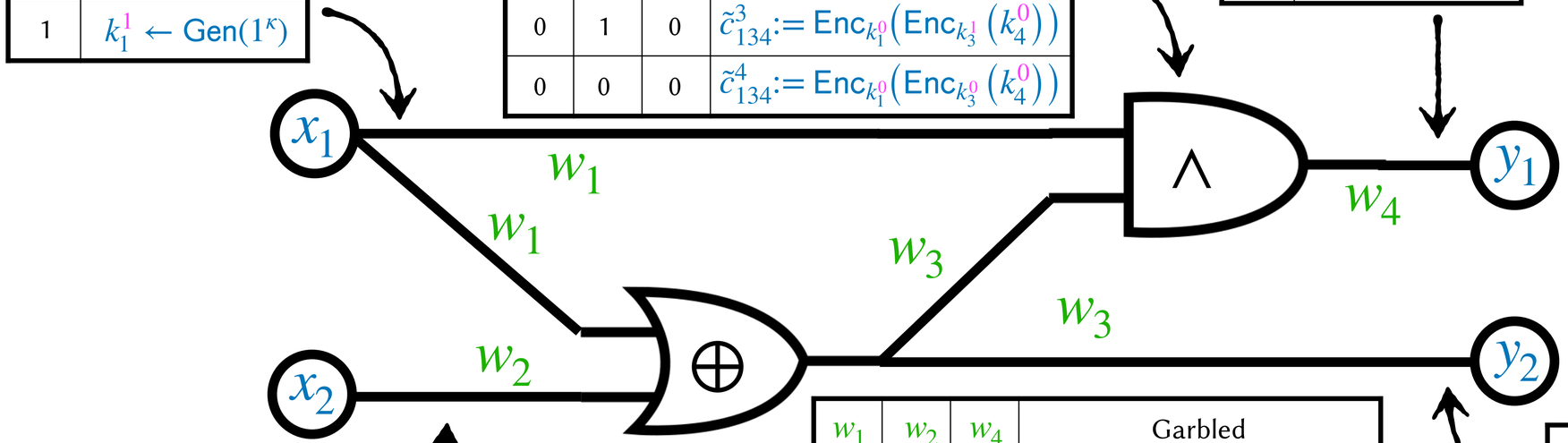
Phase 2: I/O Delivery

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

$k_1^{x_1}$

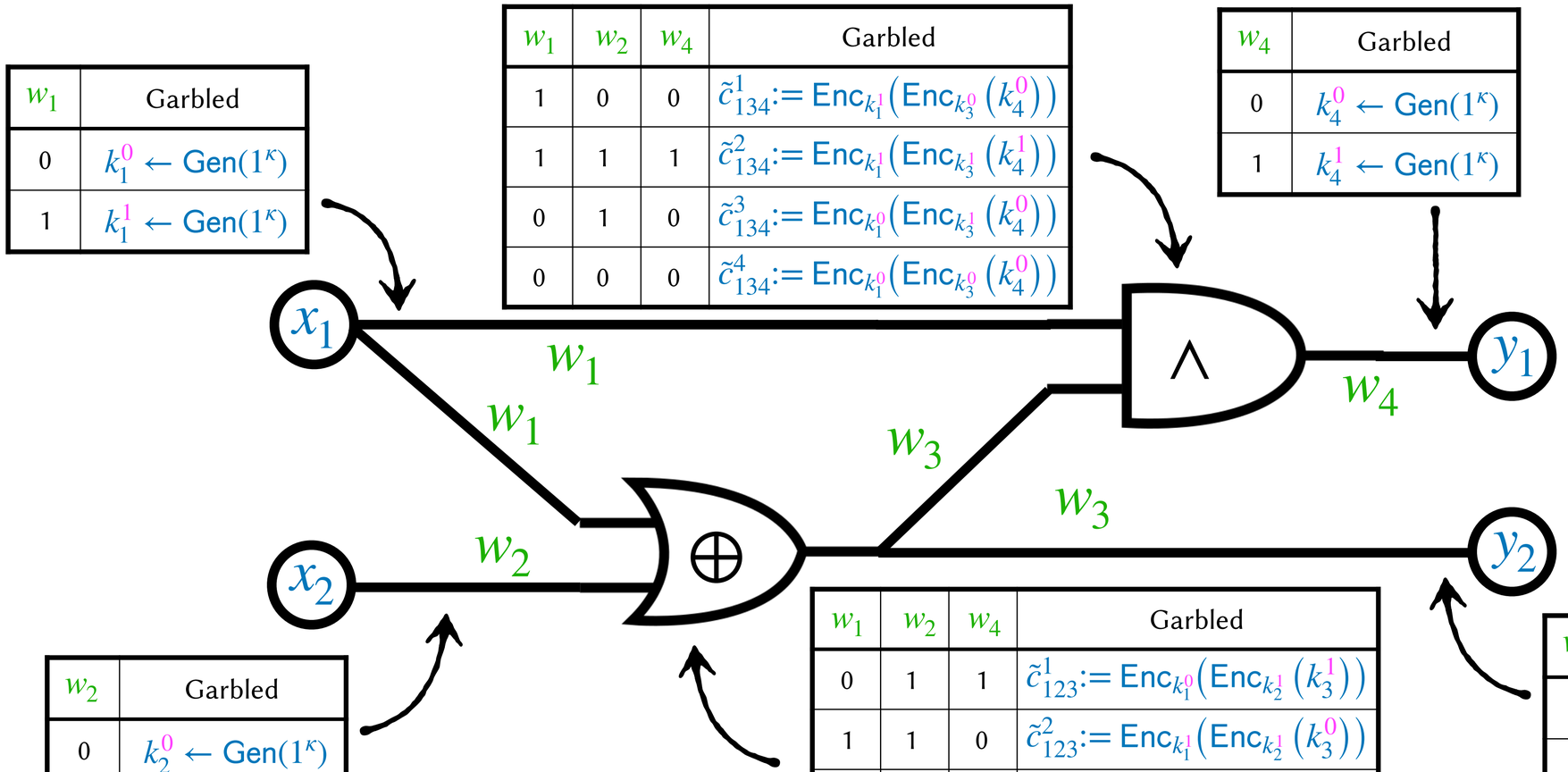
Phase 2: I/O Delivery

