Chapter 5

Pseudorandomness

5.1 Introduction

Our computers use randomness every day, but what exactly is randomness? How does your computer get this randomness? Some common sources of randomness are key-strokes, mouse movement, and power consumption, but the amount of randomness generated by these isn't a lot, and often, especially in the context of cryptography, a lot of randomness is required. We shall see concrete instances of its usage in the subsequent chapters.

This brings us to the fundamental question: Can we "expand" a few random bits into many random bits? There are many heuristic approaches to this, but this isn't good enough for cryptography. We need bits that are *"as good as truly random bits"* (to a PPT adversary). This isn't very precise, so let's define it formally.

5.2 Computational Indistinguishability and Prediction Advantage

Suppose we have n uniformly random bits, $x = x_1 \| \cdots \| x_n$, and we want to find a deterministic polynomial time algorithm G that outputs $n+1$ bits: $y = y_1 \| \cdots \| y_{n+1}$ and looks *"as good as"* a truly random string $r = r_1 \| \cdots \| r_{n+1}$. We call such a $G : \{0,1\}^n \mapsto \{0,1\}^{n+1}$ a **pseudorandom generator**, or PRG for short.

But what does *"as good as"* really mean? Intuitively it means that there should be no obvious patterns and that it should pass all statistical tests a truly random string would pass (all possible k-length substrings should occur equally). But the key point is that we only need to address our adversary, so as long as there is no efficient test that can tell $G(x)$ and r apart, then that is enough!

This gives us the notion of **computational indistinguishability** of $\{x \leftarrow\!\!\$\ \{0,1\}^n\ :\ G(x)\}$ and $\{r \leftarrow\!\!\$\ \{0,1\}^{n+1}\ :\ r\}$. We will use this notion and an equivalent one called prediction advantage in order to define pseudorandomness. Then, we will devise a complete test for pseudorandom distributions (next-bit prediction), and examine pseudorandom generators.

First, we will have to define some terms.

Definition 17 (Ensemble). *A sequence $\{X_n\}_{n\in\mathbb{N}}$ is called an **ensemble** if for each $n \in \mathbb{N}$, X_n is a probability distribution (Definition 3) over $\{0,1\}^*$.*

Typically, we will consider X_n to be a distribution over the binary strings $\{0,1\}^{\ell(n)}$, where $\ell(\cdot)$ is a polynomial.

Now, let us try to define computational indistinguishability. This captures what it means for distributions X, Y to "look alike" to any efficient test. In other words, no non-uniform PPT "distinguisher" algorithm $\mathcal{D}$ can tell X and Y apart, or the behavior of $\mathcal{D}$ on X and Y is the same.

We can try to think of this as a game of sorts. Let's say we give $\mathcal{D}$ a sample of X. Then, $\mathcal{D}$ gains a point if it says the sample is from X, and loses a point if it says the sample is from Y. Then we can encode $\mathcal{D}$'s output with one bit. In order for X and Y to be indistinguishable, $\mathcal{D}$'s average score on a sample of X should be basically the same as its average score on a sample of Y.

$$\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] \approx \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]$$

or

$$\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \leq \mu(n)$$

for negligible $\mu(\cdot)$.

It should be noted that in our definition, the distinguisher $\mathcal{D}$ will only get a *single* sample. This brings us to the formal definition of **computational indistinguishability**.

Definition 18 (Computational Indistinguishability). *Two ensembles of probability distributions $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are said to be* computationally indistinguishable *if for every non-uniform PPT distinguisher $\mathcal{D}$ there exists a negligible function $\nu(\cdot)$ such that*

$$\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \leq \nu(n).$$

We can see that this formalizes the notion of a PRG if we let X be the distribution over the PRG outputs and Y be the uniform distribution over strings of the same length as PRG outputs.

But, there is actually another model for the same idea! If we give $\mathcal{D}$ a sample from either X or Y, and ask it to identify which distribution it is from, if $\mathcal{D}$ is not right with probability better than $\frac{1}{2}$, then X and Y look the same to it! Since any adversary can trivially win with probability $\frac{1}{2}$ by guessing without looking at the sample received, we want to quantify how well an adversary does beyond simply guessing. It will be convenient to change notation a bit and set $X^{(1)} = X, X^{(0)} = Y$.

Definition 19 (Prediction Advantage). *Prediction Advantage is defined as*

$$\max_{\mathcal{A}} \left| \Pr\left[b \leftarrow\$ \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(1^n, t) = b \right] - \frac{1}{2} \right|.$$

As before, we want the prediction advantage to be negligible. Let us depict the above game pictorially as a game between a *Challenger* and an adversary $\mathcal{A}$. We say that $\mathcal{A}$ wins if the *Challenger* outputs value 1. From the prior discussion,

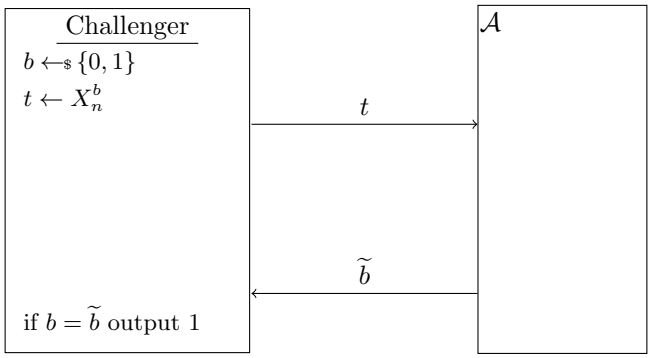


Figure 5.1: Indistinguishability Game between Challenger and adversary $\mathcal{A}$.

it is clear that probability that the challenger outputs 1 is,

$$\Pr[\text{ Challenger outputs 1}] = \Pr\left[b \xleftarrow{\$} \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(t) = b \right].$$

Proposition 1. *Prediction advantage is equivalent to computational indistinguishability.*

Proof Let's write out the prediction advantage without taking the maximum over all $\mathcal{A}$,

$$\begin{aligned}
&\left|\Pr\left[b \leftarrow \{0,1\}; z \leftarrow X^{(b)}; \mathcal{D}(1^n, z) = b\right] - \frac{1}{2}\right| \\
&= \left|\Pr_{x \leftarrow X^b}\left[\mathcal{D}(x) = b \;\middle|\; b = 1\right] \cdot \Pr[b = 1] + \Pr_{x \leftarrow X^b}\left[\mathcal{D}(x) = b \;\middle|\; b = 0\right] \cdot \Pr[b = 0] - \frac{1}{2}\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] + \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 0] - 1\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] - \left(1 - \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 0]\right)\right| \\
&= \frac{1}{2}\left|\Pr_{x \leftarrow X^1}[\mathcal{D}(x) = 1] - \Pr_{x \leftarrow X^0}[\mathcal{D}(x) = 1]\right|
\end{aligned}$$

So they are equivalent within a factor of 2.

