6.875 Lecture 4

Spring 2020

Lecturer: Shafi Goldwasser

Randomness is the foundation of cryptography:

- Cryptographic keys have to be unpredictable to the adversary
- Cryptographic algorithms use additional randomness (beyond the key)
- If the random bits are revealed (or are predictable) the entire structure collapses

Sources of Randomness

1) Specialized Hardware: e.g., Transistor noise

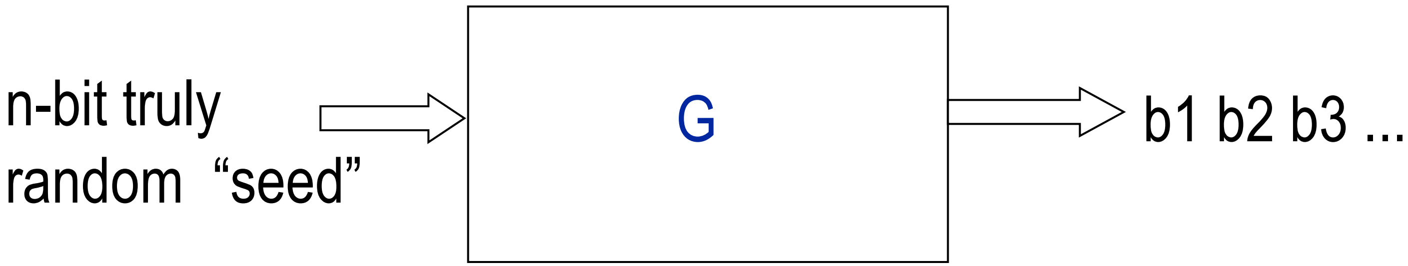
2) User Input: Every time random number used, user is queried

Usually biased, but can “extract” unbiased bits assuming the source has “some structure and enough entropy” [von Neumann, Elias, Blum]

BUT: True randomness is an expensive commodity.

If Only there were Random Number Generators…

That is: **Deterministic** Programs that stretch a truly random seed into a (much) longer sequence of truly random bits.



Can such a G exist?

Pseudo-random Generators

Informally: **Deterministic** Programs that stretch a "truly random" seed into a (much) longer sequence of **"seemingly random"** bits.

Application for One Time Pads

$Enc(m_i) = m_i \oplus pad_i$ where pad_i is the ith block output by G

TODAY

NEW NOTION: Pseudo-random Generators
(Two different definitions; Equivalence)

CONSTRUCTION [Blum-Micali’82, Yao82]:
One-way Permutations + Hardcore Bits = Pseudorandom Generator.

APPLICATIONS

Pseudo-random Generators

Informally: **Deterministic** Programs that stretch a "truly random" seed into a (much) longer sequence of **"seemingly random"** bits.

How to **Define** a Strong Pseudo Random Number Generator?

Def 1 [Indistinguishability]

"No polynomial-time algorithm can distinguis between the output of a PRG on a random seed vs. a andom string" = "as good as" a truly random string practical purposes.

Def 2 [Next-bit Unpredict]

"No polynomial-time a n can predict the (i+1)th bit of the output of a PRG gi e first i bits"

Def 3 [Incon ssibility]

"No polynomial-time algorithm can compress the output of the PRG into a shorter string"

ALL THREE DEFS EQUIVALENT!

PRG Def 1: Indistinguishability

Definition [Indistinguishability]:

A deterministic polynomial-time computable function G: $\{0,1\}^n \rightarrow \{0,1\}^m$ is a PRG which "passes all poly time statistical tests" if

(a) m > n and

(b) for every PPT algorithm D, there is a negligible function negl such that:

$$\left| \Pr[\, D(G(U_n)) = 1 \,] - \Pr[\, D(U_m) = 1 \,] \right| = \mathrm{negl}(n)$$

Notation: U_n (resp. U_m) denotes the random distribution on n-bit (resp. m-bit) strings; m is shorthand for m(n).

PRG Def 1: Indistinguishability

Definition [Indistinguishability]:

A deterministic polynomial-time computable function G: $\{0,1\}^n \to \{0,1\}^m$ is a PRG which "passes all poly time statistical tests" if

(a) $m > n$ and

(b) for every PPT algorithm D, there is a negligible function negl such that:

$$\left| \Pr[D(G(U_n)) = 1] - \Pr[D(U_m) = 1] \right| = \mathrm{negl}(n)$$

We call D that takes a sequence and outputs 0 or 1

a *statistical test*..

PRG Def 1: Indistinguishability

Def: A deterministic function $G: \{0,1\}^n \rightarrow \{0,1\}^m$ is a strong PRG if $m > n$ and for every PPT algorithm D, there is a negligible function negl such that:

$$\left| \Pr[\, D(G(U_n)) = 1 \,] - \Pr[\, D(U_m) = 1 \,] \right| = \text{negl}(n)$$

WORLD 1: The Pseudorandom World

$y \leftarrow G(U_n)$

WORLD 2: The Truly Random World

$y \leftarrow U_m$

PPT Distinguisher gets y but cannot tell which world she is in

Why is this a good definition

Good for all Applications:

As long as we can find truly random seeds, can replace true randomness by the output of PRG(seed) in ANY "computational" setting.

If it behaves differently,
can convert "application"=statistical test

But: its hard to work with. How do you show that generator G passes ALL statistical tests?