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

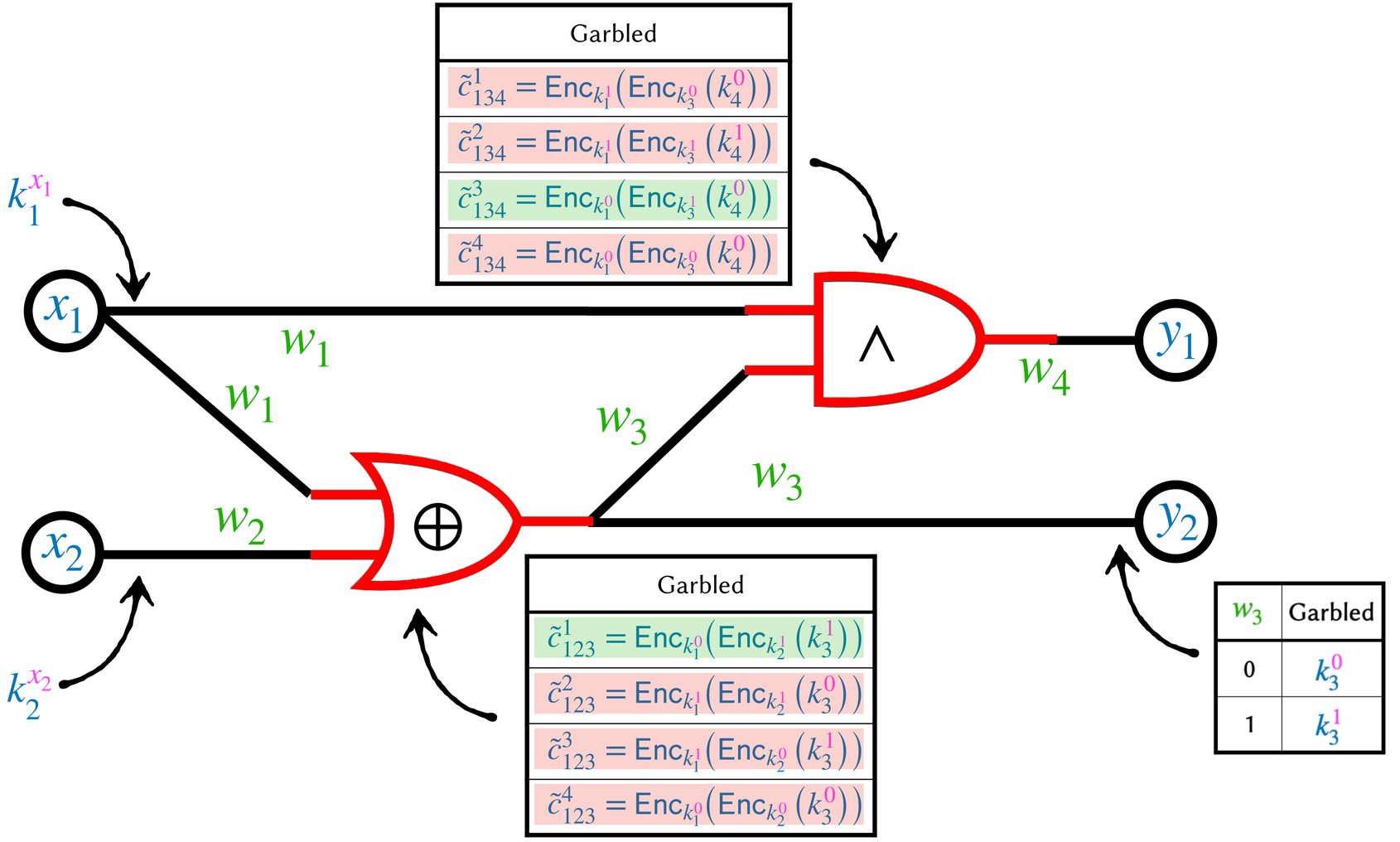
Garbled
$\tilde{c}^1_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^1}(k_3^1))$
$\tilde{c}^2_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^1}(k_3^0))$
$\tilde{c}^3_{123} := \text{Enc}_{k_1^1}(\text{Enc}_{k_2^0}(k_3^1))$
$\tilde{c}^4_{123} := \text{Enc}_{k_1^0}(\text{Enc}_{k_2^0}(k_3^0))$