In the first step of the above calculations, we use the law of total probability (see Section 2.5 and the example for the law of total probability within). □

As before, a useful notion to consider for proofs is the notion of what is means for two distributions to *not* be computationally indistinguishable.

Lemma 8 (Prediction Lemma). *Let $\left\{X_n^{(0)}\right\}_{n\in\mathbb{N}}, \left\{X_n^{(1)}\right\}_{n\in\mathbb{N}}$ be ensembles of probability distributions. Let $\mathcal{D}$ be a non-uniform PPT adversary that $\varepsilon(\cdot)$-distinguishes $\{X_n^{(0)}\}, \{X_n^{(1)}\}$ for infinitely many $n \in \mathbb{N}$. Then $\exists$ a non-uniform PPT $\mathcal{A}$ such that*

$$\Pr\left[b \xleftarrow{\$} \{0,1\}, t \leftarrow X_n^b : \mathcal{A}(t) = b\right] - \frac{1}{2} \geq \frac{\varepsilon(n)}{2}$$

for infinitely many $n \in \mathbb{N}$.

Let's now look at some properties of computational indistinguishability.

Properties of Computational Indistinguishability.

1. First, we define the notation $\{X_n\} \approx_c \{Y_n\}$ to mean computational indistinguishability. Often, when clear from context we shall drop the subscript c and denote it by $\{X_n\} \approx \{Y_n\}$.
2. If we apply an efficient algorithm on X and Y, then their images under this operation are still indistinguishable. Formally, $\forall$ non-uniform PPT M, $\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$. If this were not the case, then a distinguisher could simply use M to tell $\{X_n\}$ and $\{Y_n\}$ apart!

3. If X, Y are indistinguishable with advantage at most μ_1 and Y, Z are indistinguishable with advantage at most μ_2, then X, Z are indistinguishable with advantage at most $\mu_1 + \mu_2$. This follows from the triangle inequality, and we sketch the proof below.
We are given, for all PPT $\mathcal{D}$

$$\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \leq \mu_1(n)$$

and

$$\left|\Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \leq \mu_2(n).$$

Then we compute,

$$\begin{aligned}
&\left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \\
&= \Big|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] \\
&\quad + \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\Big| \\
&\leq \left|\Pr[x \leftarrow X;\ \mathcal{D}(1^n, x) = 1] - \Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1]\right| \\
&\quad + \left|\Pr[y \leftarrow Y;\ \mathcal{D}(1^n, y) = 1] - \Pr[z \leftarrow Z;\ \mathcal{D}(1^n, z) = 1]\right| \\
&= \mu_1(n) + \mu_2(n)
\end{aligned}$$

where we use the triangle inequality that $|a + b| \leq |a| + |b|$.

This last property is actually quite nice, and we would like to generalize it a bit. The proof follows from a simple application of the pigeonhole principle and triangle inequality.

Lemma 9 (Hybrid Lemma). *Let $X^1, \ldots, X^m$ be distribution ensembles for $m = \mathrm{poly}(n)$. Suppose $\mathcal{D}$ distinguishes X^1 and X^m with advantage ε. Then $\exists i \in \{1, \cdots, m-1\}$ such that $\mathcal{D}$ distinguishes X^i, X^{i+1} with advantage $\geq \frac{\varepsilon}{m}$.*

The pigeonhole principle states that if there are n pigeons and m holes such that $n > m$, then there must be a hole with more than one pigeon. While this may

Try to see where the Hybrid Lemma uses the pigeonhole principle.

seem like an obvious fact, this principle has found extensive use in combinatorics and computer science.

Returning to pseudorandomness, we define a bit more notation. We call the uniform distribution over $\{0,1\}^{\ell(n)}$ by $U_{\ell(n)}$. Intuitively, a distribution is pseudorandom if it looks like a uniform distribution to any efficient test. We have the tools now to formulate this:

Definition 20 (Pseudorandom Ensembles). *An ensemble $\{X_n\}$, where X_n is a distribution over $\{0,1\}^{\ell(n)}$ is said to be pseudorandom if*

$$\{X_n\} \approx_c \{U_{\ell(n)}\}.$$

This is relevant for PRGs, as their outputs should be pseudorandom.

5.3 Next-Bit Test

In this section, we consider an interesting way to characterize pseudorandomness. For a string to be considered pseudorandom, we know that at the very least it must pass all efficient tests a true random string would pass. Let's consider one such natural test, which we shall call the "next bit unpredictability". For a truly random string, given any prefix of this string, it is not possible to predict the "next bit" with probability better than $\frac{1}{2}$. Let us extend this notion to an arbitrary string. We say a sequence of bits *passes the next-bit test* if no *efficient* adversary can predict "the next bit" in the sequence with probability better than $\frac{1}{2}$ even given all previous bits of the sequence. Let us formalize this below.

Definition 21 (Next-bit Unpredictability). *An ensemble of distributions $\{X_n\}$ over $\{0,1\}^{\ell(n)}$ is next-bit unpredictable if, $\forall\ 0 \leq i < \ell(n)$, $\forall$ non-uniform PPT adversaries $\mathcal{A}$, $\exists$ negligible function $\nu(\cdot)$ such that $\forall n \in \mathbb{N}$,*

$$\Pr\Big[t = t_1 \cdots t_{\ell(n)} \leftarrow X_n : \mathcal{A}(t_1 \cdots t_i) = t_{i+1}\Big] \leq \frac{1}{2} + \nu(n).$$

When presented with such a definition, an instructive exercise is to consider what it means for an ensemble to *not* be next-bit unpredictable. This will be useful when we discuss proof by reduction.

An ensemble $\{X_n\}$ is *not* next-bit unpredictable if there exists an index i, an

adversary $\mathcal{A}_{\text{next-bit}}$ and a polynomial n such that for infinitely many n,

$$\Pr\Big[t = t_1 \cdots t_{\ell(n)} \leftarrow X_n : \mathcal{A}_{\text{next-bit}}(t_1 \cdots t_i) = t_{i+1}\Big] \geq \frac{1}{2} + \frac{1}{\mathsf{p}(n)}.$$

We will refer to $\frac{1}{\mathsf{n}}$ as the advantage $\mathcal{A}_{\text{next-bit}}$ with respect to randomly guessing.

As before, the figure below illustrates a game for the above definition.