PRG Def 2: (Next-bit) Unpredictability

Definition [Next-bit Unpredictability]:

A deterministic polynomial-time computable function $G: \{0,1\}^n \rightarrow \{0,1\}^m$ is a PRG if

(a) $m > n$ and

(b) for every PPT algorithm PRED and every $i \in [1..m]$, there is a negligible function negl such that:

$$\Pr[\, y \leftarrow G(U_n):\ \text{PRED}(y_1 y_2 \ldots y_{i-1}) = y_i\,] = \tfrac{1}{2} + \text{negl}(n)$$

Notation: y_i denotes the i-th bit of y.

$y_{1\ldots i}$ denotes the first i bits of y.

PRG Def 2: (Next-bit) Unpredictability

Definition [Next-bit Unpredictability]:

A deterministic polynomial-time computable function G: $\{0,1\}^n \rightarrow \{0,1\}^m$ is a PRG

(a) m > n and

(b) or every PPT algorithm PRED and every i ∈ [1..m], there is a negligible function negl such that:

$$\Pr[\, y \leftarrow G(U_n):\ \mathrm{PRED}(y_1 y_2 \ldots y_{i-1}) = y_i] = \tfrac{1}{2} + \mathrm{negl}(n)$$

Notation: Call PRED a "next-bit test" and if (b) holds, we say that G "passes all next bit tests "

Def 1 and Def 2 are Equivalent

Theorem: A PRG G passes all polynomial time statistical tests if and only if it passes all polynomial time next-bit tests

$\Rightarrow$

Proof: By counter positive.[if predictable then distinguishable]

- Suppose there is a next-bit test PRED, a polynomial p and an index i such that

$$\Pr[\ PRED(G(U_n)_{1\ldots i}) = G(U_n)_{i+1}\ ] > 1/2 + 1/p(n)$$

- We know that $\Pr[\ PRED(U_i) = u_{i+1}\ ] \leq 1/2$ since u_{i+1} is uniformly random and independent of $u_1, u_2, \ldots, u_i$ and this its impossible to guess it correctly better than 1/2
- Thus, PRED is a (ppt) statistical test which distinguishes between $G(U_n)$ and U_m, and thus G is not indistinguishable. QED

Def 1 and Def 2 are Equivalent

Theorem: A PRG G satisfies all polynomial time statistical tests if and only if it passes all next-bit tests

Proof: By counter positive

Suppose now that G does not pass some polynomial time statistical test DIST.

Then we will show that A can be converted into a next bit test PRED.

That is, show the existence of a bit position j s.t. for sufficiently large n, PRED can predict the value of j-th output bit of G by reading only a prefix of length j-1.

Def 1 and Def 2 are Equivalent

Theorem: A PRG G satisfies the indistinguishability def if and only if it is next-bit unpredictable.

Proof: By contradiction. TWO STEPS.

- **STEP 1:** HYBRID ARGUMENT
- **STEP 2:** From Distinguishing to Predicting

Distinguishers and Predictors

– Given a distinguisher algorithm DIST with advantage ε, we have:

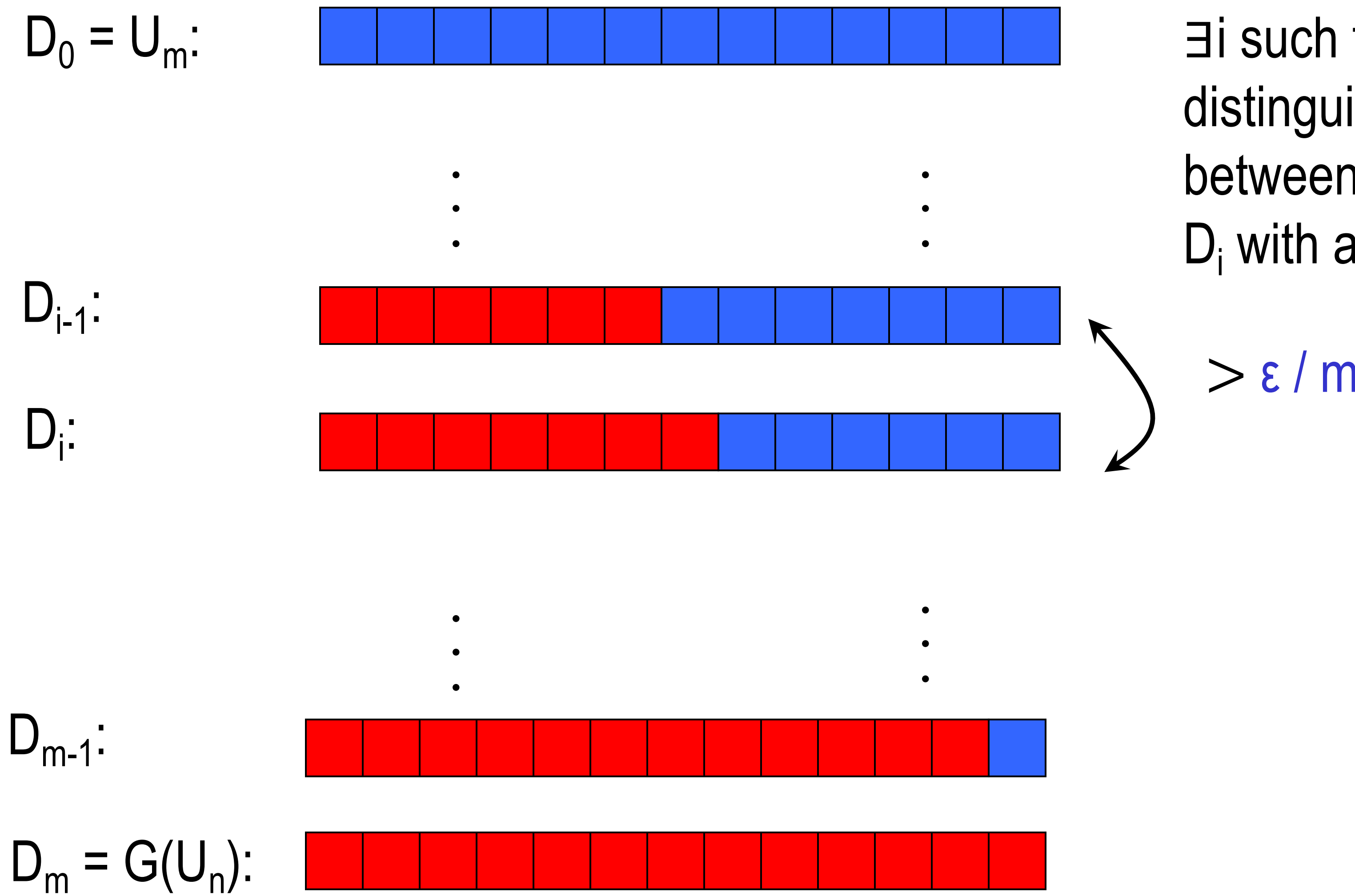
$$\left| \Pr[\,DIST(G(U_n)) = 1] - \Pr[DIST(U_m) = 1] \right| > \varepsilon$$

