COS433/Math 473: Cryptography

Mark Zhandry

Princeton University

Spring 2018

Integer Factorization

Given an integer **N**, find it's prime factors

Studied for centuries, presumed difficult

- Grade school algorithm: $O(N^{1/2})$
- Better algorithms using birthday paradox: $O(N^{1/4})$
- Even better assuming G. Riemann Hyp.: $O(N^{1/5})$
- Still better heuristic algorithms:

$$\exp(\ C\ (\log N)^{1/3}\ (\log \log N)^{2/3}\ )$$

- However, all require super-polynomial time in bit-length of **N**

(λ,t,ε)-Factoring Assumption: For any factoring algorithm  running in time at most **t**,

Pr[(p,q)← (N):
N=pq and **p,q** random **λ**-bit primes**]≤ε**

Plausible assumption: **(λ, $t=2^{\lambda^{1/3}}$, $\varepsilon=2^{-\lambda^{1/3}}$)**

Sampling Random Primes

Prime Number Theorem: A random λ-bit number is prime with probability $\approx 1/\lambda$

Primality Testing: It is possible in polynomial time to decide if an integer is prime

Fermat Primality Test (randomized, some false positives):

- Choose a random integer $a \in \{0,\ldots,N-1\}$
- Test if $a^N = a \bmod N$
- Repeat many times

Chinese Remainder Theorem

Let $\mathbf{N} = \mathbf{pq}$ for distinct prime $\mathbf{p,q}$

Let $\mathbf{x} \in \mathbb{Z}_{\mathbf{p}}$, $\mathbf{y} \in \mathbb{Z}_{\mathbf{q}}$

Then there exists a unique integer $\mathbf{z} \in \mathbb{Z}_{\mathbf{N}}$ such that

- $\mathbf{x = z \bmod p}$, and
- $\mathbf{y = z \bmod q}$

Proof: $\mathbf{z = [py(p^{-1} \bmod q)+qx(q^{-1} \bmod p)] \bmod N}$

Quadratic Residues

Definition: y is a quadratic residue mod N if there exists an x such that $y = x^2 \bmod N$. x is called a "square root" of y

Ex:

- Let p be a prime, and $y \neq 0$ a quadratic residue mod p. How many square roots of y?
- Let $N=pq$ be the product of two primes, y a quadratic residue mod N. Suppose $y \neq 0 \bmod p$ and $y \neq 0 \bmod q$. How many square roots?

(λ,t,ε)-QR Assumption: For any factoring algorithm  running in time at most **t**,

Pr[$y^2 = x^2$ mod N:

y←**(N,x^2)**

N=pq and **p,q** random **λ**-bit primes

x←$\mathbb{Z}_N$]≤ε

Theorem: If the **(λ,t,ε)**-factoring assumption holds, then the **$(\lambda,t-t',2\varepsilon)$**-QR assumption holds

Proof

To factor **N**:

- $x \leftarrow \mathbb{Z}_N$
- $y \leftarrow$  (N, x^2)
- Output $GCD(x-y, N)$

Analysis:

- Let {a,b,c,d} be the 4 square roots of x^2
-  has no idea which one you chose
- With probability ½, y will not be in $\{+x,-x\}$
- In this case, we know $x=y \bmod p$ but $x=-y \bmod q$

Solving Quadratic Equations

In general, solving quadratic equations is:

- Easy over prime moduli
- As hard as factoring over composite moduli

Other Powers?

What about $\mathbf{x} \to \mathbf{x^4 \bmod N}$? $\mathbf{x} \to \mathbf{x^6 \bmod N}$?

The function $\mathbf{x} \to \mathbf{x^3 \bmod N}$ appears quite different

- Suppose **3** is relatively prime to **p-1** and **q-1**
- Then $\mathbf{x} \to \mathbf{x^3 \bmod p}$ is injective for $\mathbf{x \neq 0}$
  - Let **a** be such that $\mathbf{3a = 1 \bmod p\text{-}1}$
  - $\mathbf{(x^3)^a = x^{1+k(p-1)} = x(x^{p-1})^k = x \bmod p}$
- By CRT, $\mathbf{x} \to \mathbf{x^3 \bmod N}$ is injective for $\mathbf{x} \in \mathbb{Z}_N^*$

x^3 mod N

What does injectivity mean?

Cannot base of factoring:

Adapt alg for square roots:

- Choose a random **z mod N**
- Compute $y = z^3 \bmod N$
- Run inverter on **y** to get a cube root **x**
- Let **p = GCD(z-x, N), q = N/p**

RSA Problem

Given

- $N = pq$,
- e such that $GCD(e,p-1)=GCD(e,q-1)=1$,
- $y=x^e \bmod N$ for a random x

Find x

Injectivity means cannot base hardness on factoring, but still conjectured to be hard

(e,t,ε)-RSA Assumption: For any factoring algorithm  running in time at most **t**,

Pr[x←**(N,x^3 mod N)**

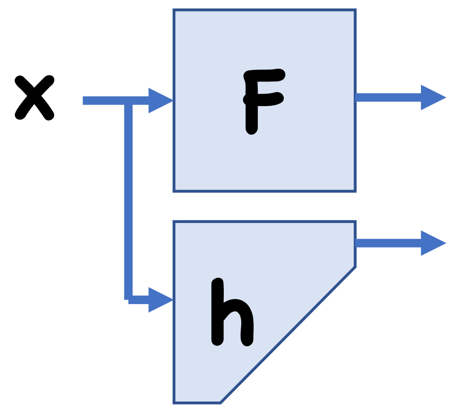
N=pq and **p,q** random **λ**-bit primes s.t.

GCD(3,p-1)=GCD(3,q-1)=1

x←$\mathbb{Z}_N^*$]≤ε

Application: PRGs

Let $\mathbf{F(x) = x^3 \bmod N}$, $\mathbf{h(x)}$ = least significant bit



Theorem: If **(e,t,ε)-RSA** Assumption holds, then
$\mathbf{G(x) = (\ F(x),\ h(x)\ )}$ is a **(t−t',ε')**-secure PRG

Crypto from Minimal Assumptions

Many ways to build crypto

We've seen many ways to build crypto

- SPN networks
- LFSR's
- Discrete Log
- Factoring

Questions:

- Can common techniques be abstracted out as theorem statements?
- Can every technique be used to build every application?

One-way Functions

The minimal assumption for crypto

Syntax:

- Domain **D**
- Range **R**
- Function **F: D → R**

No correctness properties other than deterministic

Security?

Definition: F is **(t,ε)–**One-Way if, for all  running in time at most **t**,

Pr[x←**(F(x)),x←D] < ε**

Trivial example:

F(x) = parity of **x**

Given **F(x),** impossible to predict **x**

Security

Definition: F is **(t,ε)-**One-Way if, for all  running in time at most **t**,

Pr[F(x)=F(y):y← **(F(x)),x←D] < ε**

Examples

Any PRG

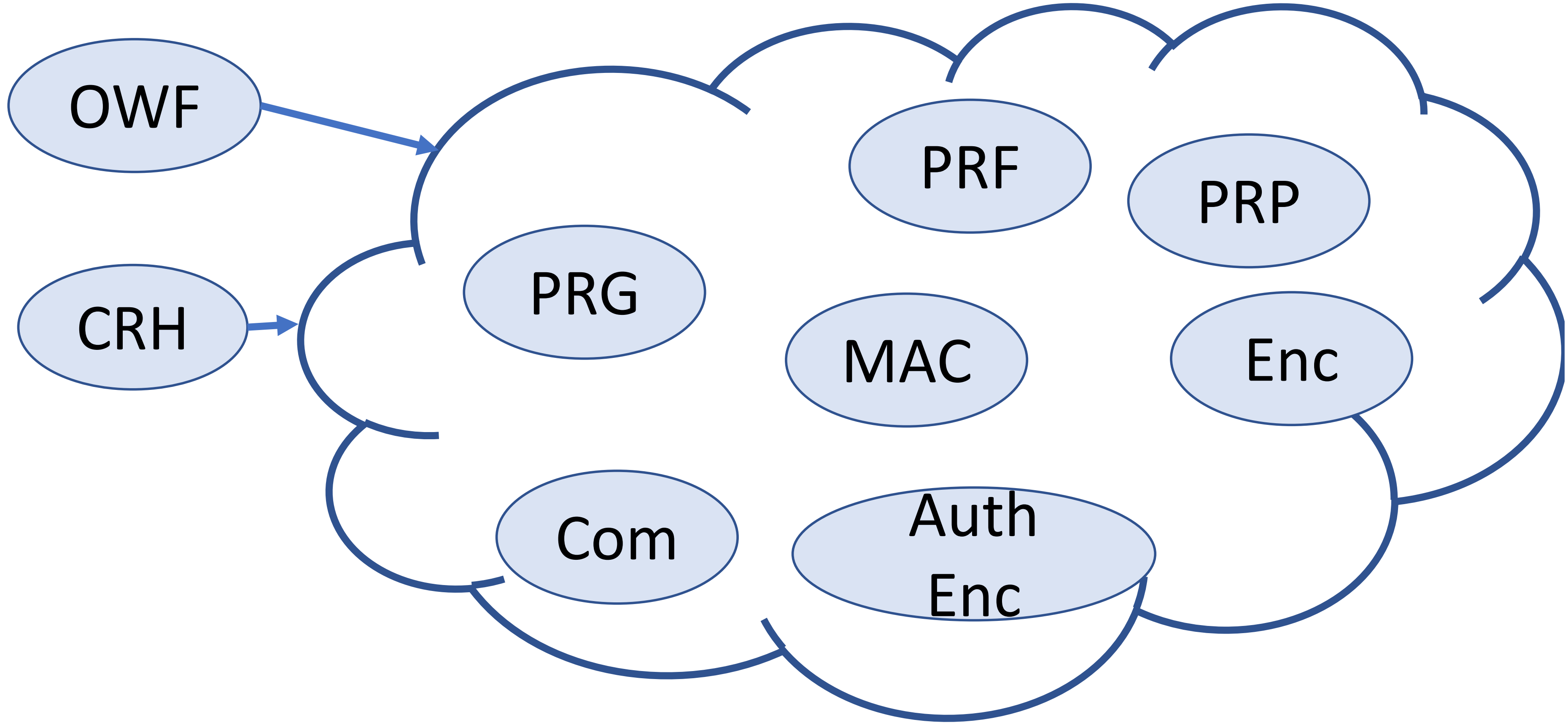
Any Collision Resistant Hash Function (with sufficient compression)

$F(p,q) = pq$

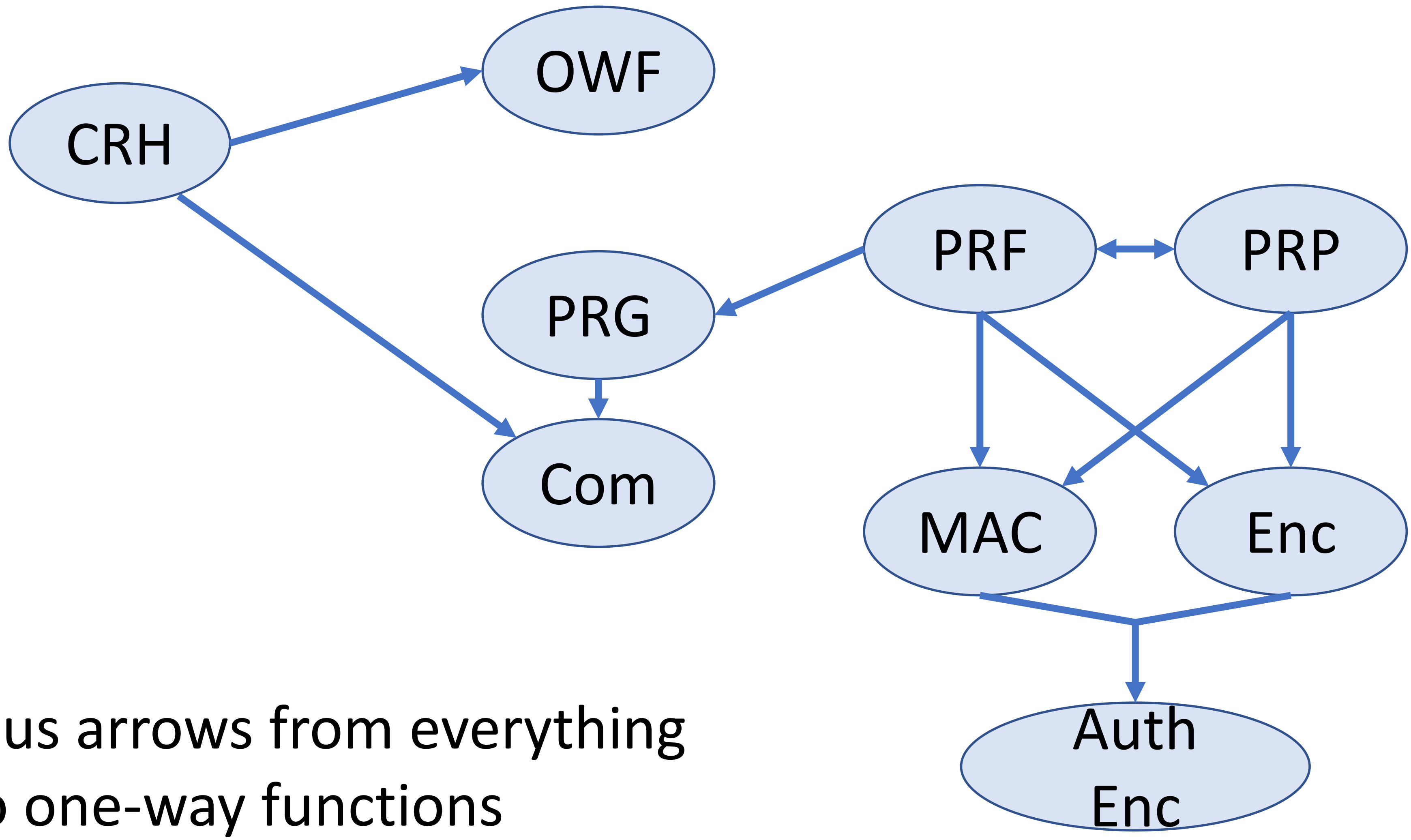
$F(g,a) = (g,g^a)$

$F(N,x) = (N,x^3 \bmod N)$ or $F(N,x) = (N,x^2 \bmod N)$

What’s Known



So Far



Plus arrows from everything to one-way functions

Our Goal: Fill in Remaining Arrows

Hardcore Bits

Let **F** be a one-way function with domain **D**, range **R**

Definition: A function $\mathbf{h:D \rightarrow \{0,1\}}$ is a $\mathbf{(t,\varepsilon)}$-hardcore bit for **F** if, for any  running in time at most **t**,

| Pr[1←**(F(x), h(x)), x←D]**
– Pr[1←**(F(x), b), x←D,b←{0,1}] | ≤ ε**

In other words, even given **F(x)**, hard to guess **h(x)**

Examples of Hardcore Bits

Define **lsb(x)** as the least significant bit of **x**

For $\mathbf{x} \in \mathbf{Z}_N$, define **Half(x)** as **1** iff $\mathbf{0 \leq x < N/2}$

Theorem: Let $\mathbf{p}$ be a prime, and $\mathbf{F:Z_p^* \rightarrow Z_p^*}$ be $\mathbf{F(x) = g^x \bmod p}$, for some generator $\mathbf{g}$

Half is a hardcore bit for **F** (assume **F** is one-way)

Theorem: Let $\mathbf{N}$ be a product of two large primes $\mathbf{p,q}$, and $\mathbf{F:Z_N^* \rightarrow Z_N^*}$ be $\mathbf{F(x) = x^e \bmod N}$ for some $\mathbf{e}$ relatively prime to $\mathbf{(p-1)(q-1)}$

Lsb and Half are hardcore bits for **F** (assuming RSA)

Theorem: Let $\mathbf{N}$ be a product of two large primes $\mathbf{p,q}$, and $\mathbf{F:Z_N^* \rightarrow Z_N^*}$ be $\mathbf{F(x) = x^2 \bmod N}$

Lsb and Half are hardcore bits for **F** (assuming factoring)