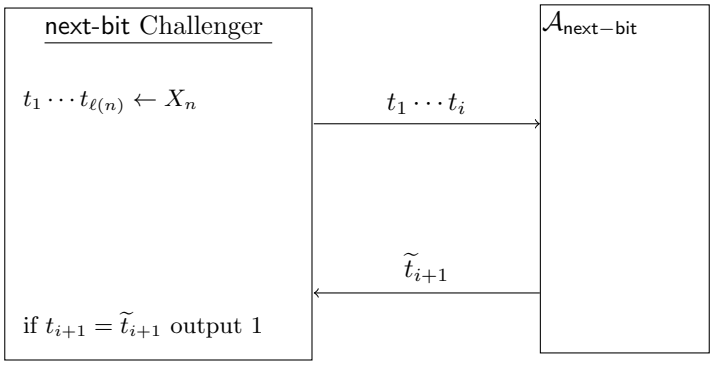


Figure 5.2: Next bit unpredictability

This intuitive test turns out to be really powerful, as we show below. As we show, the next-bit test can in fact be used to test *pseudorandomness*.

Theorem 10 (Completeness of the Next-bit Test). *If $\{X_n\}$ is next-bit unpredictable then $\{X_n\}$ is pseudorandom.*

Proof This will be our first example where we use the Hybrid Lemma (Lemma 9) in a proof. This will be an invaluable tool through this course, and therefore we shall go over this proof carefully. In subsequent proofs, we might skip some of the details with the expectation that the reader will complete them.

Let us assume to the contrary that $\{X_n\}$ is pseudorandom. From Definition 20, we have that there exists a non uniform PPT distinguisher $\mathcal{D}$, and a polynomial $p(\cdot)$ s.t. for infinitely many $n \in \mathbb{N}$, $\mathcal{D}$ distinguishes X_n and $U_{l(n)}$ with probability $\frac{1}{p(n)}$. Our goal will be to use the above distinguisher D to construct an adversary $\mathcal{A}_{\text{next-bit}}$ for the next-bit unpredictability.

Let us consider the following distributions, that we shall refer to as **hybrid distributions** or **hybrids** for short. For $i \in [\ell(n)]$, we define

$$\mathsf{H}_n^i := \Big\{x \leftarrow X_n, u \leftarrow U_{\ell(n)} : x_1 ... x_i u_{i+1} u_{i+2} ... u_{\ell(n)}\Big\}$$

where $U_{\ell(n)}$ is the uniform distribution.

Note that the first hybrid H_n^0 is the uniform distribution $U_{\ell(n)}$, and the last hybrid $\mathsf{H}_n^{\ell(n)}$ is the distribution X_n. We show the transition of the hybrids pictorially in Figure 5.3, where the darker color indicates a random bit, and the lighter a pseudorandom bit.

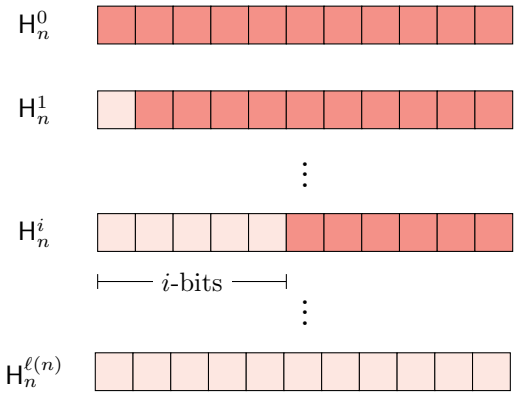


Figure 5.3: Progression of Hybrids

Think of why this is true. What would happen if it for all values of $i \in [\ell(n)]$, the probability was smaller than $\frac{1}{p(n)\ell(n)}$?

Thus, rewriting our initial claim regarding distinguisher $\mathcal{D}$ with respect to the hybrid distributions, $\mathcal{D}$ distinguishes between H_n^0 and $\mathsf{H}_n^{\ell(n)}$ with probability $\frac{1}{p(n)}$.

By the hybrid lemma (Lemma 9), $\exists$ some $i \in [0, \ell(n)]$ s.t. $\mathcal{D}$ distinguishes between H_n^i and H_n^{i+1} with probability at least $\frac{1}{p(n)\ell(n)}$.

The only difference between H^{i+1} and H^i is that in H^{i+1}, $(i+1)^{th}$ bit is x_{i+1}, and in H^i, it is u_{i+1}.

Let's define another distribution $\tilde{\mathsf{H}}_n^i$:

$$\tilde{\mathsf{H}}_n^i = \left\{ x \leftarrow X_n, u \leftarrow U_{\ell(n)} : x_1 ... x_{i-1} \bar{x_i} u_{i+1} u_{i+2} ... u_{\ell(n)} \right\},$$

where $\bar{x_i} = 1 - x_i$. This is identical to H_n^{i+1} with the $i+1$-th bit flipped.

Since H_n^{i+1} can be sampled from either H_n^i or $\tilde{\mathsf{H}}_n^i$ with equal probabilities,

$$\begin{aligned}
&\left|\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] - \Pr[t \leftarrow \mathsf{H}_n^{i+1} : \mathcal{D}(t) = 1\right]\Big| \\
&= \Bigg| \Pr\left[t \leftarrow \mathsf{H}_n^{i+1} : \mathcal{D}(t) = 1\right] - \\
&\qquad \left(\frac{1}{2}\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] + \frac{1}{2}\Pr\left[t \leftarrow \tilde{\mathsf{H}}_n^i : D(t) = 1\right]\right) \Bigg| \\
&= \frac{1}{2}\left|\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] - \Pr\left[t \leftarrow \tilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\right]\right|.
\end{aligned}$$

The observation that $\mathcal{D}$ distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{p(n)\ell(n)}$ implies that $\mathcal{D}$ distinguishes H_n^{i+1} and $\tilde{\mathsf{H}}_n^{i+1}$ with probability $\frac{2}{p(n)\ell(n)}$. We shall use $\mathcal{D}$ to construct an adversary $\mathcal{A}_{\mathsf{next-bit}}$ for the next-bit Challenger. $\mathcal{A}_{\mathsf{next-bit}}$ on receiving input $t_1 \cdots t_i$ from the next-bit Challenger, needs to prepare the input for $\mathcal{D}$. It uses $t_1 \cdots t_i$, samples a random bit b, and the rest of the string is randomly generated. If $\mathcal{D}$ responds with 1, $\mathcal{A}_{\mathsf{next-bit}}$ guesses the next bit as b, else it guesses $1 - b$. This strategy is presented in Figure 5.4. Now we need to calculate the