– Define m+1 **hybrid** distributions.

Hybrid Distributions

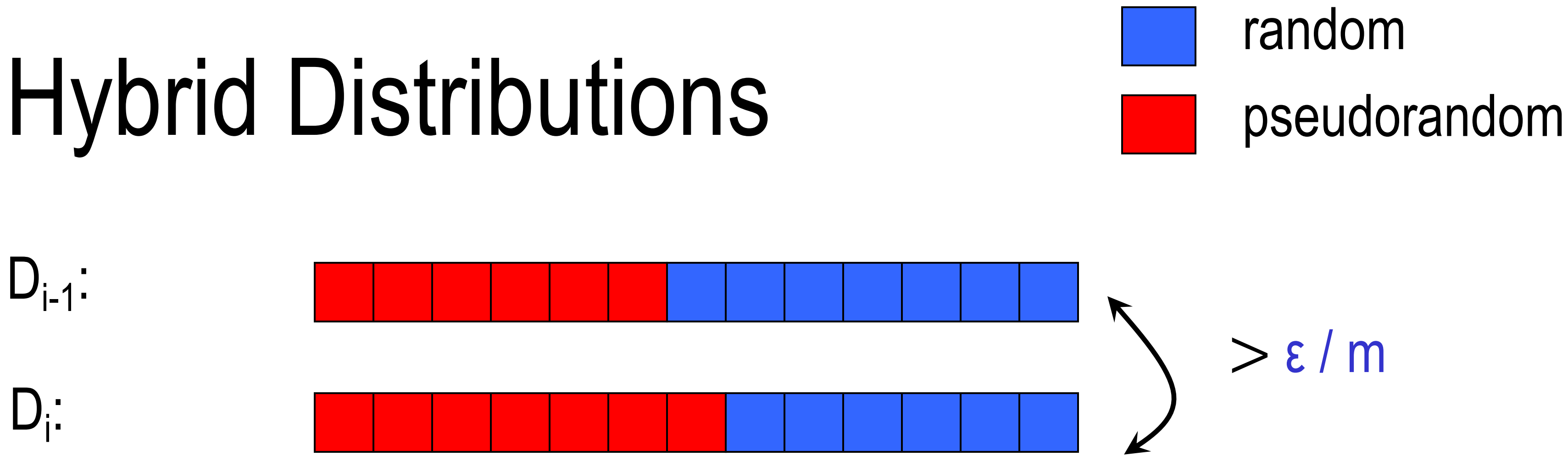
random

pseudorandom



$\exists i$ such that DIST distinguishes between D_{i-1} and D_i with advantage

Hybrid Distributions



- Define: $p_i = \Pr[y \leftarrow D_i : \text{DIST}(y) = 1]$
  - Then: $p_0 = \Pr[y \leftarrow U_m : \text{DIST}(y) = 1]$ and

    $p_m = \Pr[y \leftarrow G(U_n) : \text{DIST}(y) = 1]$
- Wlog this. implies $p_i - p_{i-1} > \varepsilon/m$. [exercise: deal with absolute values]
- THEN: Can design a predictor (next-bit test) PRED for i-th bit of pseudo-random sequences given the (i-1)-bit prefix.

Predictor PRED for i^{th} bit:

On input: $y = y_1 y_2 \ldots y_{i-1}$

PRED:

- flip a coin: **c** $\in \{0,1\}$
- $u = u_{i+1} u_{i+2} \ldots u_m \leftarrow U_{m-i}$
- Run DIST(y**c**u)
- if D outputs 1, output c;
- if D outputs 0, output $\neg c$

(intuition: 1 is a vote for psr bit since $p_i > p_{i-1}$)

