CS 594 – Advanced Crypto (Spring 2026)

Alex Block

Lecture 10

February 18, 2026

Function Secret Sharing

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$.
  - Share x as $s_1, \ldots, s_n \in \mathbb{G}$ such that $s_i \xleftarrow{\$} \mathbb{G}$ for $i < n$ and $s_n = x - \sum_{j=1}^{n-1} s_j$.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$.
  - Share x as $s_1, \ldots, s_n \in \mathbb{G}$ such that $s_i \xleftarrow{\$} \mathbb{G}$ for $i < n$ and $s_n = x - \sum_{j=1}^{n-1} s_j$.

- Additive secret sharing is *additively homomorphic*:

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$.
  - Share x as $s_1, \ldots, s_n \in \mathbb{G}$ such that $s_i \xleftarrow{\$} \mathbb{G}$ for $i < n$ and $s_n = x - \sum_{j=1}^{n-1} s_j$.

- Additive secret sharing is *additively homomorphic*:
  - Let $x, x' \in \mathbb{G}$ and with additive shares $(s_1, \ldots, s_n), (s'_1, \ldots, s'_n)$.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$.
  - Share x as $s_1, \ldots, s_n \in \mathbb{G}$ such that $s_i \xleftarrow{\$} \mathbb{G}$ for $i < n$ and $s_n = x - \sum_{j=1}^{n-1} s_j$.

- Additive secret sharing is *additively homomorphic*:
  - Let $x, x' \in \mathbb{G}$ and with additive shares $(s_1, \ldots, s_n), (s'_1, \ldots, s'_n)$.
  - Then, $(s_1 + s'_1, \ldots, s_n + s'_n)$ is an additive sharing of $x + x'$.

Function Secret Sharing

- So far, we have only discussed secret sharing fixed values.
- Function Secret Sharing asks:

> What if we could secret share a *function*?

- We've already seen a form of function secret sharing: *additive secret sharing*.
  - Fix secret $x \in \mathbb{G}$. $= \{0,1\}^4$
  - Share x as $s_1, \ldots, s_n \in \mathbb{G}$ such that $s_i \xleftarrow{\$} \mathbb{G}$ for $i < n$ and $s_n = x - \sum_{j=1}^{n-1} s_j$.

- Additive secret sharing is *additively homomorphic*:
  - Let $x, x' \in \mathbb{G}$ and with additive shares $(s_1, \ldots, s_n), (s'_1, \ldots, s'_n)$.
  - Then, $(s_1 + s'_1, \ldots, s_n + s'_n)$ is an additive sharing of $x + x'$.

- Function being shared: $f(X)\colon \{0,1\}^\ell \to \mathbb{G}$ where $f(\mathbf{y}) = x$, where $\mathbf{y}$ is the binary representation of $x \in \mathbb{G}$.

Function Secret Sharing

Function Secret Sharing

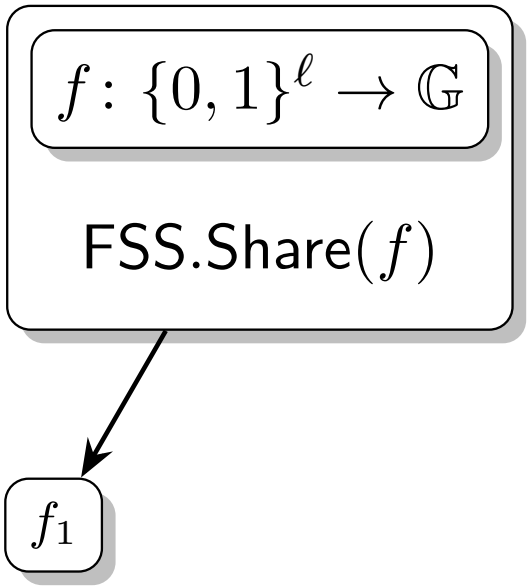
$f : \{0,1\}^{\ell} \to \mathbb{G}$

Function Secret Sharing

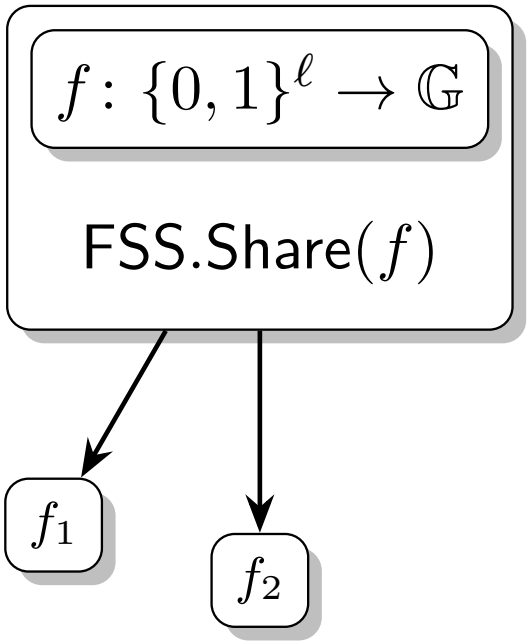
$f : \{0,1\}^\ell \to \mathbb{G}$

$\mathsf{FSS.Share}(f)$

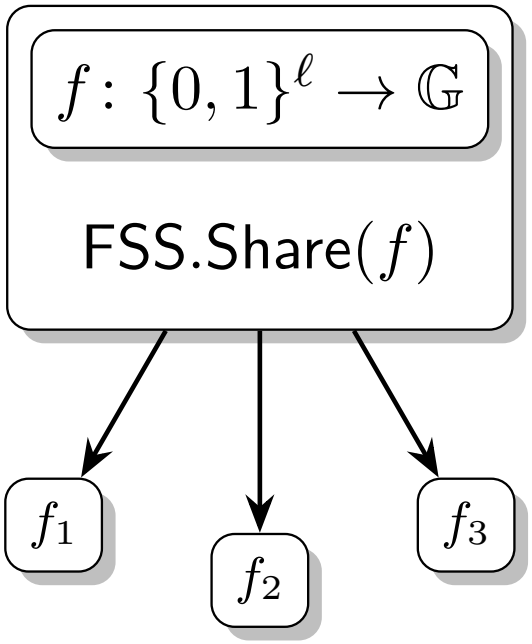
Function Secret Sharing



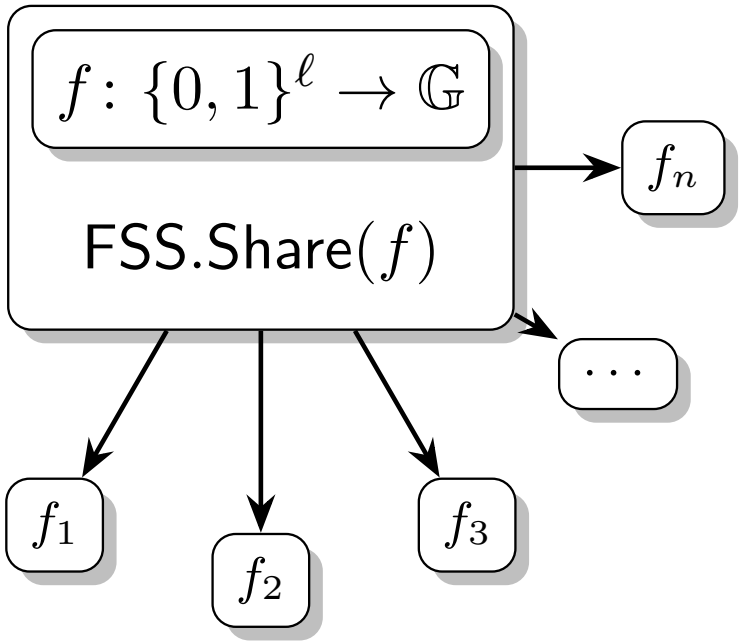
Function Secret Sharing



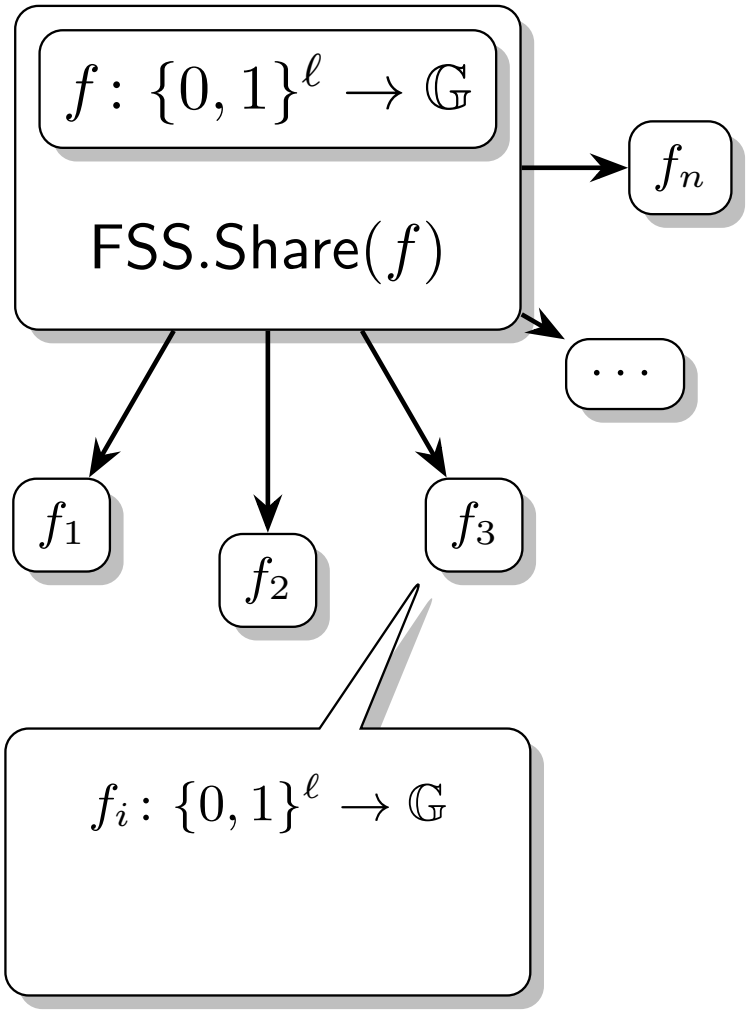
Function Secret Sharing



Function Secret Sharing

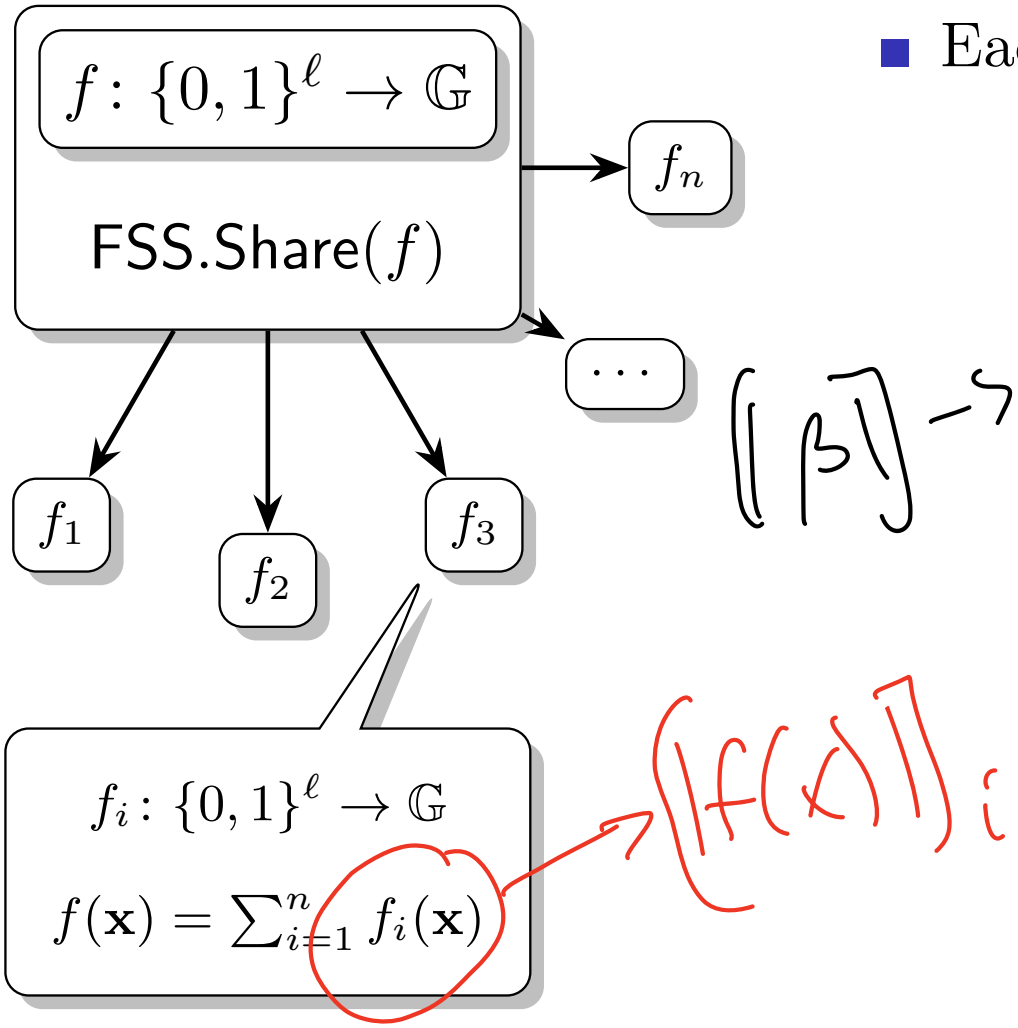


Function Secret Sharing



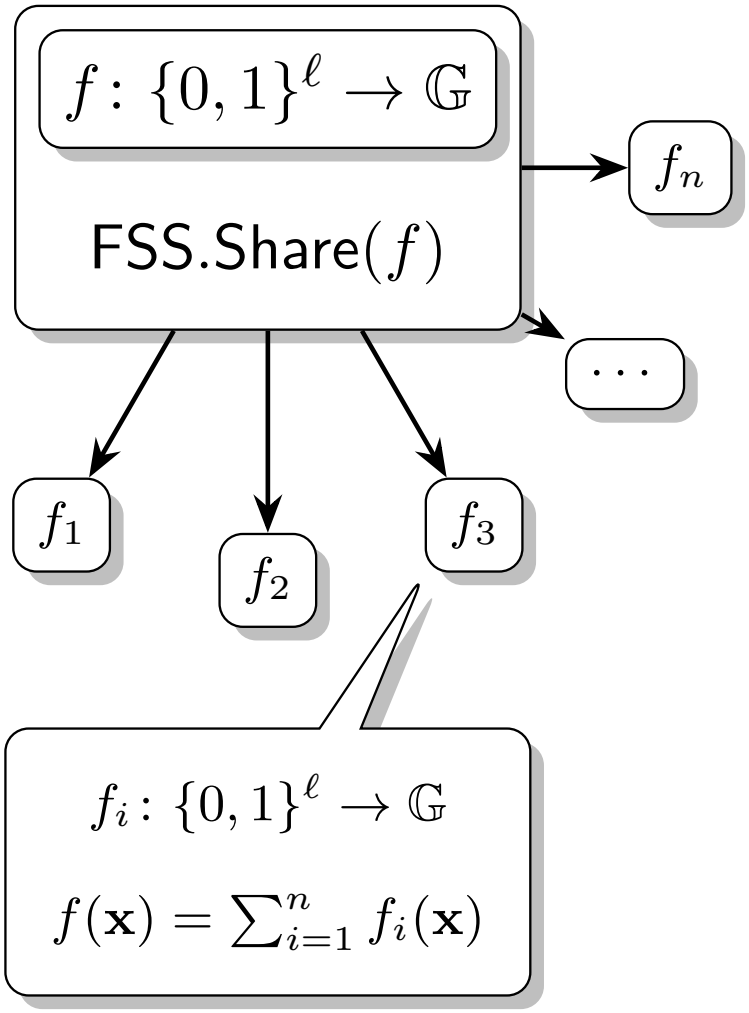
- Each f_i is an additive share of f.

Function Secret Sharing



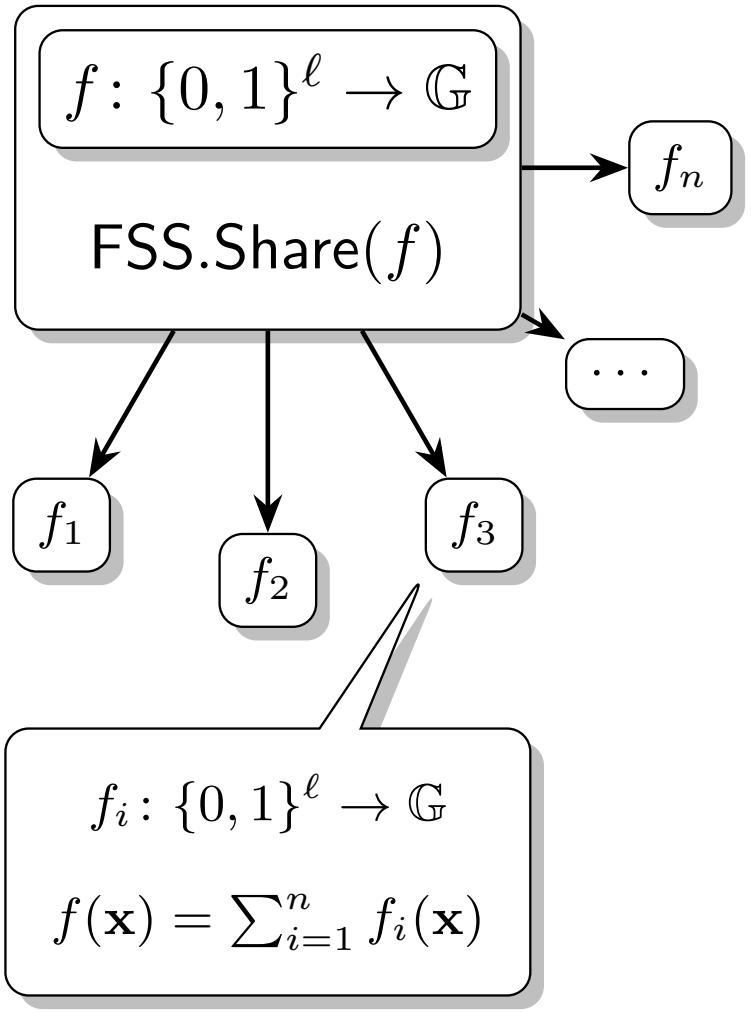
- Each f_i is an additive share of f.

Function Secret Sharing



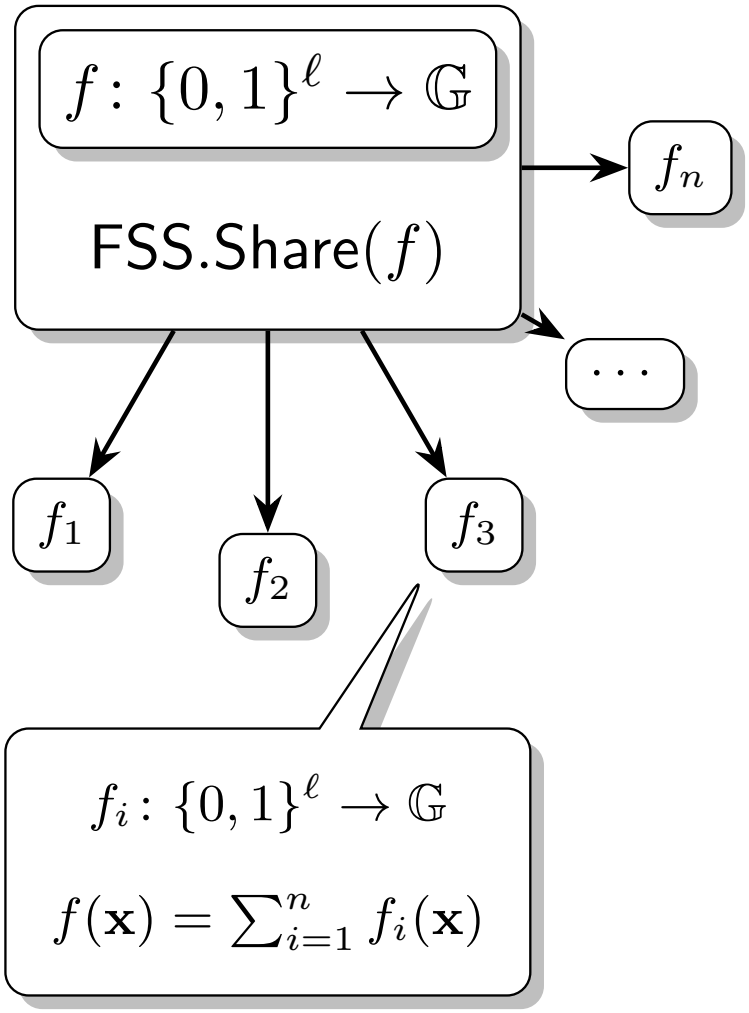
- Each f_i is an additive share of f.
- Any subset of $< n$ functions reveal nothing about f.

Function Secret Sharing



- Each f_i is an additive share of f.
- Any subset of $< n$ functions reveal nothing about f.
- Trivial Solution: Additively share the truth table of f.

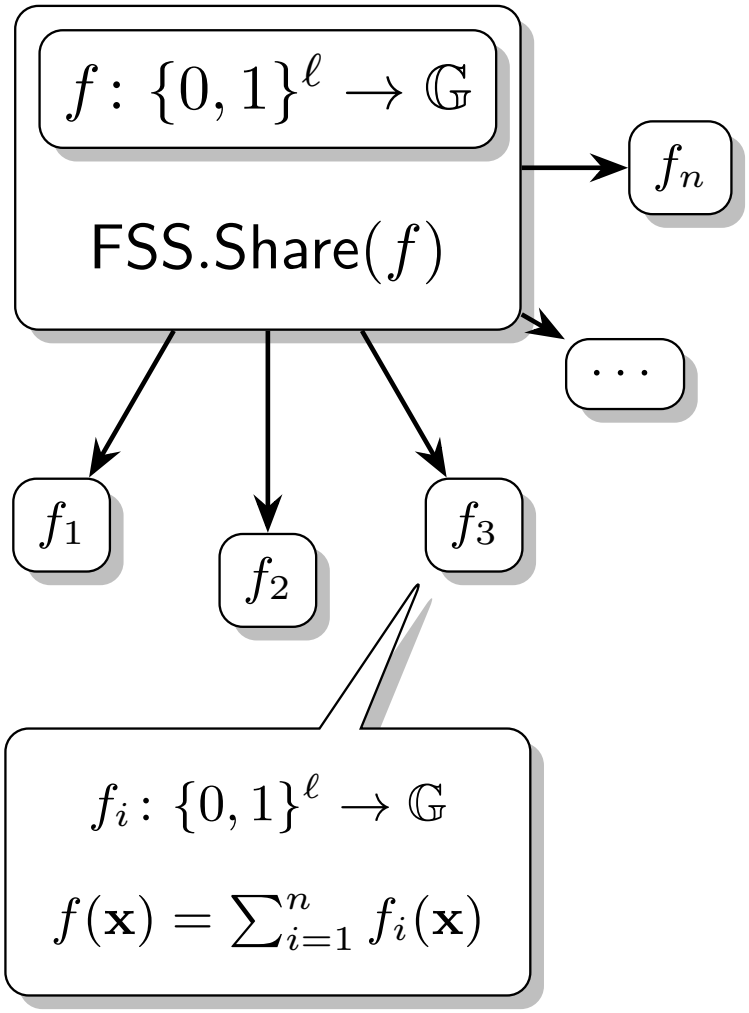
Function Secret Sharing



- Each f_i is an additive share of f.
- Any subset of $< n$ functions reveal nothing about f.
- Trivial Solution: Additively share the truth table of f.

$f : \{0,1\}^2 \to \mathbb{G}$	
$\mathbf{x}$	$f(\mathbf{x})$
00	α
01	β
10	1
11	g

Function Secret Sharing



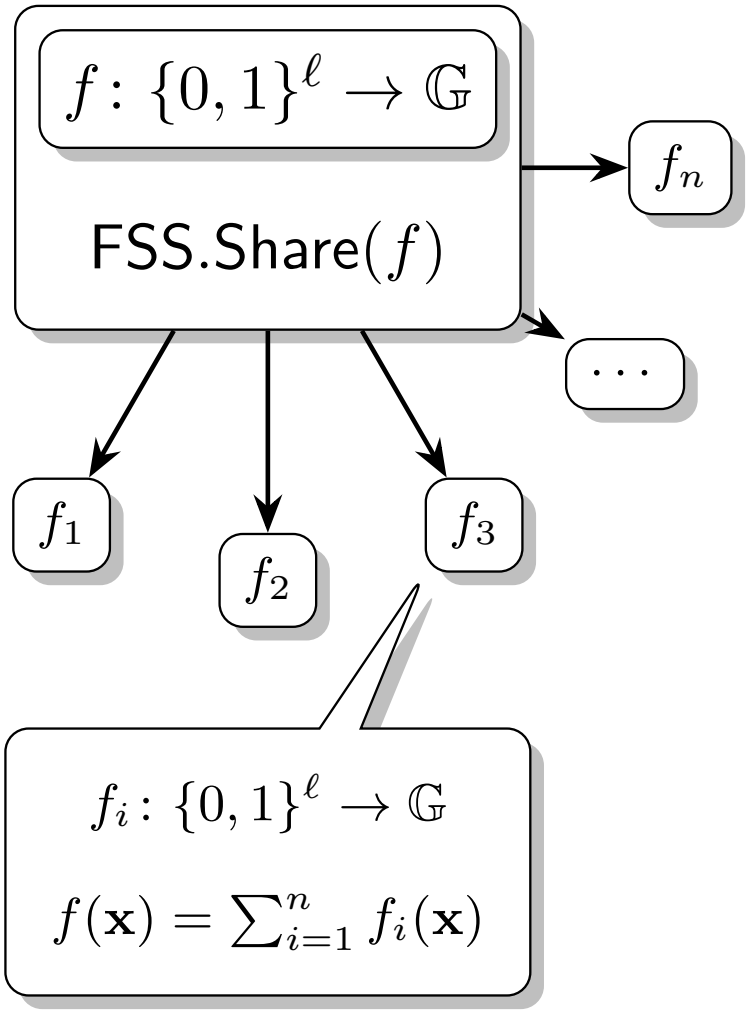
- Each f_i is an additive share of f.
- Any subset of $< n$ functions reveal nothing about f.
- Trivial Solution: Additively share the truth table of f.