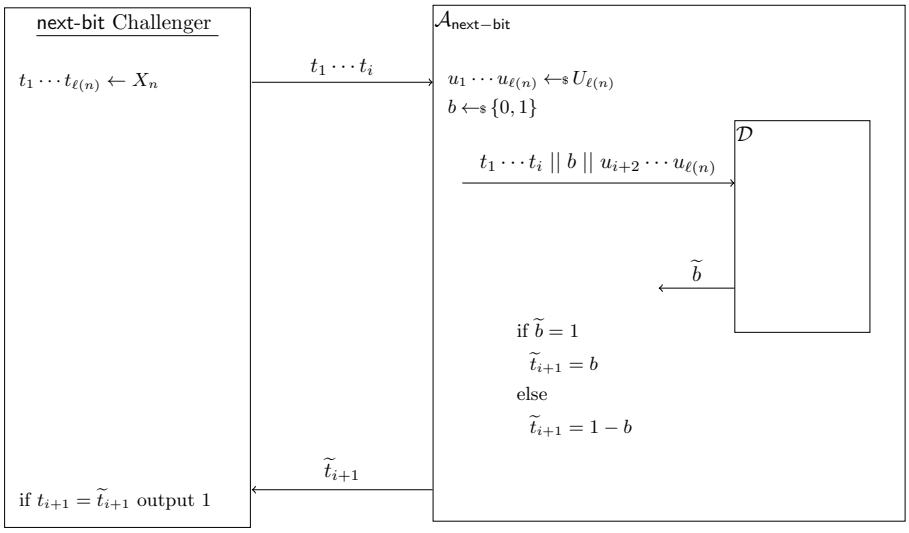


Figure 5.4: Completeness of Next-bit unpredictability

probability that $\mathcal{A}_{\mathsf{next-bit}}$ wins.

$$\begin{aligned}
\Pr[\mathcal{A}_{\mathsf{next-bit}} \text{ wins }] &= \frac{1}{2} \cdot \Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] + \frac{1}{2} \cdot \Pr\left[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) \neq 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] + \frac{1}{2} \cdot \left(1 - \Pr\left[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr\left[t \leftarrow \mathsf{H}_n^i : \mathcal{D}(t) = 1\right] - \Pr\left[t \leftarrow \widetilde{\mathsf{H}}_n^i : \mathcal{D}(t) = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{\mathsf{p}(n)\ell(n)}
\end{aligned}$$

This follows from the fact that with probability $\frac{1}{2}$ we will provide $\mathcal{D}$ with input either from H_n^i or $\widetilde{\mathsf{H}}_n^i$. We have constructed an adversary $\mathcal{A}_{\mathsf{next-bit}}$ that breaks our assumption that $\{X_n\}$ is next-bit unpredictable. Therefore it must be the case that $\{X_n\}$ is pseudorandom. □

We can now rely on next bit unpredictability to prove distributions are pseudorandom.

5.4 Pseudorandom Generators (PRG)

We have defined the notion of pseudorandomness and next-bit unpredictability. Now, we turn to our goal of constructing pseudorandom generators, that will take a small amount of randomness and expand it into a large amount of pseudorandomness.

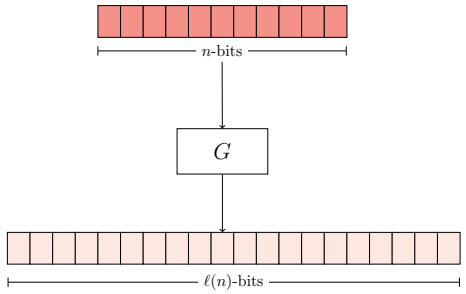


Figure 5.5: Pseudorandom Generator G

Before we can proceed with a construction, we need to define the properties we need from a pseudrandom generator.

Definition 22 (Pseudorandom Generator). *A deterministic algorithm* $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ *is called a pseudorandom generator (PRG) if:*

1. *(efficiency): the output of* G *can be computed in polynomial time*
2. *(expansion):* $|G(x)| > |x|$
3. *(pseudorandomness):* $\{x \leftarrow\!\!\$\ \{0,1\}^n : G(x)\} \approx_c \{U_{\ell(n)}\}$ *where* $\ell(n) = |G(0^n)|$

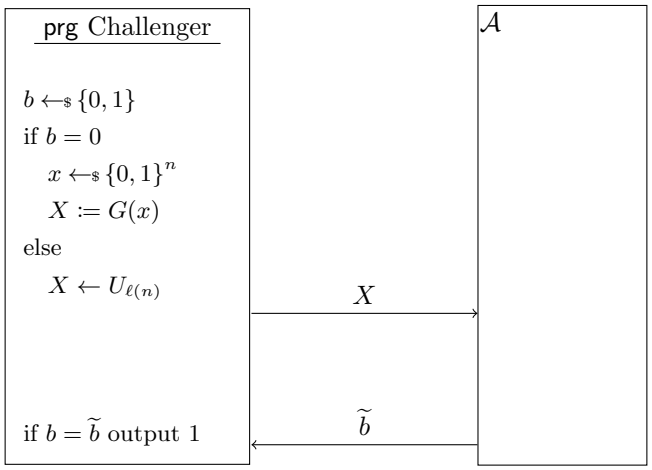


Figure 5.6: Indistinguishability Game for PRG.

A few remarks about the above definition are in order:

- **Deterministic:** It is clear that the above definition is only meaningful if G is deterministic since any additional randomness can otherwise be directly output by G.
- **Seed:** The input to the PRG is referred to as the *seed* of the PRG.
- **Randomness of the seed:** Similar to the one-way functions, the security of a PRG is defined only when the seed is chosen uniformly at random.

To evaluate the "effectiveness" of a PRG, we define the *stretch* of the PRG below.

Definition 23 (Stretch). *The stretch of* G *is defined as:* $|G(x)| - |x|$

We will first construct a PRG by 1-bit stretch, which already seems non trivial. Then, we will show how to generically transform a PRG with 1-bit stretch into one that achieves polynomial-bit stretch.

5.5 PRG with 1-bit Stretch

Armed with what we've learned so far, a natural candidate for our construction of PRG is

$$G(s) = f(s)||h(s)$$

where f is a one-way function and h is the hardcore predicate associated with f.

While $h(s)$ is indeed unpredictable even given $f(s)$, this construction is not really a PRG because of the following reasons:

- $|f(s)|$ might be less than $|s|$.
- $f(x)$ may always start with a prefix, which is not random. Indeed, OWF doesn't promise random outputs.

It turns out that PRGs can in fact be constructed from one-way functions, but the construction is quite involved. Instead, we'll settle on a slightly easier problem of constructing PRG from a one-way *permutation* (OWP) over $\{0,1\}^n$.

A one-way permutation clearly address both of the above issues:

- Since f is a permutation, the domain and range have the same number of bits, i.e., $|f(s)| = |s| = n$.
- $f(s)$ is uniformly random (not just pseudorandom) over $\{0,1\}^n$ if s is randomly chosen. In particular $\forall y$:

$$\Pr[s \leftarrow\$ \{0,1\}^n : f(s) = y] = \Pr\Big[s \leftarrow\$ \{0,1\}^n : s = f^{-1}(y)\Big] = \frac{1}{2^n}.$$

  Thus, if s is uniform, $f(s)$ is uniform.