<u>**Claim**</u>:

$$\Pr[\mathrm{PRED}(y_{1\ldots i-1}) = y_i] > ½ + \varepsilon/m.$$

Distinguishing to Prediction: Analysis

- Recall: $p_i - p_{i-1} > \varepsilon/m$

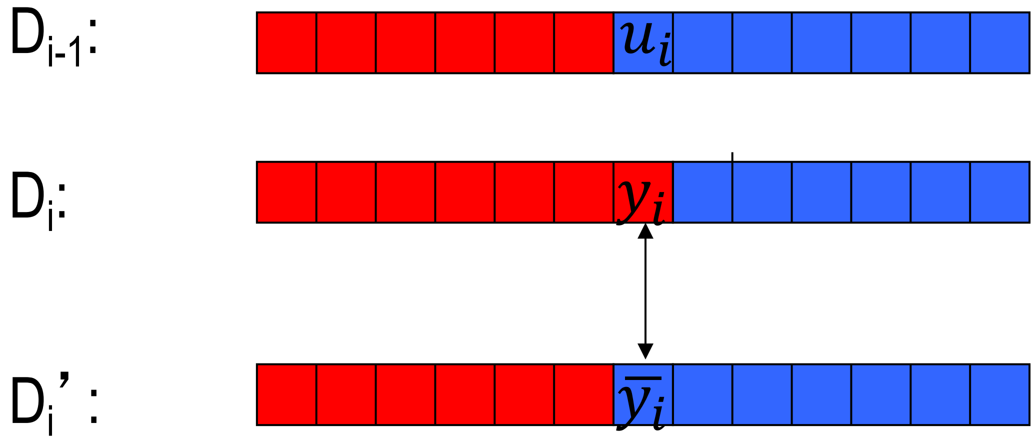
  (i.e prob D outputs 1 higher when i-th bit is from the output of the PRG as opposed to random)

- Let distribution D_i' be D_i with i-th bit flipped and

  $$p_i' = \Pr[y \leftarrow D_i' : \mathrm{DIST}(y) = 1]$$

Claim: $p_{i-1} = (p_i + p_i')/2$

Proof: Exercise.



Proof of Claim

$y = y_1 y_2 \ldots y_{j-1}$

$\Pr[y \leftarrow D_i: \text{PRED}(y_{1\ldots i-1}) = y_i] =$

$\Pr_c[y_i = c \text{ and } \text{DIST}(ycu) = 1] +$

$\Pr_c[y_j = \neg c \text{ and } \text{DIST}(ycu) = 0] =$

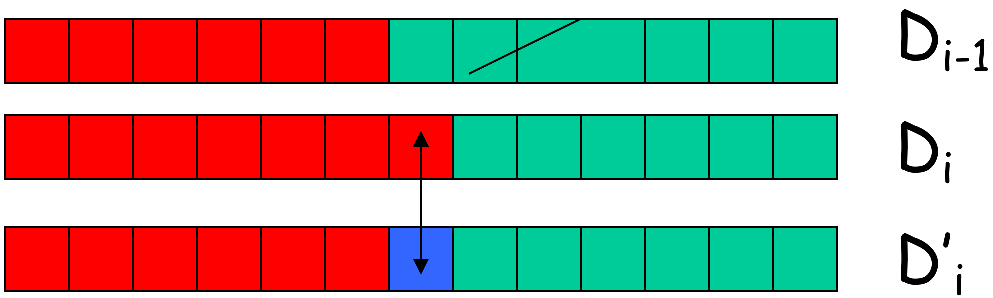
$\Pr_c[c = y_i] \Pr[\text{DIST}(ycu) = 1 | y_i = c\,] +$

$\Pr[| \neg c = y_i) \Pr[\text{DIST}(ycu) = 0 | y_i = \neg c\,|) =$

$½(p_i + (1 - p_i')) = 1/2 + 1/2(p_i - p_i') =$

$½ + ½(p_i - (2p_{i-1} - p_i)) =$

$½ + (p_i - p_{i-1}) = 1/2 + \varepsilon/m$



We used that

- $p_{i-1} = (p_i + p_i')/2$ and thus $p_i' = 2p_{i-1} - p_i$
- $p_i - p_{i-1} > \varepsilon/m$

Lets call a PRG that satisfied passes all polynomial time statistical tests a Cryptographically Strong PRG

(CSPRG)

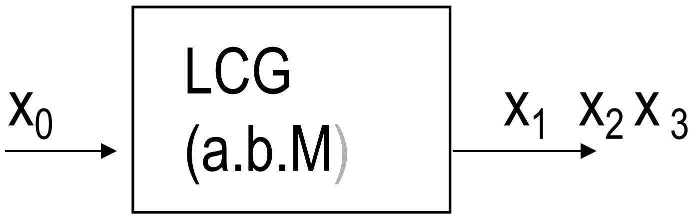
Part 2:
One-way Permutation + Hardcore Bits = Pseudorandom Generator

Linear Congruential Generators

k_0 truly random seed

$x_{i+1} = a\, x_i + b \mod M$

(where a,b,M define the generator)



Predictable !!!

Even if a,b,M unknown [Pl]

Even if truncated [FHLK]

Of course, predictability

insecurity within any crypto application as the pseudo random sequence of x_i' s can be hidden (in particular: can' t use prediction algorithms) But should raise great concern

Cryptographically Strong- PSRG from one-way **permutations**

Idea: Let f be one-way permutation.

