# Motivation.

- You should have seen in your homework that PRGs become insecure if the seed is not chosen uniformly at random.

- Annoying!

- Would be much nicer if we had a "magic PRG" such that, even if we don't choose its seed uniformly, its output still "looks random" to any adversary.

- Here's a candidate construction for such an object. Let's say we want a "magic function" from m bits to n bits.

Just choose a uniformly random function from the set of all functions mapping m bits to n bits.

Equivalent to making a table:

| $x$ | $F(x)$ |
|---|---|
| $0000\cdots0$ | $r_0$ |
| $0000\cdots1$ | $r_1$ |

$\vdots$

$1\,1 \cdots 1$ | $r_{2^m}$

choose these uniformly from $\{0,1\}^n$.

- No matter which "seed" $x$ an adversary picks, this function's output is perfectly random! (over the choice of function)

- What's the problem with this?

---

## PRF security.

- OK, so we can't hope for such a "magic function".

- But maybe we <u>can</u> get a function that is computationally indistinguishable from such a function!

- More precisely: remember even our exponentially complicated "magic function" was only random looking to an adversary over the choice of function. There is nothing random looking about a fixed function!

- We want to compare apples to apples here, so we can only require our "computationally random-looking function" to also be random-looking over the choice of function.

- Formally: let's define PRF security in terms of a game between a challenger and an adversary.
  Basically: we want the PPT adversary to be unable to tell the difference between a function picked randomly from a small set $\mathcal{F}_{small}$ (which only contains functions we can efficiently evaluate) and a function picked randomly from the set of all possible $\{0,1\}^m \to \{0,1\}^n$ functions, $\mathcal{F}_{all}$.

$\mathcal{F}_{small}$ $\qquad$ $\mathcal{F}_{all}$

$\mathcal{C}$ $\qquad\qquad\qquad\qquad$ $\mathcal{A}$

$f \xleftarrow{R} \begin{pmatrix} \mathcal{F}_{small} \text{ OR} \\ \mathcal{F}_{all} \end{pmatrix}$ (prob. $\frac{1}{2}$ each)

note: $\mathcal{A}$ never knows what $f$ is! Only gets to query it

$\xleftarrow{\quad x_1 \quad}$

$\xrightarrow{\quad f(x_1) \quad}$

$\vdots$

$x$ — some number = poly(n)

$$\xleftarrow{\quad x_q \quad}$$

$$\xrightarrow{\quad f(x_q) \quad}$$

$$\xleftarrow{\quad \mathcal{F}_{small} \text{ or } \mathcal{F}_{all} \ ? \quad}$$

We say $\mathcal{F}_{small}$ is a secure PRF family if no PPT adversary can win this game with probability better than $\frac{1}{2} + \mathrm{negl}(m)$.

ok, really $\mathrm{negl}(\lambda)$ for your security parameter $\lambda$ — we assume $m(\lambda), n(\lambda) = \mathrm{poly}(\lambda)$ for this lecture

**Remark.** Notice that the challenger's behaviour when it chooses $\mathcal{F}_{all}$ is identical to just returning uniformly random $n$-bit strings in response to all the adversary's queries. This is conceptually useful to keep in mind.

but keeping track of duplicates!

**Remark 2.** For notational convenience, we like to give the functions in $\mathcal{F}_{small}$ names. So we label each function in $\mathcal{F}_{small}$ with a "key" $k$, and we may write

secret! ("A never knows what $f$ is")

$$\mathcal{F}_{small} = \{ f_k : \{0,1\}^m \to \{0,1\}^n \}_{k \in \mathcal{K}},$$

where $\mathcal{K}$ is some set of keys.

**Remark 3.** We can wlog assume the adversary never makes the same query twice. (Why?)

---

## Constructing PRFs from PRGs.

- The canonical construction is the "GGM" construction (Goldreich - Goldwasser - Micali).

- Highly sequential (not super fast), but conceptually beautiful!

- Assumption: there exists a length-doubling PRG

  $G : \{0,1\}^n \to \{0,1\}^{2n}$

  same $n$ as in the output length of the PRF.

  Notation: $G(s) := \underbrace{G_0(s)}_{n \text{ bits}} \,\|\, \underbrace{G_1(s)}_{n \text{ bits}}$

- The construction:

  Secret key $k$ = seed $s$ for the PRG.
  (defines a different function in $\mathcal{F}_{small}$ for each different seed)