$f : \{0,1\}^2 \to \mathbb{G}$	
$\mathbf{x}$	$f(\mathbf{x})$
00	α
01	β
10	1
11	g

(n, n) additive shares for each entry

↳ n shares, give each party 1 share per row

Function Secret Sharing



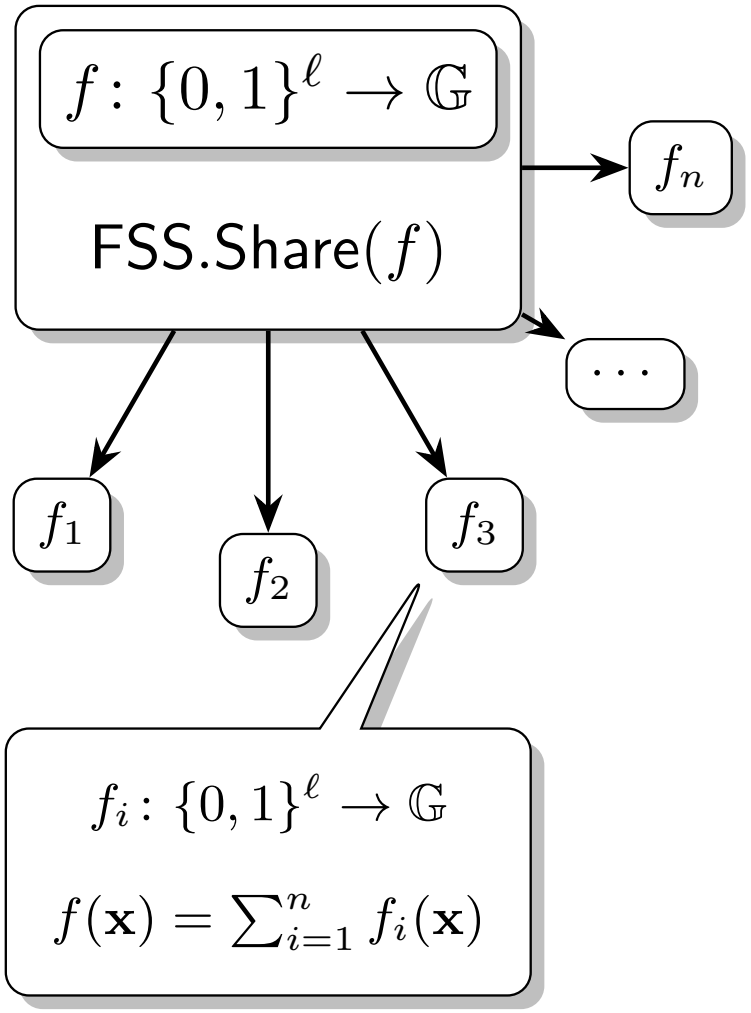
- Each f_i is an additive share of f.
- Any subset of $< n$ functions reveal nothing about f.
- Trivial Solution: Additively share the truth table of f.

$f : \{0,1\}^2 \to \mathbb{G}$	
$\mathbf{x}$	$f(\mathbf{x})$
00	α
01	β
10	1
11	g

(n, n) additive shares for each entry

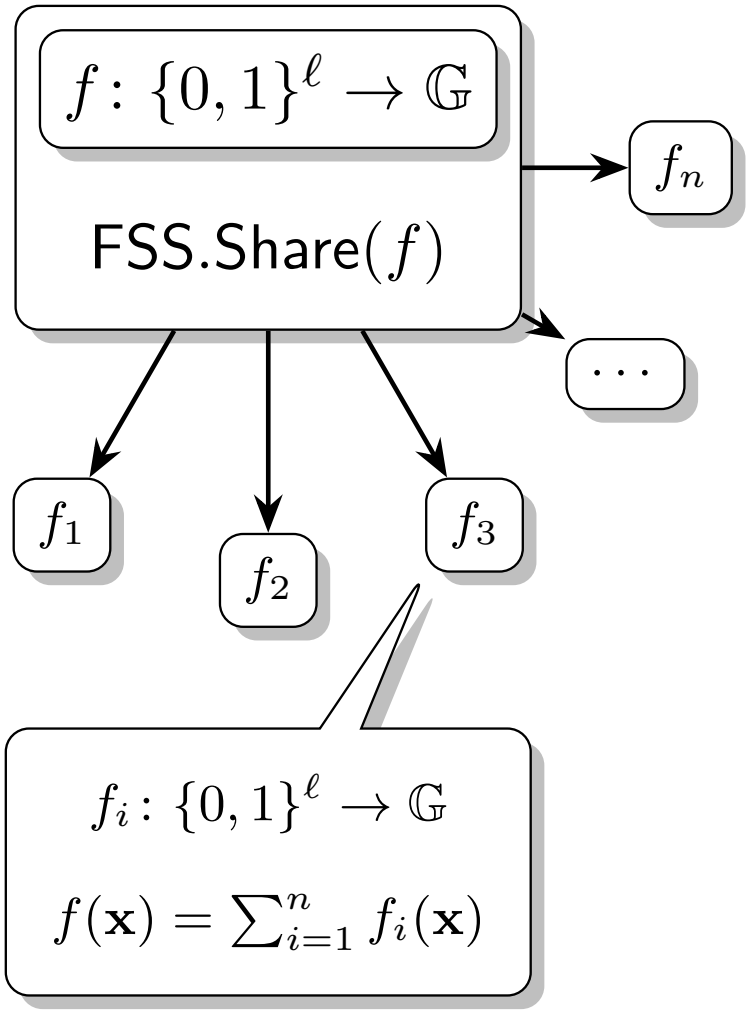
- Reconstruction: pick the right set of secret shares depending on $\mathbf{x}$.

Function Secret Sharing



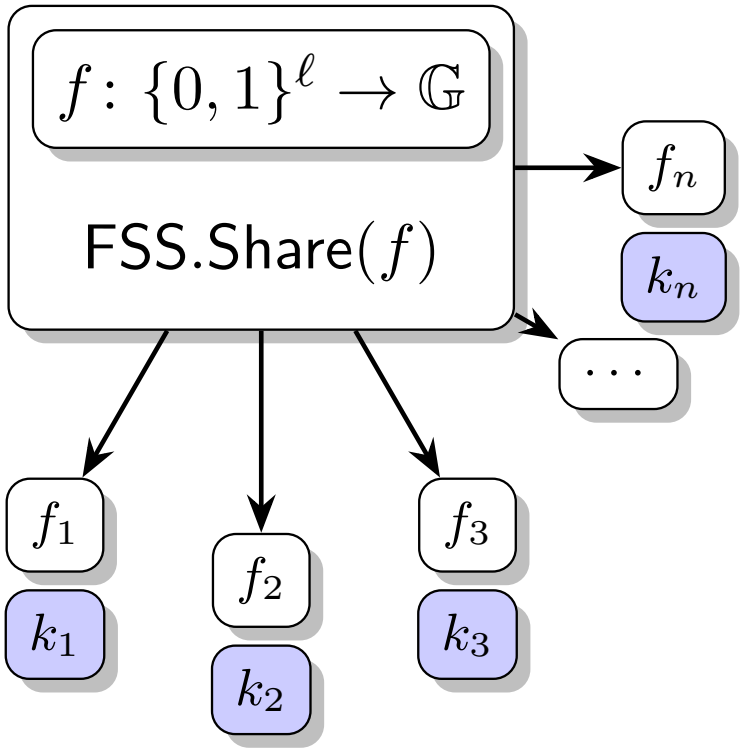
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!

Function Secret Sharing



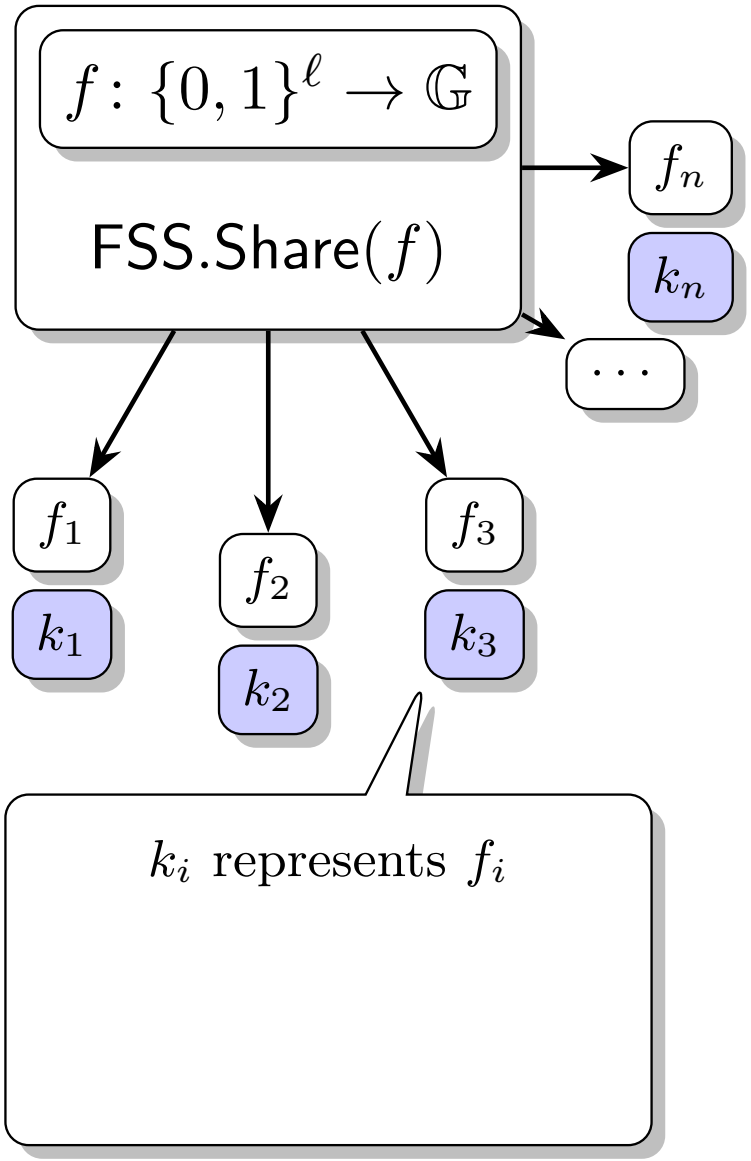
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.

Function Secret Sharing



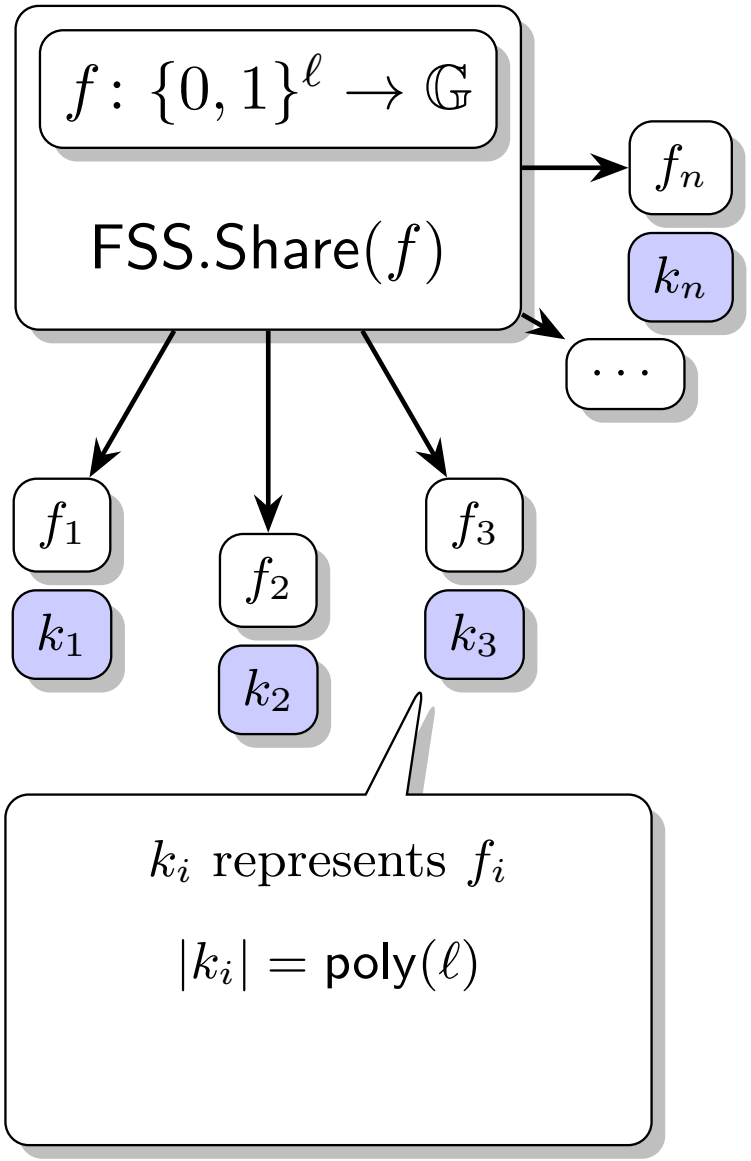
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.

Function Secret Sharing



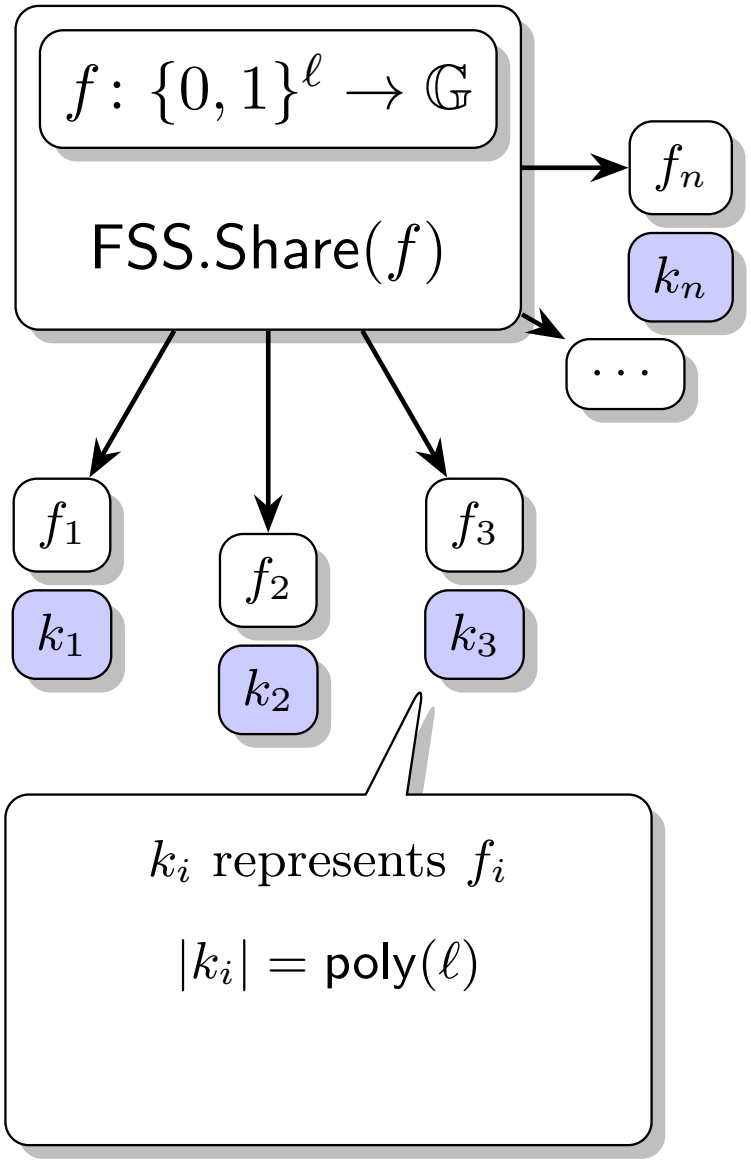
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.

Function Secret Sharing



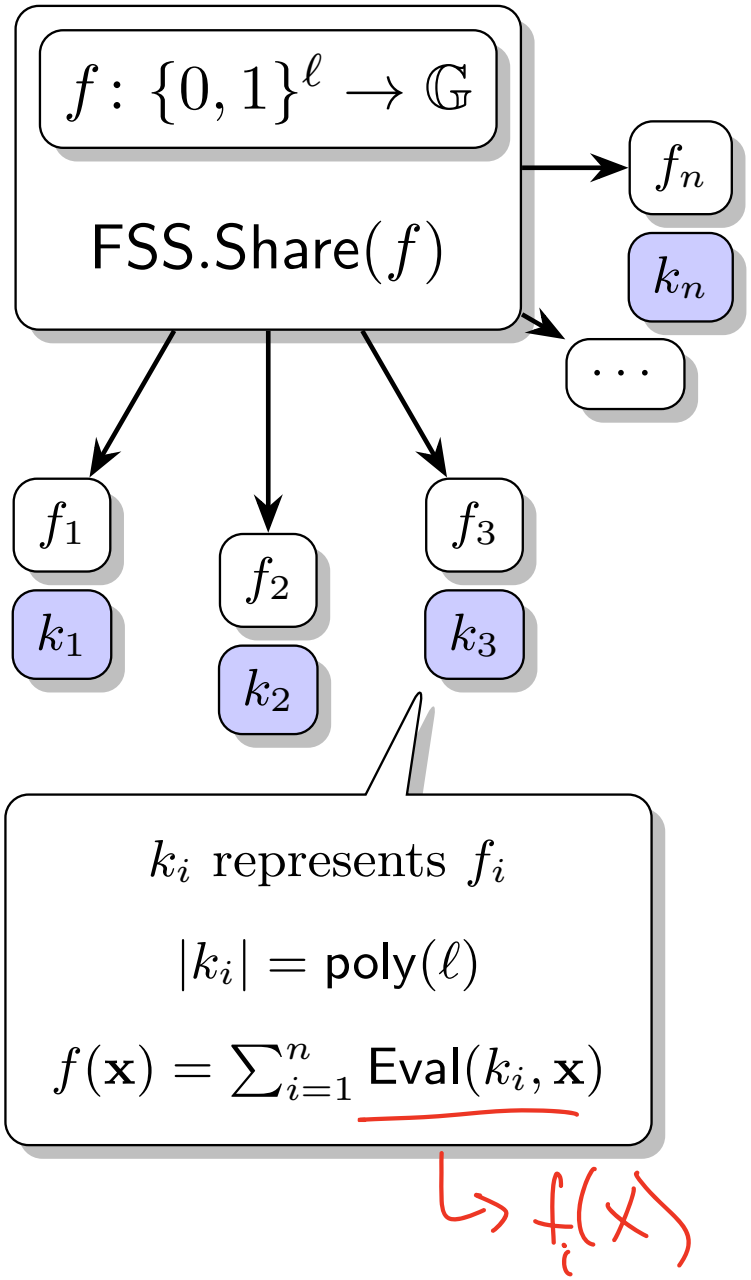
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.

Function Secret Sharing



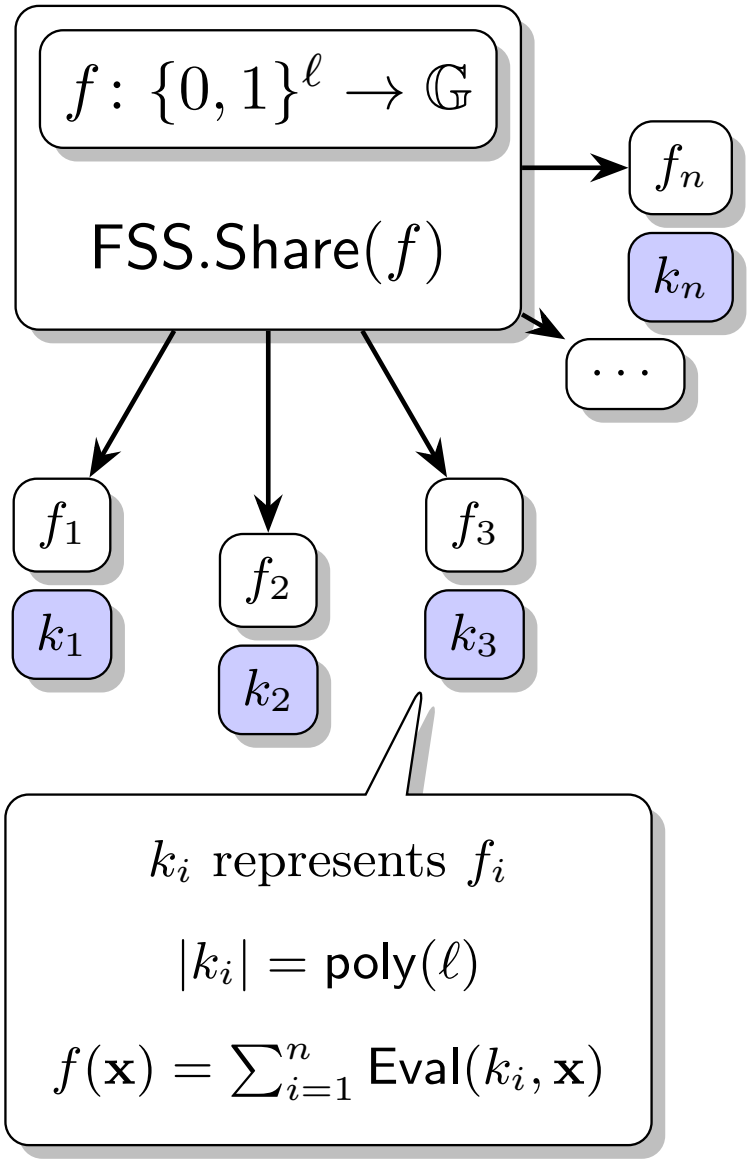
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.
  - Any set of $< n$ keys hide f.

Function Secret Sharing



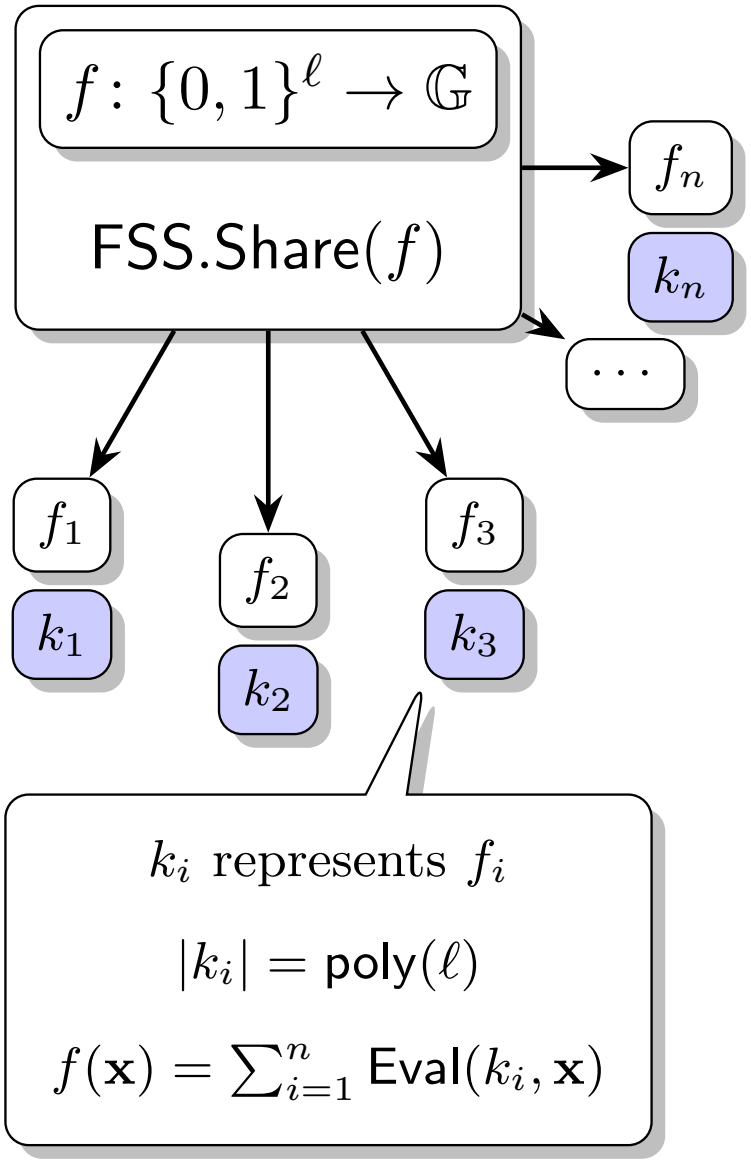
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.
  - Any set of $< n$ keys hide f.
- Additive reconstruction with respect to publicly known Eval function.