- Choose random seed s in $\{0,1\}^n$
- Compute $f(s)\ f^2(s)\ f^3(s)\ \ldots\ f^m(s)$
- Output in reverse order

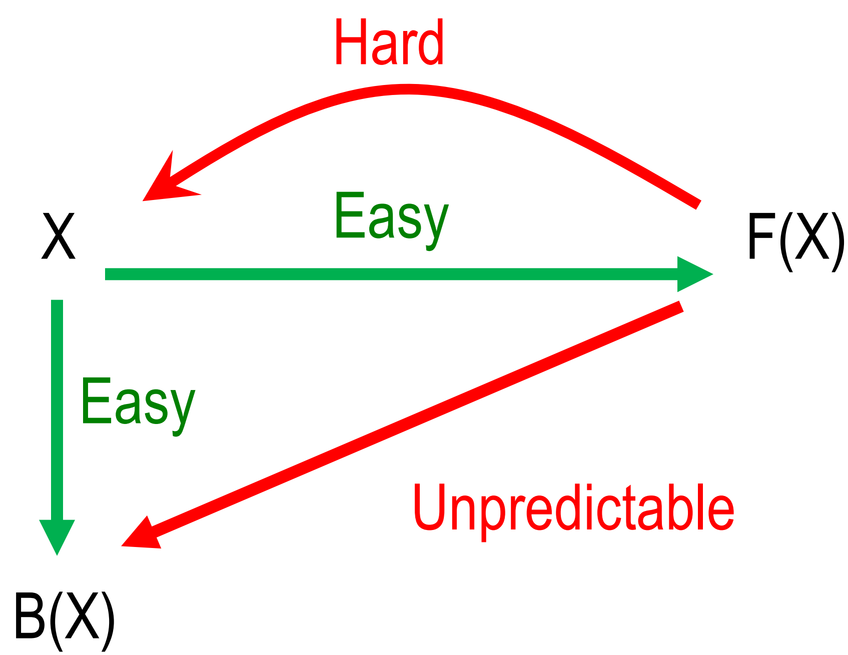
- Intuitively, Why good?
  - Unpredictable: From $f^i(s)$ can't compute $f^{i-1}(s)$
- Why not so good ?
  - Even though you cannot predict $f^{i-1}(s)$ some bits of it may be predictable.

Recall: Hard Core Predicates for OWF

DEFINITION: A hard-core predicate for a one-way function $F:\{0,1\}^* \rightarrow \{0,1\}^*$ is a Boolean predicate $B: \{0,1\}^* \rightarrow \{0,1\}$ such that:

∀ PPT algorithm **PRED** ("predictor"), there is a negligible function negl(.) such that:

Prob [PRED(f(x)) = B(x)]= ½ + negl(n)
(probability over random x in $\{0,1\}^n$ and P's coins)



Constructing PSRG

Theorem: If there exist one-way-**permutations** f with hard core bit B, then there exist

CS PRG $G:\{0,1\}^n \to \{0,1\}^{m(n)}$ for any polynomial m.

Proof: Let m be a polynomial function, set m=m(n)

On input seed s from U_n,

G: (1) compute $f(s)$ $f(f(s))$ … $f(f^{m-1}(s))$

(2) compute $B(s)$ $B(f(s))$ … $B(f^{m-1}(s))$

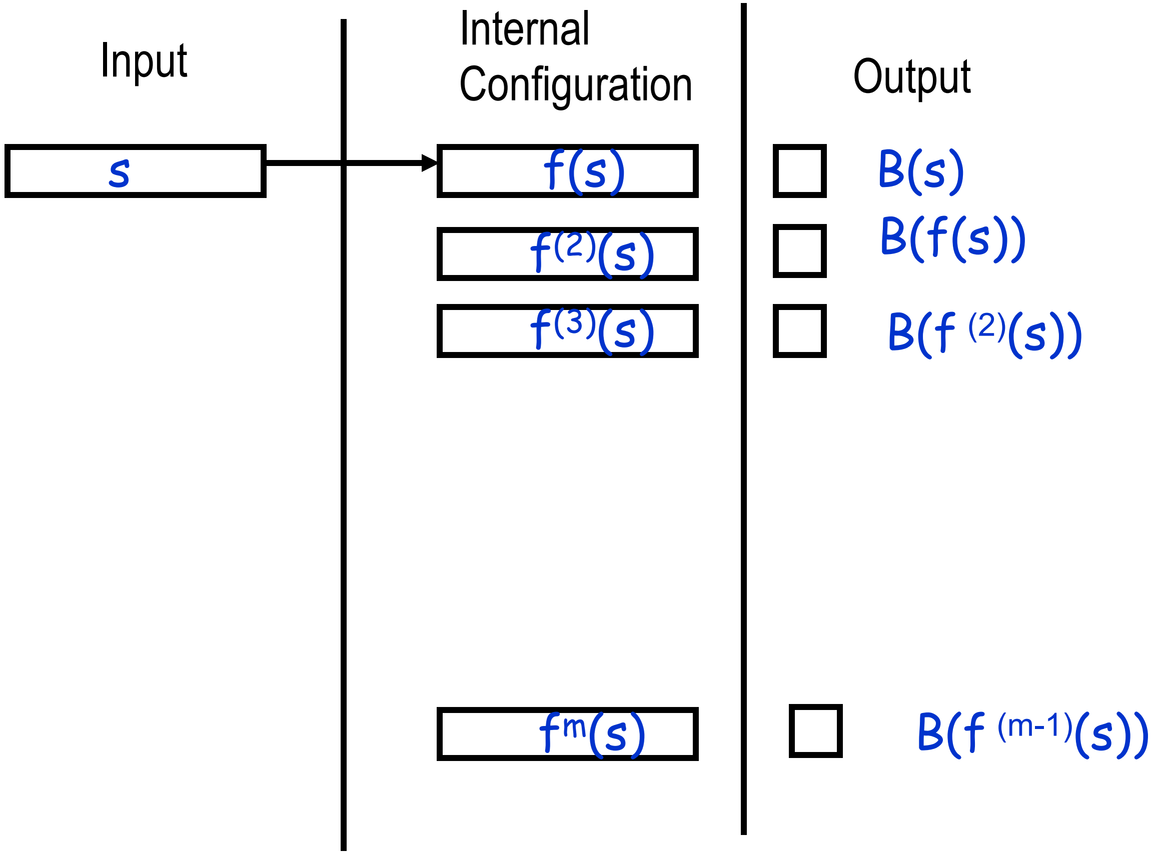
|| || ||

output y_m y_{m-1} … y_1

Note: Cost of computing i-th bit is

O(i*cost of evaluating f)