Goldreich Levin Hardcore Bit

Let **F** be a OWF with domain $\{0,1\}^n$ and range **R**

Let $F':\{0,1\}^{2n} \to \{0,1\}^n \times R$ be:

$$F'(r,x) = r,F(x)$$

Define $h(r,x) = \langle r,x \rangle = \Sigma r_i x_i \bmod 2$

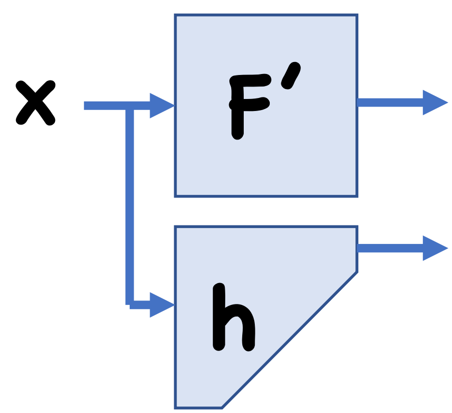
Theorem (Goldreich-Levin): If **F** is (t,ε)–one-way, then **h** is a $(\ \text{poly}(t,1/\varepsilon),\ \text{poly}(\varepsilon)\ )$–hc bit for **F'**

Application: PRGs

Suppose **F** was a permutation (**D=R** and **F** is one-to-one)

Let **F'**, **h** be from Goldreich-Levin

- **F'** is also a permutation



Hardcore Bits

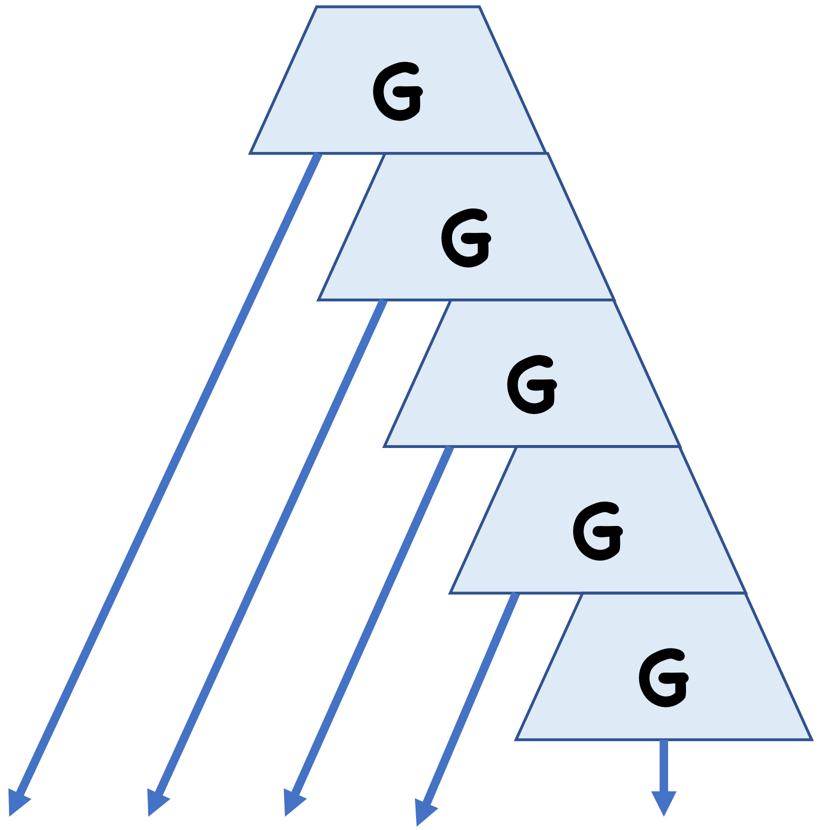
A hc bit for any OWF

Implies PRG from any one-way *permutation*
- PRG from Dlog (Blum-Micali)
- PRG from RSA
- PRG from Factoring

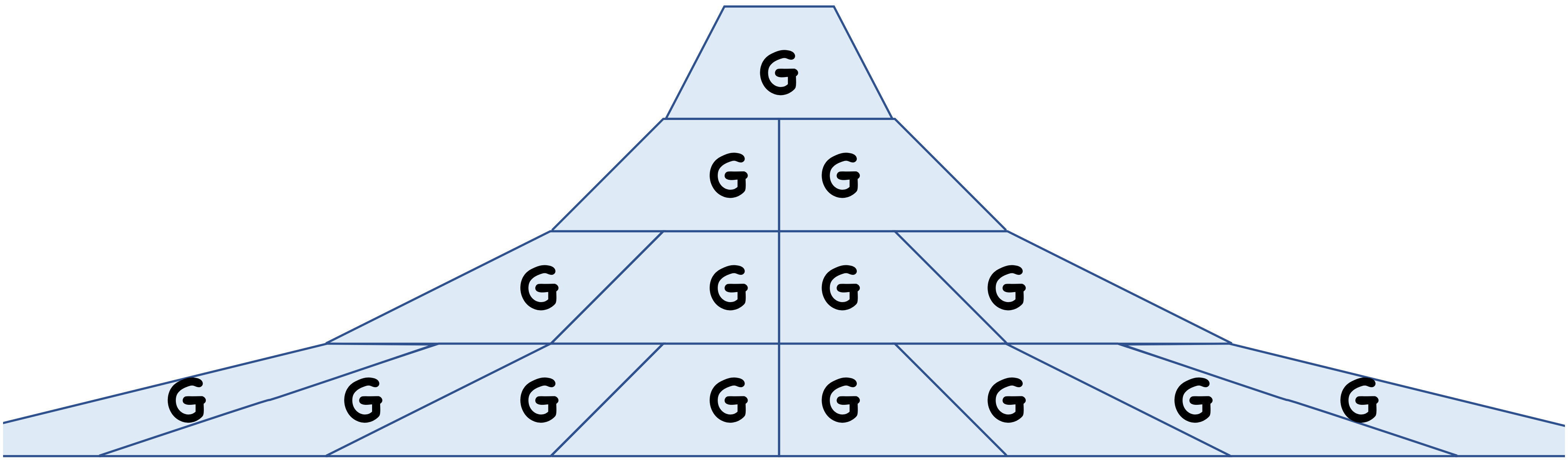
Actually, can construct PRG from any OWF
- Proof beyond scope of course

PRGs → PRFs

First: Expanding Length of PRGs



A Different Approach



Advantage of Tree-based Approach

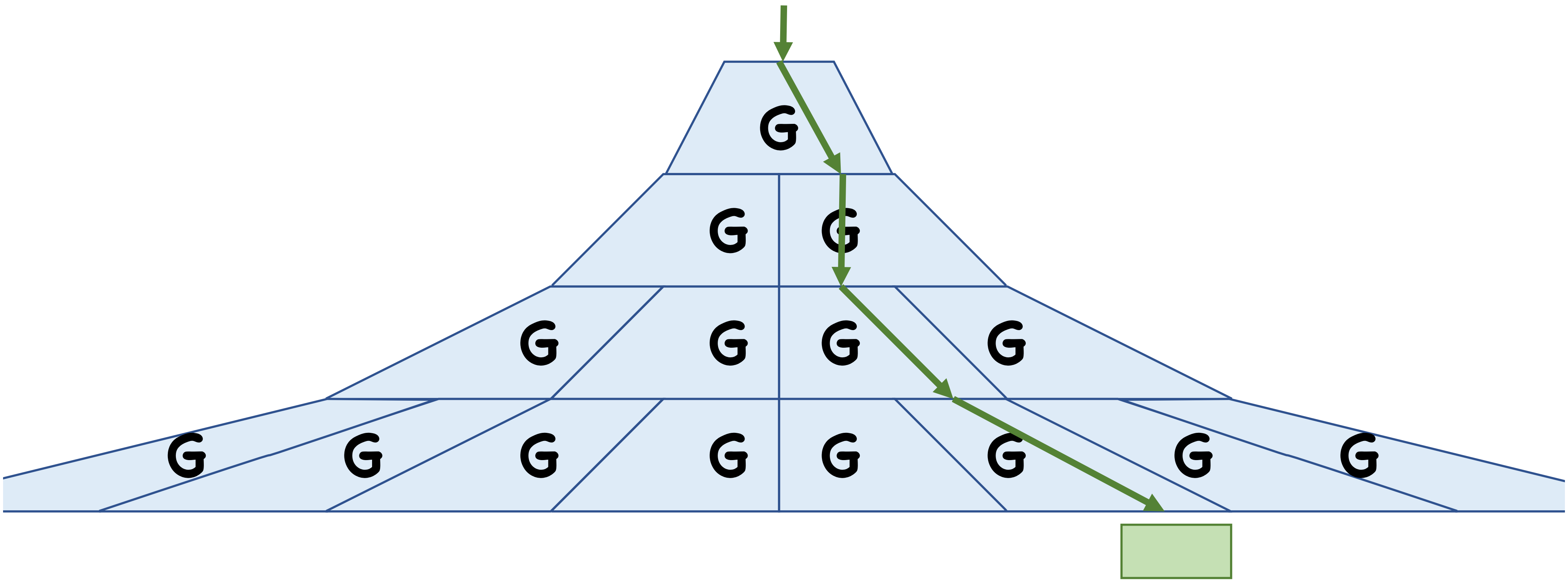
To expand λ bits into $2^h\lambda$ bits, need h levels