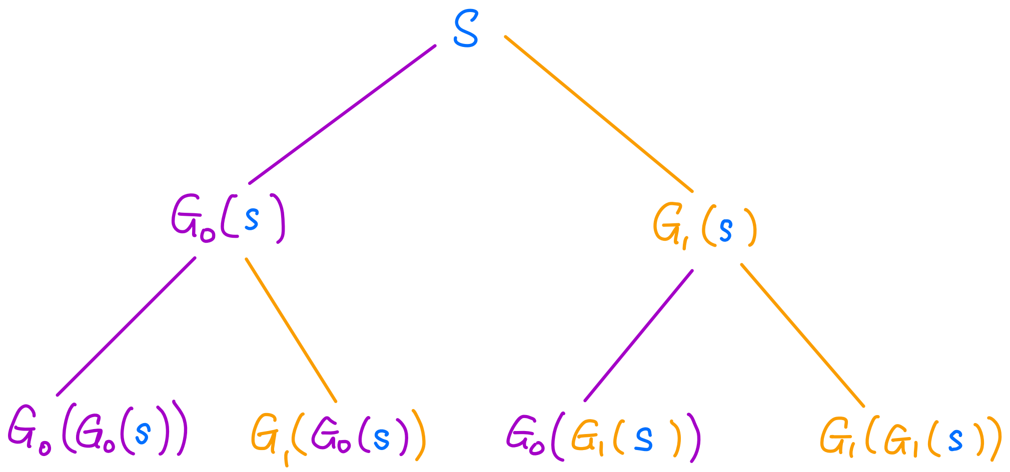
  For a fixed seed $s$ :

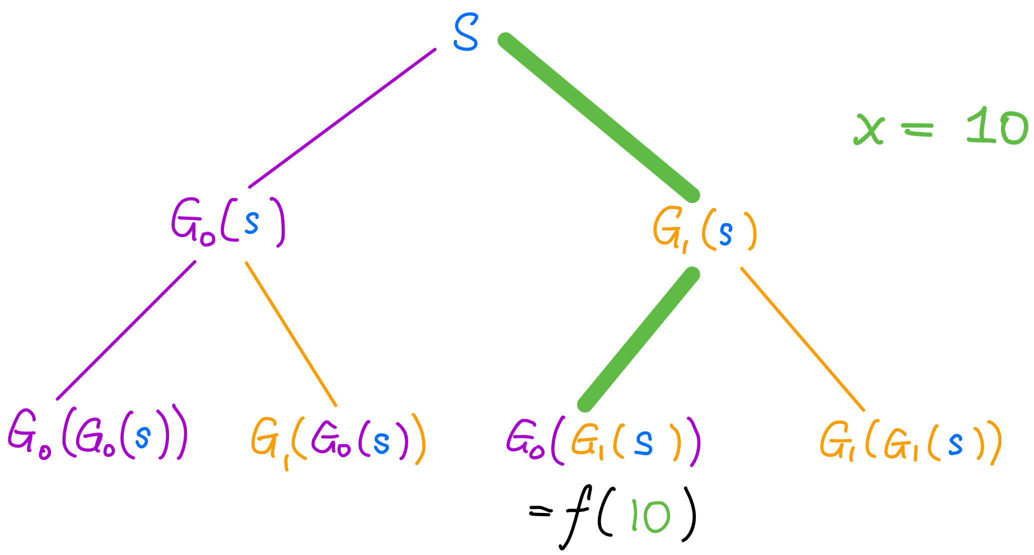
$$f_s(x) := G_{x_m} \circ \cdots \circ G_{x_2} \circ G_{x_1}(s)$$

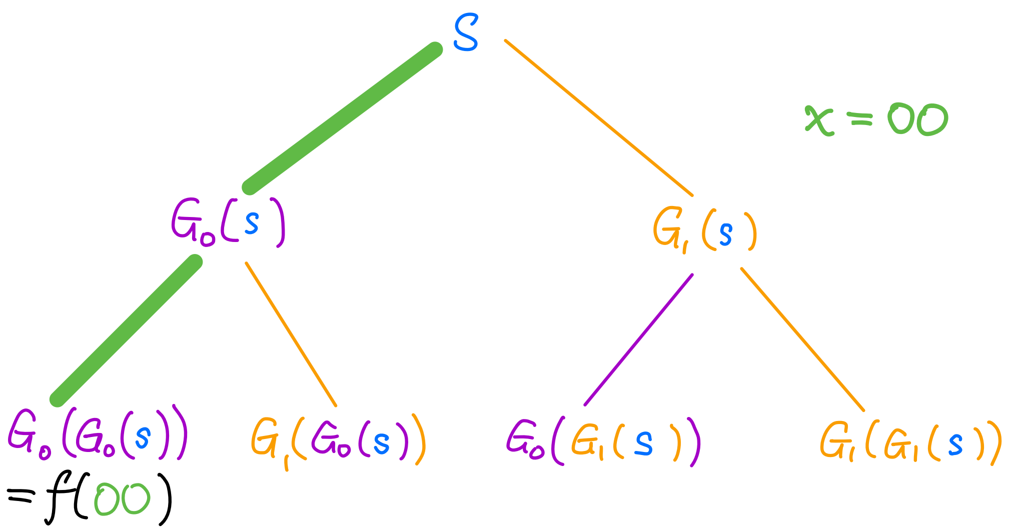
m bits

read this way.

Picture! $(m=2)$



$x$ picks out a path in the tree.