As it turns out, our initial proposed construction for works when f is a one-way permutation. Let us now prove the following theorem:

Theorem 11 (PRG based on OWP). *Let f be a one-way permutation, and h be a hard-core predicate for f. Then $G(s) = f(s)||h(s)$ is a PRG with 1-bit stretch.*

Proof Let us assume, to the contrary, that G is not a PRG. Then from Definition 22, we know that the output of G is not pseudorandom. Since we've established that the next-bit unpredictability is complete (Theorem 10), if the output of G is not pseudorandom, it must not satisfy next-bit unpredictability.

Putting this all together, if G is not a PRG, there exists an index i, a non-uniform PPT adversary $\mathcal{A}_{\mathsf{next-bit}}$, a polynomial $\mathsf{p}(\cdot)$ such that for infinitely many

values of n,

$$\Pr[s \leftarrow\$ \{0,1\}^n, y_1 \cdots y_n y_{n+1} \coloneqq G(s) : \mathcal{A}_{\mathsf{next-bit}}(y_1 \cdots y_i) = y_{i+1}] = \frac{1}{2} + \frac{1}{\mathsf{p}(\cdot)}.$$

Let us take a closer look at what values i can take. From our construction, we know that if s is random, the first n bits are random and no adversary can guess any of its bits with probability better than $\frac{1}{2}$. So it must be the case that $i = n$. We will use this information to construct an adversary $\mathcal{A}_{\mathsf{hcp}}$ for the hard-core predicate. The strategy is quite simple, when $\mathcal{A}_{\mathsf{hcp}}$ is given $y \coloneqq f(x)$, this is passed along to $\mathcal{A}_{\mathsf{next-bit}}$ and the corresponding response from $\mathcal{A}_{\mathsf{next-bit}}$ is used as the response to the hcp Challenger. We describe this pictorially below:

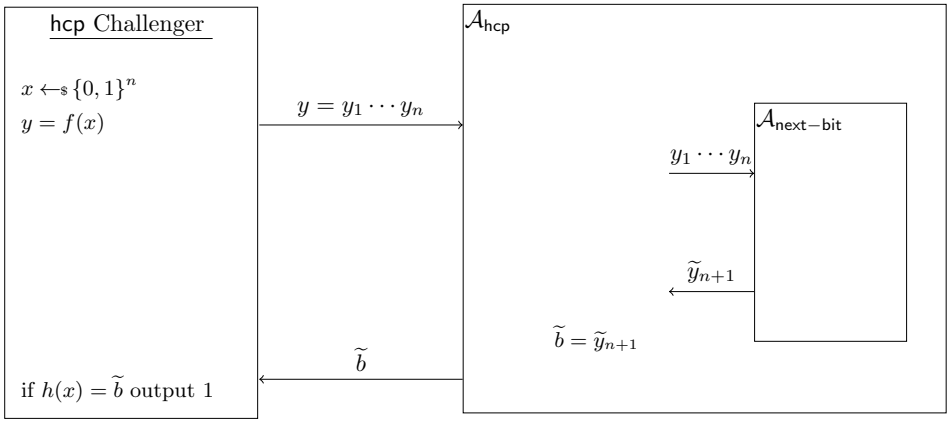


Figure 5.7: 1-bit Stretch PRG

From the description above it is clear that $\mathcal{A}_{\mathsf{hcp}}$ succeeds if and only if $\mathcal{A}_{\mathsf{next-bit}}$ succeeds. Therefore, the probability $\mathcal{A}_{\mathsf{hcp}}$ succeeds is given by

$$\Pr[\mathcal{A}_{\mathsf{hcp}} \text{ wins }] = \Pr[\mathcal{A}_{\mathsf{next-bit}} \text{ wins }] \geq \frac{1}{2} + \frac{1}{\mathsf{p}(n)}.$$

This contradicts the assumption that h is a hard-core predicate for f. Thus, it must be the case that G is a PRG. □

5.6 PRG with Poly-Stretch

To be useful, we will need to go beyond a 1-bit stretch, to any arbitrary polynomial. From our previous section, why stop at 1 bit? Why don't we repeatedly apply the same procedure. Indeed, this will be how we will construct our PRG.

See Figure 5.8 for visual representation of this idea, where we collect the last bit from each iteration and this forms the output of the PRG G. The first n bits from each output are fed into the next iteration of G.

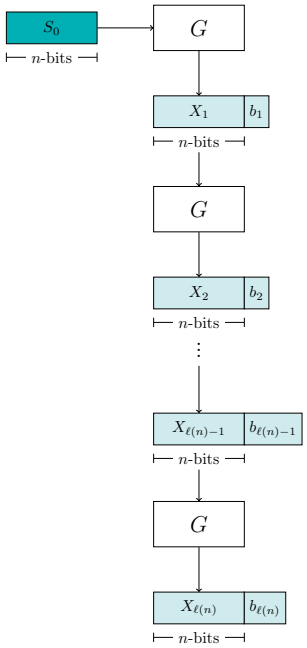


Figure 5.8: PRG with polynomial Stretch from PRG with a 1-bit Stretch.

We formally prove the lemma below.

Lemma 12. *Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. For any polynomial ℓ, $G' : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is defined as:*

$$G'(s) = b_1 ... b_{\ell(n)} \text{ where}$$
$$X_0 := s$$
$$X_{i+1} \| b_{i+1} := G(X_i)$$

Then, G' is a PRG.

Proof We first establish some notation. Let $G'(s) = G^{\ell(n)}(s)$, where

$$G^0(x) = \epsilon$$
$$G^i(x) = b || G^{i-1}(x') \text{ where } x' || b := G(x)$$

and ϵ denotes the empty string.

Let us assume, to the contrary, that G' is not a PRG. Then, from Definition 22, there exists a distinguisher $\mathcal{D}$, and a polynomial $\mathsf{p}(\cdot)$ such that for infinitely many n, $\mathcal{D}$ distinguishes

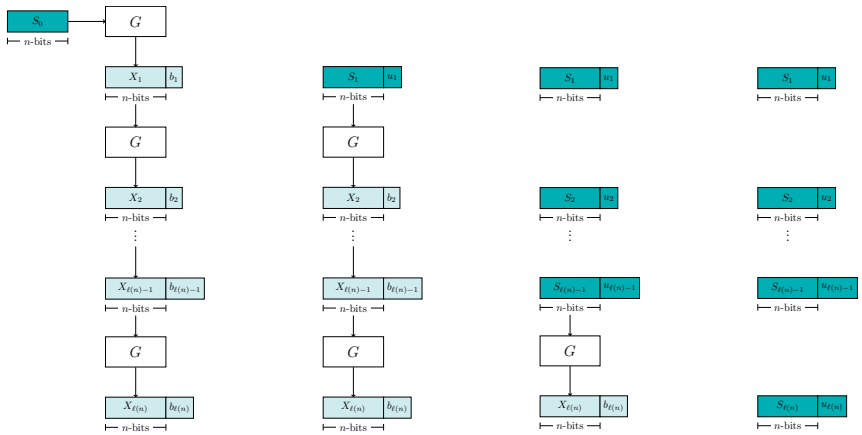
$$\left\{U_{\ell(n)}\right\} \text{ and } \{G'(U_n)\}$$

with non-negligible probability $\frac{1}{p(n)}$.