Function Secret Sharing



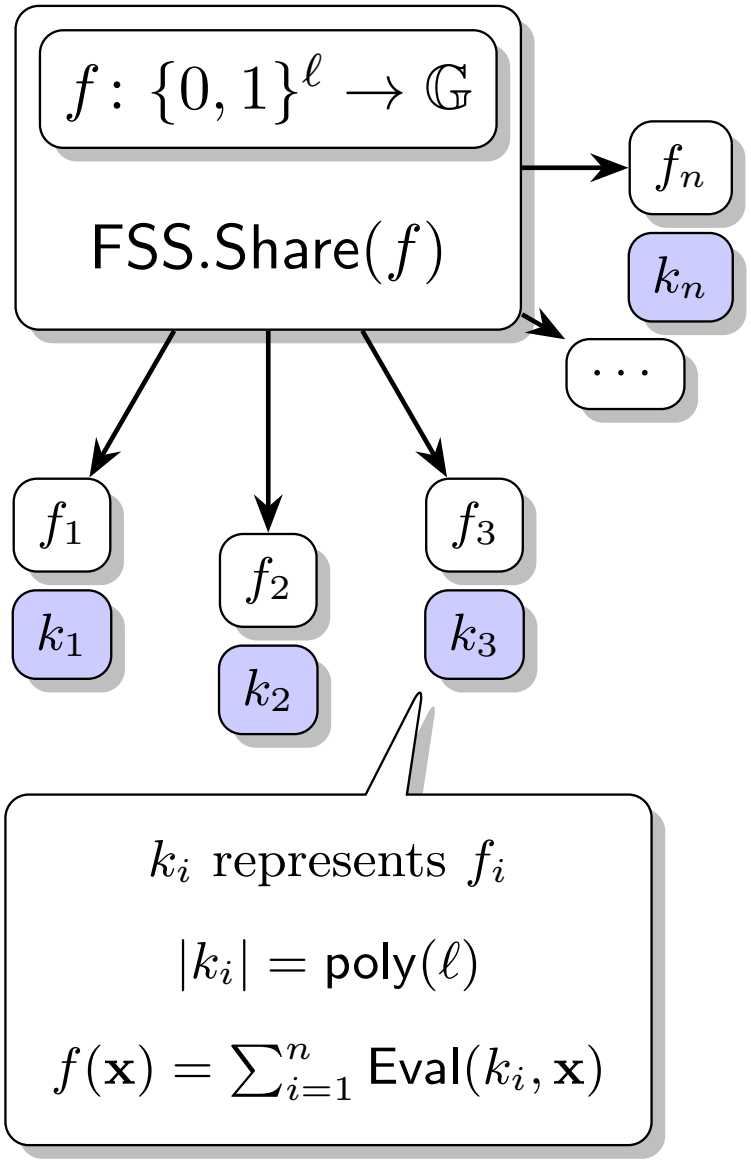
- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.
  - Any set of $< n$ keys hide f.
- Additive reconstruction with respect to publicly known Eval function.
- FSS thus has the following requirements:

Function Secret Sharing



- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.
  - Any set of $< n$ keys hide f.
- Additive reconstruction with respect to publicly known Eval function.
- FSS thus has the following requirements:
  - $f \in \mathcal{F}$: a set of structured functions with $|f| = \mathsf{poly}(\ell)$; and

Function Secret Sharing



- For $f : \{0,1\}^\ell \to \mathbb{G}$, trivial solution requires 2^ℓ group elements to be stored per party!
- Want to do better.
- Share each f_i as a key k_i.
  - k_i is a *compressed additive share* of f.
  - Any set of $< n$ keys hide f.
- Additive reconstruction with respect to publicly known Eval function.
- FSS thus has the following requirements:
  - $f \in \mathcal{F}$: a set of structured functions with $|f| = \mathsf{poly}(\ell)$; and
  - k_i's only computationally hide f.

FSS: Definition

Definition 1 (Function Secret Sharing)

FSS: Definition

Definition 1 (Function Secret Sharing)

Let $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}_{\ell \in \mathbb{N}}$ be a class of functions such that $|f| = \text{poly}(\ell)$ for all $f \in \mathcal{F}$.

FSS: Definition

Definition 1 (Function Secret Sharing)

Let $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}_{\ell \in \mathbb{N}}$ be a class of functions such that $|f| = \text{poly}(\ell)$ for all $f \in \mathcal{F}$. A (n,t)-*function secret sharing* scheme with respect to $\mathcal{F}$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

FSS: Definition

Definition 1 (Function Secret Sharing)

Let $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}_{\ell \in \mathbb{N}}$ be a class of functions such that $|f| = \text{poly}(\ell)$ for all $f \in \mathcal{F}$. A (n,t)-*function secret sharing* scheme with respect to $\mathcal{F}$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, f)$: on input security parameter 1^λ and function description $f \in \mathcal{F}$, the probabilistic *key generation algorithm* outputs n keys $(k_1, \ldots, k_n)$.

FSS: Definition

Definition 1 (Function Secret Sharing)

Let $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}_{\ell \in \mathbb{N}}$ be a class of functions such that $|f| = \text{poly}(\ell)$ for all $f \in \mathcal{F}$. A (n,t)-*function secret sharing* scheme with respect to $\mathcal{F}$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, f)$: on input security parameter 1^λ and function description $f \in \mathcal{F}$, the probabilistic *key generation algorithm* outputs n keys $(k_1, \ldots, k_n)$.
- $\mathsf{Eval}(i, k_i, x)$: on input party index i, key k_i, and input string $\mathbf{x} \in \{0,1\}^\ell$, the *evaluation algorithm* output $y_i \in \mathbb{G}$ (the value $f_i(\mathbf{x})$, the i^{th} share of $f(\mathbf{x})$).

FSS: Definition

Definition 1 (Function Secret Sharing)

Let $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}_{\ell \in \mathbb{N}}$ be a class of functions such that $|f| = \text{poly}(\ell)$ for all $f \in \mathcal{F}$. A (n,t)-*function secret sharing* scheme with respect to $\mathcal{F}$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, f)$: on input security parameter 1^λ and function description $f \in \mathcal{F}$, the probabilistic *key generation algorithm* outputs n keys $(k_1, \ldots, k_n)$.
- $\mathsf{Eval}(i, k_i, x)$: on input party index i, key k_i, and input string $\mathbf{x} \in \{0,1\}^\ell$, the *evaluation algorithm* output $y_i \in \mathbb{G}$ (the value $f_i(\mathbf{x})$, the i^{th} share of $f(\mathbf{x})$).

Moreover, $(\mathsf{Gen}, \mathsf{Eval})$ also satisfy the following correctness and security requirements (given in the next two definitions).

FSS: Correctness

Definition 2 (FSS Correctness)

FSS: Correctness

Definition 2 (FSS Correctness)

For all $f \in \mathcal{F}$ and $\mathbf{x} \in \{0,1\}^{\ell}$, we have

FSS: Correctness

Definition 2 (FSS Correctness)

For all $f \in \mathcal{F}$ and $\mathbf{x} \in \{0,1\}^{\ell}$, we have

$$\Pr\left[(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^{\lambda}, f) \ : \ f(\mathbf{x}) = \sum_{i=1}^{n} \mathsf{Eval}(i, k_i, \mathbf{x})\right] = 1.$$

FSS: Security

Definition 3 (FSS Security)

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

Let $\mathsf{Adv}(1^\lambda, \mathcal{A}) := |\Pr[b = b'] - 1/2|$ be the advantage of the adversary in the above game.

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

Let $\mathsf{Adv}(1^\lambda, \mathcal{A}) := |\Pr[b = b'] - 1/2|$ be the advantage of the adversary in the above game. We say that $(\mathsf{Gen}, \mathsf{Eval})$ is *t-secure* if

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

Let $\mathsf{Adv}(1^\lambda, \mathcal{A}) := |\Pr[b = b'] - 1/2|$ be the advantage of the adversary in the above game. We say that $(\mathsf{Gen}, \mathsf{Eval})$ is *t-secure* if there exists a negligible function negl such that

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

Let $\mathsf{Adv}(1^\lambda, \mathcal{A}) := |\Pr[b = b'] - 1/2|$ be the advantage of the adversary in the above game. We say that $(\mathsf{Gen}, \mathsf{Eval})$ is *t-secure* if there exists a negligible function negl such that for all (non-uniform) PPT adversaries $\mathcal{A}$, it holds that

FSS: Security

Definition 3 (FSS Security)

Consider the following experiment for corrupt parties $T \subset [n]$ with $|T| \leqslant t$, controlled by adversary $\mathcal{A}$:

1. The adversary outputs $(f_0, f_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda)$, where $f_0, f_1 \in \mathcal{F}$ and both have domain $\{0,1\}^\ell$, and sends (f_0, f_1) and T to the challenger.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$ and sends $(k_i)_{i \in T}$ to the adversary.
3. The adversary outputs a guess $b' \xleftarrow{\$} \mathcal{A}((k_i)_{i \in T}, \mathsf{state})$.

Let $\mathsf{Adv}(1^\lambda, \mathcal{A}) := |\Pr[b = b'] - 1/2|$ be the advantage of the adversary in the above game. We say that $(\mathsf{Gen}, \mathsf{Eval})$ is *t-secure* if there exists a negligible function negl such that for all (non-uniform) PPT adversaries $\mathcal{A}$, it holds that $\mathsf{Adv}(1^\lambda, \mathcal{A}) \leqslant \mathsf{negl}(\lambda)$.

Distributed Point Functions

Distributed Point Functions

Distributed Point Functions

- We'll build a FSS scheme for the special class of *point functions*.

Distributed Point Functions

- We'll build a FSS scheme for the special class of *point functions*.

Definition 4 (Point Function)

Distributed Point Functions

- We'll build a FSS scheme for the special class of *point functions*.

Definition 4 (Point Function)

For $\alpha \in \{0,1\}^\ell$ and $\beta \in \mathbb{G}$, the *point function* $f_{\alpha,\beta}$ is defined as

Distributed Point Functions

- We'll build a FSS scheme for the special class of *point functions*.

Definition 4 (Point Function)

For $\alpha \in \{0,1\}^\ell$ and $\beta \in \mathbb{G}$, the *point function* $f_{\alpha,\beta}$ is defined as

$$f_{\alpha,\beta}(\mathbf{x}) = \begin{cases} \beta & \text{if } \mathbf{x} = \alpha \\ 0 & \text{otherwise} \end{cases}.$$

Distributed Point Functions

- We'll build a FSS scheme for the special class of *point functions*.

Definition 4 (Point Function)

For $\alpha \in \{0,1\}^\ell$ and $\beta \in \mathbb{G}$, the *point function* $f_{\alpha,\beta}$ is defined as

$$f_{\alpha,\beta}(\mathbf{x}) = \begin{cases} \beta & \text{if } \mathbf{x} = \alpha \\ 0 & \text{otherwise} \end{cases}.$$

Definition 5 (Distributed Point Function)

A *distributed point function* (DPF) is a $(2,2)$-FSS for the family of all point functions $\mathcal{F} = \{f_{\alpha,\beta} \colon \alpha \in \{0,1\}^\ell, \beta \in \mathbb{G}\}_{\ell \in \mathbb{N}}$.

(Failed) DPFs

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.
  - Set $k_i = (\alpha, \llbracket\beta\rrbracket_i)$.

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.
  - Set $k_i = (\alpha, \llbracket\beta\rrbracket_i)$.
  - Violates security/privacy of FSS!

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.
  - Set $k_i = (\alpha, \llbracket\beta\rrbracket_i)$.
  - Violates security/privacy of FSS!

- Failed Attempt 2: trivial sharing of the truth table

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.
  - Set $k_i = (\alpha, \llbracket\beta\rrbracket_i)$.
  - Violates security/privacy of FSS!

- Failed Attempt 2: trivial sharing of the truth table
  - Keys k_i are of size 2^ℓ, violates $k_i = \text{poly}(\ell)$.

(Failed) DPFs

- Goal: design a $(2, 2)$-FSS for point functions (a DPF).
- We will take $\mathbb{G} = \{0, 1\}$, so $\beta \in \mathbb{G}$ is a single bit.
  - Known as *single-bit DPF*.

- Already tricky to do!

- Failed Attempt 1:
  - Additively share β as $(\llbracket\beta\rrbracket_0, \llbracket\beta\rrbracket_1)$.
  - Set $k_i = (\alpha, \llbracket\beta\rrbracket_i)$.
  - Violates security/privacy of FSS!

- Failed Attempt 2: trivial sharing of the truth table
  - Keys k_i are of size 2^ℓ, violates $k_i = \text{poly}(\ell)$.

- To build a (single-bit) DPF, we will need two primitives: *puncturable PRF* and *PRGs*.

Puncturable PRF and PRGs

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

Puncturable PRF and PRGs

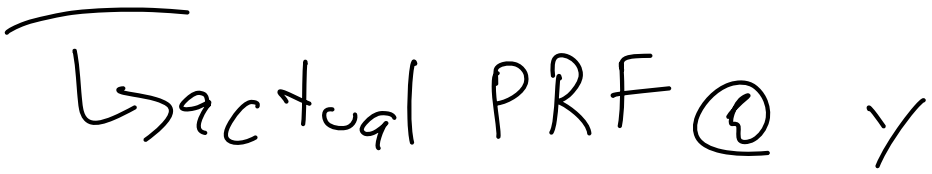
- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;



Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and

computationally

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and
  - k' (computationally) hides x.

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and
  - k' (computationally) hides x.

- We also need *pseudorandom generators* (PRGs).

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and
  - k' (computationally) hides x.

- We also need *pseudorandom generators* (PRGs).
  - Informally, PRGs take a λ-bit random seed and stretch it to some longer *pseudorandom* string.

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and
  - k' (computationally) hides x.

- We also need *pseudorandom generators* (PRGs).
  - Informally, PRGs take a λ-bit random seed and stretch it to some longer *pseudorandom* string.
  - A (poly-time computable) deterministic function $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+s}$ is a *PRG with stretch* s if $G(k) \approx_{\mathsf{negl}} U_{\lambda+s}$ for $k \xleftarrow{\$} \{0,1\}^\lambda$.

Puncturable PRF and PRGs

- Let $\{F_k : \{0,1\}^n \to \{0,1\}^m\}_k$ be a PRF family.

- Informally, for key k and PRF $F_k = F(k, \cdot)$, a *puncturable PRF* allows us to "puncture" (i.e., transform) k at a point x into new key $k' := k\{x\}$ such that
  - $F(k', y) = F(k, y)$ for all $y \neq x$;
  - $F(k', x)$ is indistinguishable from random; and
  - k' (computationally) hides x.

- We also need *pseudorandom generators* (PRGs).
  - Informally, PRGs take a λ-bit random seed and stretch it to some longer *pseudorandom* string.
  - A (poly-time computable) deterministic function $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+s}$ is a *PRG with stretch* s if $G(k) \approx_{\mathsf{negl}} U_{\lambda+s}$ for $k \xleftarrow{\$} \{0,1\}^\lambda$.
  - When $s = \lambda$, we call G a *length-doubling PRG*.

PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

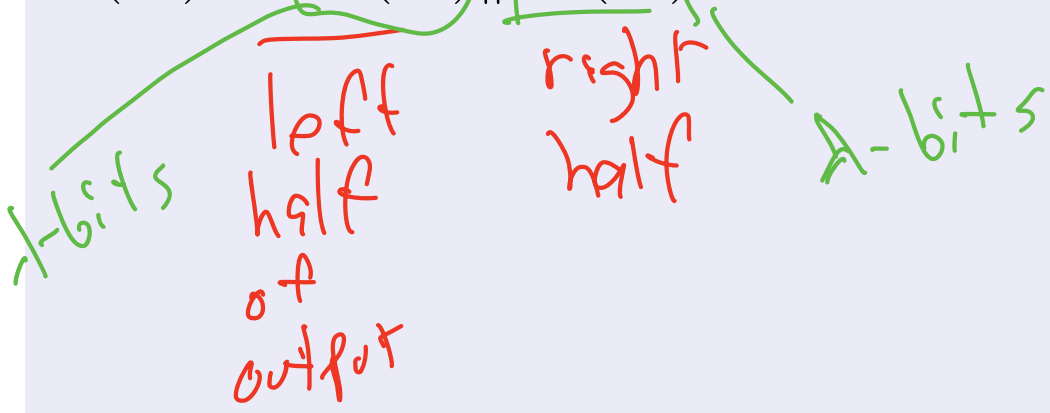
Theorem 1

PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

Theorem 1

Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling PRG, and let* $G(K) = G_0(K) \| G_1(K)$ *denote the left and right halves of the output.*



PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

Theorem 1

Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling PRG, and let* $G(K) = G_0(K) \| G_1(K)$ *denote the left and right halves of the output. Then, the following function family is a PRF family: for any* $K \in \{0,1\}^\lambda$*, the function* $F_K\colon \{0,1\}^n \to \{0,1\}^\lambda$ *is defined as:*

$n = \text{poly}(\lambda)$

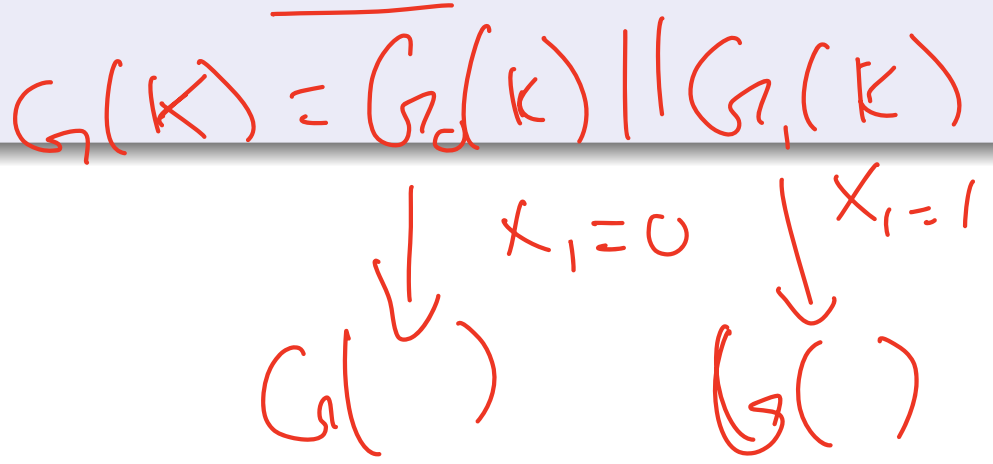
PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

Theorem 1

Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling PRG, and let* $G(K) = G_0(K)\|G_1(K)$ *denote the left and right halves of the output. Then, the following function family is a PRF family: for any* $K \in \{0,1\}^\lambda$*, the function* $F_K\colon \{0,1\}^n \to \{0,1\}^\lambda$ *is defined as:*

$$F_K(x) = G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots)),$$



PRFs from PRGs via GGM Construction

- We can construct PRFs from PRGs using the Goldreich-Goldwasser-Micali Construction.

Theorem 1

Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ *be a length-doubling PRG, and let* $G(K) = G_0(K) \| G_1(K)$ *denote the left and right halves of the output. Then, the following function family is a PRF family: for any* $K \in \{0,1\}^\lambda$*, the function* $F_K\colon \{0,1\}^n \to \{0,1\}^\lambda$ *is defined as:*

$$F_K(x) = G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots)),$$

where $x = (x_1, \ldots, x_n)$ *is the input to the function.*

PRFs from PRGs via GGM Construction

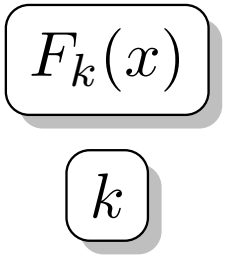
Tree is of size 2^{n+1}.

$F_k(x)$

$F_k(x)$ computable in poly(n) time for any x.

PRFs from PRGs via GGM Construction

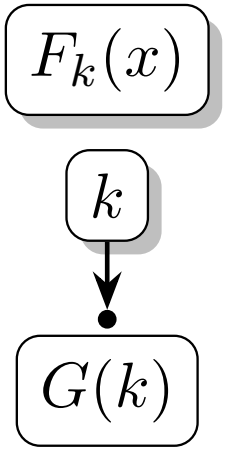
Tree is of size 2^{n+1}.



$F_k(x)$ computable in poly(n) time for any x.

PRFs from PRGs via GGM Construction

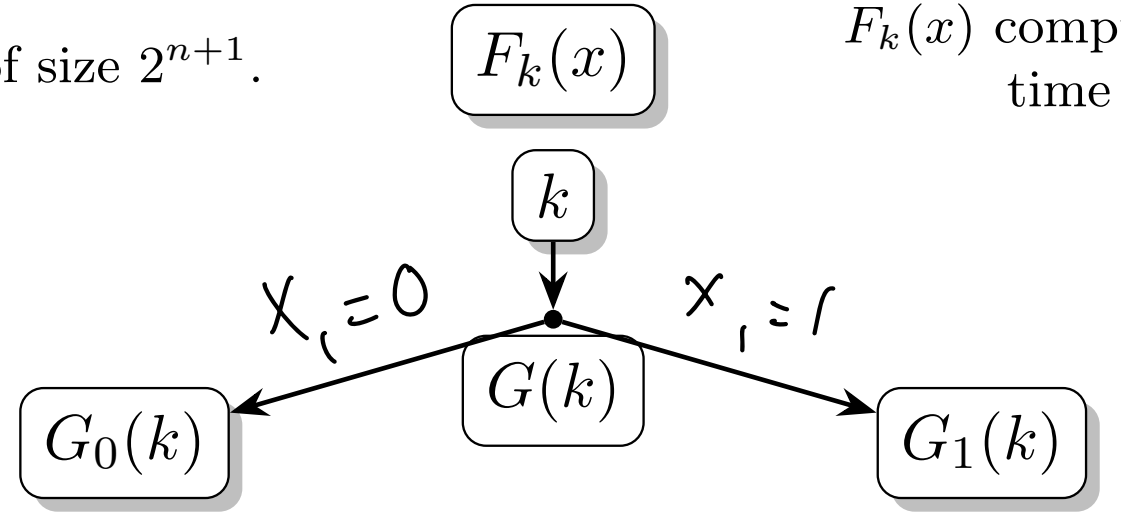
Tree is of size 2^{n+1}.



$F_k(x)$ computable in poly(n) time for any x.

PRFs from PRGs via GGM Construction

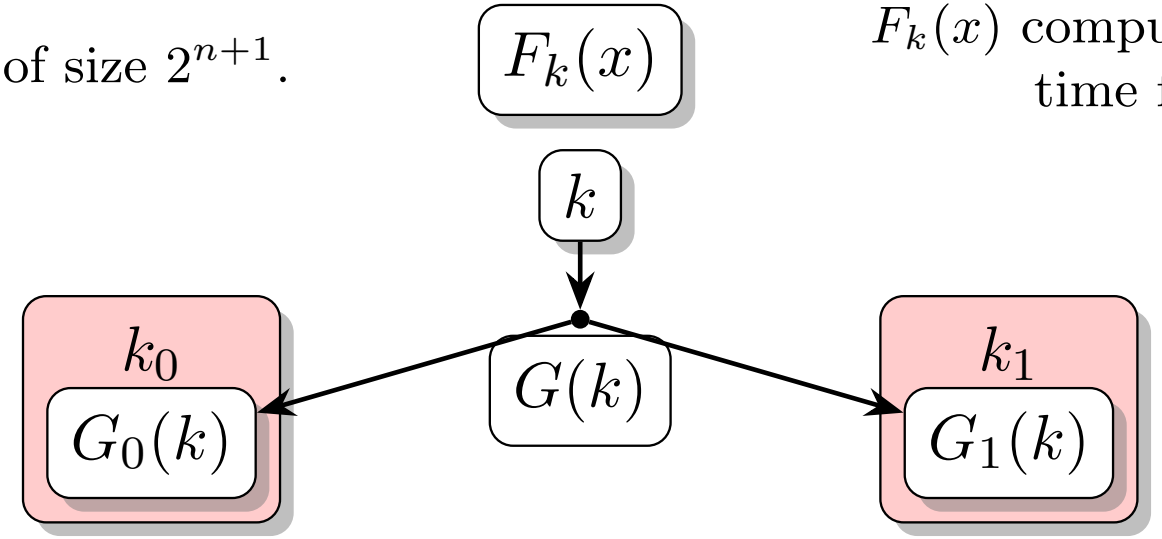
Tree is of size 2^{n+1}.



$F_k(x)$ computable in poly(n) time for any x.

PRFs from PRGs via GGM Construction

Tree is of size 2^{n+1}.

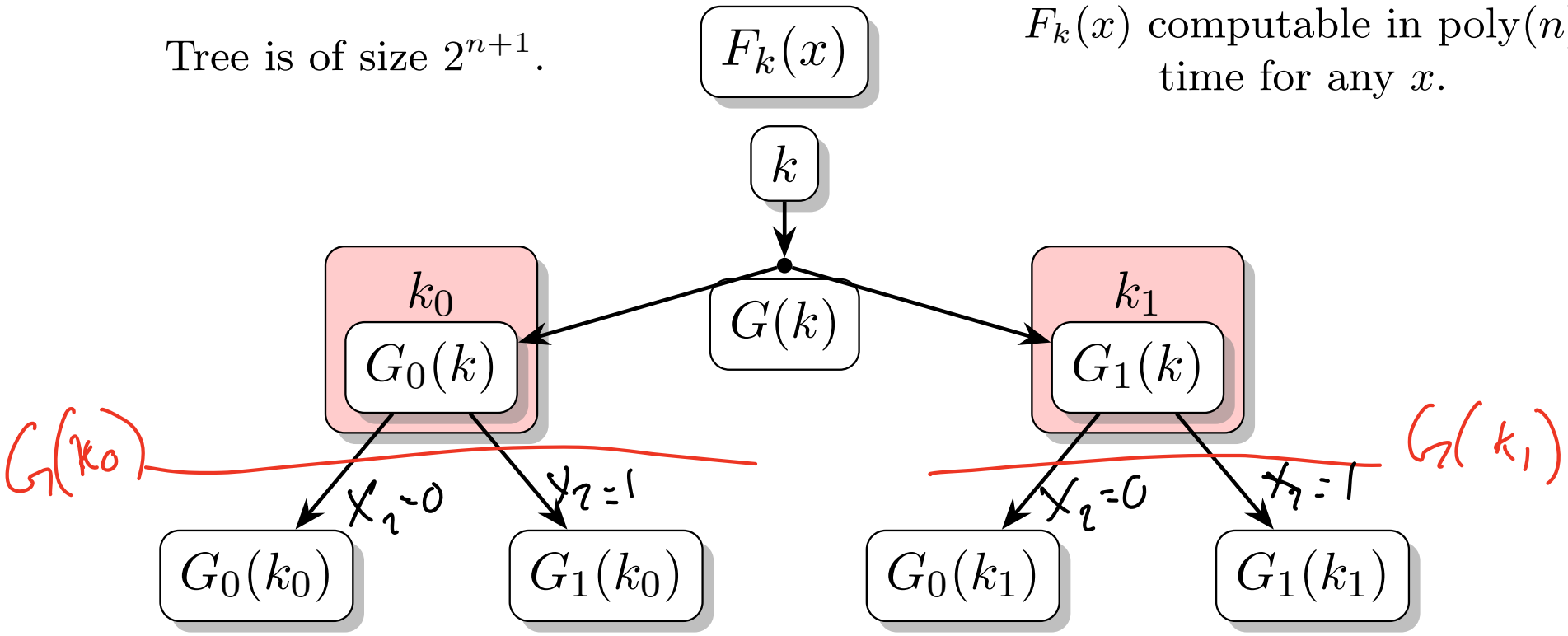


$F_k(x)$ computable in poly(n) time for any x.

PRFs from PRGs via GGM Construction

Tree is of size 2^{n+1}.