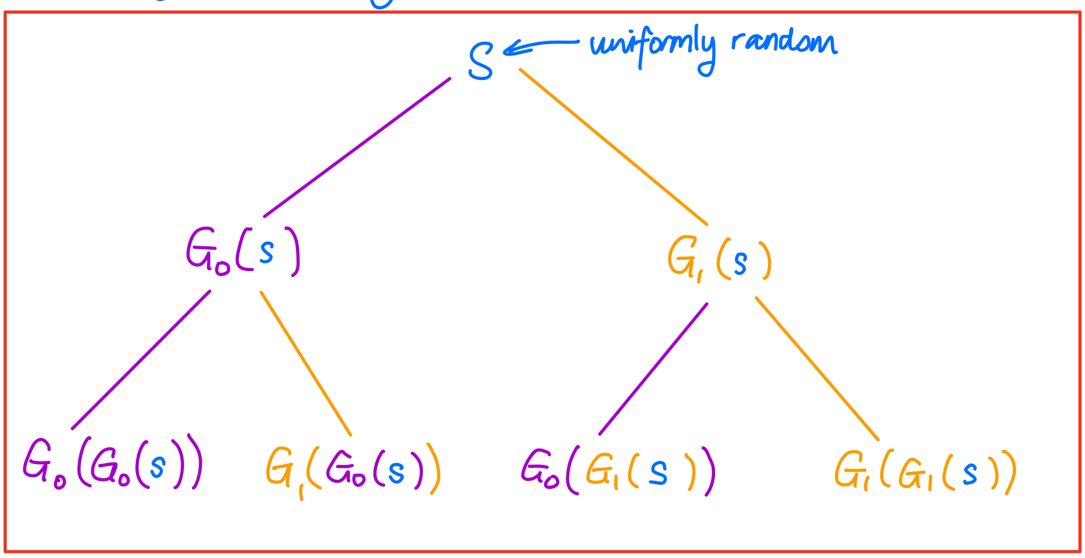
---

## Security.

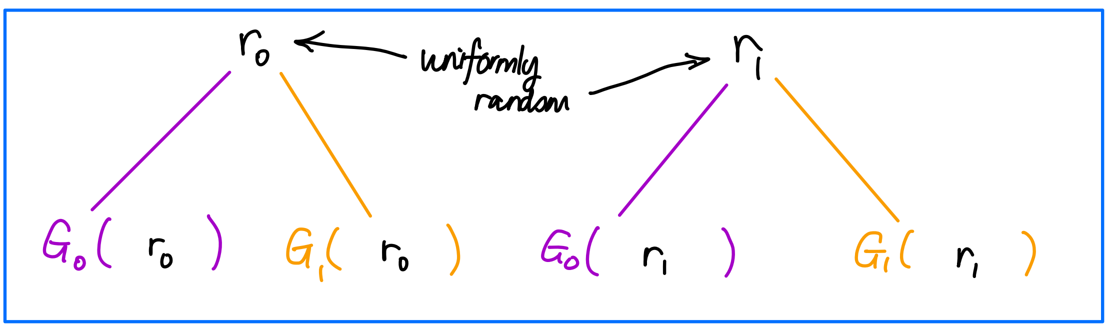
- We want to reduce to PRG security: we want to construct an adversary $\mathcal{B}$ who uses $\mathcal{A}$ to break $G$.

  PRF adversary

- Hybrid argument. Show that




$$\approx_c$$




Formal interpretation:

If $\mathcal{B}$ answers $\mathcal{A}$'s queries as if $f$ were defined according to TREE 1,

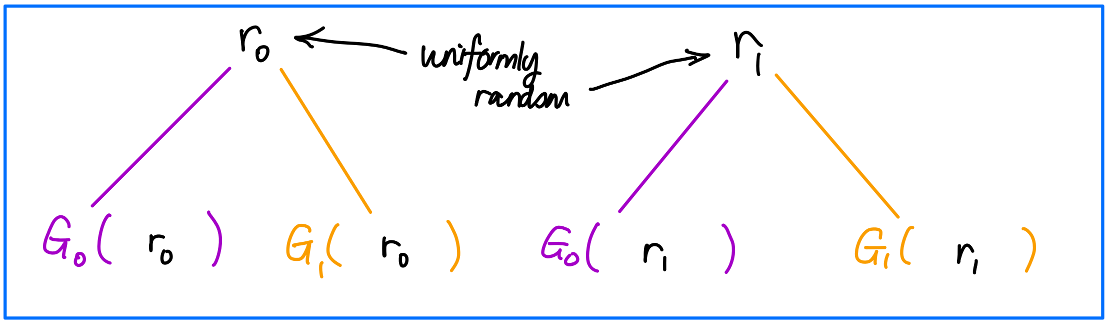
PPT $\mathcal{A}$ cannot tell the difference between that and

If $\mathcal{B}$ answers $\mathcal{A}$'s queries as if $f$ were defined according to TREE 2

i.e., in Tree 2, $f_s(x) := G_{x_m} \circ \cdots \circ G_{x_2}(r_{x_1})$.