Picture Better than 1000 words



- Remark: Can make $f^m(x)$ public
  - But not any other internal state

Proof : Show outputs of G pass all next-bit tests.

Suppose, for contradiction, ∃bit location $j<m(n)$ and predictor **P** s.t. **$\Pr[\, y \leftarrow G(U_n):\ P(y_1 y_2 \ldots y_{j-1}) = y_i] > ½ + \varepsilon$**

Then show a predictor PRED for Hard Core B

PRED(f(x)):

1. compute f(x) f(f(x)) … $f(f^{j-1}(x))$
2. compute B(f(x)) … $B(f^{j-1}(x))$

   y_{j-1} (= B(f(x))) … y_1 (= $B(f^{j-1}(x))$)

3. Output $P(y_1 \ \ldots \ y_{j-1})$

EUREKA: the next bit $y_{I.}$ in the sequence should be B(f(x))

And we assumed that P predicts next bit y_i with pron. ½+ ε

Proof : Show outputs of G pass all next-bit tests.

Suppose, for contradiction, ∃bit location $j<m(n)$ and predictor **P** s.t. **Pr[$y \leftarrow G(U_n)$: P($y_1 y_2 \ldots y_{j-1}$) = y_j] > ½ +** ε

Then show a predictor PRED for Hard Core B

PRED(f(x)):

1. compute   f(x) f(f(x)) … $f(f^{j-1}(x))$
2. compute   B(f(x)) … $B(f^{j-1}(x))$

   B(f(x)) = y_{j-1} … $B(f^{j-1}(x))$ = y_1

3. Output   P(y_1 … y_{j-1})

Claim: Pr[PRED(f(x)=B(x)]=Prob[P(b_1 … b_{j-1})=b_j]>½ + ε

Essential to Pf: f is a permutation ⇒y_1 … y_{j-1} is the same distribution as P is expecting and will perform well on.

We just went through A sequence of reductions

- Since B is hard-core for one-way function f Pred cannot exist

$\Rightarrow$ Next bit test P cannot exist

$\Rightarrow$ G passes all next bit tests

$\Rightarrow$ G passes all polynomial time statistical tests

$\Rightarrow$ G outputs are computationally indistinguishable from random

Recall: Every OWF Has an Associated Hard Core Bit

Theorem [GoldreichLevin]:

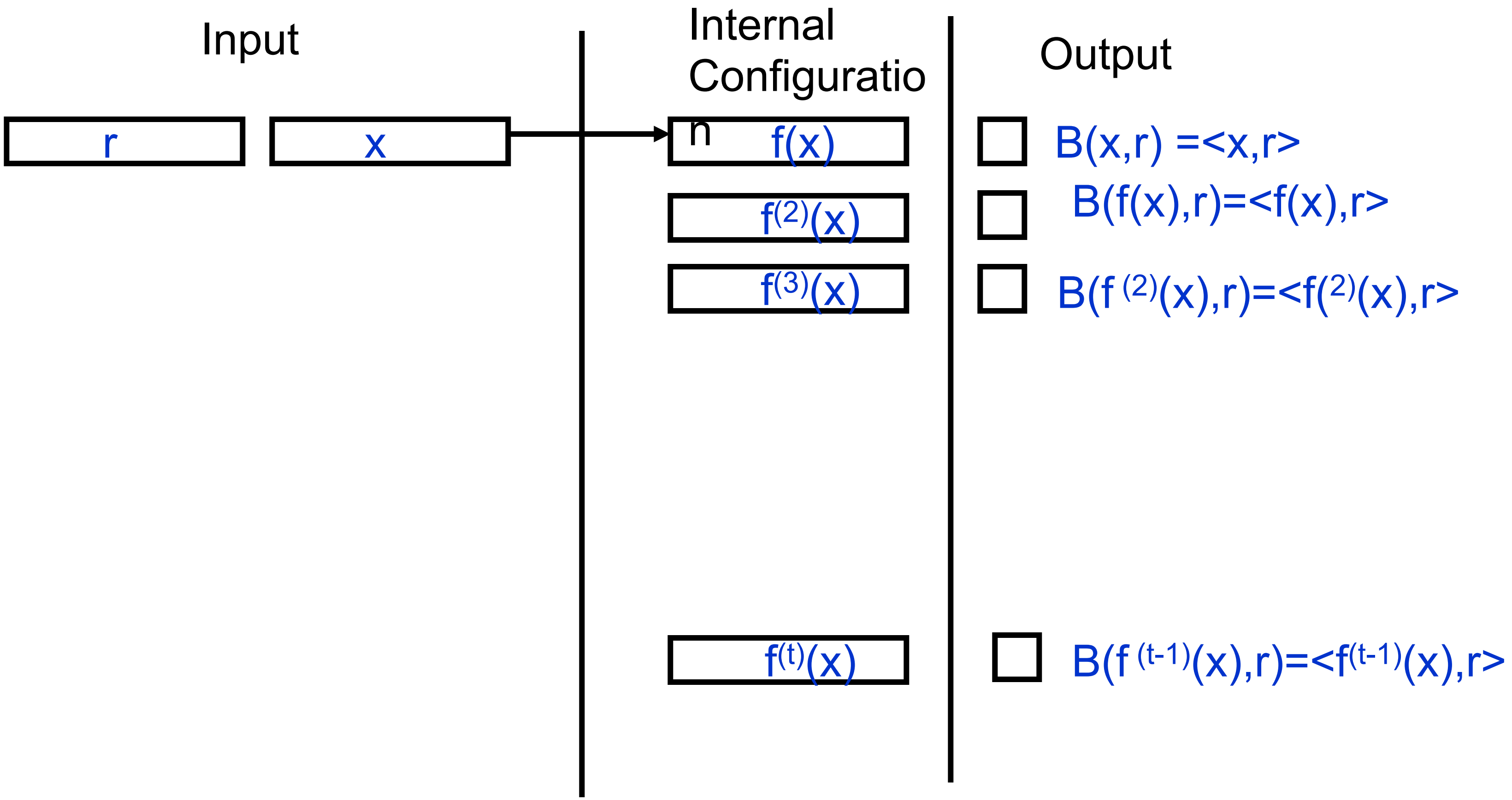
Let f be a One-way Function.