Can compute output locally:

- To compute ith chunk of λ bits, only need h PRG evaluations

In other words, can locally compute in logarithmic time

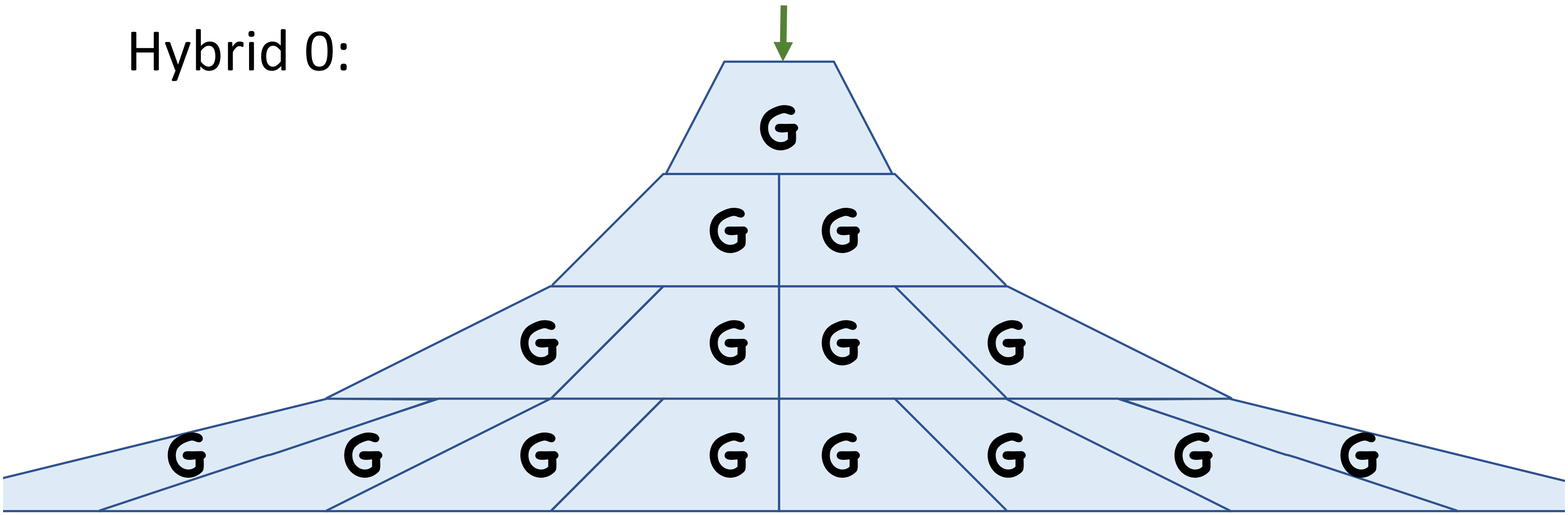
Advantage of Tree-based Approach



Theorem: For any logarithmic **h**, if **G** is a **(t,ε)**-secure PRG, then tree-based PRG is **(t−t′, L(h)ε)**-secure for some function **L(h)**

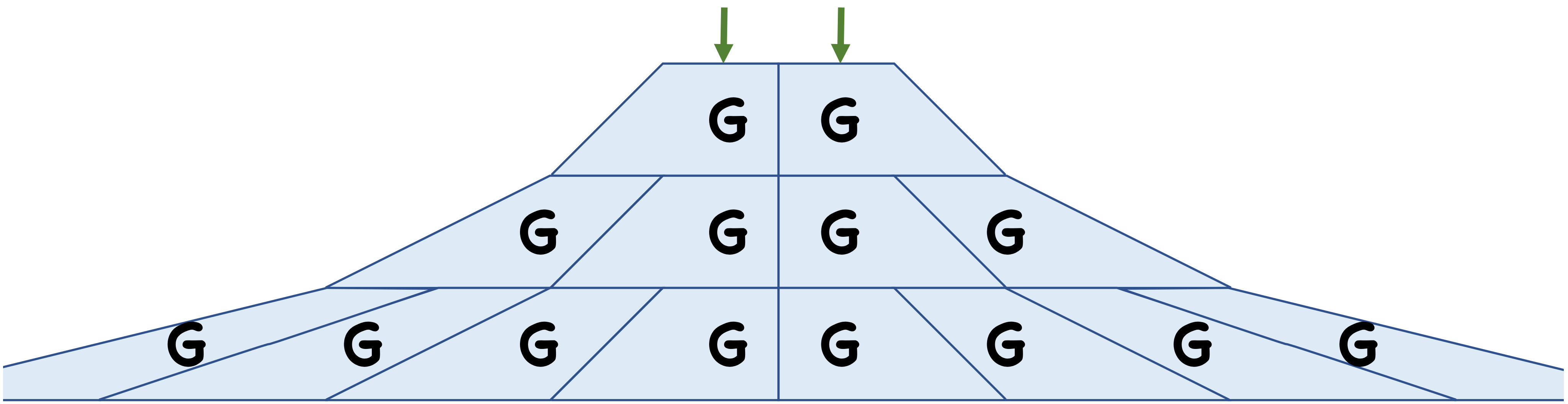
Proof

Hybrid 0:



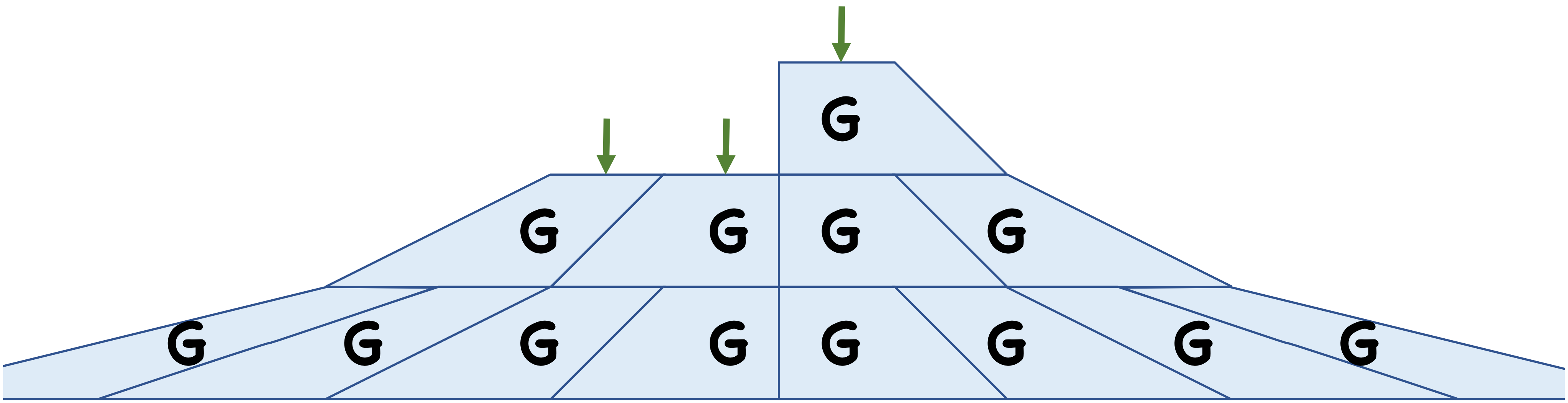
Proof

Hybrid 1:



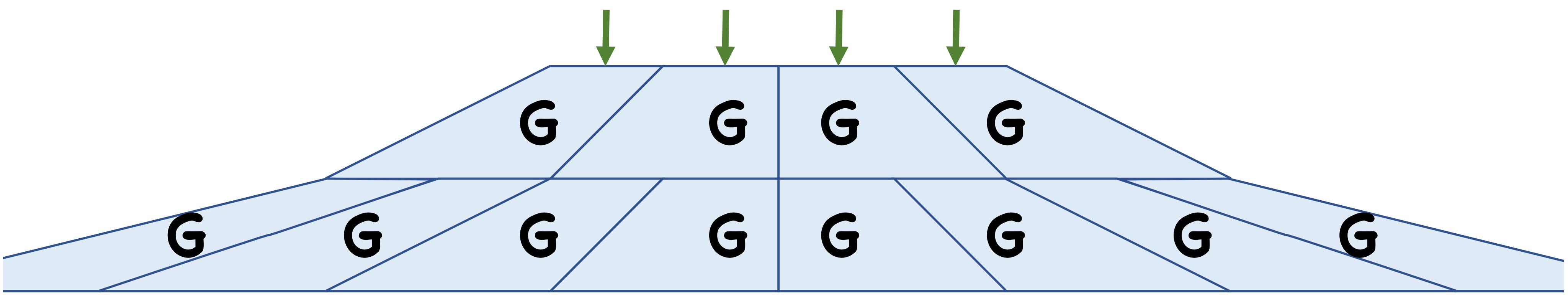
Proof

Hybrid 2:



Proof

Hybrid 3:



Proof

Hybrid **t**:

Proof

What is **L(h)**?

PRG adversary distinguishes Hybrid 0 from Hybrid **t** with advantage **L(h)ε**

- $\exists$ **i** such that adversary distinguishes Hybrid **i-1** from Hybrid **i** with advantage **ε**
- Can use to construct adversary for **G** with advantage **ε**

A PRF

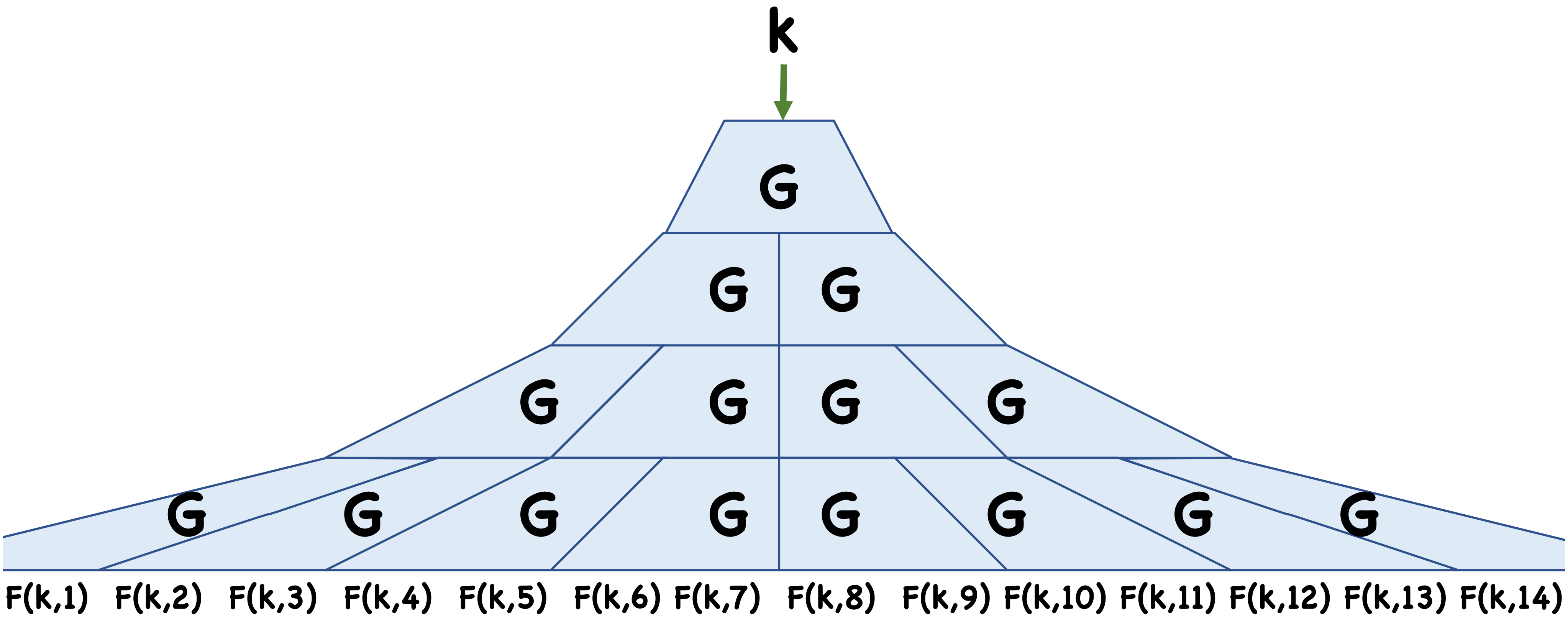
Domain $\{0,1\}^n$

Set $h = n$

$F(k, x)$ is the xth block of λ bits

- Computation involves h evals of G, so efficient

A PRF



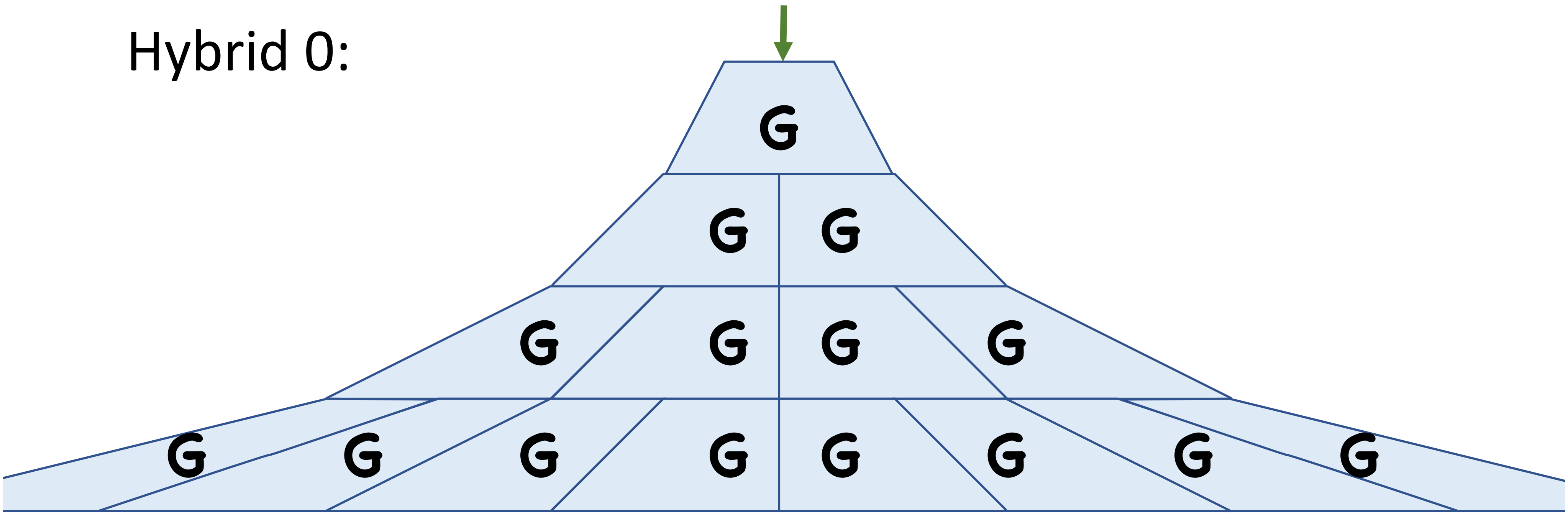
k
G
G
G
G
G
G
G
G
G
G
G
G
G
G
G
F(k,1)
F(k,2)
F(k,3)
F(k,4)
F(k,5)
F(k,6)
F(k,7)
F(k,8)
F(k,9)
F(k,10)
F(k,11)
F(k,12)
F(k,13)
F(k,14)

Problem with Security Proof

Suppose I have a PRF adversary with advantage ε'. In the proof, what is the advantage of the derived PRG adversary?