$k_1^{x_1}$

w_3	Garbled
0	k_3^0
1	k_3^1

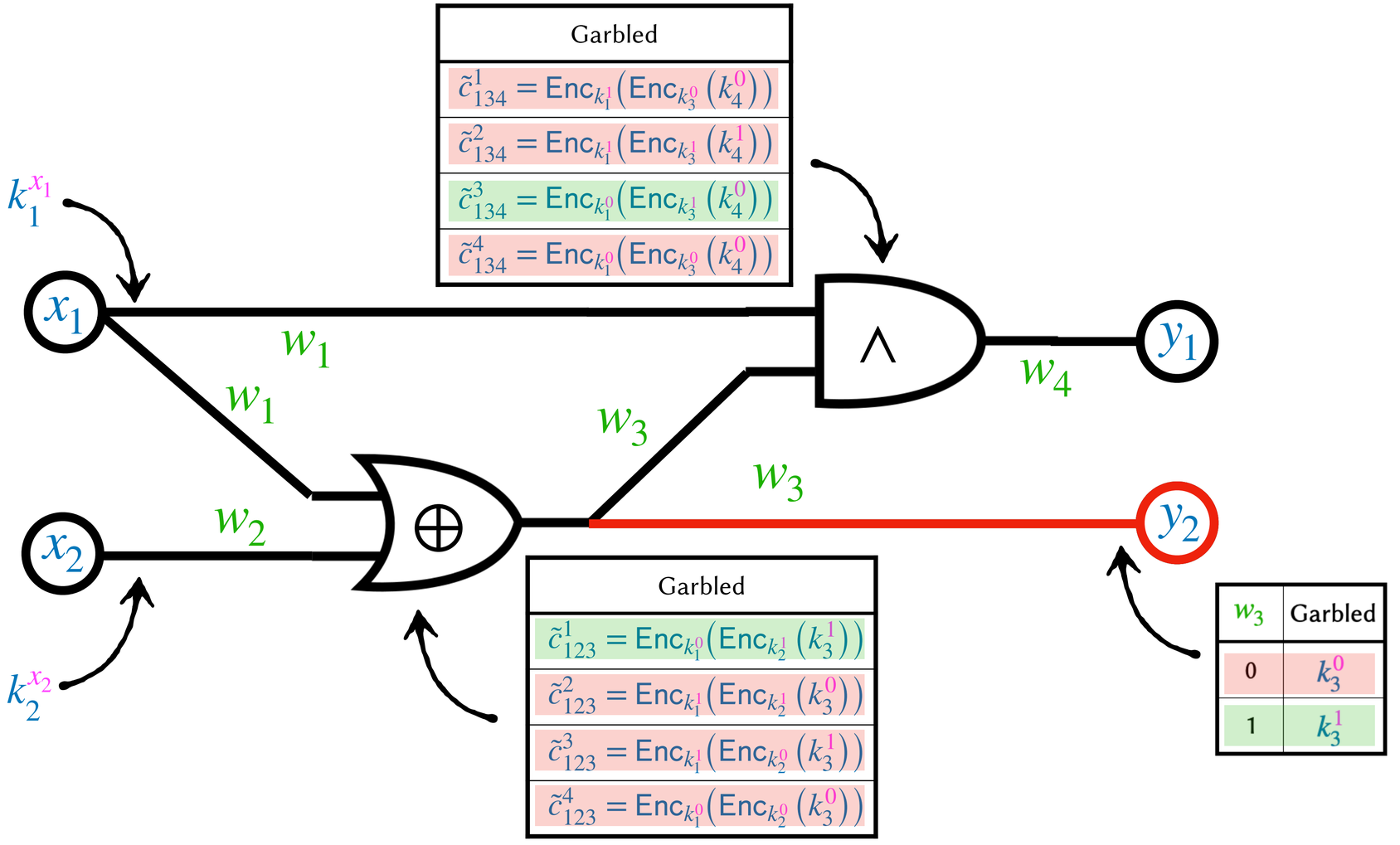
Phase 3: Evaluation

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.