Define f'(x,r) = f(x) || r where |r|=|x|=n.

Then **B(x,r) = $\sum x_i r_i$ mod 2 = <x,r>** is a hard-core predicate for f'.

(Alternatively, {$B_r(x)$ = <x,r> mod 2}$_r$ is a collection of hardcore predicates for f_i.)

Example: Any one-way permutation based on Goldreich-Levin Hard Core Bit



- Use the same r and even can make r public

One Way **Functions vs.** One Way **Permutations**

Theorem: If ∃**one-way-functions** ,

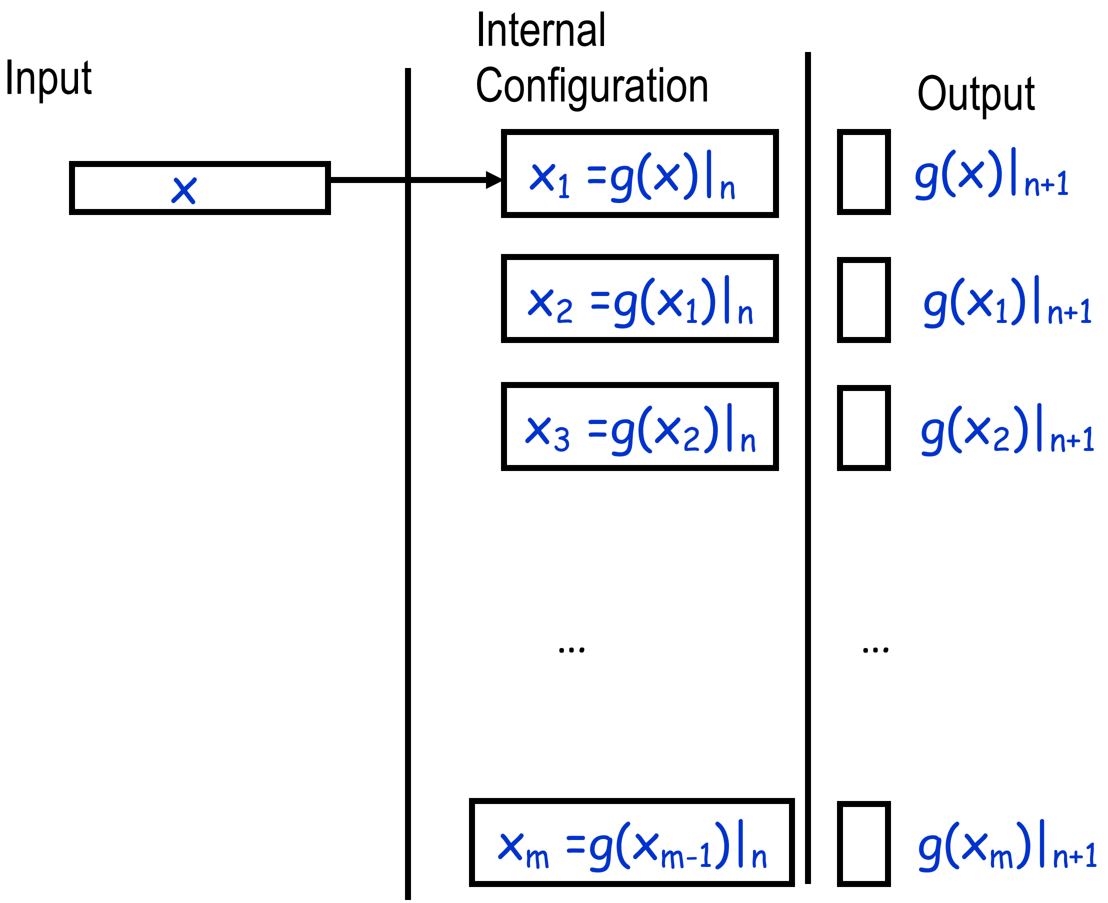
then ∃CS-PSRG

G:$\{0,1\}^n$->$\{0,1\}^{P(n)}$ for any polynomial P.