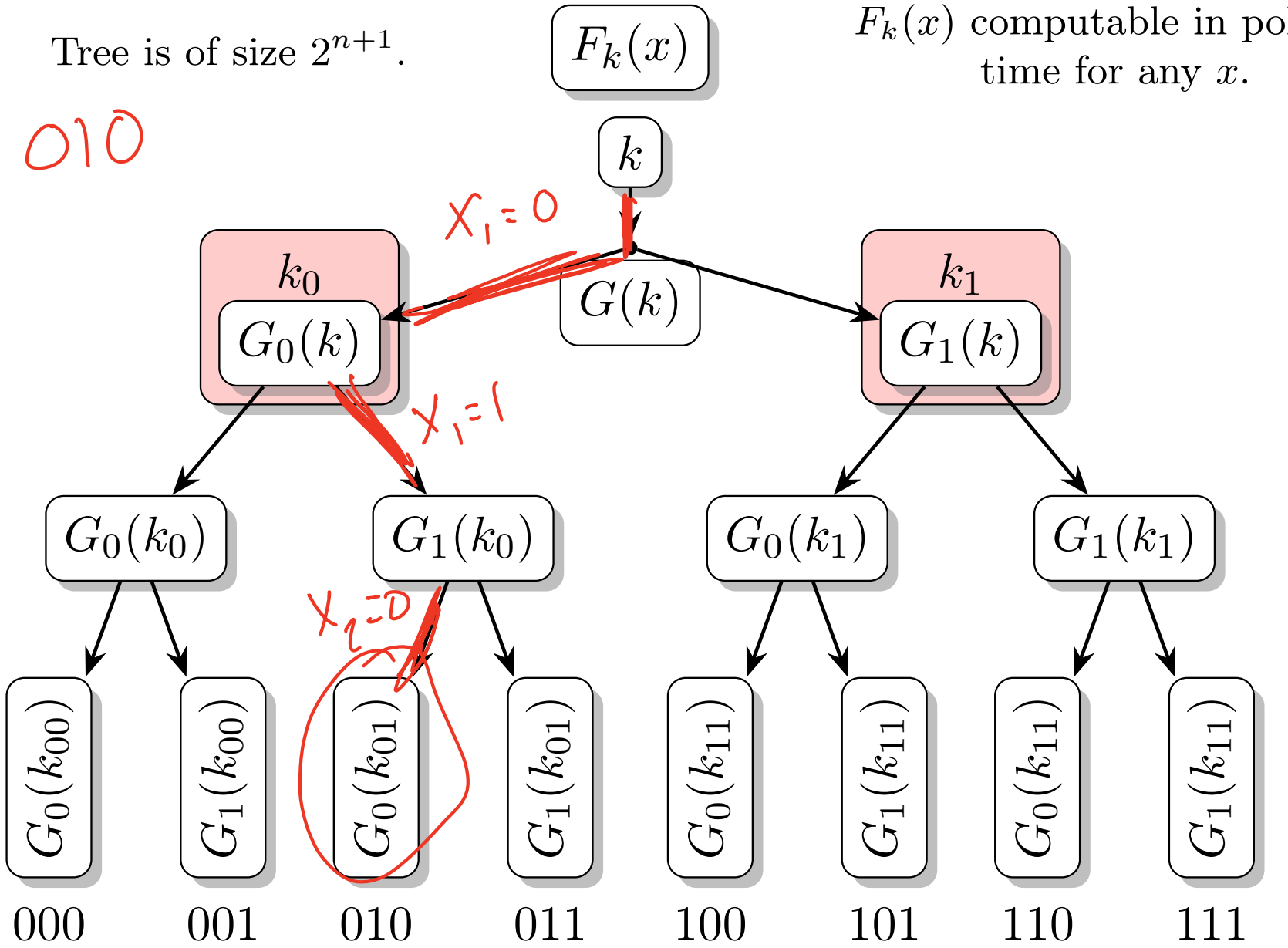
$F_k(x)$ computable in poly(n) time for any x.



PRFs from PRGs via GGM Construction

Tree is of size 2^{n+1}.

$F_k(x)$ computable in poly(n) time for any x.



$X = 010$

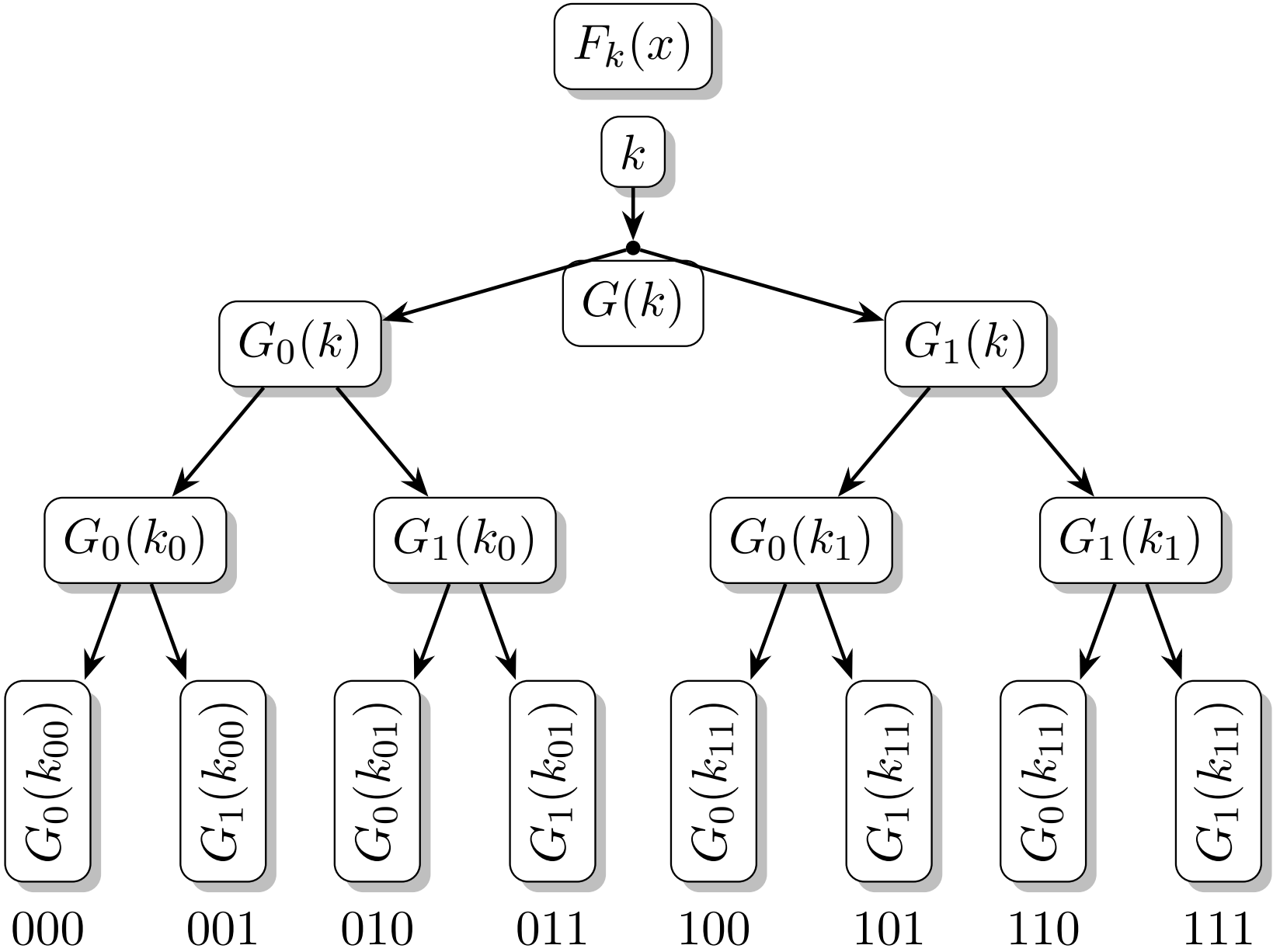
$X_1 = 0$

$X_1 = 1$

$X_2 = 0$

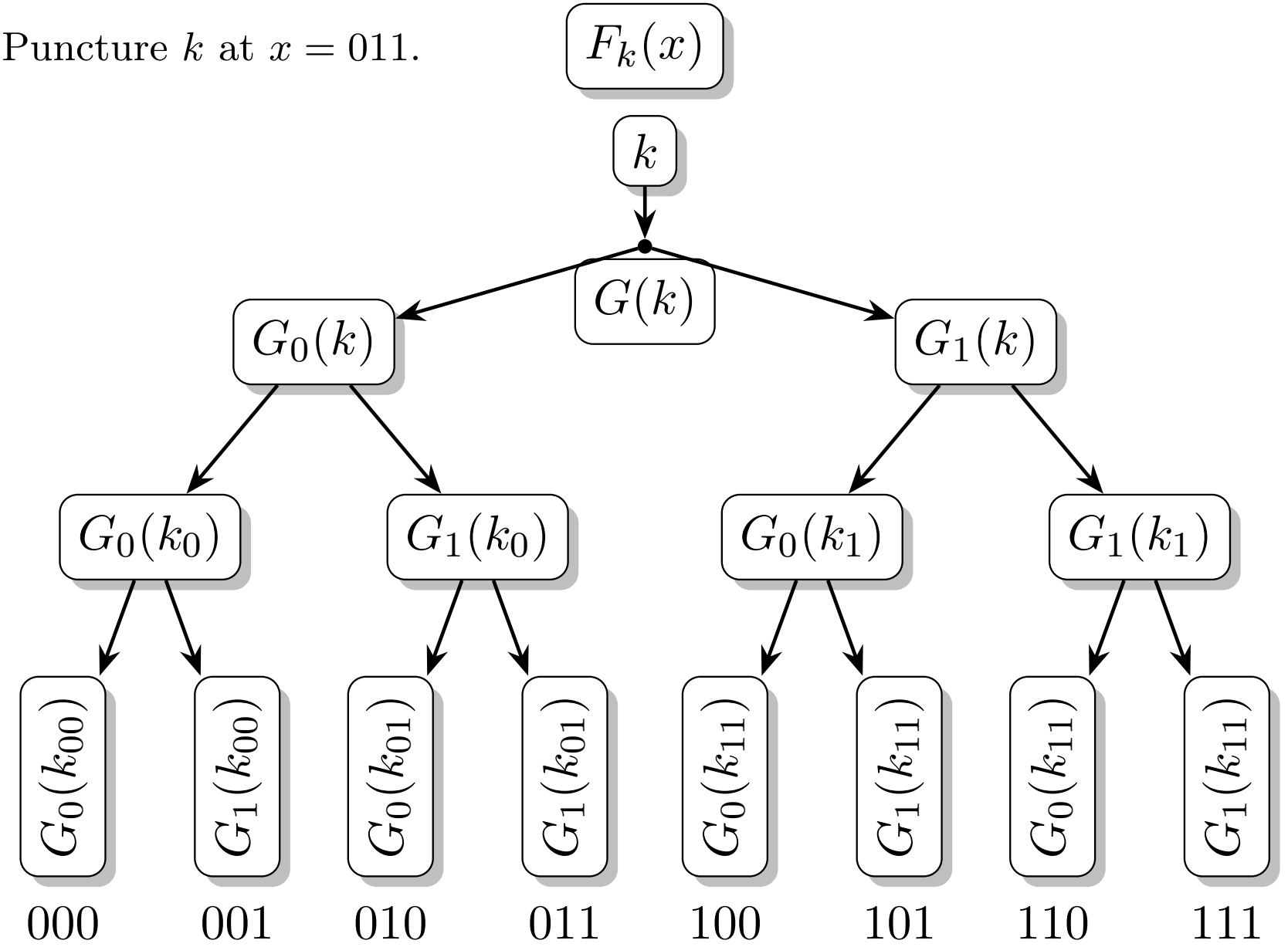
$F_k(010) = G_0(k_{01}) = k_{010} = G_0(G_1(G_0(k)))$

PPRFs from PRGs via GGM Construction



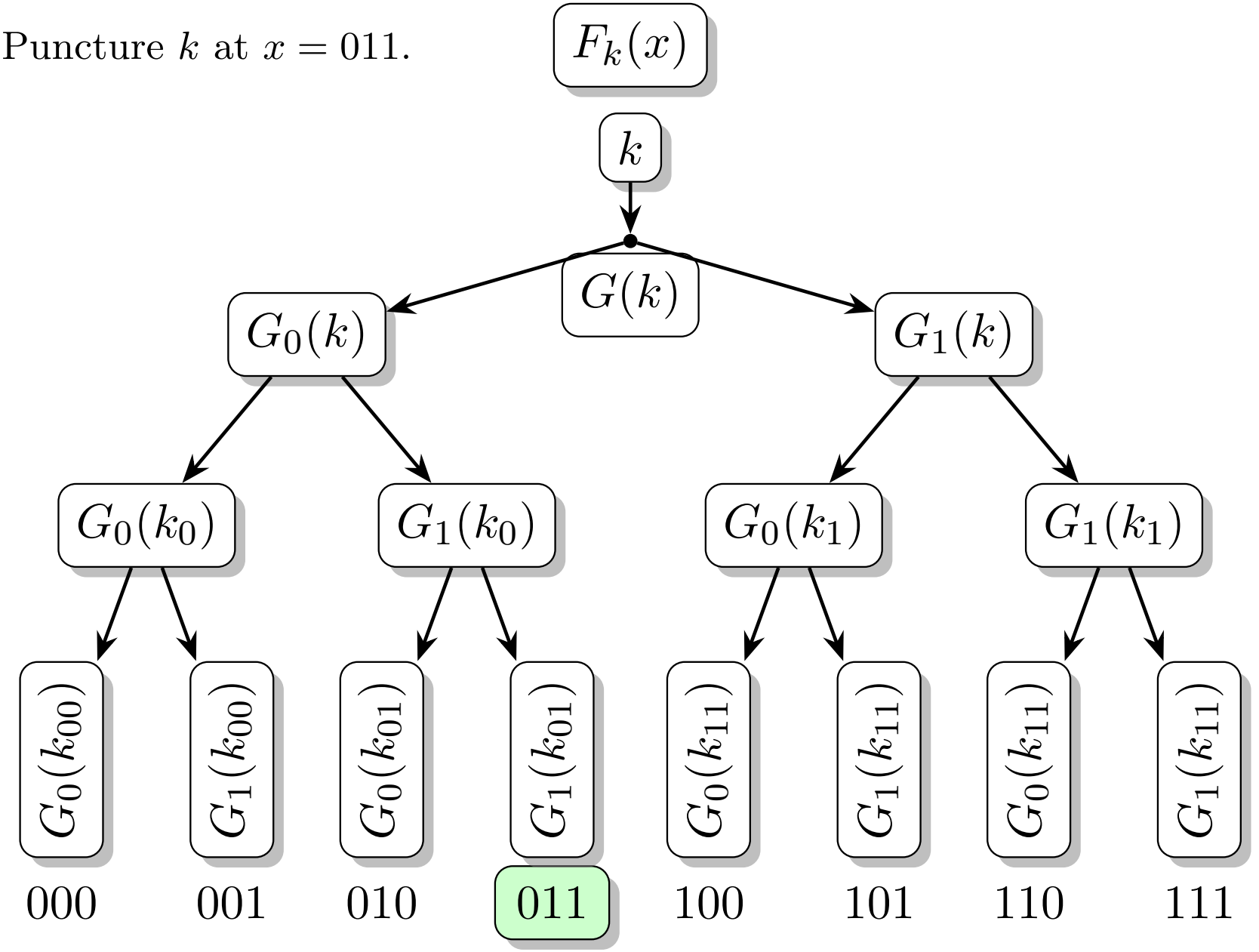
PPRFs from PRGs via GGM Construction

Puncture k at $x = 011$.



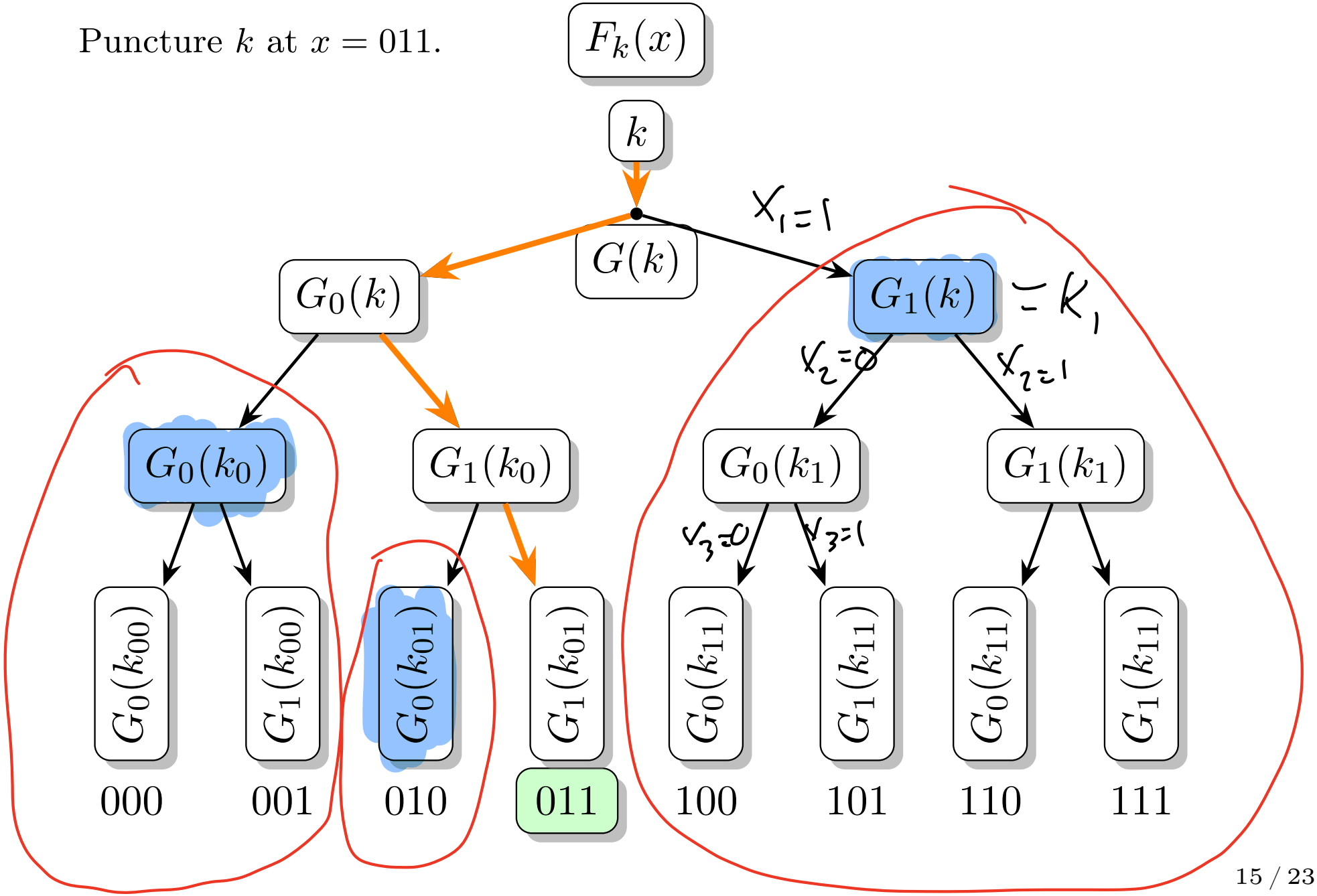
PPRFs from PRGs via GGM Construction

Puncture k at $x = 011$.



PPRFs from PRGs via GGM Construction

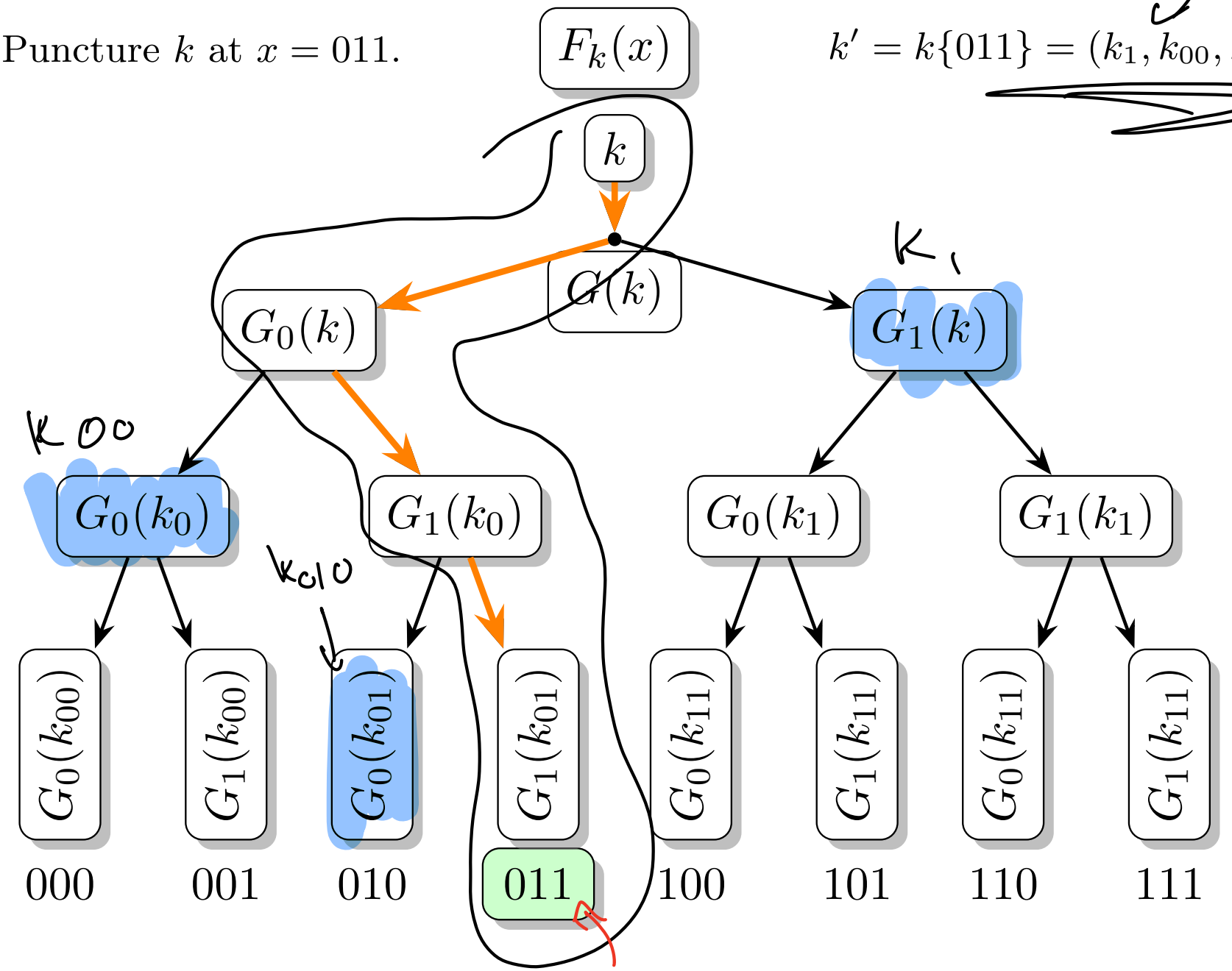
Puncture k at $x = 011$.



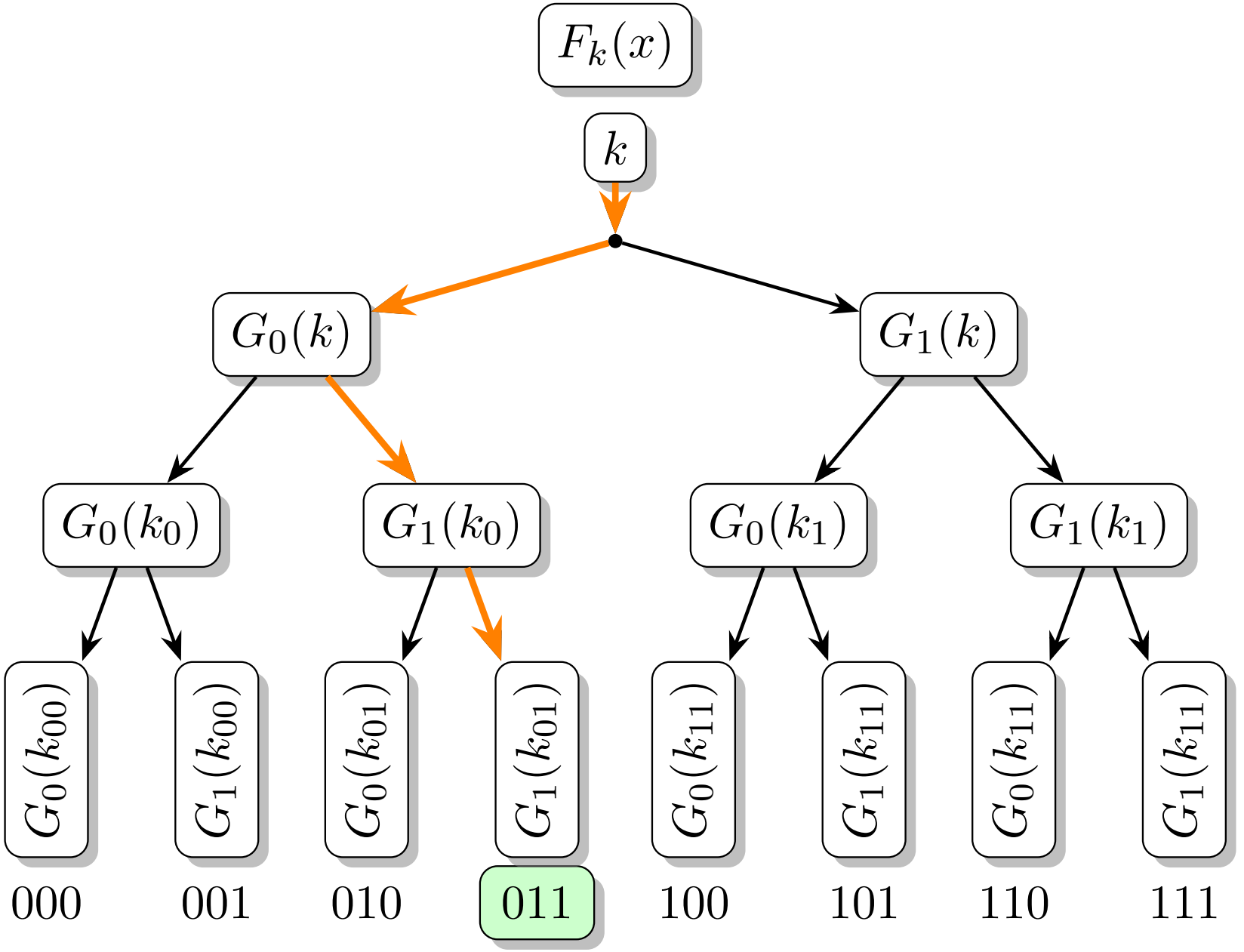
PPRFs from PRGs via GGM Construction

Puncture k at $x = 011$.