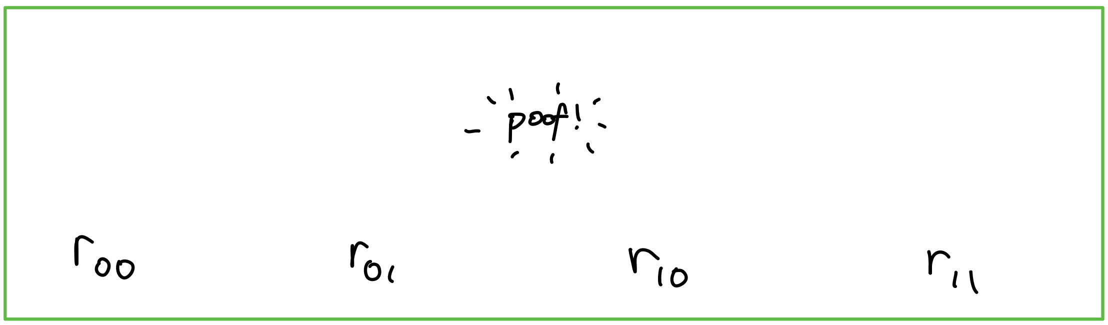
Then we just keep doing this.

In Tree 3, $f_s(x) := G_{x_m} \circ \cdots \circ G_{x_3}(r_{x_1 x_2})$




TREE 2

$$\approx_c$$




TREE 3

Wait a minute, TREE 3 would just be $\mathcal{B}$ answering all of $\mathcal{A}$'s queries with uniformly random strings!

↳ That's exactly the $\mathcal{F}_{all}$ case (and TREE 1 was the $\mathcal{F}_{small}$ case).

So these hybrids bridge the gap between $\mathcal{F}_{small}$ and $\mathcal{F}_{all}$, as expected.

As usual, we assume that, if $\mathcal{A}$ can tell the difference between $\mathcal{F}^{\alpha}_{\text{small}}$ (TREE 1) and $\mathcal{F}^{\alpha}_{\text{all}}$ (TREE 3), then it can tell the difference between ONE OF

TREE 1 and TREE 2

OR

TREE 2 and TREE 3.

## The one-query case.

- For simplicity, let's do the case first where $\mathcal{A}$ (PRF adversary) only makes one query.

- How to reduce to PRG security? $\mathcal{B}$ (PRG adv.) should somehow plant its challenge when it's answering $\mathcal{A}$'s queries.

- Let's say $\mathcal{B}$ gets a challenge $c_\alpha$.

$$c_\alpha = \begin{cases} G_0(s) \,\|\, G_1(s) & \alpha = 0 \\ r_{2n} & \alpha = 1 \end{cases}$$

- $\mathcal{B}$ pretends to be $\mathcal{A}$'s challenger:

$\mathcal{C}$ (PRG challenger) $\qquad\qquad \mathcal{B} \qquad\qquad \mathcal{A}$

$c_\alpha$

single query

$x$

??

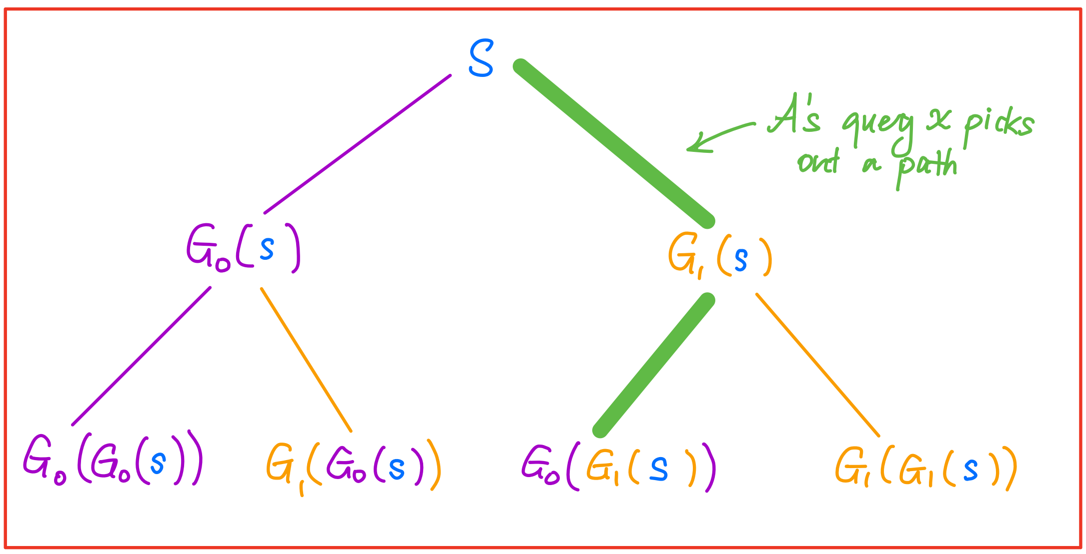
$\mathcal{A}$ is supposed to tell the difference between these

TREE 1 or TREE 2 ?

(we'll show that the other case is basically the same)

$\alpha = 0$ or $\alpha = 1$ ?

Let's try to fill in the ??: let's look at TREE 1.