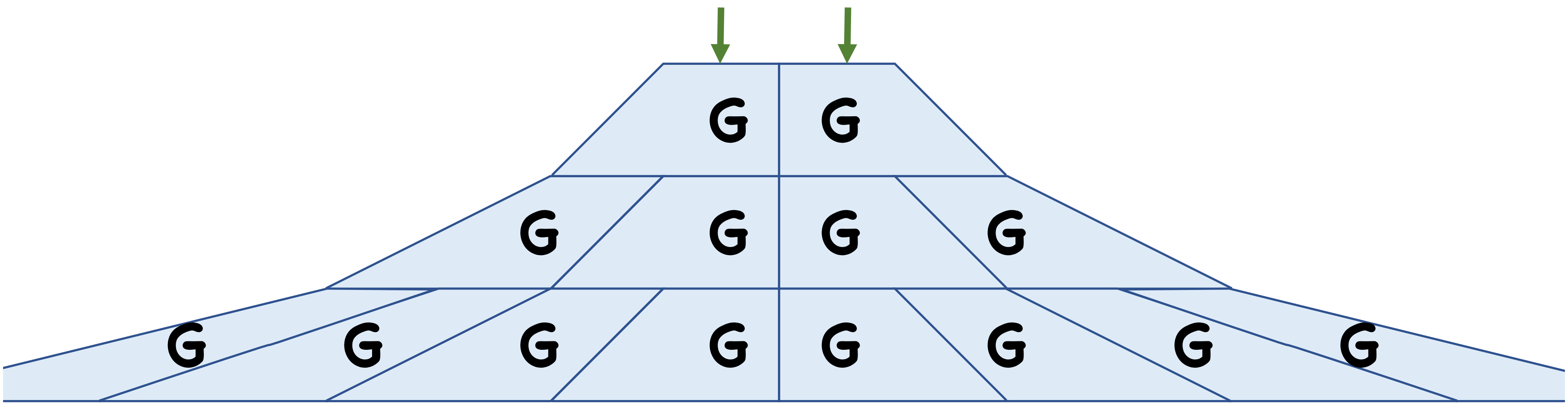
A Better Proof

Hybrid 0:



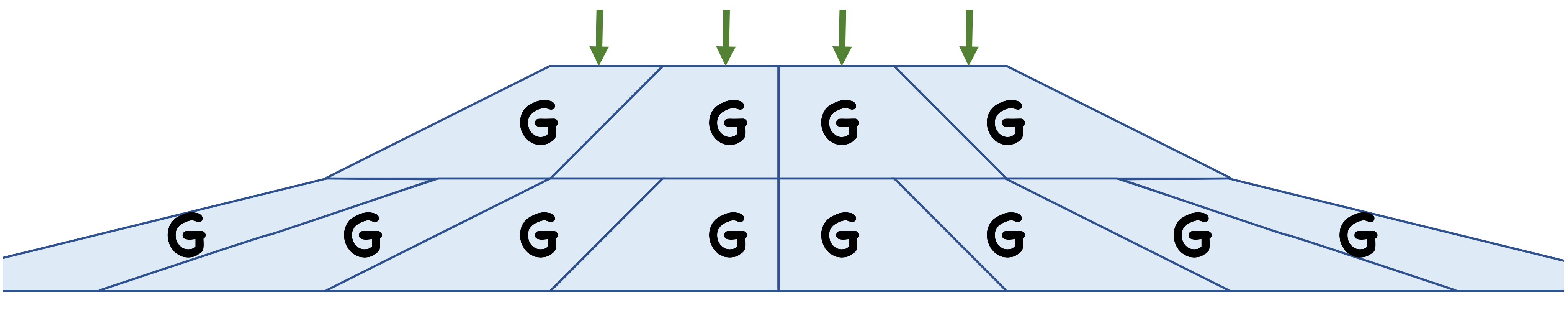
A Better Proof

Hybrid 1:



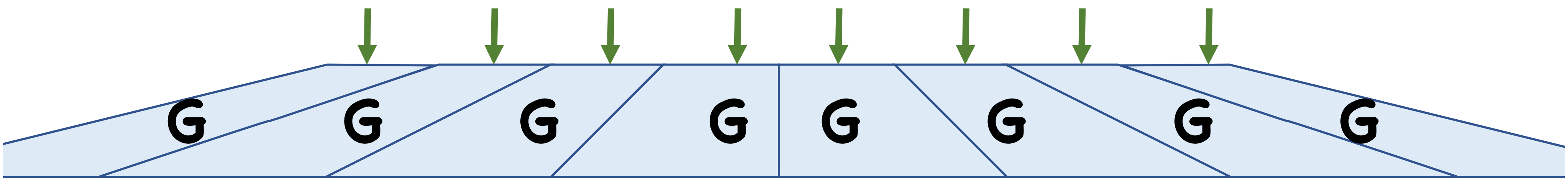
A Better Proof

Hybrid 2:



A Better Proof

Hybrid 3:



A Better Proof

Hybrid **h=n**:

A Better Proof

Now if PRF adversary distinguishes Hybrid 0 from Hybrid $h=n$ with advantage ε', $\exists i$ such that adversary distinguishes Hybrid $i-1$ from Hybrid i with advantage ε'/n

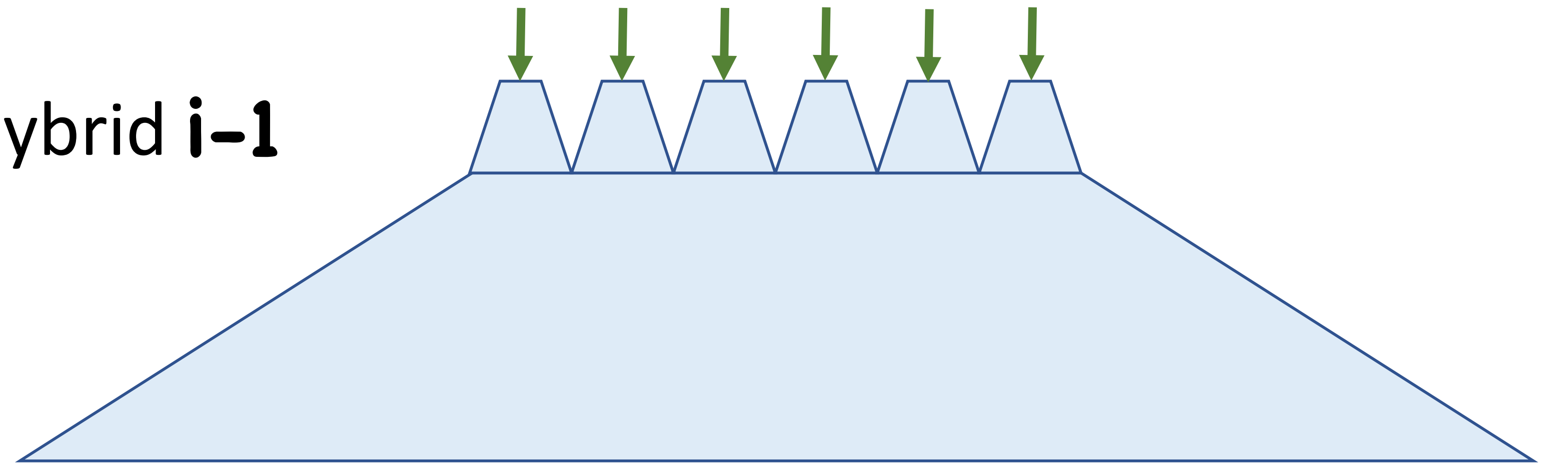
- Non-negligible advantage

Not quite done: Distinguishing Hybrid $i-1$ from Hybrid i does not immediately give a PRG distinguisher