$k' = k\{011\} = (k_1, k_{00}, k_{010})$

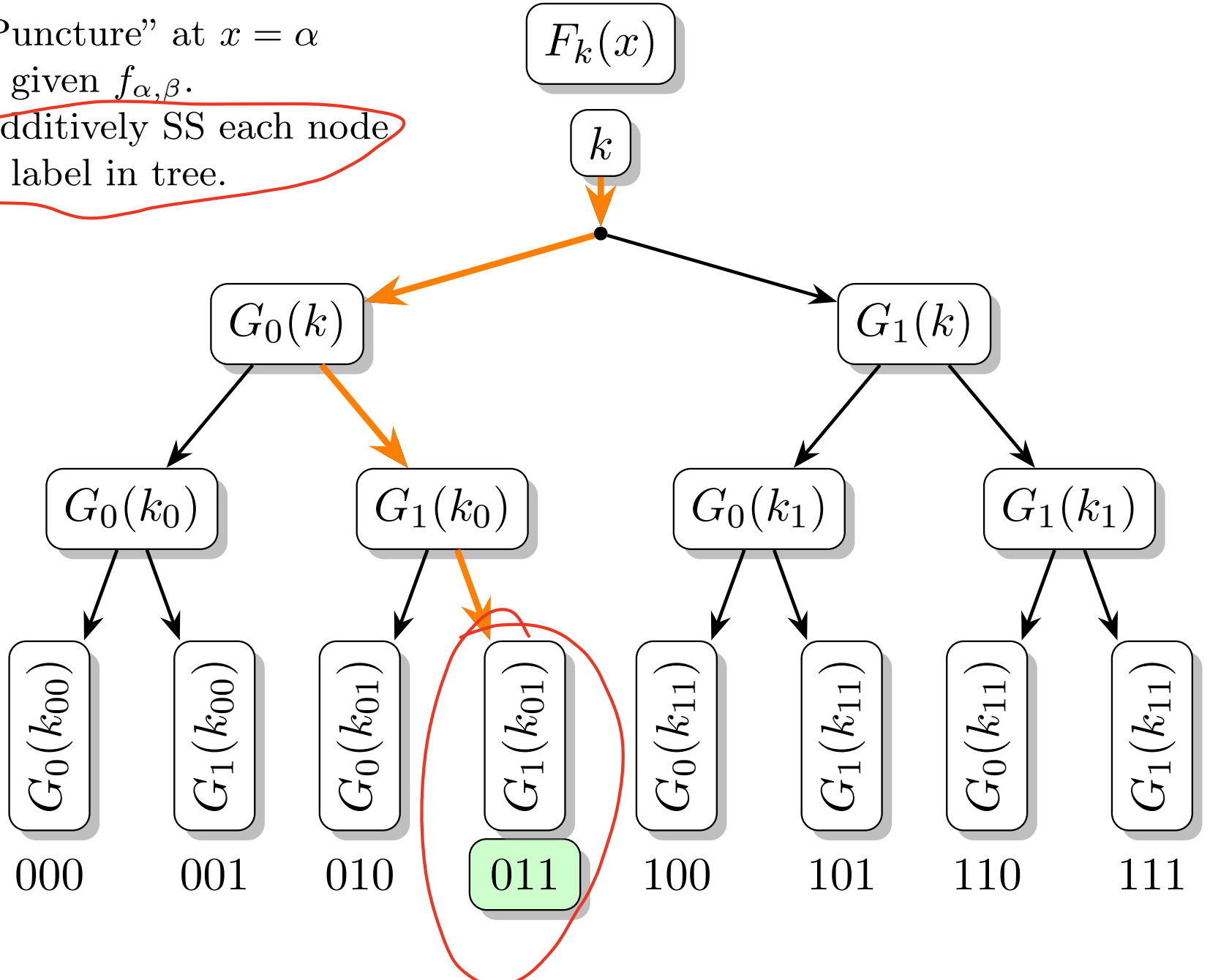


DPFs from PRGs via GGM-like Construction



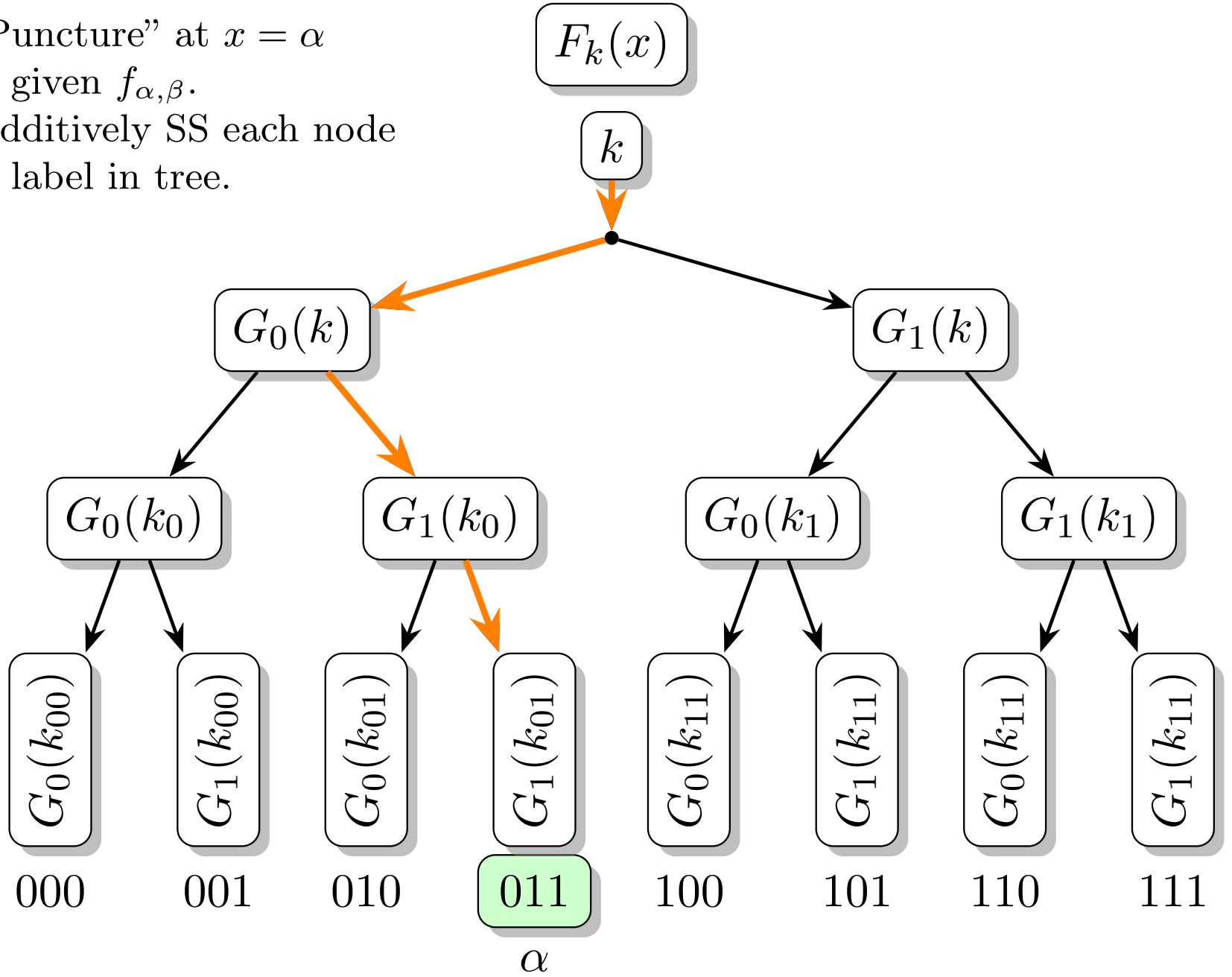
DPFs from PRGs via GGM-like Construction

"Puncture" at $x = \alpha$ given $f_{\alpha,\beta}$.
Additively SS each node label in tree.



DPFs from PRGs via GGM-like Construction

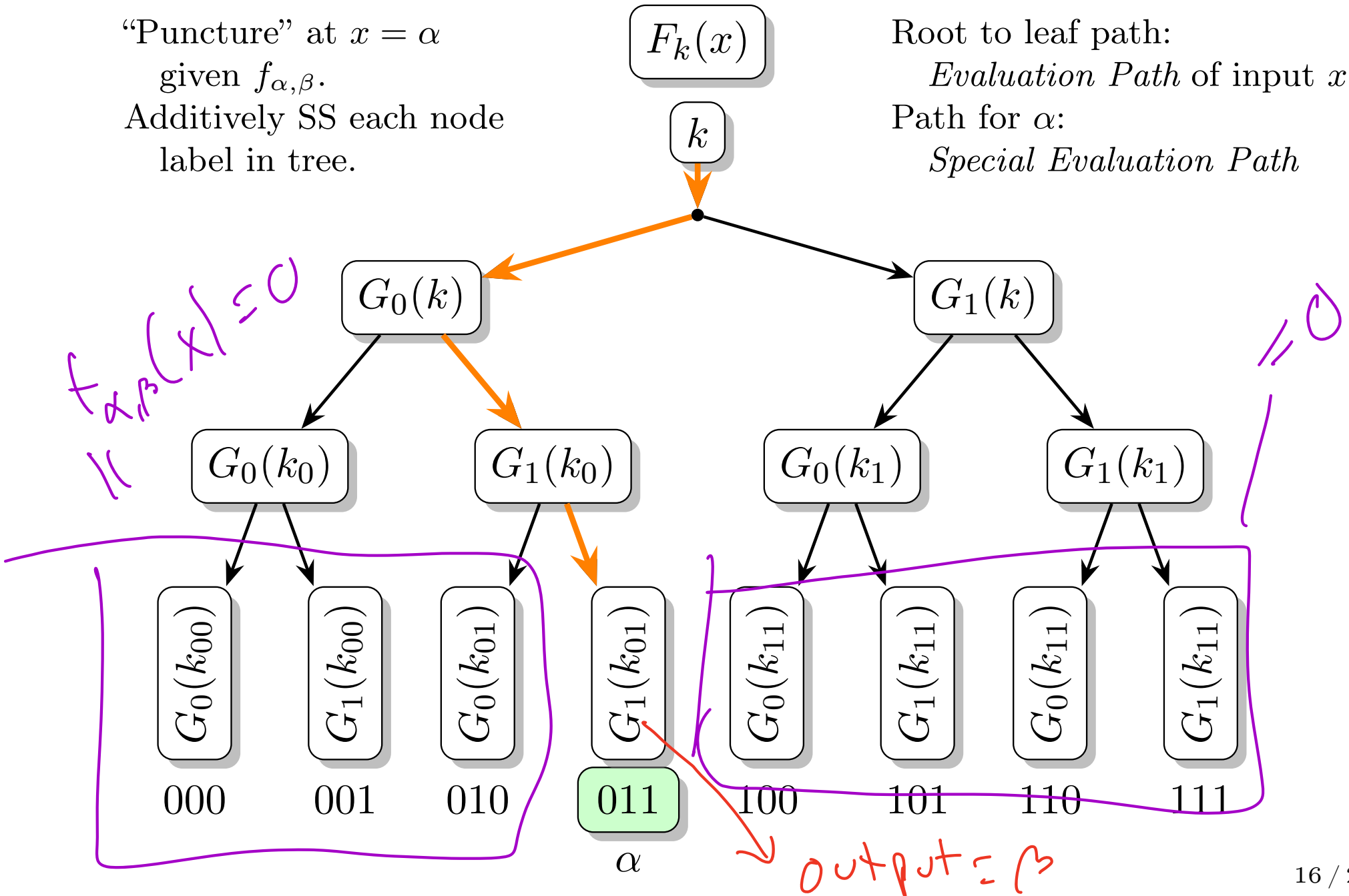
"Puncture" at $x = \alpha$ given $f_{\alpha,\beta}$.
Additively SS each node label in tree.



DPFs from PRGs via GGM-like Construction

"Puncture" at $x = \alpha$ given $f_{\alpha,\beta}$.
Additively SS each node label in tree.

Root to leaf path:
Evaluation Path of input x
Path for α:
Special Evaluation Path



DPFs from PRGs via GGM-like Construction

DPFs from PRGs via GGM-like Construction

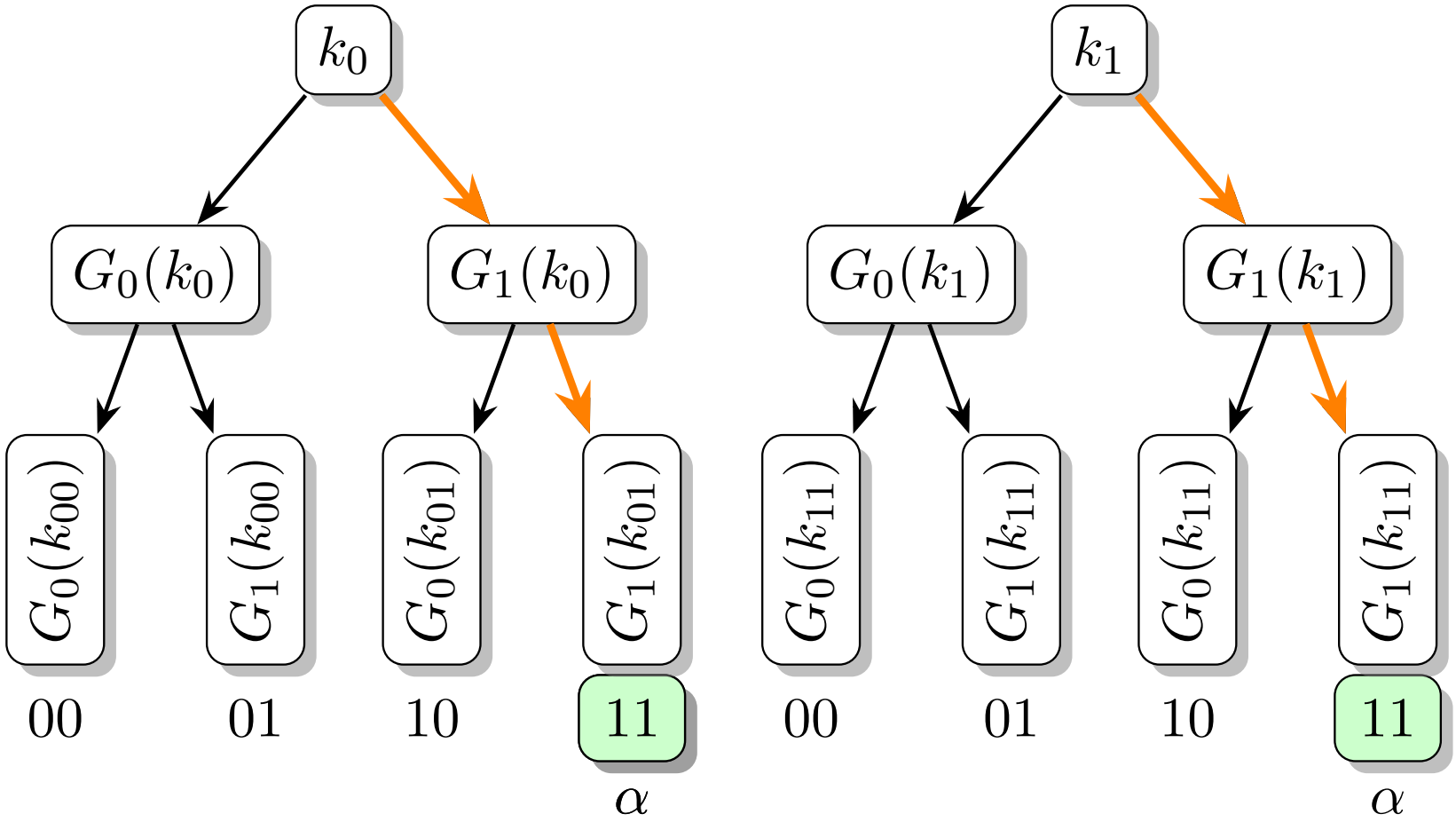
- Idea: to share $f_{\alpha,\beta}$, specify two keys K_0, K_1 where each allows you to compute a GGM-style binary tree.

DPFs from PRGs via GGM-like Construction

- Idea: to share $f_{\alpha,\beta}$, specify two keys K_0, K_1 where each allows you to compute a GGM-style binary tree.
- Each node in the tree will be labeled with $\lambda + 1$ bits: A λ-bit seed or key, and a single control bit t.

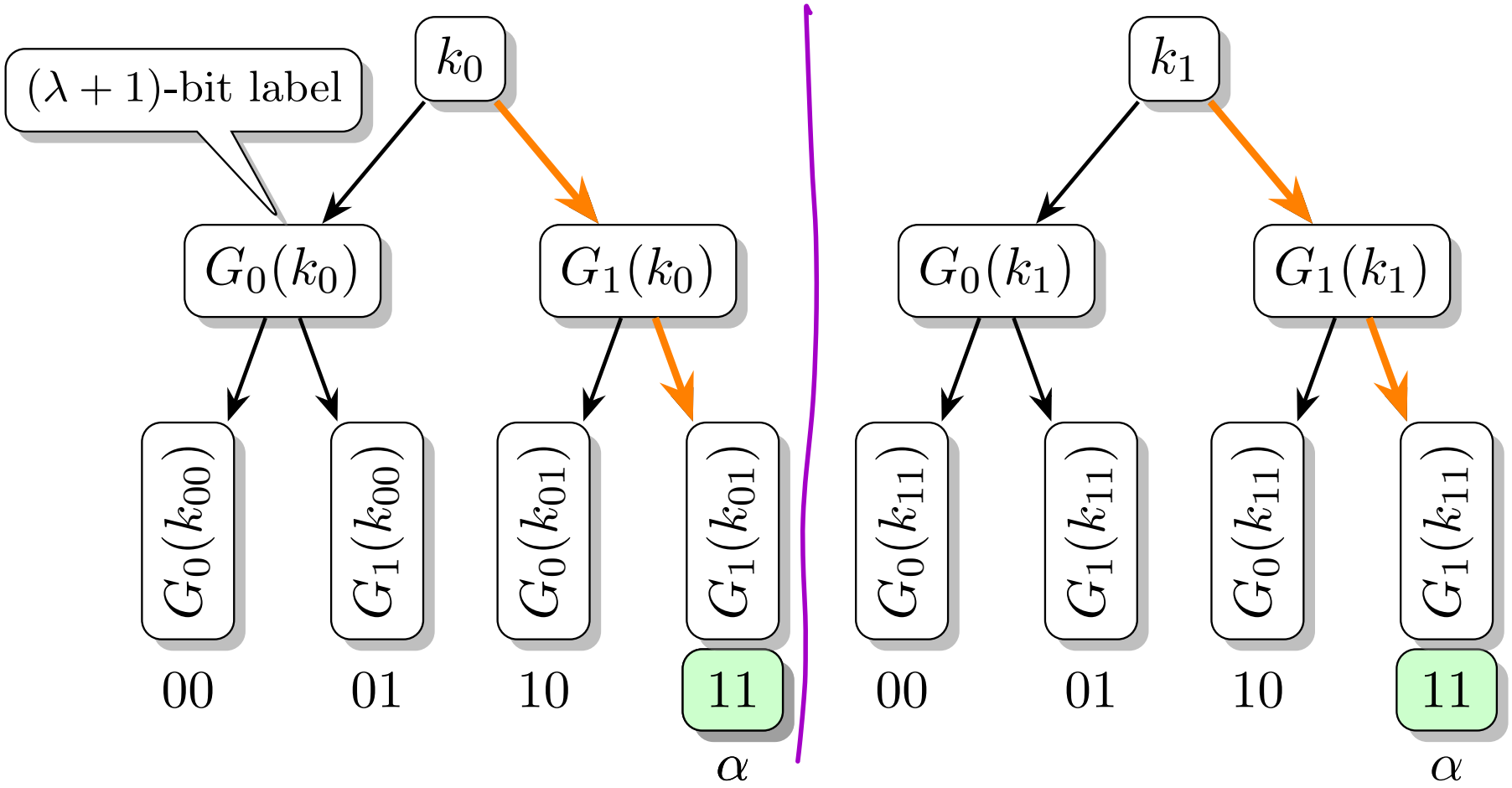
DPFs from PRGs via GGM-like Construction

- Idea: to share $f_{\alpha,\beta}$, specify two keys K_0, K_1 where each allows you to compute a GGM-style binary tree.
- Each node in the tree will be labeled with $\lambda + 1$ bits: A λ-bit seed or key, and a single control bit t.



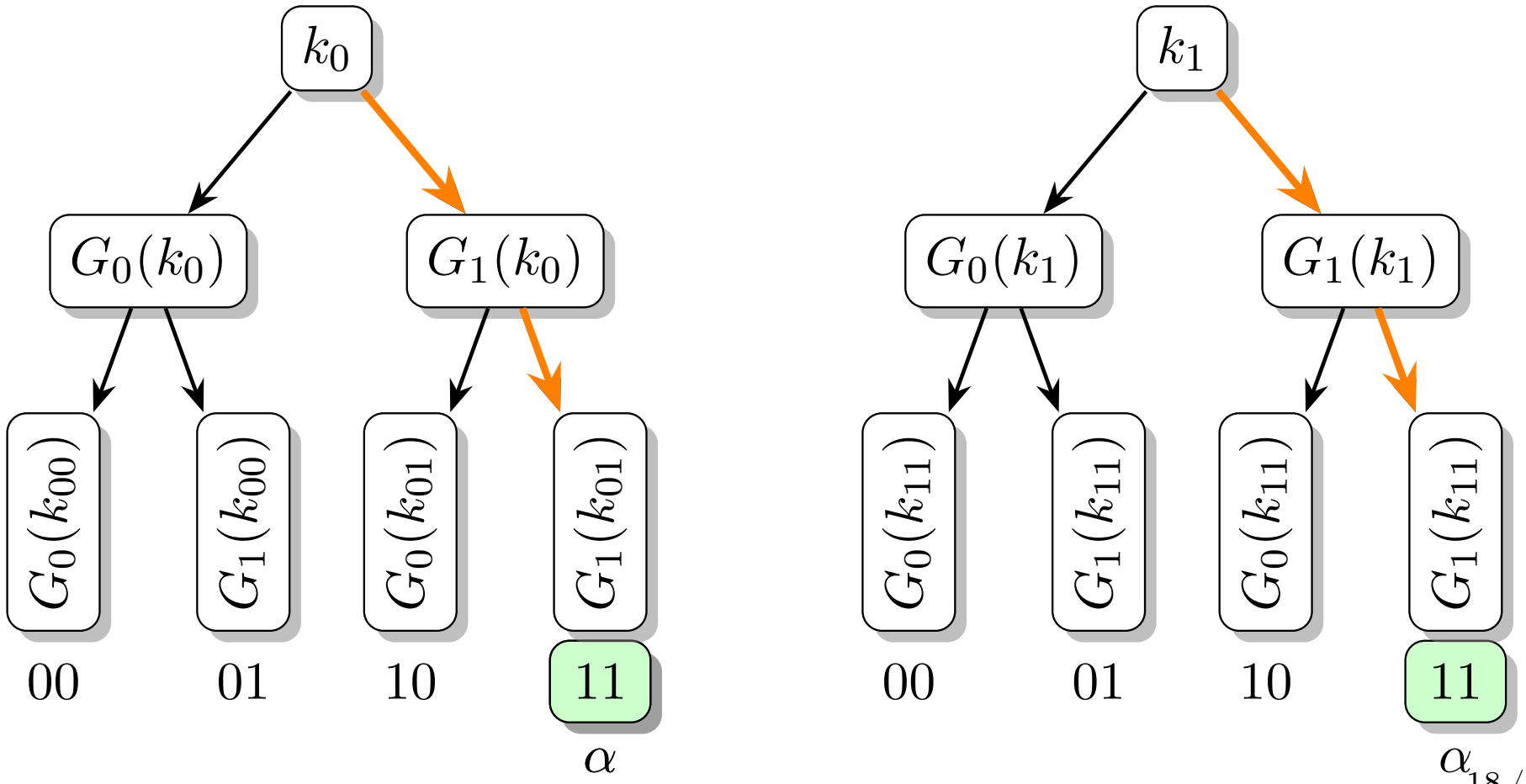
DPFs from PRGs via GGM-like Construction

- Idea: to share $f_{\alpha,\beta}$, specify two keys K_0, K_1 where each allows you to compute a GGM-style binary tree.
- Each node in the tree will be labeled with $\lambda + 1$ bits: A λ-bit seed or key, and a single control bit t.