TREE 1

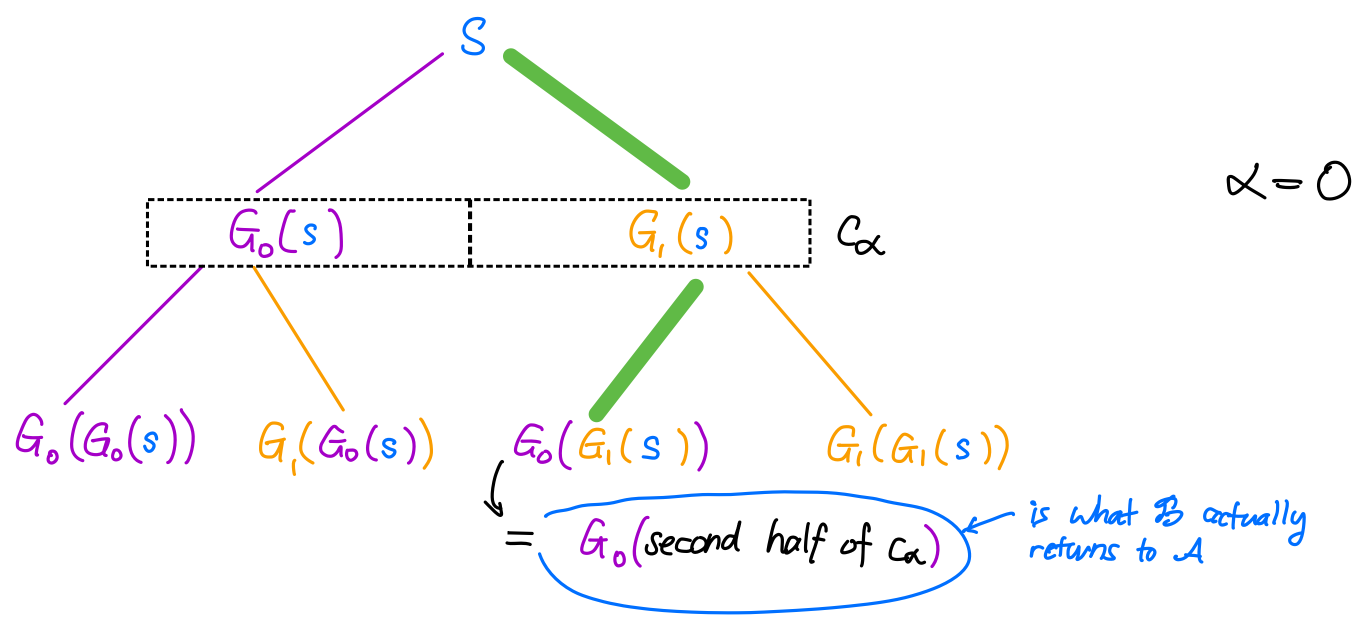
Remember $C_\alpha$ is of the form

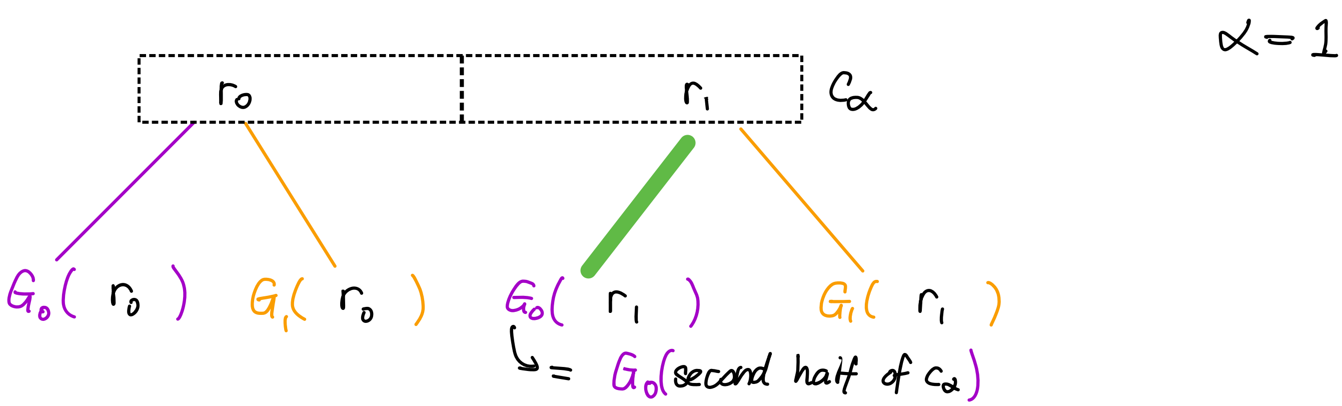
$$C_\alpha = \begin{cases} G_0(s) \,\|\, G_1(s) & \alpha = 0 \\ r_{2n} & \alpha = 1 \end{cases}$$

uniformly random

| | | |
|---|---|---|
| $\alpha = 0$ | $G_0(s)$ | $G_1(s)$ |
| $\alpha = 1$ | $r_0$ | $r_1$ |

So what if we had $\mathcal{B}$ answer $\mathcal{A}$'s query according to the following tree:

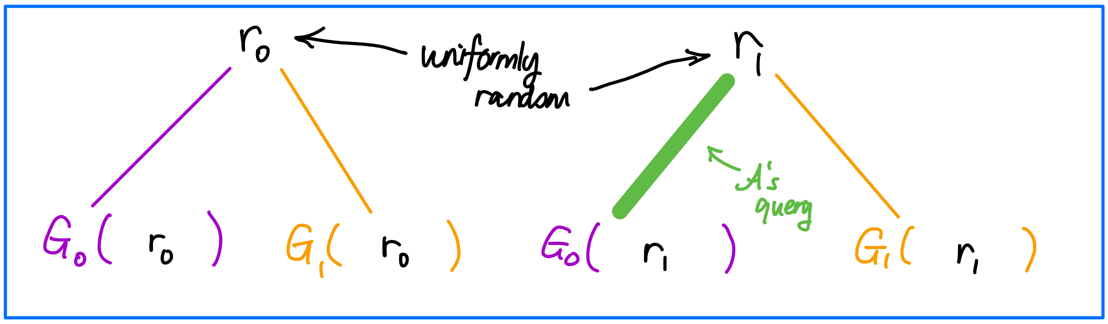



$\alpha = 0$ case looks exactly like TREE 1,
$\alpha = 1$ case looks exactly like TREE 2!

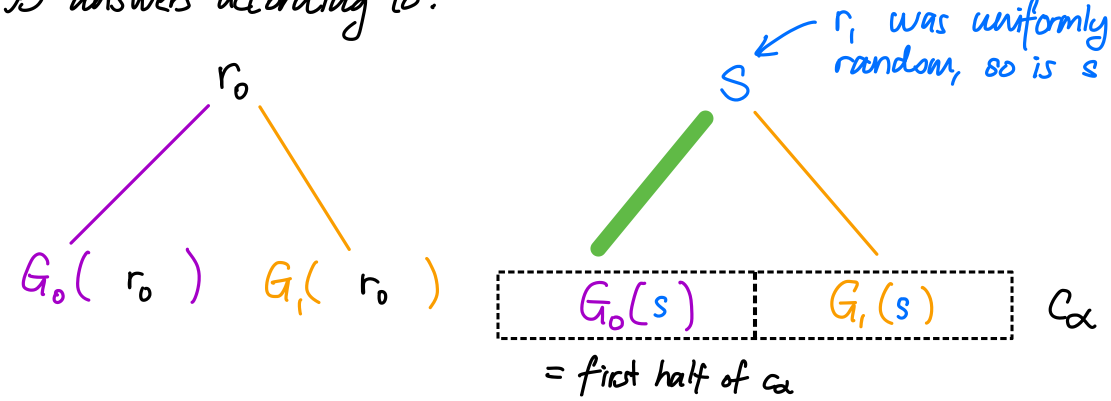
$\Rightarrow$ $\mathcal{B}$ has the same advantage as $\mathcal{A}$. (Karp reduction)

Similar for TREE 2 vs. TREE 3.



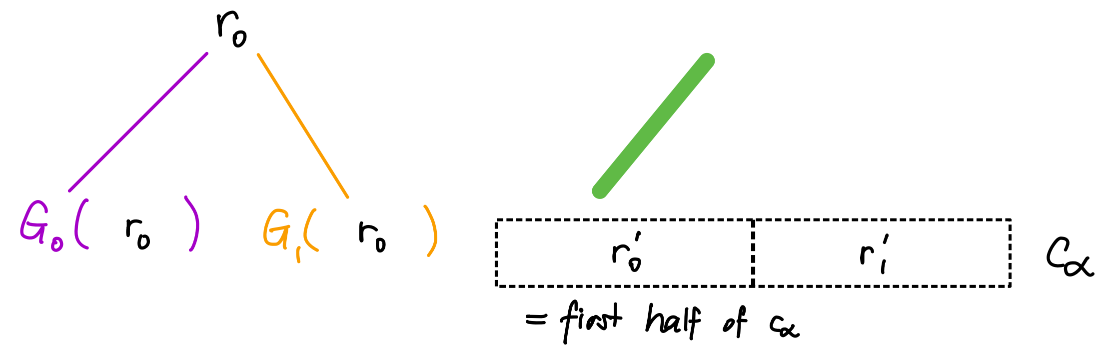

TREE 2

$\mathcal{B}$ answers according to:




$\alpha = 0$

identical to TREE 2




$\alpha = 1$

identical to TREE 3 *
(from A's point of view)

* Actually, only half of it looks like TREE 3 — but $\mathcal{A}$ never sees the other half anyway!

---

## Many queries.

- In order to handle many queries, $\mathcal{B}$ has to be playing a modified version of the PRG security game, where