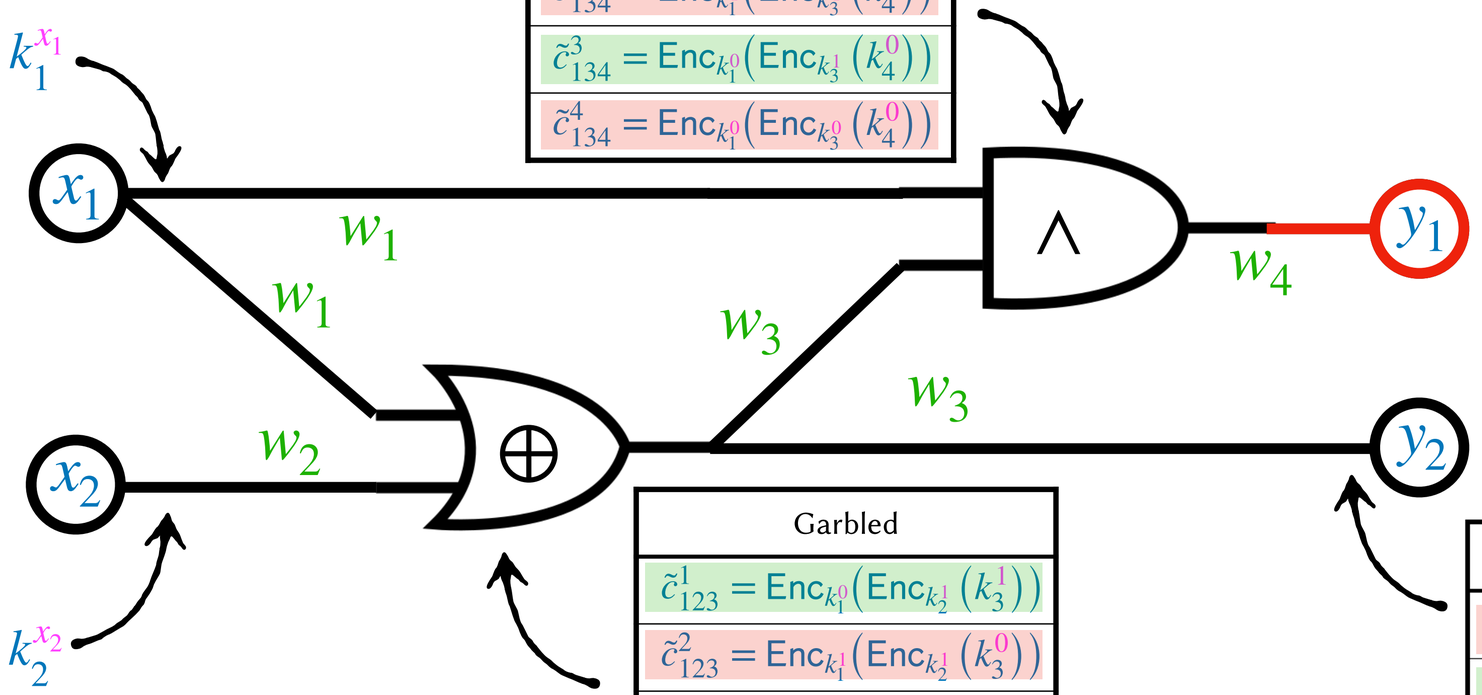
Phase 3: Evaluation

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

3c. Security for Yao (Preliminaries)

Recall: Even Simpler Def for Deterministic f

Definition 9. Computational Semi-Honest Deterministic SFE:

Let $\kappa, n, t \in \mathbb{N}$ such that $t < n$ and n is upper-bounded by some polynomial in κ, let $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$ be a n-ary function, **(which does not use any randomness)** and let π be a n-party protocol that runs in PPT relative to κ.

We say that π *securely computes* f in the presence of a bounded semi-honest adversary that statically corrupts up to t parties if

1. There exists some negligible function ε such that for every input vector $\vec{x} \in \mathcal{X}_1 \times \ldots \mathcal{X}_n$ it holds that $\Pr[f(\vec{x}) \neq \mathsf{OUTPUT}_\pi] \leq \varepsilon(\kappa)$.
2. There exists a PPT *simulator algorithm* Sim such that for every $I \subseteq [n]$ of size $|I| \leq t$ and every vector $\vec{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ it holds that

$$\left\{ \mathsf{Sim}\left(1^\kappa, I, \vec{x}_I, f_I(\vec{x})\right) \right\}_{\kappa \in \mathbb{N}} \approx_c \{\mathsf{VIEW}_I\}_{\kappa \in \mathbb{N}}$$

Correctness:

The canonical IND-CPA-secure SKE we introduced earlier is *perfectly* correct.

However, in order to give it an evasive range, we appended an extra κ zeros to the end of the message, and then checked before decrypting that they must be there.

Specifically, we said:

> "Dec_k verifies that c is in range by checking that its last κ bits are the same as those of $G(F_k(r))$.
>
> By pseudorandomness of F_k and G, finding r and the κ bits of $G(F_k(r))$ is *hard* without k."

If we sample $k' \leftarrow \mathsf{Gen}(1^\kappa)$, then with probability at most negligible in κ, $G(F_k(r)) = G(F_{k'}(r))$ and the ciphertext c is in range for k', even though it was not encrypted under k'.