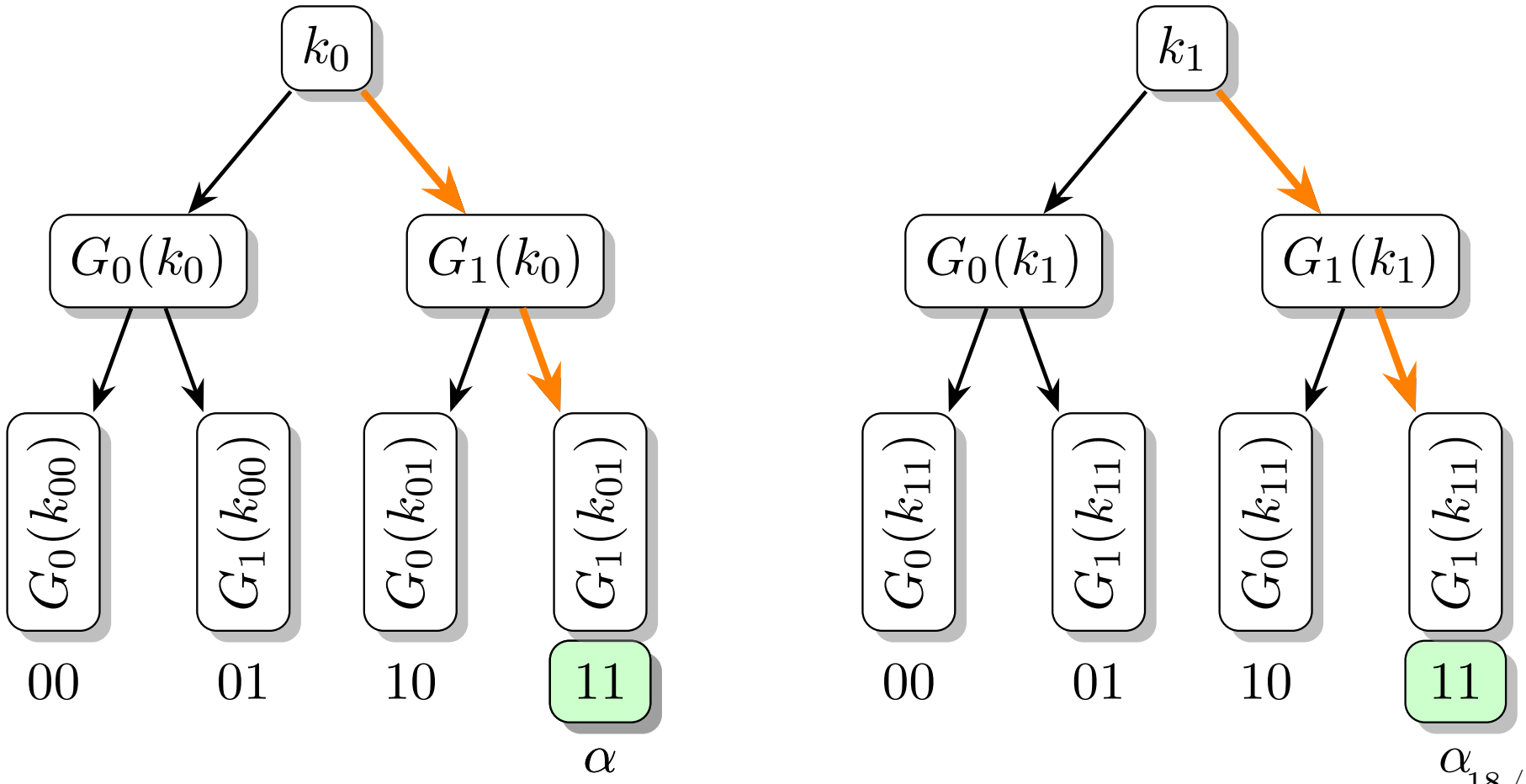
DPFs from PRGs via GGM-like Construction

- Need the following invariants:



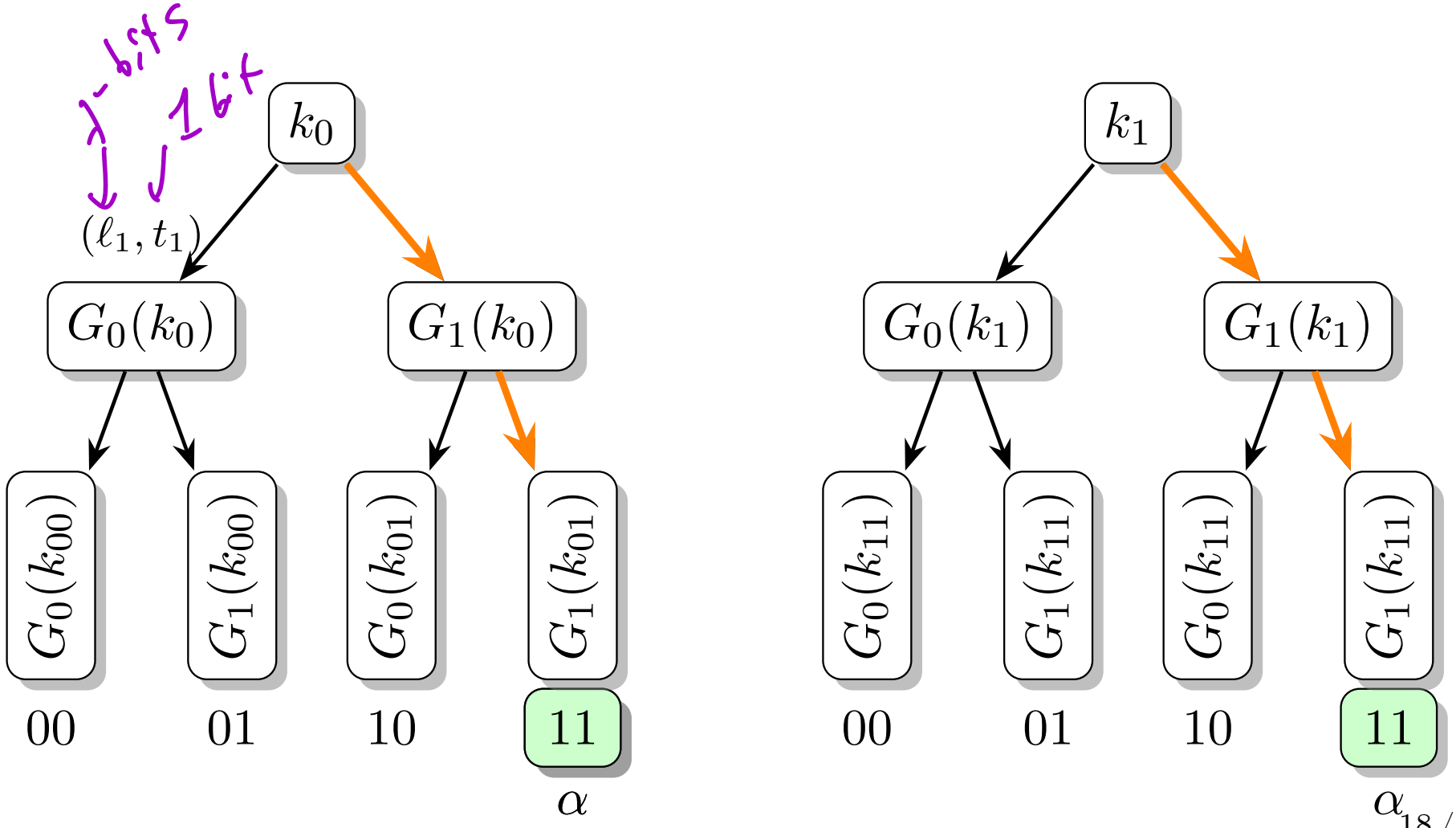
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



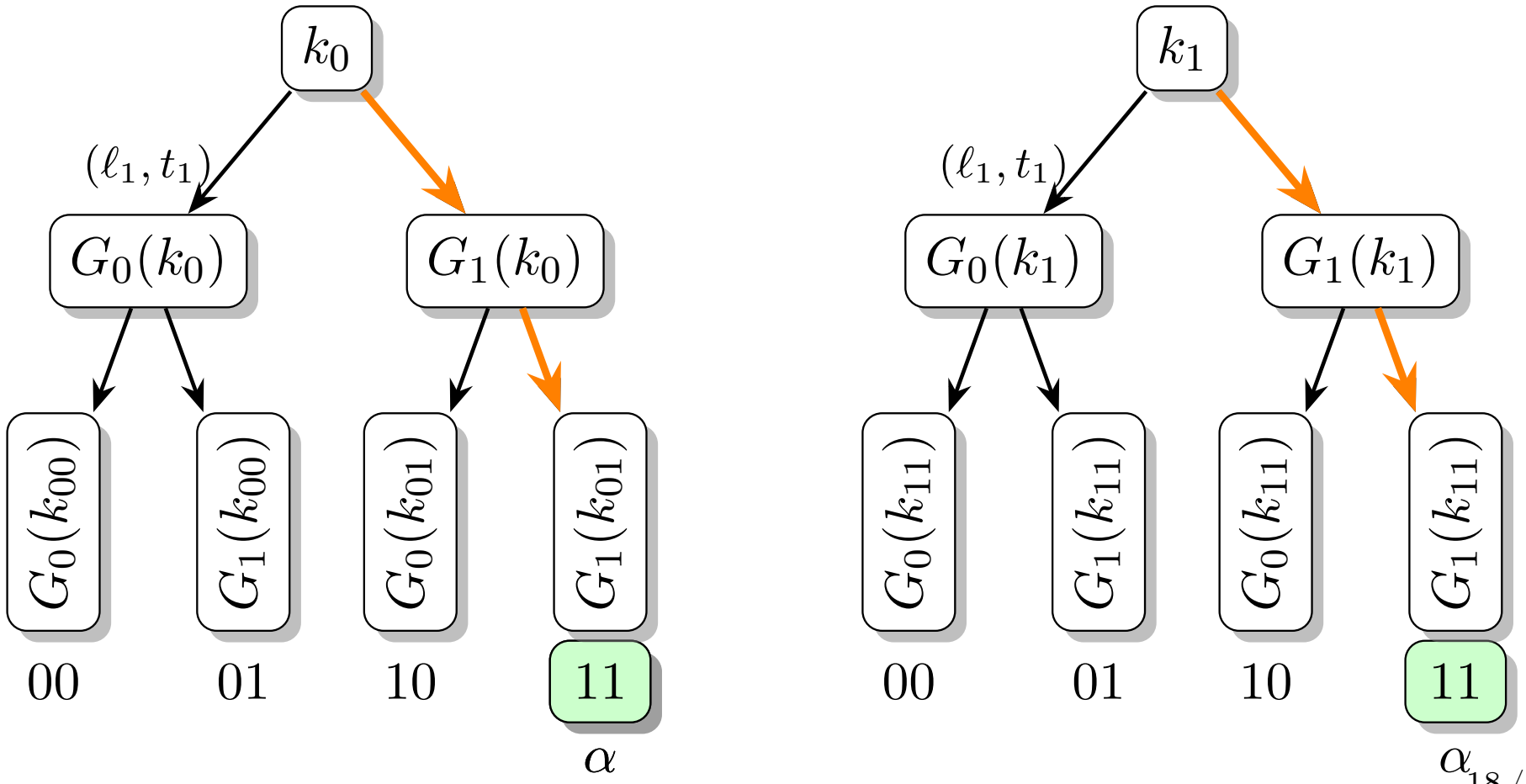
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



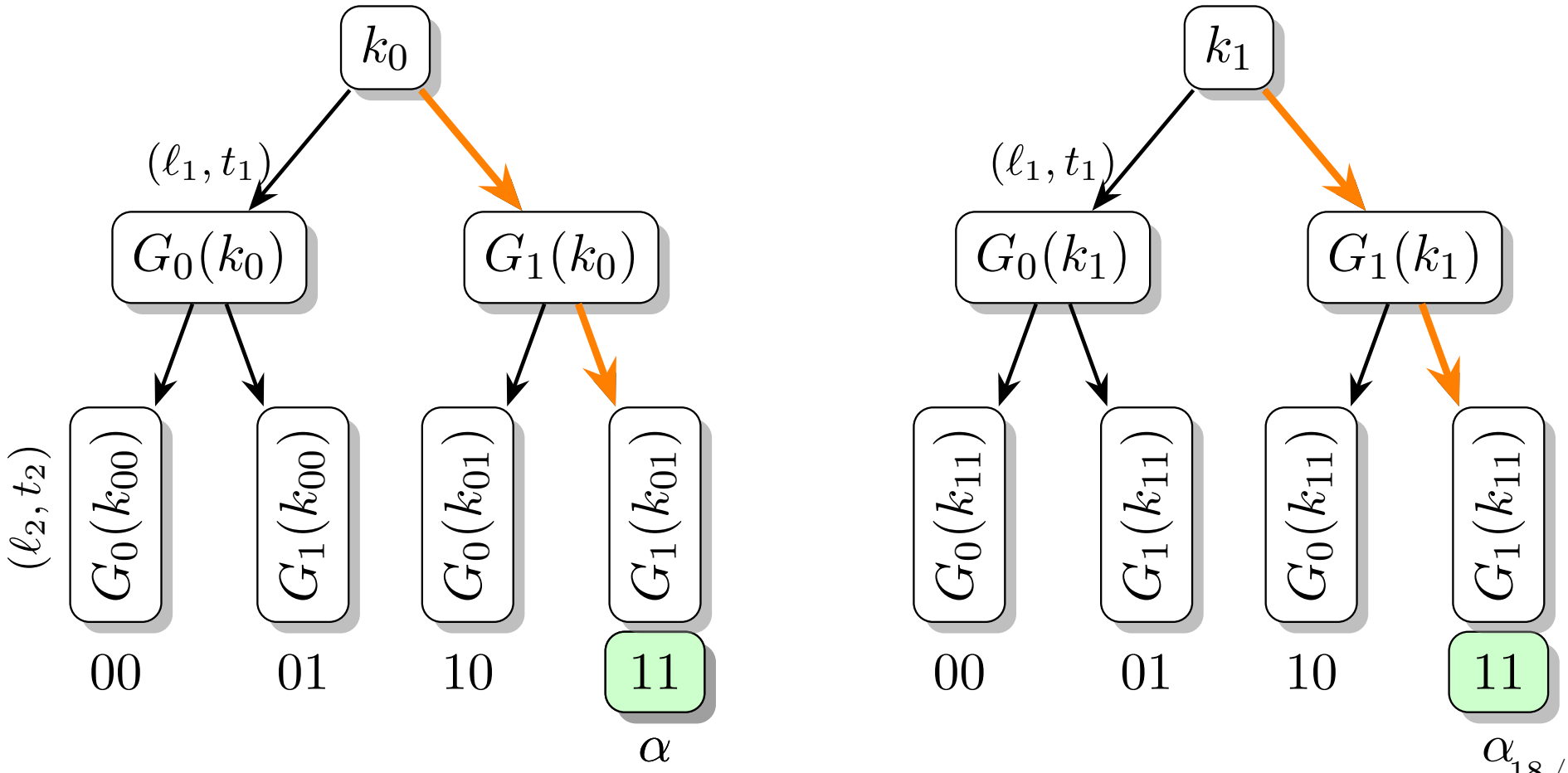
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



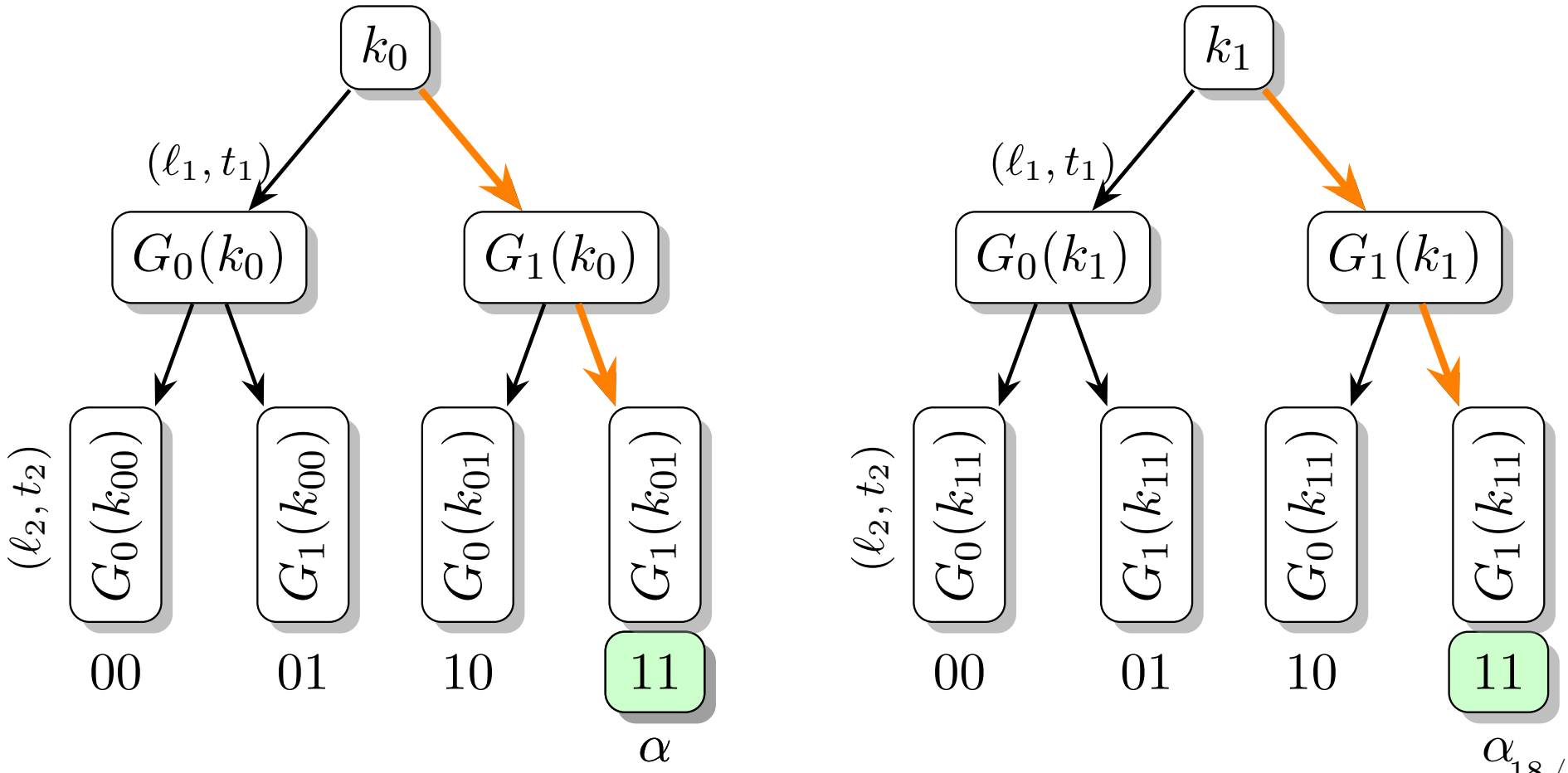
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



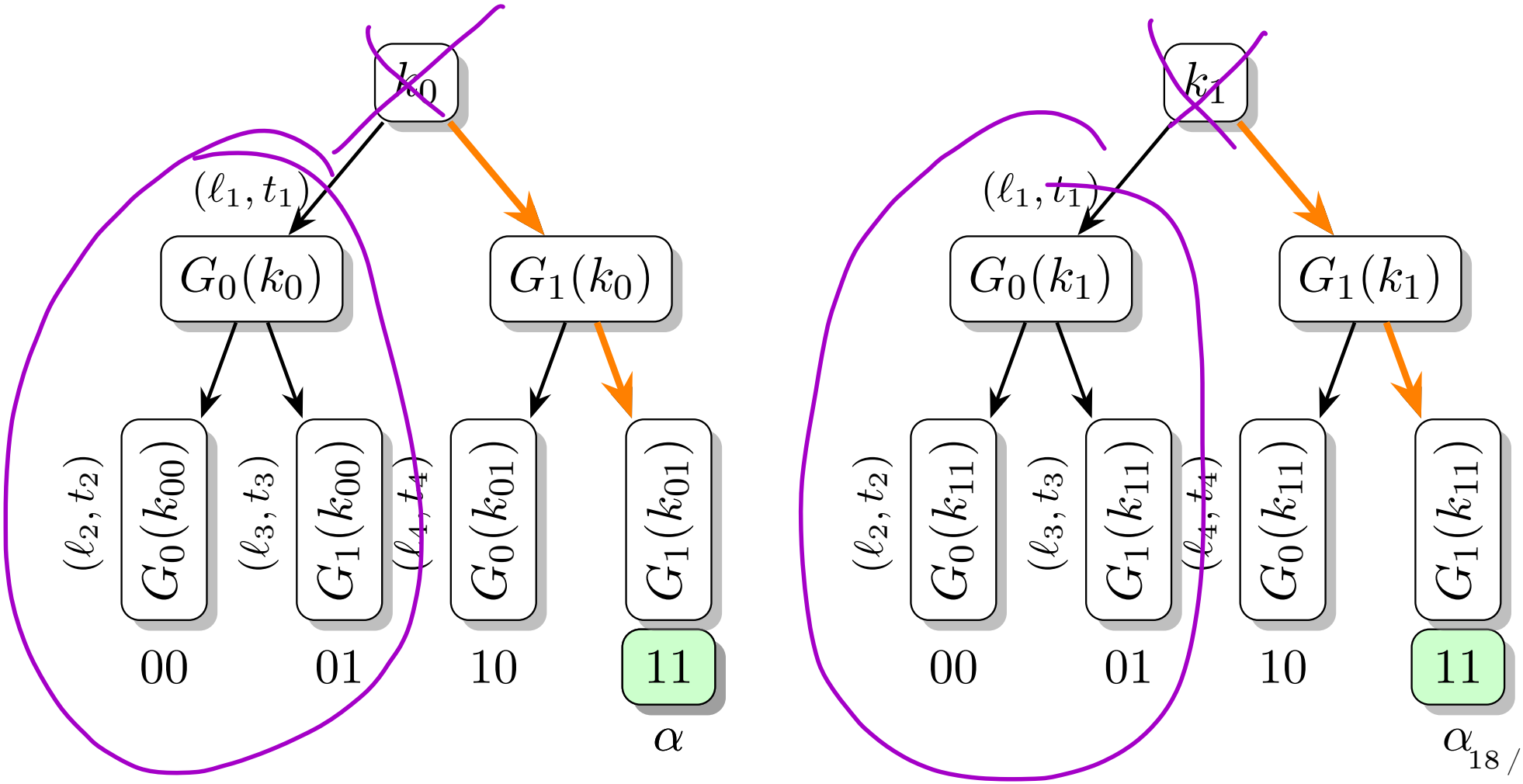
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



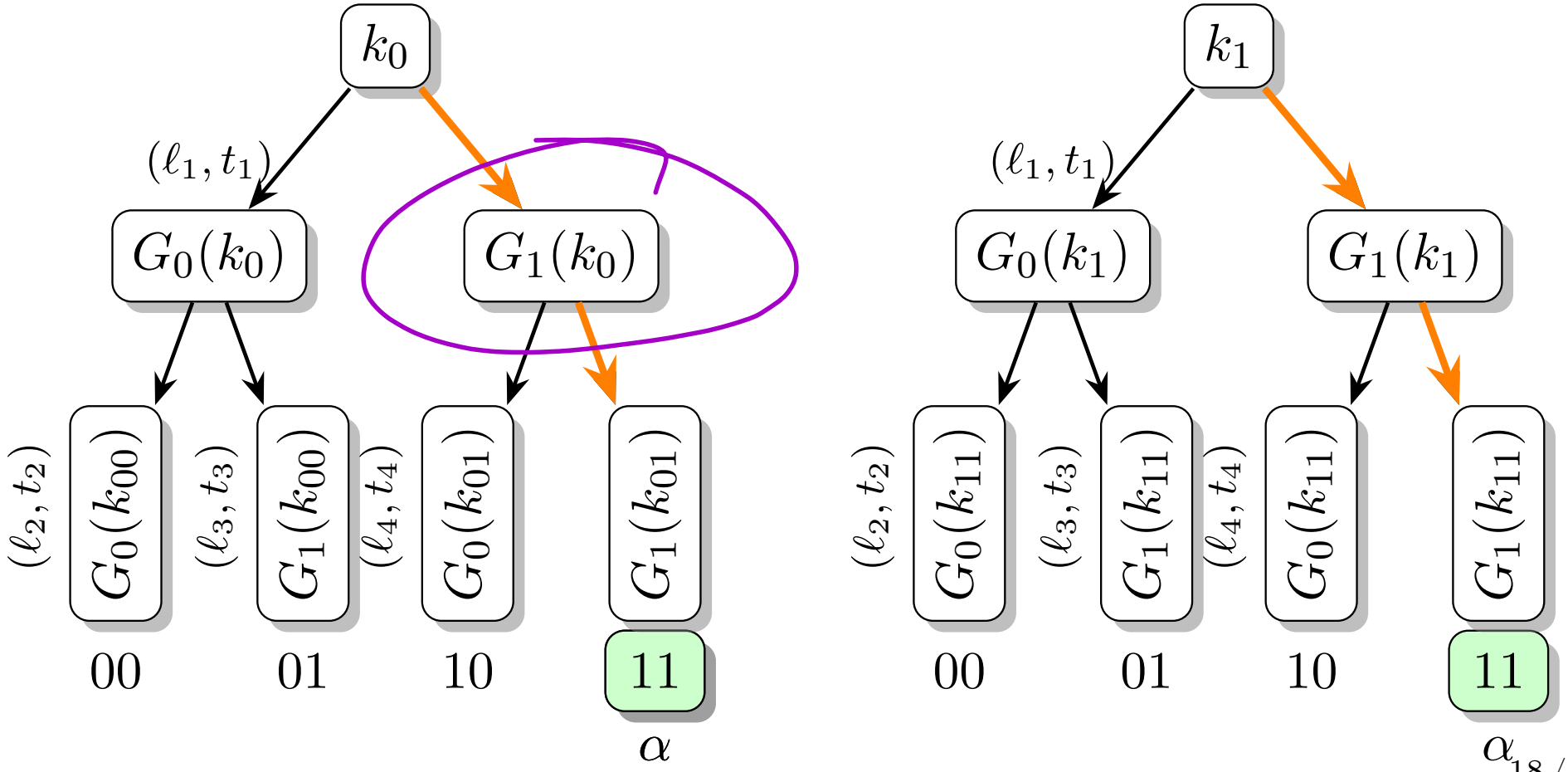
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.