We shall use the Hybrid Lemma (Lemma 9) as the main tool for our proof. To do so, let us define the following hybrid distributions $\forall i \in [\ell(n)]$:

$$\mathsf{H}_n^i = U_i \| G^{\ell(n)-1}(U_n),$$

i.e. in hybrid H_n^i, the first i bits are random, while the rest are obtained as in the process above. Figure 5.9 gives a visual depiction of these hybrids. Then, $\mathsf{H}_n^0 =$



(a) H_0 (b) H_1 (c) $\mathsf{H}_{\ell(n)-1}$ (d) $\mathsf{H}_{\ell(n)}$

Figure 5.9: Hybrids in

$G^{\ell(n)}(U_n)$ and $\mathsf{H}_n^{\ell(n)} = U_{\ell(n)}$ and $\mathcal{D}$ distinguishes H_n^0 and $\mathsf{H}_n^{\ell(n)}$ with probability $\frac{1}{\mathsf{p}(n)}$.

By the hybrid lemma, for infinitely many n, there exists index i such that $\mathcal{D}$ distinguishes H_n^i and H_n^{i+1} with probability $\frac{1}{\ell(n)\mathsf{p}(n)}$. Since $\ell(n)$ is polynomial, so is $\ell(n)\mathsf{p}(n)$

Note that the only difference between H_n^i and H_n^{i+1} can be seen by writing

out the hybrids explicitly

$$\begin{aligned}\mathsf{H}_n^i &= U_i \| G^{\ell(n)-i}(U_n) \\ &= U_i \| b \| G^{\ell(n)-i-1}(x) \\ \mathsf{H}_n^{i+1} &= U_{i+1} \| G^{\ell(n)-i-1}(U_n) \\ &= U_i \| U_1 \| G^{\ell(n)-i-1}(U_n)\end{aligned}$$

where $x \| b \coloneqq G(U_n)$. Thus, only the $i+1$-th bit either comes from a PRG output or is truly random. Now we shall use the adversary $\mathcal{D}$ to build an adversary $\mathcal{A}_{\mathsf{prg}}$ that breaks the pseudorandomness of G. $\mathcal{A}_{\mathsf{prg}}$ gets as input either a random $n+1$ bit string, or an output of a PRG of the same length. The strategy is to provide to $\mathcal{D}$ as input something from either H_n^i or H_n^{i+1} using the inputs $\mathcal{A}_{\mathsf{prg}}$ receives. From the construction, and the description of the hybrids, it is clear that we should use the input to $\mathcal{A}_{\mathsf{prg}}$ for the $i+1$-th bit for the input to $\mathcal{D}$. Formally, this is shown in the diagram below, If $b = 0$, then the input constructed by $\mathcal{A}_{\mathsf{prg}}$ for $\mathcal{D}$ is identical

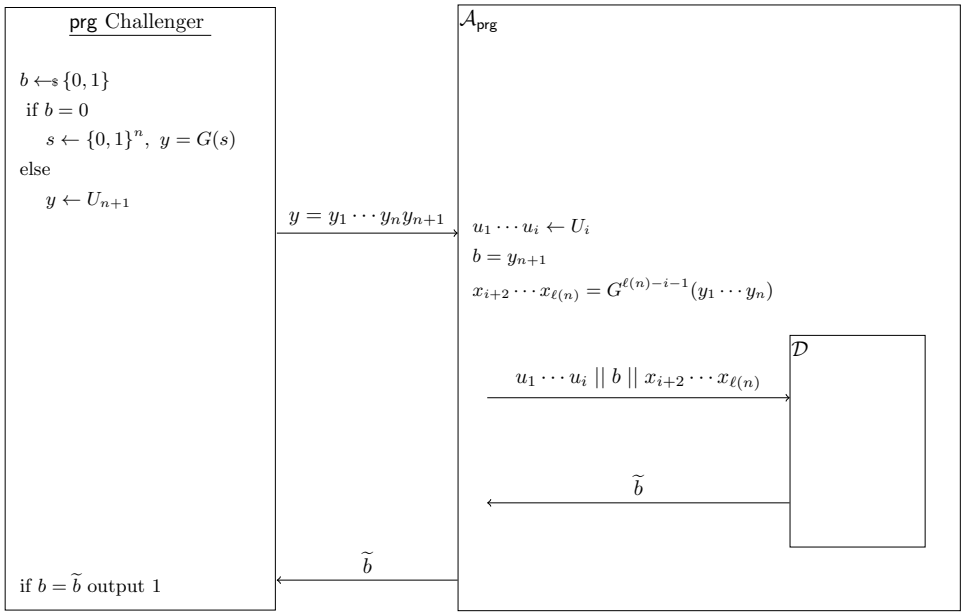


Figure 5.10: 1-bit to poly-bit stretch PRG

to H_n^i, while if $b = 1$, then it is identical to H_n^{i+1}. Therefore, the value $\widetilde{b}$ returned by $\mathcal{D}$ also indicates whether the input to $\mathcal{A}_{\mathsf{prg}}$ was output of a PRG or random. In

fact $\mathcal{A}_{\mathsf{prg}}$ has the same probability of success as $\mathcal{D}$. Therefore,

$$\Pr[\mathcal{A}_{\mathsf{prg}} \text{ wins }] = \Pr[\mathcal{D} \text{ wins }] \geq \frac{1}{2} + \frac{1}{\ell(n)\mathsf{p}(n)}.$$

Since G runs in polynomial time, so does $\mathcal{A}_{\mathsf{prg}}$. Therefore our constructed $\mathcal{A}_{\mathsf{prg}}$ contradicts the assumption that G is a PRG. It must then be the case that G' is a PRG.

What happened if we got greedy and tried to output say X_1? Would it still be secure?

□

5.7 Pseudorandom Functions (PRF)

Going beyond Poly Stretch. PRGs can only generate polynomially long pseudorandom strings. What if we want exponentially long pseudorandom strings? How can we efficiently generate them? One way to do this is by using functions that index exponentially long pseudorandom strings.

Towards that end, let us start by looking at the notion of a random function. Consider a function $f : \{0,1\}^n \mapsto \{0,1\}^n$. As we've seen in Example 2, the total number of functions that map n bits to n bits is 2^{n2^n}.
To define a random function, we can use one of the two methods:

1. Select a random function F *uniformly at random* from all 2^{n2^n} functions that map n bits to n bits
2. Use a randomized algorithm to describe the function. This is called *lazy sampling*, and often more convenient implementation for proofs. Specifically, we describe below a randomized program M that maintains a table $\mathcal{T}$ to mimic a random function.