Thus there is a negligible probability that when traversing the circuit, Bob finds a particular ciphertext to be in range, even though it is not the correct ciphertext.

Since there are polynomially many ciphertexts overall, the overall probability that this will induce an error is negligible, and thus our scheme satisfies the correctness criteria.

Simulation for Corrupted Alice:

What is in Alice's view?

She samples keys for all the wires, and garbles the gates. All of this can be done by a simulator, using exactly the same code as she uses.

The only message she receives is her output key, from Bob. Since the simulator knows her output, and it sampled her output keys, it can use the appropriate one as Bob's message.

This simulation is perfect, and the simulator is PPT.

Simulation for Corrupted Bob:

What is in Bob's view?

A garbled circuit, but he didn't sample it and he only knows one out of each pair of input keys.

The simulator must sample a garbling which evaluates in a way that is consistent with his input and output.

The simulator does not know Alice's input and recovering it could be as hard as inverting a one-way function. Fortunately, Alice's input keys are randomly chosen,
and the tables are shuffled.

Approach: sample a garbled circuit where *every gate always outputs the constant 1.* Give Bob input keys corresponding to 1 as well (for himself and for Alice).

Due to the shuffling of the tables, the rows that he decrypts in the gate tables will be distributed identically to the ones he decrypts if he receives a real garbling.

IND-CPA security will ensure the tables are indistinguishable from garblings of the true gates.

To ensure this garbling is consistent with his output, simply adjust his output table at the end.

Yao's Garbled Circuits Simulator, for Bob

There are Three Phases:

1. **Circuit Garbling:** For the output wire of every in, $\oplus$, and $\wedge$ gate in the circuit, $\mathsf{Sim_B}$ samples two keys $k_o^0, k_o^1 \leftarrow \mathsf{Gen}(1^\kappa)$. For every $(\neg, i, o)$ gate, she sets $k_o^0 := k_i^1$ and $k_o^1 := k_i^0$. The simulator iterates though the $\oplus$, $\wedge$, and out gates in any order:

   - Suppose $\mathsf{Sim_B}$ arrives at the gate $(\oplus, i, j, o)$ or $(\wedge, i, j, o)$. It computes:

$$c_{ijo}^1 := \mathsf{Enc}_{k_i^0}\left(\mathsf{Enc}_{k_j^0}\left(k_o^1\right)\right)$$
$$c_{ijo}^2 := \mathsf{Enc}_{k_i^0}\left(\mathsf{Enc}_{k_j^1}\left(k_o^1\right)\right)$$
$$c_{ijo}^3 := \mathsf{Enc}_{k_i^1}\left(\mathsf{Enc}_{k_j^0}\left(k_o^1\right)\right)$$
$$c_{ijo}^4 := \mathsf{Enc}_{k_i^1}\left(\mathsf{Enc}_{k_j^1}\left(k_o^1\right)\right)$$

Notice: all 1 labels.

and sets $g_{ijo} = \left(\tilde{c}_{ijo}^1, \tilde{c}_{ijo}^2, \tilde{c}_{ijo}^3, \tilde{c}_{ijo}^4\right)$ to be a random shuffling of $\left(c_{ijo}^1, c_{ijo}^2, c_{ijo}^3, c_{ijo}^4\right)$.

$\mathsf{Sim_B}$ then adds (j, k, o, g_{ijo}) to Bob's view.