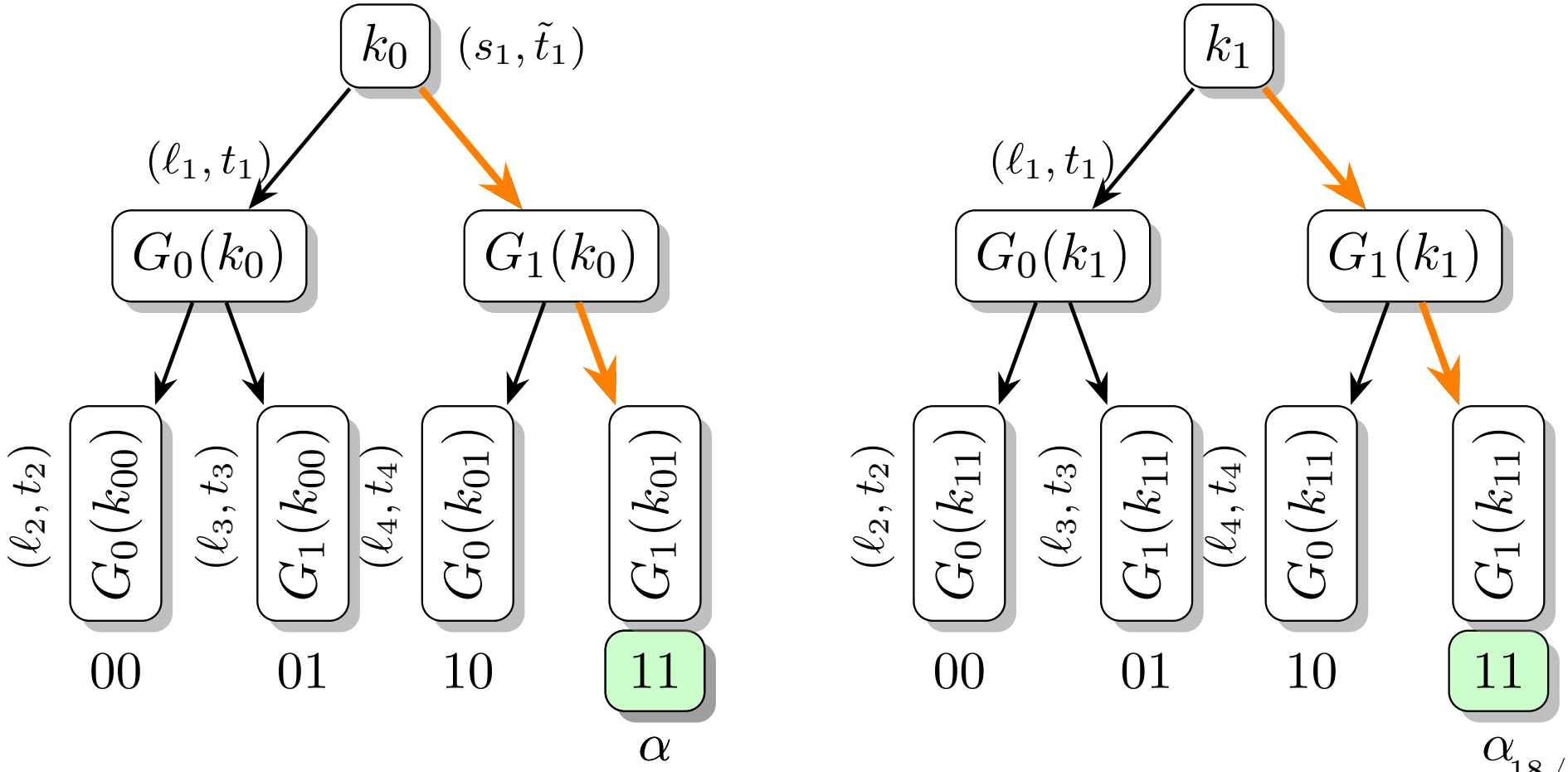
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.



DPFs from PRGs via GGM-like Construction

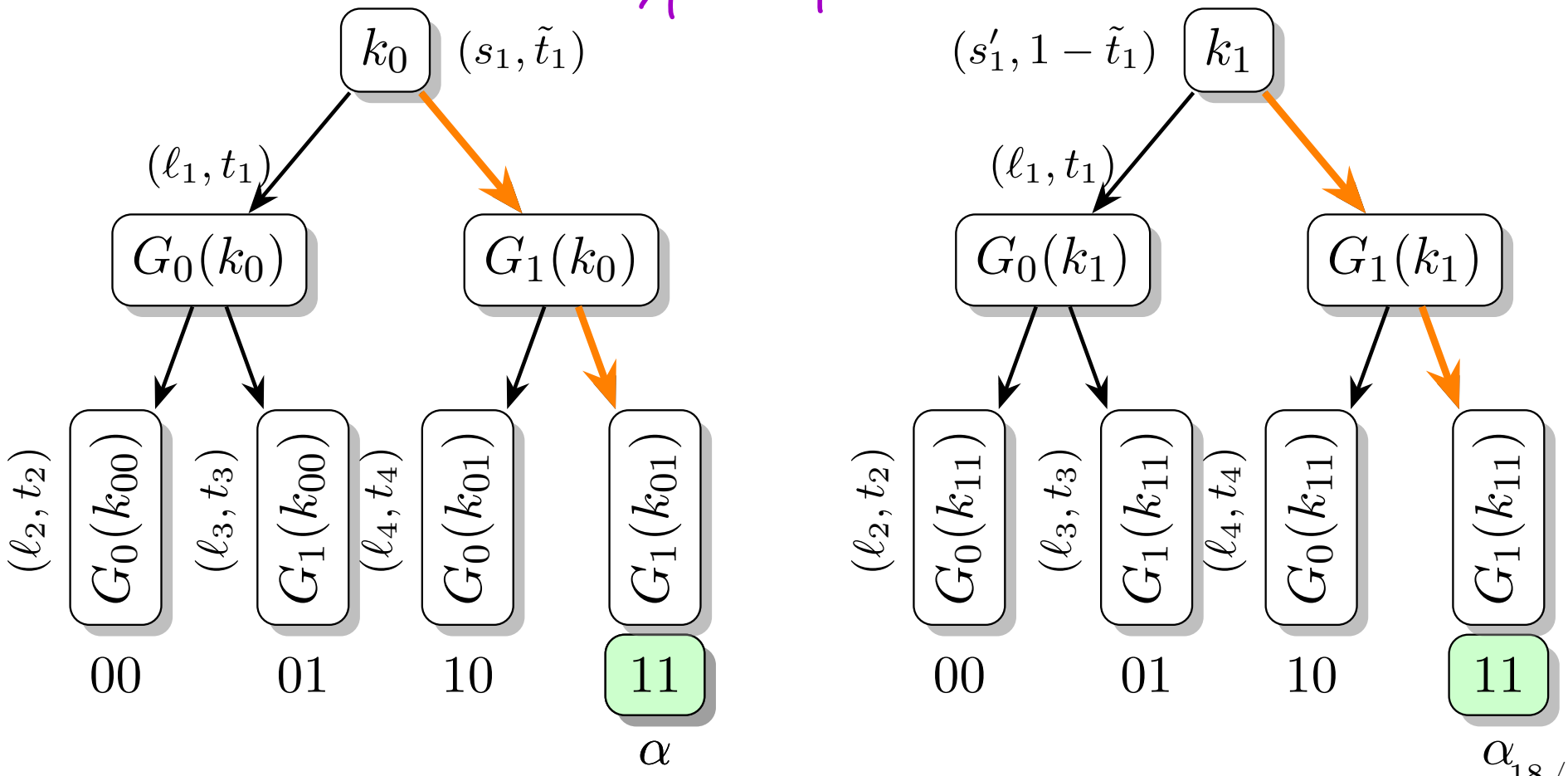
- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.



DPFs from PRGs via GGM-like Construction

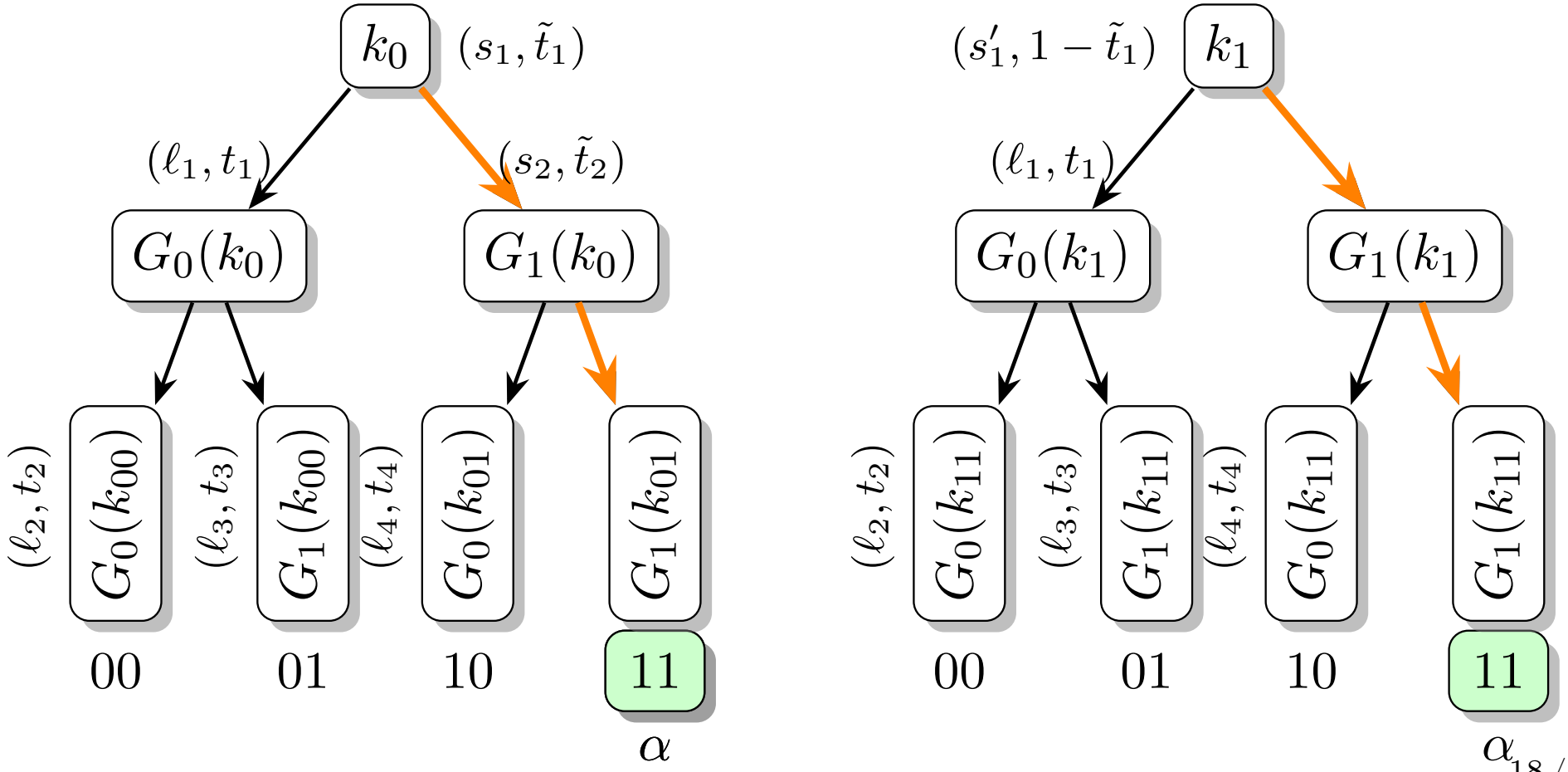
- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.

$s_1 \approx s_1'$



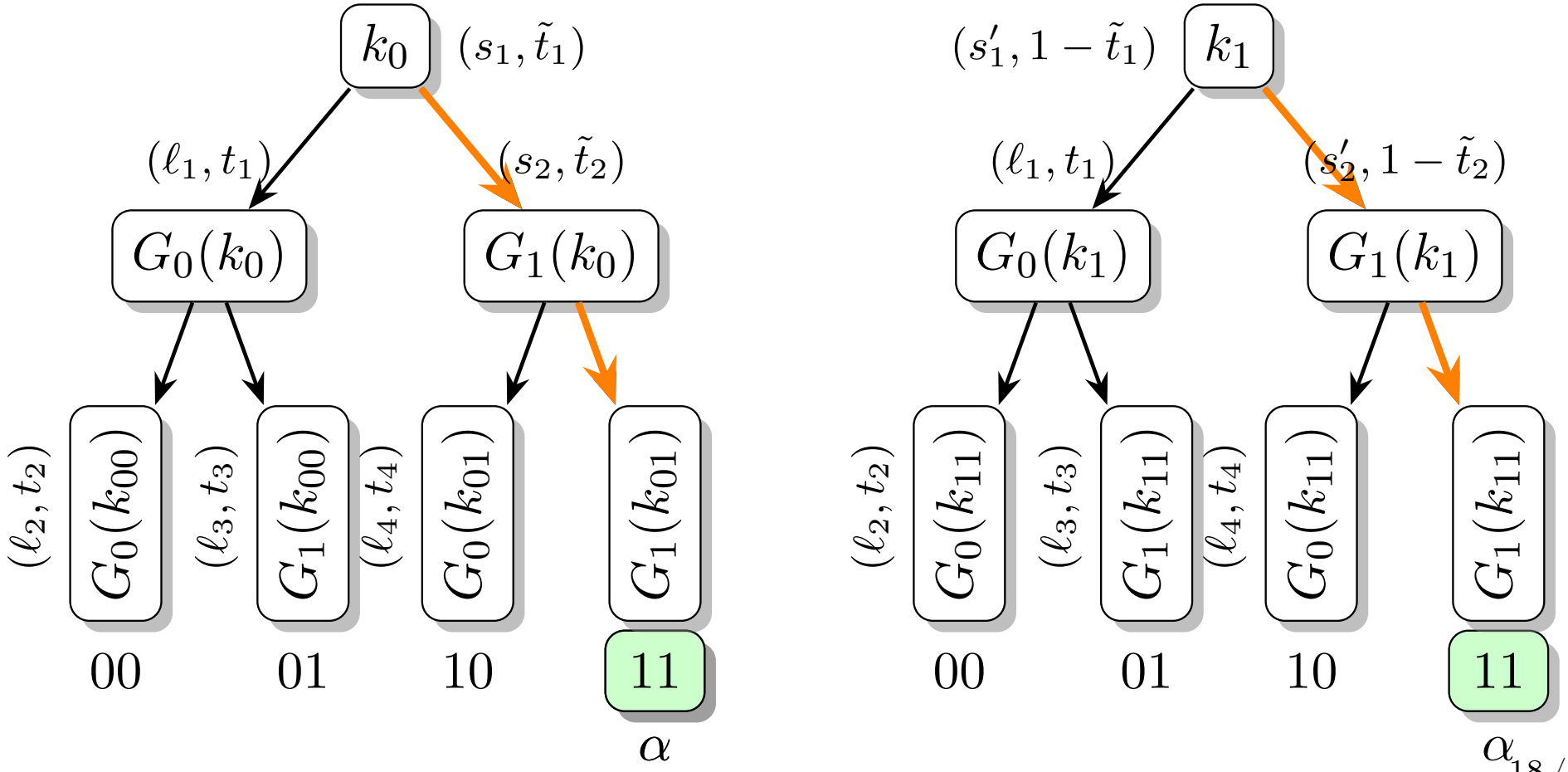
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.



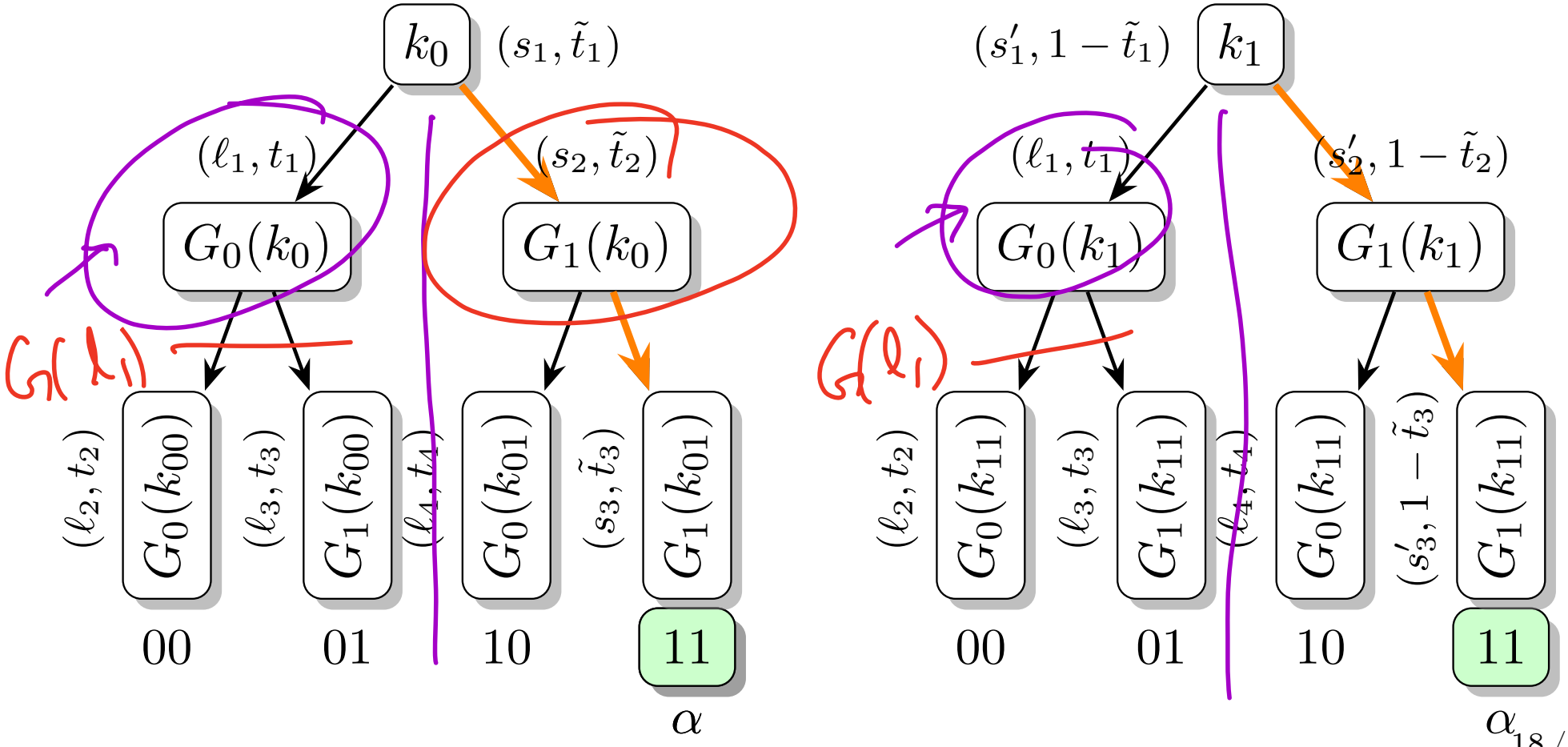
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.



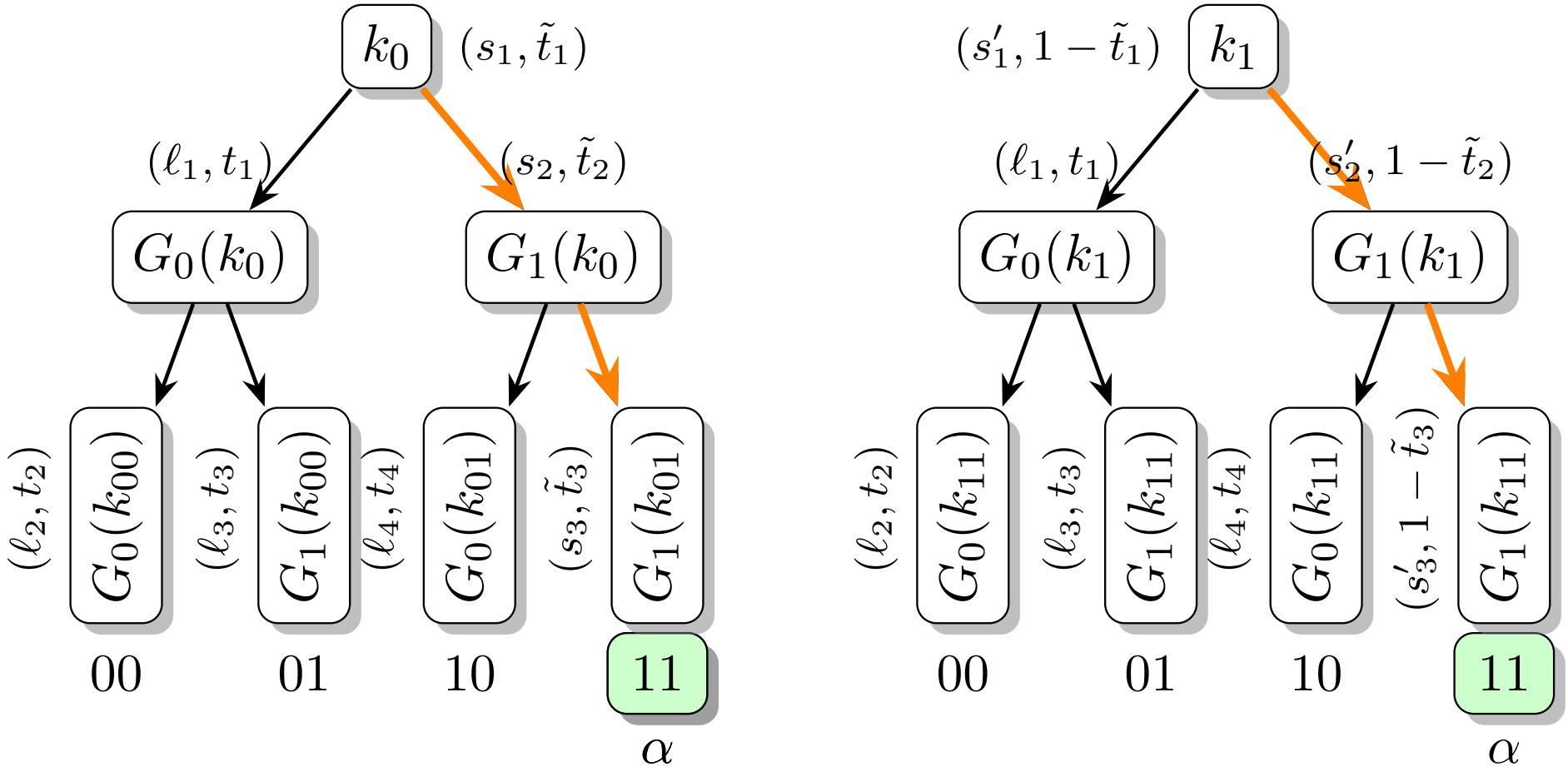
DPFs from PRGs via GGM-like Construction

- Need the following invariants:
  1. Each node *not* on the special path in each tree has the *same* label.
  2. Each node on the special path in each tree has different control bits *and* has seeds which are indistinguishable from being random and independent.



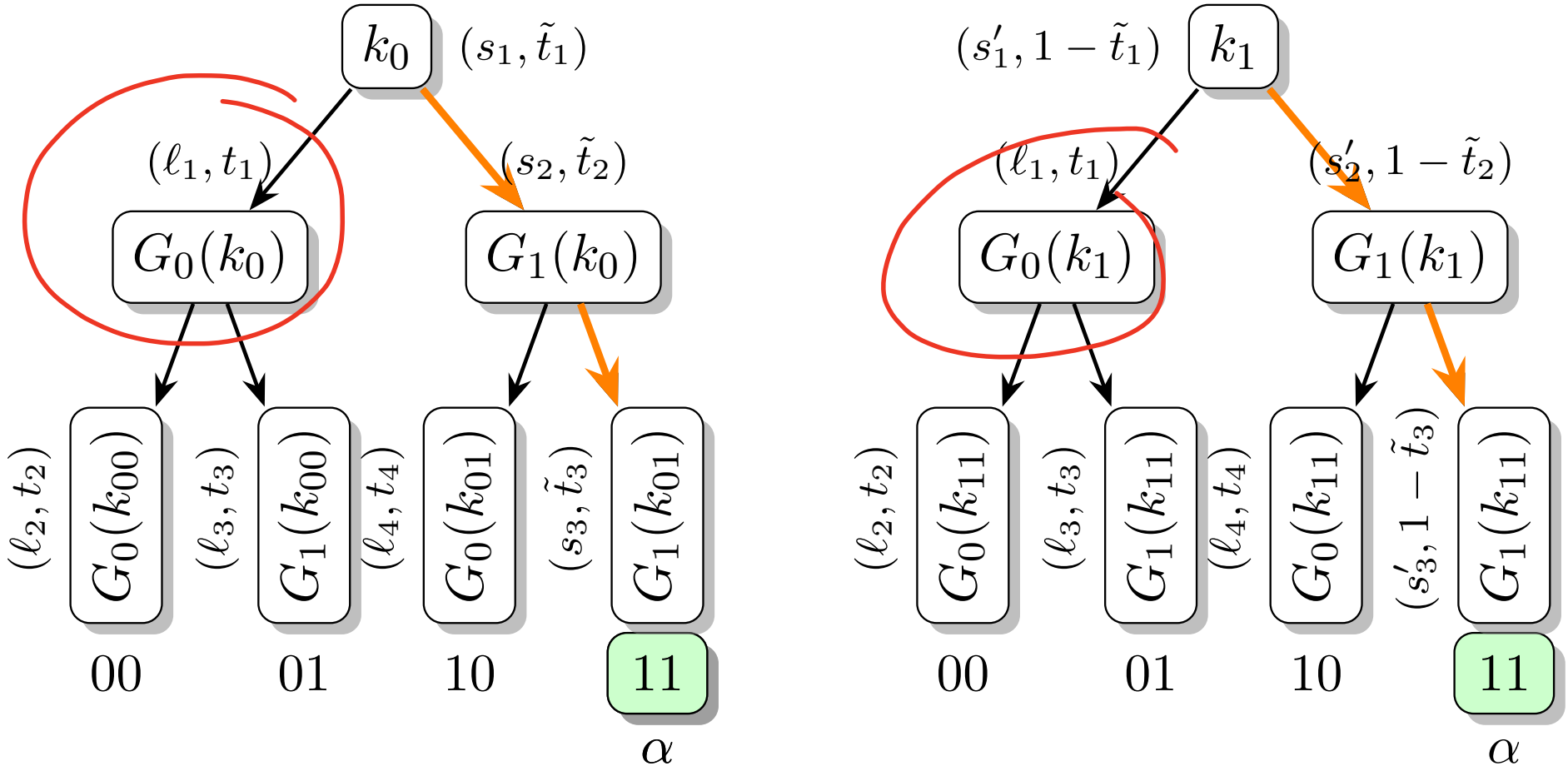
DPFs from PRGs via GGM-like Construction

- Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ be the PRG used in the previous slide.