$M(1^n)$

1 : $\mathcal{T} = \emptyset$
2 : **input** x
3 : **if** $\exists y'$ s.t. $(x, y') \in \mathcal{T}$
4 : **output** y'
5 : **else**
6 : $y \leftarrow\$ \{0,1\}^n$
7 : **add** (x, y) **to** $\mathcal{T}$
8 : **output** y

Note that the distribution of M's output is identical to that of a random function F.

Truly random functions are huge random objects. Neither of the methods allows us to store the entire function efficiently. But with the second method, M will only need polynomial space and time to store and query $\mathcal{T}$, if one is limited to making only *polynomial* calls to the random function.

We said we would use these functions to index exponentially long pseudorandom strings. We shall come to pseudorandom aspect shortly, but these terms make sense even for exponentially long random strings. We've seen that the size of function itself was large, it may not be immediately obvious what the exponentially long string or index corresponds to with respect to the functions. We will think of the exponentially long string to be all the concatenation of the output strings of the random function F, i.e. the string is $F(0)||F(1)||\cdots||F(2^n)$. Given that each output is n-bits, the size of the string is $n \cdot 2^n$. With this view, it is easy to see what the index corresponds to, it is simply the input to the function F, which returns an n bit chunk of the long string.

Now that we've seen a random function, what is a pseudorandom function (PRF). A PRF "looks" like a random function, but unlike a random function it can be described using only *polynomially* many bits. At first, it seems like a good idea to use computational indistinguishability to make PRF "look like" a random function. However, there are issues to this approach that we expand upon below:

Issue: Since the description of a random function is of exponential size, $\mathcal{D}$ might not even be able to read the input efficiently in case of random function and can easy tell the difference just be looking at the size of input.

Suggested Solution: What if $\mathcal{D}$ is not given the description of functions as input, but is instead allowed to only query the functions on inputs of its choice, and view the output. But the question here is whether the entire implementation of the PRF is hidden from the distinguisher or only a part of it.

Keeping the entire description of PRF secret from $\mathcal{D}$, is similar to providing security by obscurity which in general is not a good idea according to Kerchoff's principle. Therefore in accordance with the principle, we use the notion of keyed functions. It is better to define PRFs as keyed functions. So only the key remains secret while the PRF evaluation algorithm can be made public.

Since this is the first time we are talking about a keyed function/function family, we can think of functions $F(\cdot,\cdot)$ that take in two inputs, a key (or index of the family) k and the input x and outputs $F(k,x)$. To separate out the two types of inputs, we will find it convenient to represent keyed functions as $F_k(x)$. Once

the key or index is fixed, the function $F_k(\cdot)$ behaves like the functions we have encountered so far. We will often interchangeably use the term 'keyed function' and function family.

Now we are finally ready to talk about the security of PRFs.

5.7.1 Security of PRFs via Game Based Definition

As we have seen previously, the security is described via a game between the PRF challenger and an Adversary/Distinguisher $\mathcal{D}$. The game is described in Figure 5.11. Note that unlike previous security games, here the adversary $\mathcal{D}$ can repeatedly ask queries x_i to the challenger before it decides to output the bit $\widetilde{b}$.

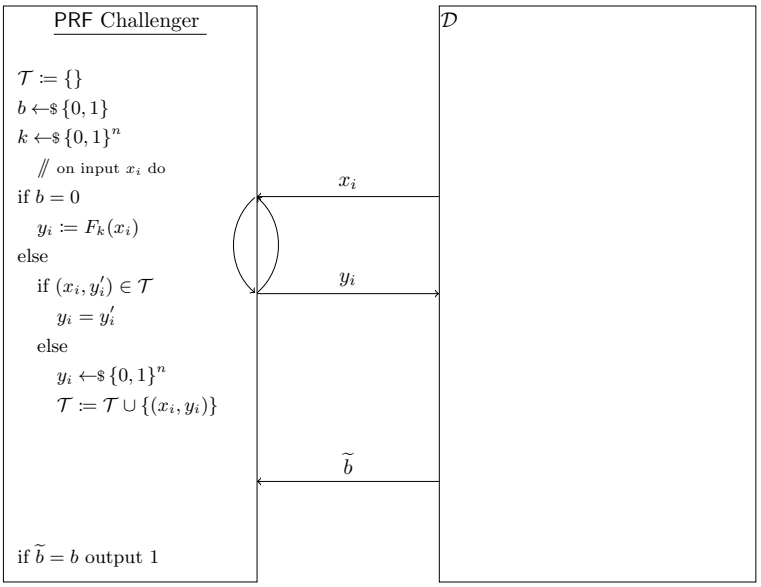


Figure 5.11: PRF Challenger Game

We say that $\mathcal{D}$ wins the game if the PRF Challenger outputs 1. Intuitively, we want that any adversary $\mathcal{D}$ wins with probability only negligibly greater than $\frac{1}{2}$. We now define the PRFs below.

Definition 24 (Pseudorandom Functions). *A family $\{F_k\}_{k\in\{0,1\}^n}$ of functions, where $F_k : \{0,1\}^n \to \{0,1\}^n$ for all k is* pseudorandom *if, it is:*

- ***Easy to Compute:*** *There is an efficient algorithm M such that $\forall k, x$: $M(k,x) = F_k(x)$.*

– ***Hard to Distinguish:*** *For every non-uniform PPT $\mathcal{D}$, there exists a negligible function ν such that $\forall n \in N$:*

$$Pr[\mathcal{D} \text{ wins}] \leq \frac{1}{2} + \nu(n) \tag{5.1}$$

Remark 6. *We are ensuring that the challenger is efficient by implementing the random function (when $b = 1$) using the second method -* lazy sampling*. It is important to construct challengers that are efficient. As we have seen while building reductions, the outer adversary acts like a challenger to the inner adversary. And if the challenger is not efficient, the outer adversary would not be efficient, and would thus lead to the construction of an* inefficient *reduction. All of our definitions are defined to be secure only against non-uniform* PPT *adversaries, therefore in our reduction, to arrive at a contradiction the outer adversary needs to run in polynomial. So an inefficient outer adversary, i.e. one whose running time is not polynomial, would not give rise to the necessary contradiction.*

Now that we have our definition, the next step is construct a PRF using the tools we've developed so far. Let's start off with a simple warm-up example of PRF whose input is a single bit.