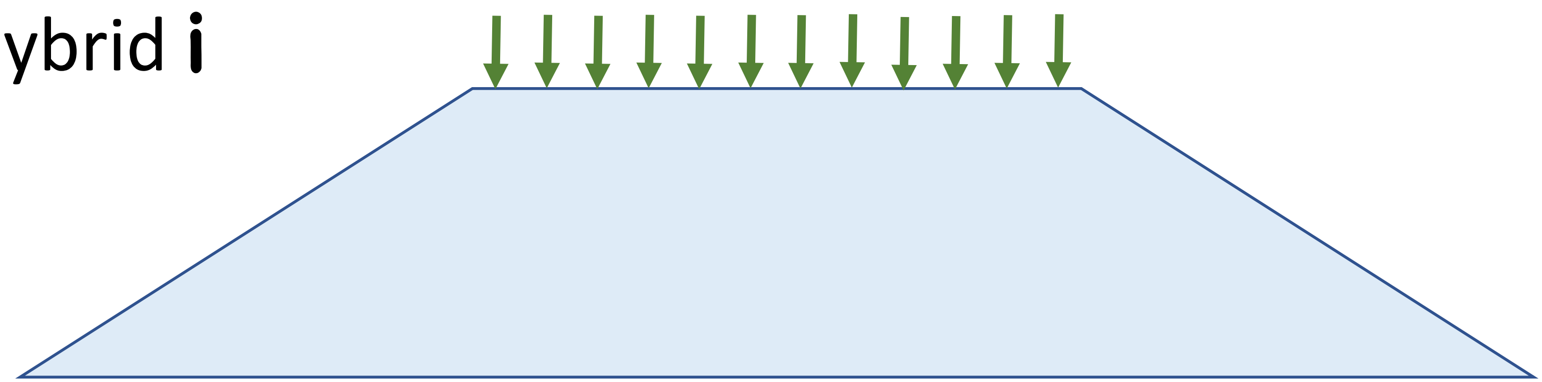
- Exponentially many PRG values changed!

A Better Proof

Hybrid **i-1**

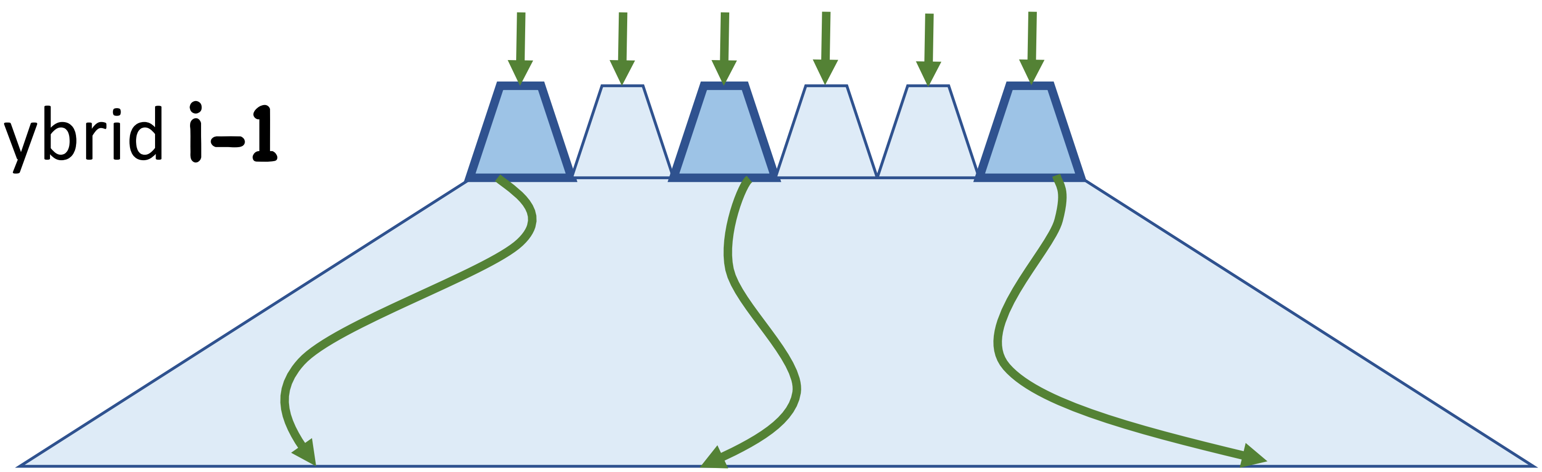


Hybrid **i**



Key Observation:

Hybrid **i-1**



Adversary only queries polynomially many outputs
$\Rightarrow$ Only need to worry about polynomially many PRG instances in level **i**

A Better Proof

More Formally:

Given distinguisher **A** for Hybrid $\mathbf{i-1}$ and Hybrid $\mathbf{i}$, can construct distinguisher **B** for the following two oracles from $\{0,1\}^{i-1} \rightarrow \{0,1\}^{2\lambda}$

- $\mathbf{H_0}$: each output is a fresh random PRG sample
- $\mathbf{H_1}$: each output is uniformly random

If **A** makes **q** queries, **B** makes at most **q** queries

A Better Proof

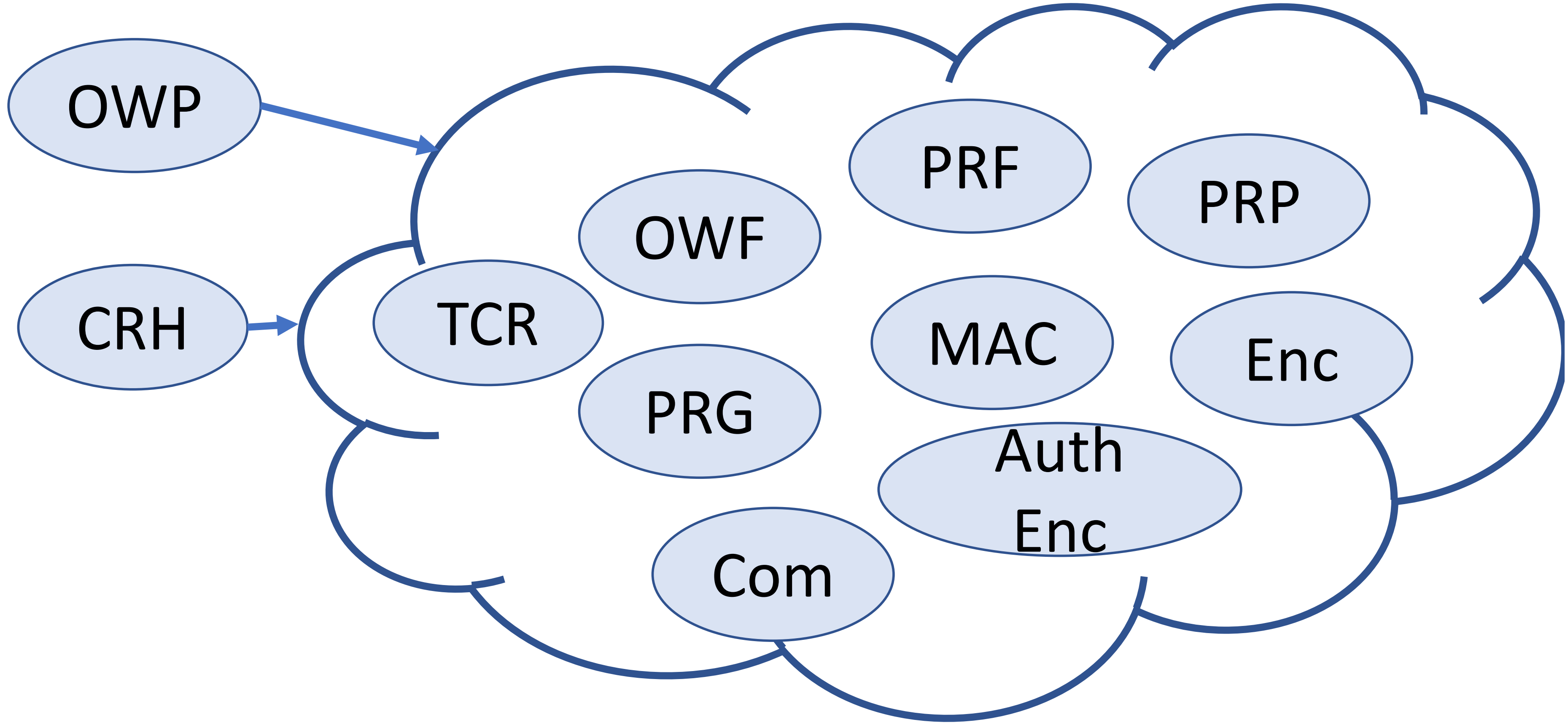
Now we have a distinguisher B with advantage $\boldsymbol{\varepsilon'/n}$ that sees at most $\mathbf{q}$ values, where either

- Each value is a random output of the PRG, or
- Each value is uniformly random

By introducing $\mathbf{q}$ hybrids, can construct a PRG distinguisher with advantage $\boldsymbol{\varepsilon'/qn}$

By setting $\boldsymbol{\varepsilon'=qn\varepsilon}$, we get security

What's Known



What about OWP, CRH?