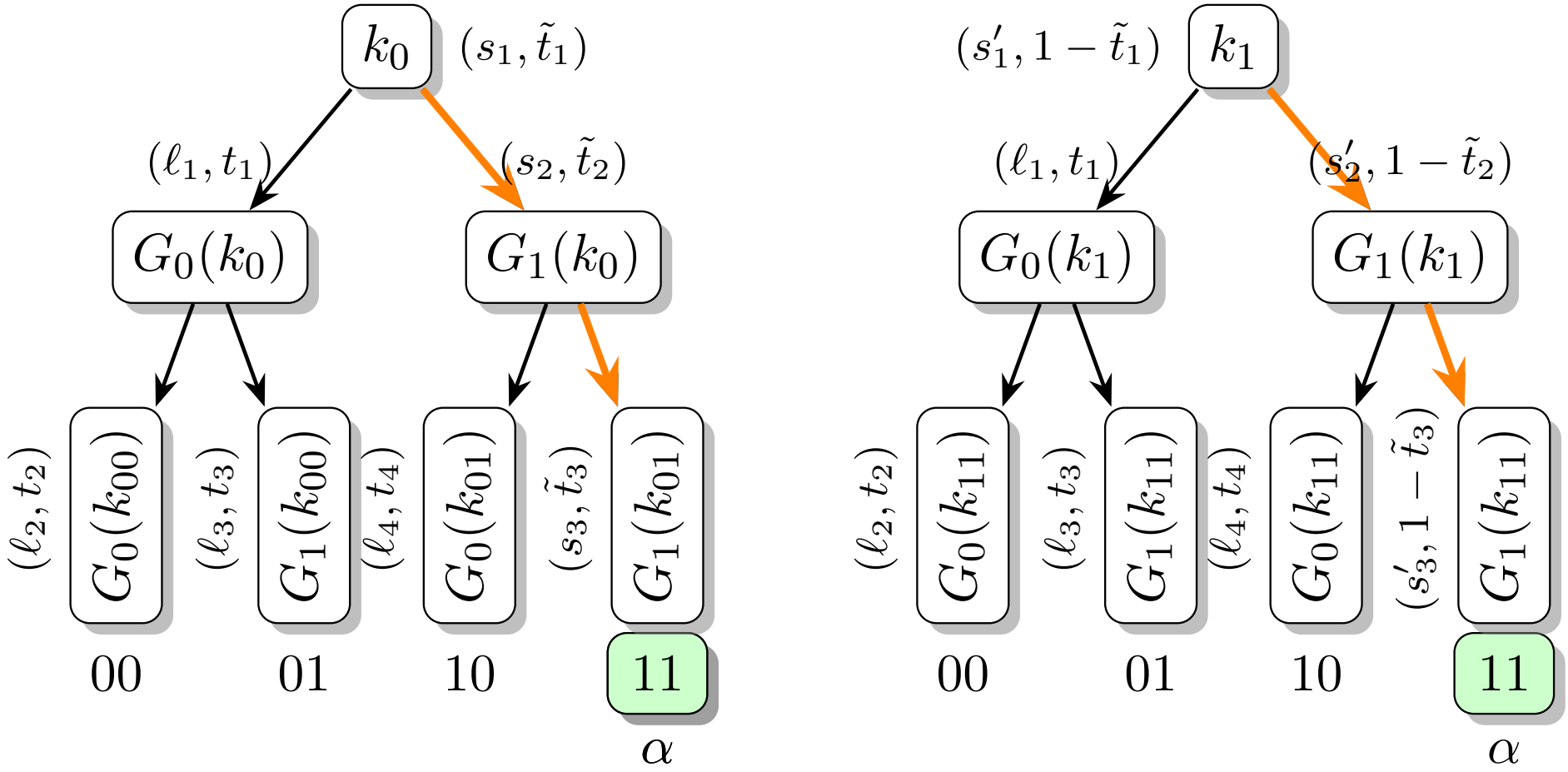
DPFs from PRGs via GGM-like Construction

- Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ be the PRG used in the previous slide.
- Not hard to see: if a node meets invariant (1), then all of its children meet this invariant (in both trees).



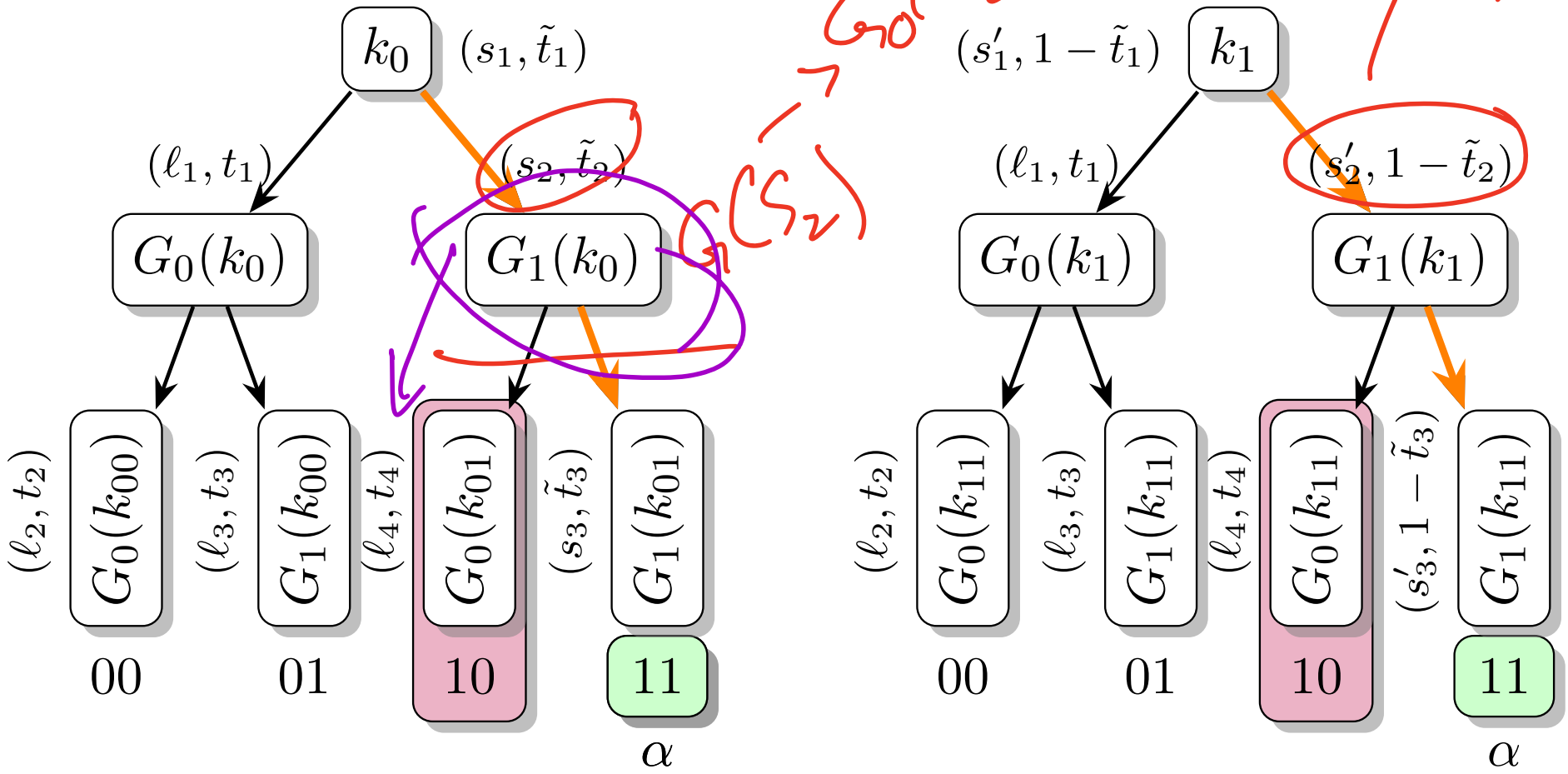
DPFs from PRGs via GGM-like Construction

- Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ be the PRG used in the previous slide.
- Not hard to see: if a node meets invariant (1), then all of its children meet this invariant (in both trees).
- Difficult case: *leaving* the special path.



DPFs from PRGs via GGM-like Construction

- Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ be the PRG used in the previous slide.
- Not hard to see: if a node meets invariant (1), then all of its children meet this invariant (in both trees).
- Difficult case: *leaving* the special path.



DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

- The construction uses the following simple ideas:

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

- The construction uses the following simple ideas:

  1. Weak homomorphism of additive secret sharing w.r.t. PRG G: if $s_0 \oplus s_1 = 0^\lambda$, then $G(s_0), G(s_1)$ extends each of these shares to longer additive shares of 0; that is, $G(s_0) \oplus G(s_1) = 0$.

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

- The construction uses the following simple ideas:

  1. Weak homomorphism of additive secret sharing w.r.t. PRG G: if $s_0 \oplus s_1 = 0$, then $G(s_0), G(s_1)$ extends each of these shares to longer additive shares of 0; that is, $G(s_0) \oplus G(s_1) = 0$.

  2. Additive homomorphism: given *public correction codeword* $CW \in \{0,1\}^\lambda$, shares $(s_0, s_1), (t_0, t_1)$, parties can locally compute shares of $s \oplus (t \cdot CW)$.

$t \in \{0,1\}$ $CW \cdot t = \begin{cases} CW & t=1 \\ 0 & t=0 \end{cases}$

$(s_0 \oplus (t_0 \cdot CW)) \oplus (s_1 \oplus (t_1 \cdot CW)) = s \oplus \underbrace{(t_0 \oplus t_1)}_{t} \cdot CW$

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

- The construction uses the following simple ideas:

  1. Weak homomorphism of additive secret sharing w.r.t. PRG G: if $s_0 \oplus s_1 = 0$, then $G(s_0), G(s_1)$ extends each of these shares to longer additive shares of 0; that is, $G(s_0) \oplus G(s_1) = 0$.

  2. Additive homomorphism: given *public correction codeword* $CW \in \{0,1\}^\lambda$, shares $(s_0, s_1), (t_0, t_1)$, parties can locally compute shares of $s \oplus (t \cdot CW)$.
     - Party $i \in \{0, 1\}$ computes $s_i \oplus (t_i \cdot CW)$

DPFs from PRGs via GGM-like Construction

- Now, view all labels of nodes in the tree as mod-2 additive secret shares (shared between trees).
  - E.g., for label (s, t), we view it as (s_0, s_1, t_0, t_1) such that $s = s_0 \oplus s_1$ and $t = t_0 \oplus t_1$.

- The construction uses the following simple ideas:

  1. Weak homomorphism of additive secret sharing w.r.t. PRG G: if $s_0 \oplus s_1 = 0$, then $G(s_0), G(s_1)$ extends each of these shares to longer additive shares of 0; that is, $G(s_0) \oplus G(s_1) = 0$.

  2. Additive homomorphism: given *public correction codeword* $CW \in \{0,1\}^\lambda$, shares $(s_0, s_1), (t_0, t_1)$, parties can locally compute shares of $s \oplus (t \cdot CW)$.
     - Party $i \in \{0, 1\}$ computes $s_i \oplus (t_i \cdot CW)$
     - This is called *conditional correction* (conditioned on $t = 1$)

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path.

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path. Let $(s_0^{(i)}, s_1^{(i)}), (t_0^{(i)}, t_1^{(i)})$ be the additive shares of its label.

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path. Let $(s_0^{(i)}, s_1^{(i)}), (t_0^{(i)}, t_1^{(i)})$ be the additive shares of its label.

- Need to compute the label of node v_{i+1} on the evaluation path.

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path. Let $(s_0^{(i)}, s_1^{(i)}), (t_0^{(i)}, t_1^{(i)})$ be the additive shares of its label.

- Need to compute the label of node v_{i+1} on the evaluation path.
  - Party $b \in \{0,1\}$ locally computes $S_b^{(i+1)} = G(s_b^{(i)})$, parsing as $(s_0^{(i+1)}, t_0^{(i+1)}, s_1^{(i+1)}, t_1^{(i+1)})$.

DPFs from PRGs via GGM-like Construction

- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path. Let $(s_0^{(i)}, s_1^{(i)}), (t_0^{(i)}, t_1^{(i)})$ be the additive shares of its label.

- Need to compute the label of node v_{i+1} on the evaluation path.
  - Party $b \in \{0, 1\}$ locally computes $S_b^{(i+1)} = G(s_b^{(i)})$, parsing as $(s_0^{(i+1)}, t_0^{(i+1)}, s_1^{(i+1)}, t_1^{(i+1)})$.

- Each level i will include a correction codeword $CW^{(i)}$.

DPFs from PRGs via GGM-like Construction

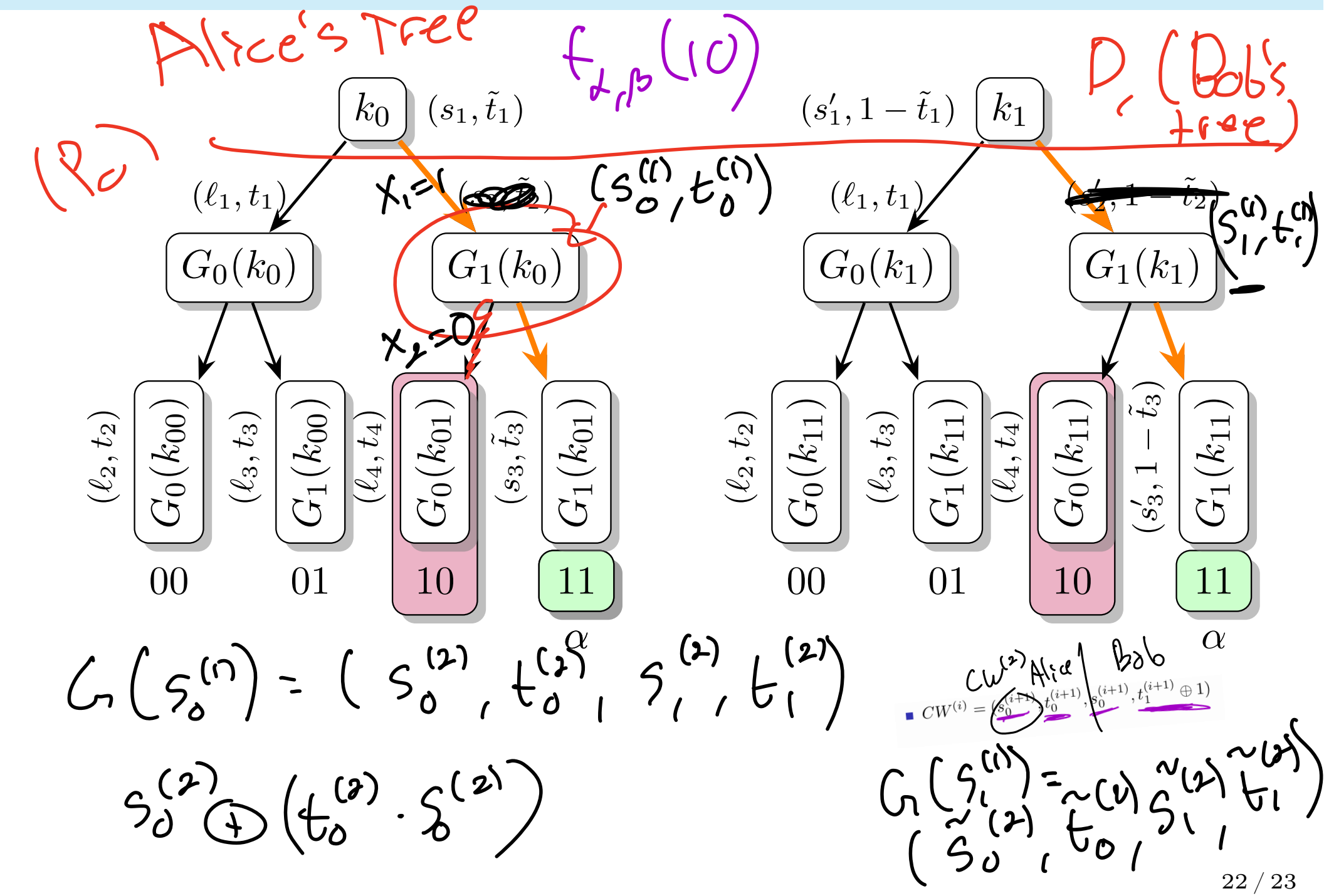
- We use these homomorphisms to maintain invariants 1 (when leaving from the special path) and (2).

- Let v_i be the i^{th} node on the special evaluation path. Let $(s_0^{(i)}, s_1^{(i)}), (t_0^{(i)}, t_1^{(i)})$ be the additive shares of its label.

- Need to compute the label of node v_{i+1} on the evaluation path.
  - Party $b \in \{0,1\}$ locally computes $S_b^{(i+1)} = G(s_b^{(i)})$, parsing as $(s_0^{(i+1)}, t_0^{(i+1)}, s_1^{(i+1)}, t_1^{(i+1)})$.

- Each level i will include a correction codeword $CW^{(i)}$.
  - $CW^{(i)} = (s_0^{(i+1)}, t_0^{(i+1)}, s_0^{(i+1)}, t_1^{(i+1)} \oplus 1)$

DPFs from PRGs via GGM-like Construction



$x_2 = 0$

$G(s_0^{(1)}) = (s_0^{(2)}, t_0^{(2)}, s_1^{(2)}, t_1^{(2)})$

$s_0^{(2)} \oplus (t_0^{(2)} \cdot s_0^{(2)})$

$\rightarrow$

$t_0^{(2)} = 0$

$\rightarrow s_0^{(2)}$

Alice

$G(\tilde{s}_1^{(1)}) = (\tilde{s}_0^{(2)}, \tilde{t}_0^{(2)}, \tilde{s}_1^{(2)}, \tilde{t}_1^{(2)})$

$\tilde{s}_0^{(2)} \oplus (\tilde{t}_0^{(2)} \cdot s_0^{(2)})$

$\tilde{t}_0^{(2)} = 1$

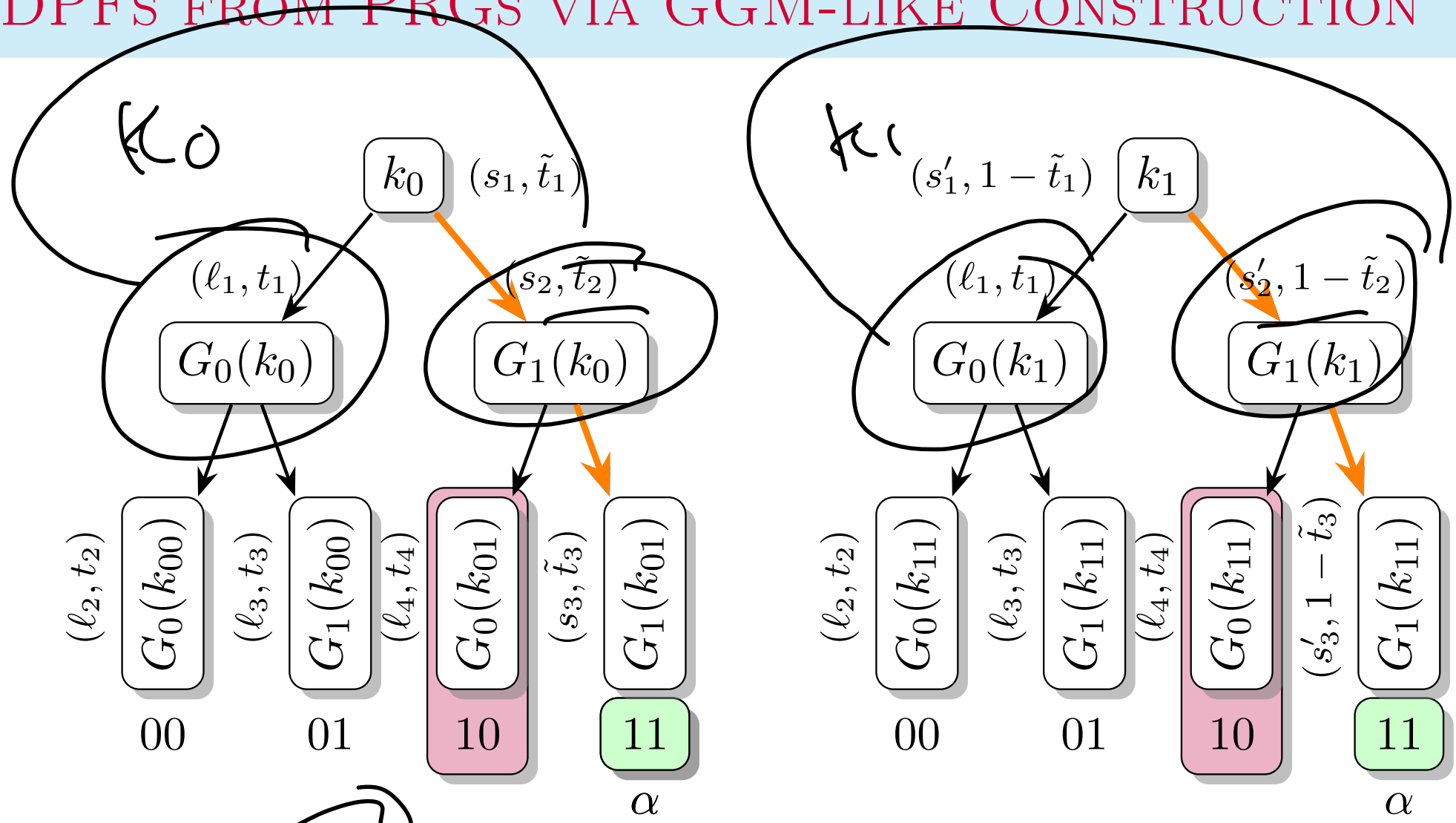
$\rightarrow s_0^{(2)}$

$CW^{(2)}$

Alice | Bob

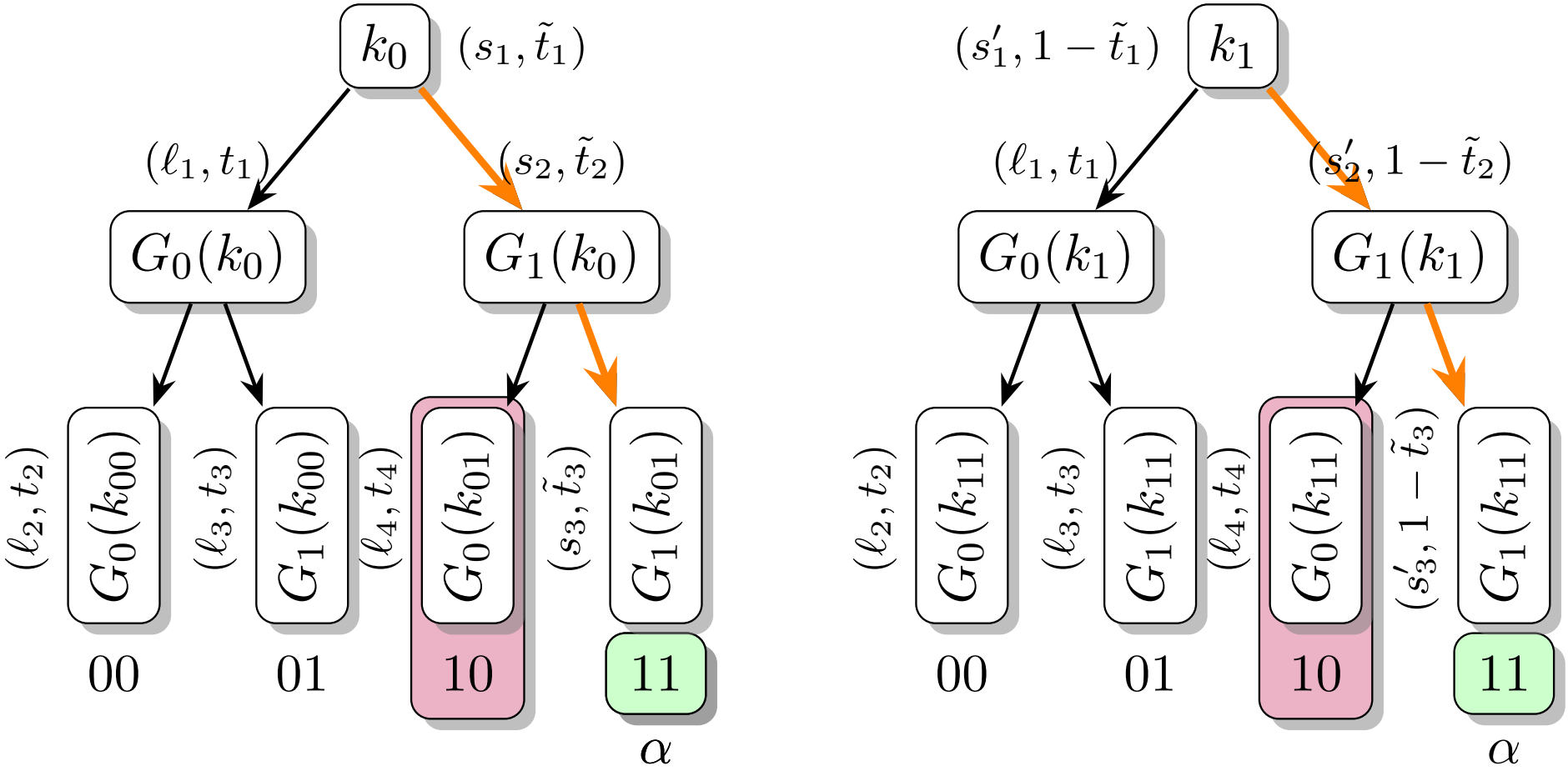
- $CW^{(i)} = (s_0^{(i+1)}, t_0^{(i+1)}, s_0^{(i+1)}, t_1^{(i+1)} \oplus 1)$

DPFs from PRGs via GGM-like Construction



- DPF keys: $CW^{(i)}$ included in both, (ℓ_1, t_1) included in both, labels of $G_1(k_0)$ and $G_1(k_1)$ are additively secret shared, each party gets one of the shares.

DPFs from PRGs via GGM-like Construction



- DPF keys: $CW^{(i)}$ included in both, (ℓ_1, t_1) included in both, labels of $G_1(k_0)$ and $G_1(k_1)$ are additively secret shared, each party gets one of the shares.
- Reconstruction: use $CW^{(i)}$ to compute the label of the leaf corresponding to α, output the (sum) of shares of t for this node.

Next Time: Zero-knowledge Proofs