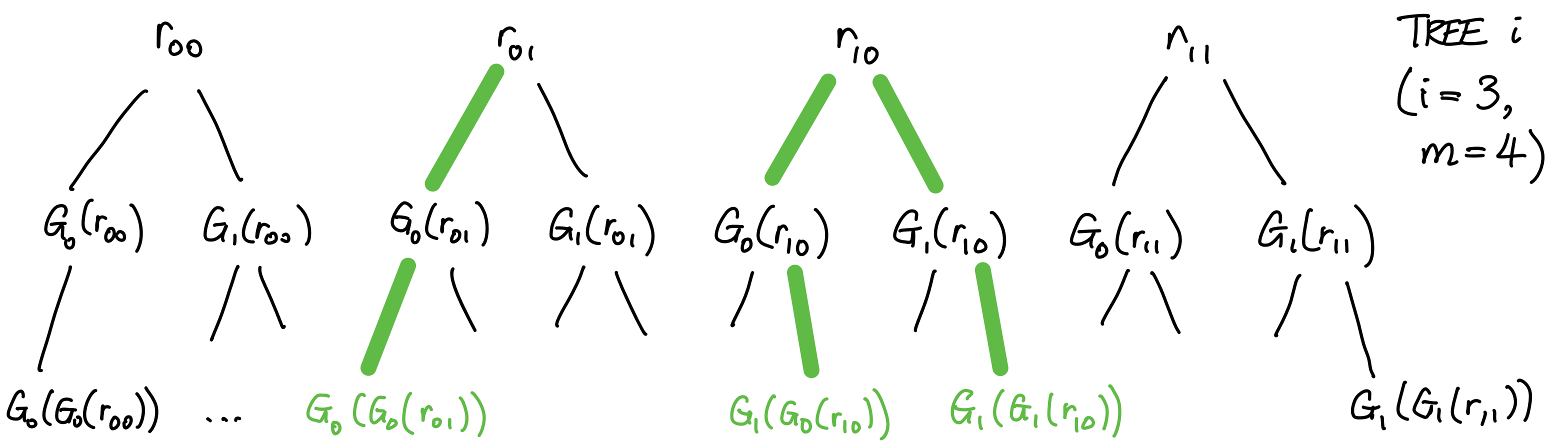
$$C_\alpha = \begin{cases} \underbrace{\overset{c^0_{\alpha 1}}{G_0(s_1)} \,\|\, \overset{c^1_{\alpha 1}}{G_1(s_1)}}_{C_{\alpha 1}}, \ \underbrace{G_0(s_2) \,\|\, G_1(s_2)}_{C_{\alpha 2}}, \ \ldots, \ G_0(s_q) \,\|\, G_1(s_q) & \alpha = 0 \\ r_{2n \cdot q} & \alpha = 1 \end{cases}$$

$q$: # queries $A$ makes

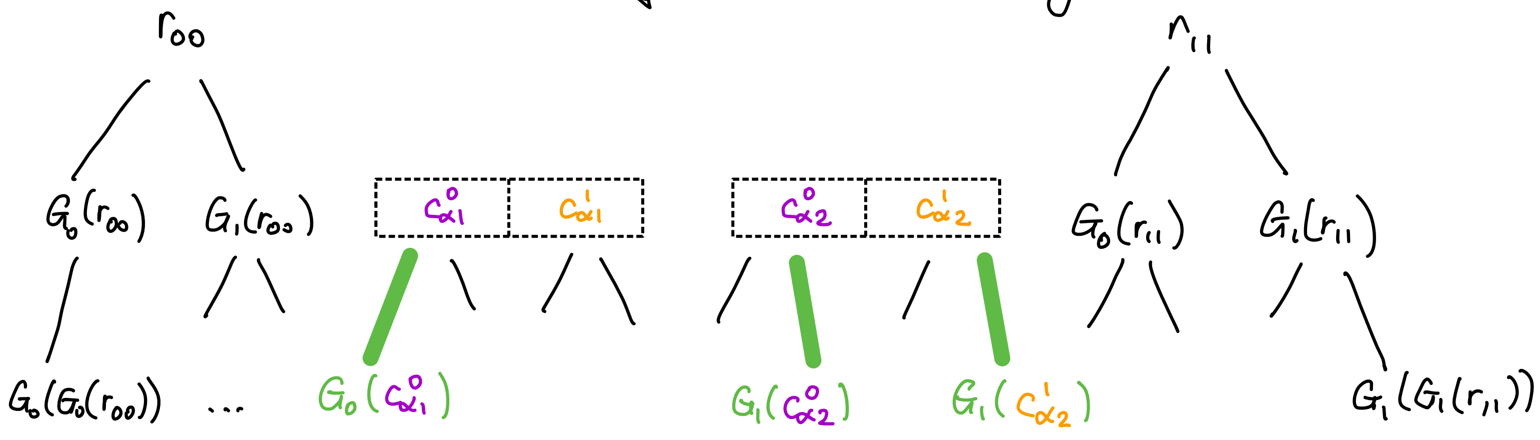
- We can prove that this game is not winnable with prob. better than $\frac{1}{2} + negl(n)$, provided $G$ is a secure PRG. (By a hybrid argument — exercise for you. ☺)

- Now $B$ just plants its $q$ challenges in the (at most) $q$ different places that $A$'s query paths intersect the relevant level of the tree.

A's queries in green ($q = 3$)




B answers according to



**Remark.** In this case, $\mathcal{B}$ did not need to use $C_{\alpha 3}$.

**Remark.** There is no need for $\mathcal{B}$ to keep track of the whole tree — it can keep track dynamically of where $A$'s queries have gone so far. As such, $\mathcal{B}$ is efficient provided that $A$ is efficient.

---

## Stateless encryption using a PRF.

- This was our chief motivation last time — let's see how to do it.