Generally Believed That…

Cannot construct OWP from OWF

Cannot construct CRH from OWF

Cannot construct CRH from OWP

Cannot construct OWP from CRH

Black Box Separations

How do we argue that you cannot build collision resistance from one-way functions?

- We generally believe both exist!

Observation: most natural constructions treat underlying objects as black boxes (don’t look at code, just input/output)

Maybe we can rule out such natural constructions

Black Box Separations

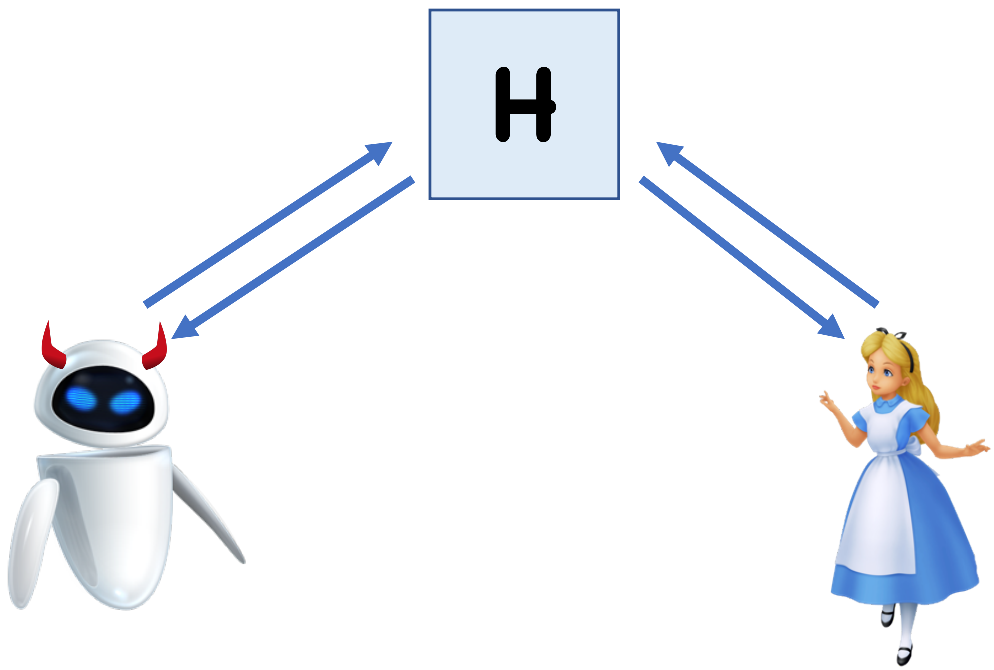
Present a world where one-way functions exist, but collision resistance does not

Hopefully, natural (black box) constructions make sense in this world

- Can construct PRGs, PRFs, PRPs, Auth-Enc, etc

Separating CRH from OWF

Starting point: random oracle model



Computation power is unlimited, but number of calls to random oracle is polynomial

Separating CRH from OWF

In ROM, despite unlimited computational power, one-way functions exist

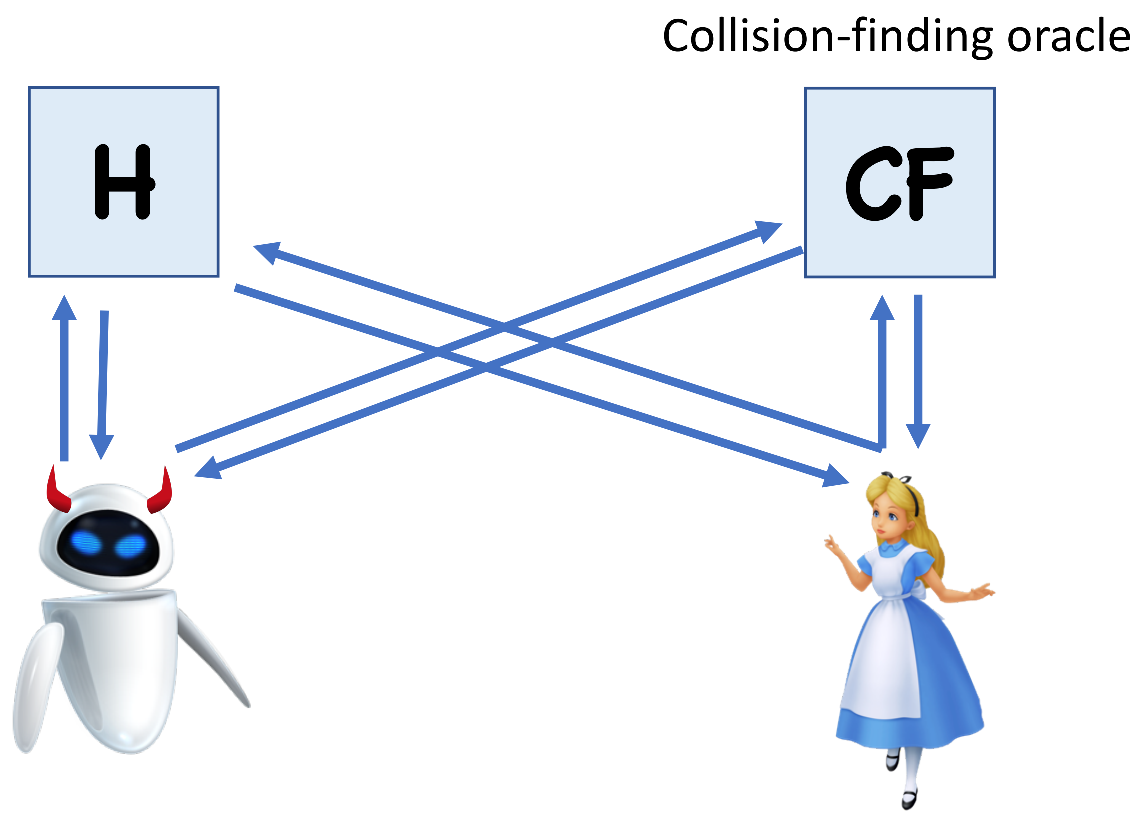
- **F(x) = H(x)**
- Can only invert oracle by making exponentially-many calls

Unfortunately, collision resistant hashing exists too!

- **F(x) = H(x)**

To fix, also add collision finding oracle

Separating CRH from OWF



Separating CRH from OWF

What does **CF** do?

- Takes as input a circuit **C**
- Circuit may have “oracle gates” that make calls to **H** or **CF**
- Outputs a collision for **C**

Impossibility of Collision Resistance?

- Consider BB construction of CRHF from OWF
- Replace calls to OWF with **H** queries
- Feed circuit computing CRHF to **CF** to find collision

Separating CRH from OWF

So we have a world in which collision resistance does not exist

However, maybe **CF** can be used to invert **H**
- So maybe one-way functions don't exist either

Must be careful in defining **CF**
- Random pair of colliding inputs will allow for inverting **H**

Separating CRH from OWF

Correct **CF**:

- Choose random input **x** to circuit
- Choose random input **y** that collides with **x**

Note that **x** will sometimes equal **y**. However, if circuit shrinks input, then with probability at least ½ **x≠y**

Careful analysis shows that **H** is still one-way

Next Time

Begin public key cryptography

Key agreement: how to exchange keys without ever meeting