Yao's Garbled Circuits Simulator, for Bob

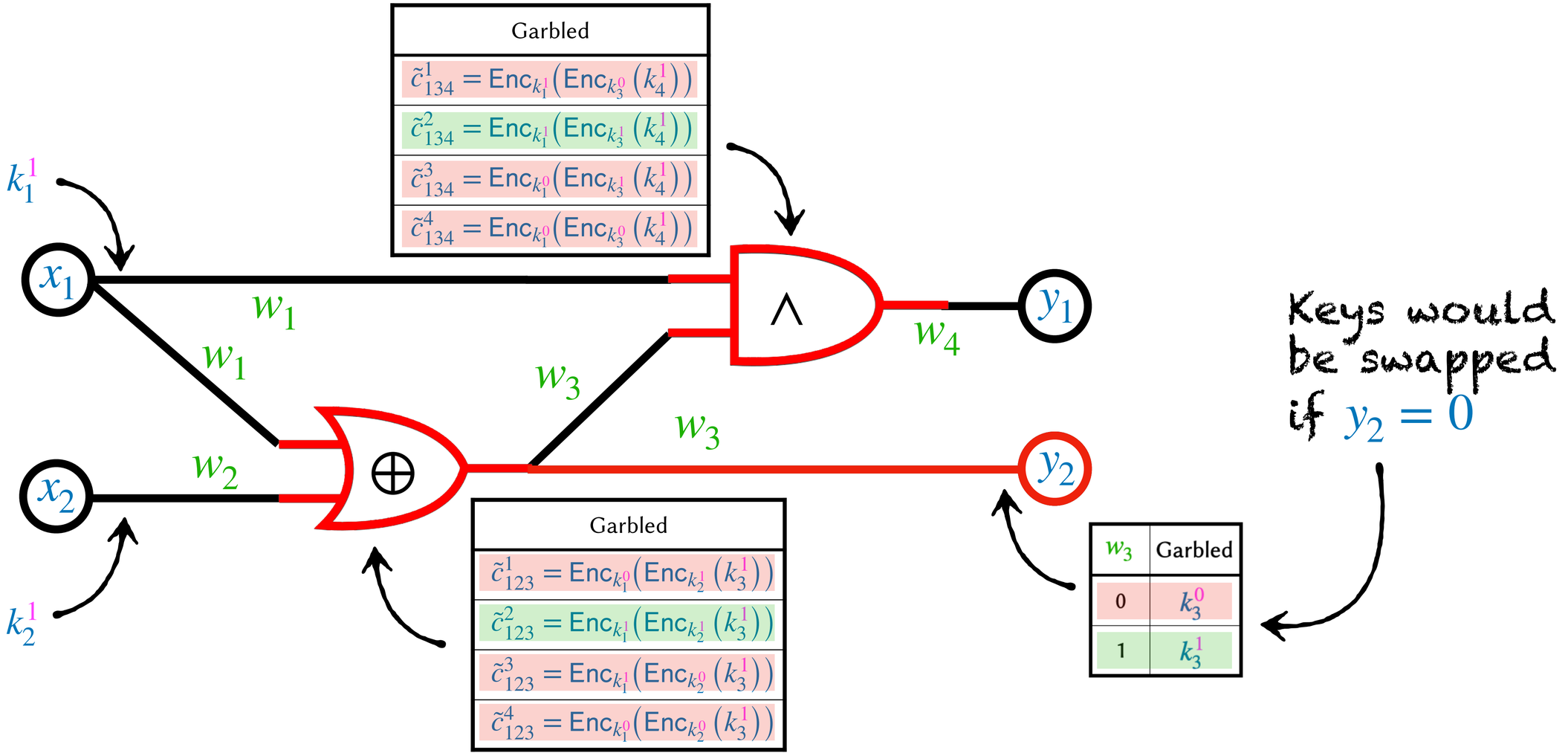
2. **Input/Output Delivery:**

   - $\mathsf{Sim_B}$ finds $(\mathsf{in},1,o) \in C$, if it exists, and adds $\left(o, k_o^1\right)$ to Bob's view as a message transmitted by Alice.
   - $\mathsf{Sim_B}$ finds $(\mathsf{in},2,o) \in C$, if it exists, and adds $\left(o, k_o^1\right)$ to Bob's view as a message transmitted by Alice.
   - $\mathsf{Sim_B}$ finds $(\mathsf{out},2,i) \in C$, if it exists. If $y_2 = 1$, then $\mathsf{Sim_B}$ adds $\left(i, k_i^0, k_i^1\right)$ to Bob's view as a message transmitted by Alice. Otherwise, it adds $\left(i, k_i^1, k_i^0\right)$ to Bob's view.