Proof: Much Harder

See web site [HILL]

More Generally: CS PRG with a Single bit extension can be converted to many bit extension (same proof idea)



Building Block:

Single Bit Expanding

CS-PSRG

$g:\{0,1\}^n \mapsto \{0,1\}^{n+1}$

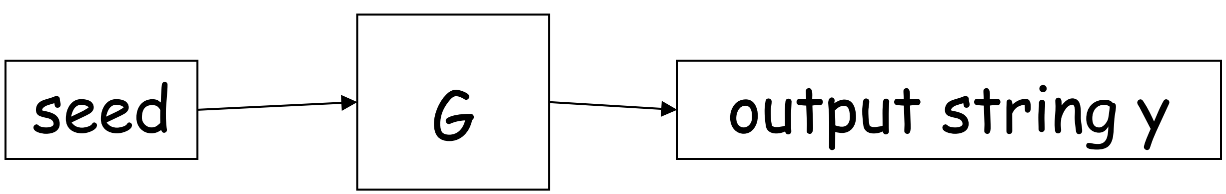
- Question: what are the hybrids you would define to prove that this works?

Application: De-randomization

- Goal: simulate BPP in sub-exponential time
- Recall: $L \in \mathbf{BPP}$ implies $\exists$algorithm M

$$x \in L \Rightarrow \Pr_{\text{coins } y}[M(x,y) \text{ accepts}] > 2/3$$
$$x \notin L \Rightarrow \Pr_{\text{coins } y}[M(x,y) \text{ rejects}] > 2/3$$

- Use Pseudo-Random Generator (PRG) to generate randomness y:



Run M(x,y)

Theorem: if one way functions exist, then $BPP \subseteq \cap_{\varepsilon>0} DTIME\left(2^{n^{\varepsilon}}\right)$

Given L in BPP

Convert BPP algorithm M for L into M ‘:

- On n-bit input x, say M uses $n' = n^c$ bits of randomness
- Let $m = n^{\varepsilon}$
- Take CS-PRG $G:\{0,1\}^m \longrightarrow \{0,1\}^{n'}$
- Output majority{M(x,G(s)): s of length m}

Observation 1:

Runtime of M’ is $O\left(2^{n^{\varepsilon}}\right)$

Theorem: if f one-way function, then $BPP \subseteq \cap_{\varepsilon>0} DTIME(2^{n^{\varepsilon}})$

Convert BPP algorithm M into M':

- On n-bit input x, say M uses n' bits of randomness
- Let $m = n^{\varepsilon}$
- Take CS-PRG $G:\{0,1\}^m \rightarrow \{0,1\}^{n'}$
- Output the majority{M(x,G(s)): s of length m}

Proof:

Suppose not. ∃L & ε s.t. for inf. many n

Case 1: ∃x in L but M' (x) rejects which means that M(x,y) behaves differently

when using true randomness y (>2/3 of M(x,y) accept) vs.

when using pseudo-random y= G(s) (<1/2 of M(x,y) accept)

⇒ M(x,) is a distinguisher between true randomness and G(s)

Case 2: ∃x not in L which is accepted by M' (x), then argue similarly….

Theorem: if f one-way function, then $BPP \subseteq \cap_{\varepsilon>0} DTIME(2^{n^{\varepsilon}})$

Proof (continued)

Use M as a distinguisher between $U_{n'}$ and $G(U_m)$.

Hardwire x to M get distinguisher D_x (y)= M(x,y) that

On input y can distinguish if $y=G(U_m)$ or $y= U_{n'}$

•$x \in L \Rightarrow \Pr[D_x(U_{n'})=1] \geq 2/3$, but $\Pr[D_x(G(U_m)) = 1] < 1/2$

Namely: if D_x (y) =1, conclude y random else pseudo-random

•$x \notin L \Rightarrow \Pr[D_x(U_{n'}) = 1] \leq 1/3$, but $\Pr[D_x(U_m) = 1] > 1/2$

Namely, If D_x (y)= 1, conclude y pseudo=random else random

Simulating **BPP** in sub-exponential time

Proof (remarks)

D_x is a non-uniform algorithm (also called a circuit)

Sequence of algorithms, one for each length n for which there exists x of length n on which M and M' behave differently.

Contradicts the fact that f is a one-way function with respect to non-uniform algorithms

Application: Symmetric Encryption for long messages with short keys

Let G be CS-PRG which stretches n to l(n)-bits

based on one-way function f.

Key Generation Gen(1^n): randomly chose n-bit seed *s* in the domain of one-way function f

Encryption Enc(m): for l(n)-bit message m

–compute G (*s*) , Send c=G(s) $\oplus$m

Decryption D(c):

–compute G(s), let m=c$\oplus$G(s)

Claim: Computational Secrecy

Proof: G(s) $\approx_c$ uniform implies

c=m$\oplus$G(s) $\approx_c$ uniform (for any m you can find)

Stateful encryption for many messages:

Let G be CS-PSRG which stretches n to l(n)-bits

based on one-way function f.

Gen(1^n): randomly chose n-bit seed *s* in the domain of one-way function f . Initialize state i=0

Enc(m_i):

–compute and send c="ith block of G(s)" $\oplus m_i$

–set i=i+1

Dec(c_i):

–set m_i= "ith block of G(s)" $\oplus$c

–Set i=i+1

Need to maintain state. Is that inherent?

Questions:

Can you access directly the i-th block output of G?

Can you do Stateless Encryption of many messages?