$\mathrm{Gen}(1^\lambda)$ = pick a PRF key $k$ uniformly at random
output $k$

$\mathrm{Enc}(k, m)$ = pick $x$ uniformly at random (from domain of $f_k$)
output $\boxed{\underbrace{f_k(x) \oplus m}_{c_1}, \underbrace{x}_{c_2}}$

$\mathrm{Dec}(k, c)$ = output $c_1 \oplus f_k(c_2)$

**Security:**

Imagine $f_k$ is actually a uniformly random "magic function". Why is this scheme secure then?

Intuition: PRF behaves "computationally indistinguishably" from uniformly random function.

Proof: you can do it!

$$\mathcal{C} \qquad\qquad\qquad\qquad\qquad\qquad \mathcal{A}^{\mathrm{Enc}(k,\cdot)}$$

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

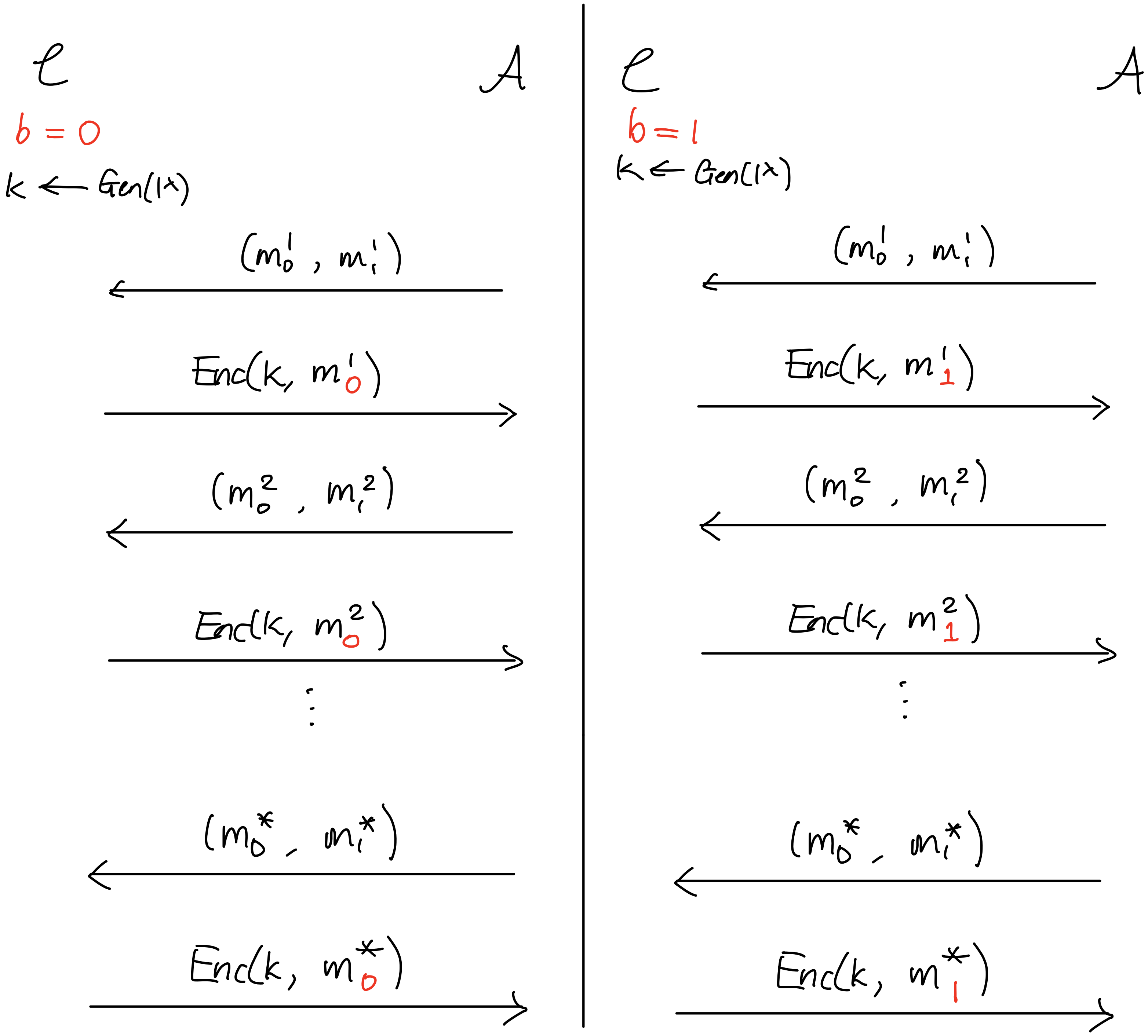
$k \leftarrow \mathrm{Gen}(1^\lambda)$

$$\xleftarrow{\quad m_0,\ m_1 \quad} \qquad |m_0| = |m_1|$$

$$\xrightarrow{\quad \mathrm{Enc}(k, m_b) \quad}$$

$$\xleftarrow{\quad \text{guess } b \quad}$$

$$\underbrace{\mathrm{IND}}_{\text{indistinguishable}} - \underbrace{\mathrm{CPA}}_{\text{"chosen plaintext attack"}}$$




$\mathcal{A}$ then tries to guess $b$.

We say (Gen, Enc, Dec) is IND-CPA secure if no adversary can guess $b$ with probability better than $\frac{1}{2} + negl(\lambda)$.