3. **Circuit Evaluation:** Bob receives no additional messages during this phase, so $\mathsf{Sim_B}$ simply follows his code, given the values it sampled in the previous phases.

Bob's Simulated View

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



IND-CPA implies Double-IND-CPA

We need to show that the simulated garbling of the circuit is computationally indistinguishable from Alice's real garbling. The garbling is made up entirely of *double-encryptions* of randomly sampled keys.

All of the *keys* used to perform the encryptions are identically distributed between the real and simulated worlds. The *values*, on the other hand, are different in many places.

We will need to formulate a sequence of hybrids experiments such that between each neighboring pair of experiments, at most one encrypted value changes. This must always be a value that Bob cannot decrypt.

First, let us convince ourselves that double-encryptions are indistinguishable so long as *at least one* of the encryption keys is unknown to the adversary.

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

Proof Sketch: Suppose that for some k_2 and $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1)$ the claim is false and there exists a distinguisher $\mathscr{D}$ with non-negligible advantage. Then we can construct a pair $(\mathscr{A}', \mathscr{D}')$ with k_2 and $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1)$ hardcoded into them that contradict the IND-CPA security of Π.

$\mathscr{A}'$ outputs $\left(\mathsf{Enc}_{k_2}(m^*_0), \mathsf{Enc}_{k_2}(m^*_1)\right)$ to the challenger, and then $\mathscr{D}'$ receives c^*_b. $\mathscr{D}'$ queries the challenger on $\mathsf{Enc}_{k_2}(m'_i)$ in order to learn c'_i for every $i \in [\ell(\kappa)]$. Then $\mathscr{D}'$ runs $\mathscr{D}$ as a subroutine and outputs whatever $\mathscr{D}\left(1^\kappa, \left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_b\right)\right)$ does.

Notice that the inputs of $\mathscr{D}$ are identically distributed to those in the claim. Thus $(\mathscr{A}', \mathscr{D}')$ has the same advantage as $\mathscr{D}$, and by the fact that Π is IND-CPA secure, no such $\mathscr{D}$ can exist. ∎

IND-CPA implies Double-IND-CPA

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 $\left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_0\right)\right\}_{\kappa\in\mathbb{N}} \approx_c \left\{\left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_1\right)\right\}_{\kappa\in\mathbb{N}}$. Notice that we fix k_1 this time.

Proof Sketch: Suppose that for some k_1 and $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1)$ the claim is false and there exists a distinguisher $\mathcal{D}$ with non-negligible advantage. Then we can construct a pair $(\mathcal{A}', \mathcal{D}')$ with k_1 and $(m'_1, \ldots, m'_{\ell(\kappa)}, m^*_0, m^*_1)$ hardcoded into them that contradict the IND-CPA security of Π.

$\mathcal{A}'$ outputs (m^*_0, m^*_1) to the challenger, and then $\mathcal{D}'$ receives $\mathsf{Enc}_{k_2}(m^*_b)$. $\mathcal{D}'$ queries the challenger on m'_i in order to learn $\mathsf{Enc}_{k_2}(m'_i)$ for every $i \in [\ell(\kappa)]$. Then, using its knowledge of k_1 to perform the outer encryptions exactly as in the claim, $\mathcal{D}'$ outputs what $\mathcal{D}\left(1^\kappa, \left(c'_1, \ldots, c'_{\ell(\kappa)}, c^*_b\right)\right)$ does.

Notice that the inputs of $\mathcal{D}$ are identically distributed to those in the claim. Thus $(\mathcal{A}', \mathcal{D}')$ has the same advantage as $\mathcal{D}$, and by the fact that Π is IND-CPA secure, no such $\mathcal{D}$ can exist. ∎

..and we will put these pieces together into a proof that Bob's simulation is computationally indistinguishable from his real-world view in the next lecture!

CS4501 Cryptographic Protocols
Lecture 16: Pseudorandom Functions, Garbled Circuits

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