5.7.2 PRF with 1-bit input

Intuitively, PRFs with 1- bit input, can be constructed using PRGs. Since PRFs are keyed functions, the key of PRF can be used as the random seed for PRG. We can construct a PRF $F_k : \{0,1\} \mapsto \{0,1\}^n$ using a length doubling PRG G as follows: Let G be the length-doubling PRG such that $G(s) = y_0||y_1$ where $|y_0| = |y_1| = n$.

$F_k(x)$

1 : $s \coloneqq k$
2 : $y_0||y_1 \coloneqq G(s)$
3 : **if** $x = 0$
4 : **output** y_0
5 : **else**
6 : **output** y_1

Let's argue the security of this PRF construction by relying on the security of the underlying PRG.

Proof Let us assume that the above construction is *not* a PRF. Then there exists an adversary $\mathcal{D}_{\mathsf{PRF}}$ that can distinguish between the above construction and a random function F. We will use $\mathcal{A}_{\mathsf{PRF}}$ to construct a distinguisher $\mathcal{D}_{\mathsf{PRG}}$ that is able to distinguish between the output of the PRG G and a random string. The reduction is presented in Figure 5.12.

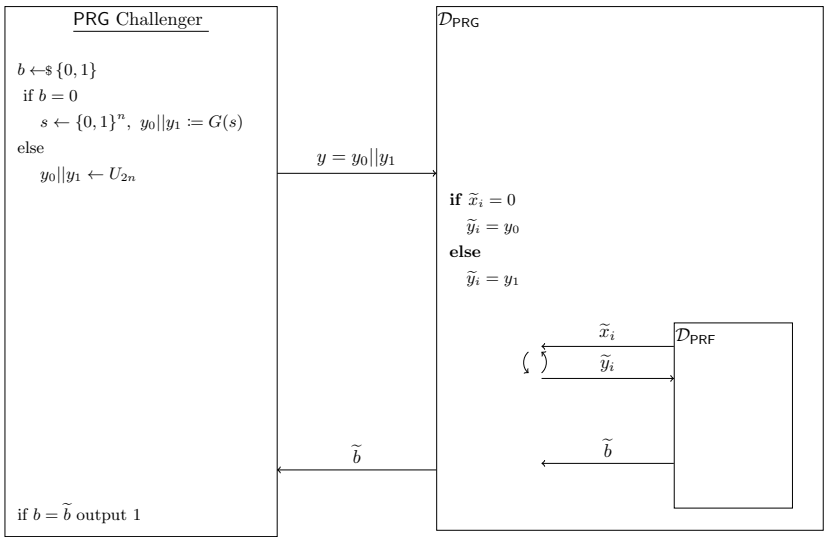


Figure 5.12: Security of 1-bit PRF

Since the distinguisher $\mathcal{D}_{\mathsf{PRF}}$ can distinguish between a random function F and the constructed PRF with noticeable probability. From the described reduction (Figure 5.12), it follows that $\mathcal{D}_{\mathsf{PRG}}$ can also distinguish between a random string and the output of the PRG with the same noticeable probability. But from the security of the PRG G, we know that no such distinguisher exists and therefore no such distinguisher for the PRF can exist. □

5.7.3 PRF with n-bit input

Now that we have a PRF construction with a 1-bit input, can we extend the same idea to n-bit outputs? Let's think of the naive extension where instead of partitioning the output of the PRG G into two parts, we start with a PRG G with polynomial size output and partition the output into 2^ℓ parts where ℓ is the length of the PRF input. Since G can only output $\mathsf{poly}(n)$ bits, ℓ is limited to being $\log(n)$. This yields a PRF $F_k : \{0,1\}^{\log(n)} \mapsto \{0,1\}^n$ from a PRG $G(s) = y_1||y_2||\cdots||y_{\mathsf{poly}(n)}$, where each y_i is n-bits.

This is a useful start, but doesn't give us what we want: going beyond polynomial stretch. Instead, for n-bit inputs, we will use the "double and choose" policy used in out 1-bit PRF construction repeatedly. This wonderful construction was first described by Golreich, Goldwasser and Micali; and is referred to as the *GGM* construction.

Theorem 13 (Goldreich-Goldwasser-Micali (GGM) [GGM84]). *If pseudorandom generators exist, then pseudorandom functions exist.*

Proof Let us fix some terminology, we start with a length-doubling PRG $G : \{0,1\}^n \mapsto \{0,1\}^{2n}$. For convenience, we define two functions G_0 and G_1 that refer to either the left or right half of G's output, i.e. $G_0(s) \coloneqq G(s)[1:n]$, and $G_1(s) \coloneqq G(s)[n+1:2n]$. The construction, in a concise manner, is given below.

$F_k(x)$

1 : $s \coloneqq k$

2 : **output** $G_{x_n}(G_{x_{n-1}}(.....(G_{x_1}(k))..)$

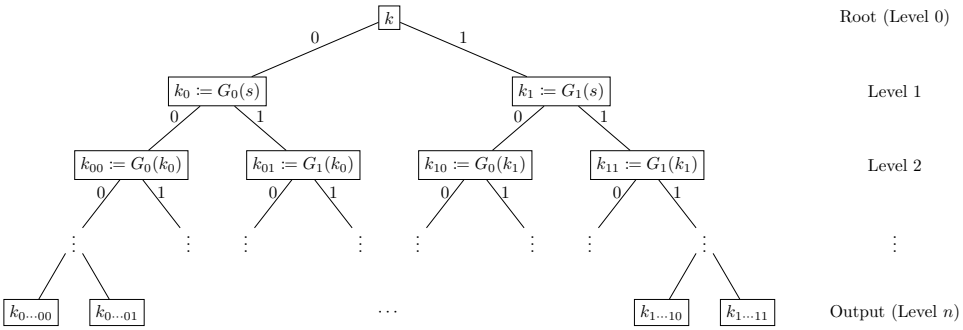


Figure 5.13: PRF Construction

Let us look at the construction in more detail. It is convenient to think of the construction of F_k as a binary tree of size 2^n, see Figure 5.13. Here k denotes the root (chosen randomly); the left and right child at level 1 of the tree are denoted as $k_0 \coloneqq G_0(k)$ and $k_1 \coloneqq G_1(k)$. The output of the PRG, k_0 and k_1, now being fed as the seed to the PRG to generate the next level of the tree. It is easy to see from Figure 5.13 how this extends to higher levels. Specifically, at level ℓ, there are 2^ℓ nodes; and each of these nodes are labeled $k_{x_1....x_\ell}$ for $x_1, \cdots, x_\ell \in \{0,1\}$.

The evaluation of function F_k on an input string $x_1x_2....x_n$ can be thought of as a walk down the branches of the tree up till the leaf nodes. Starting from

the root, based on the value of x_1 (0 or 1), either the path traverses through the left branch ($x_1 = 0$) or the right branch ($x_1 = 1$). Subsequently, the choice of branches is based on individual bits of the input string. Every input would correspond to a unique path and thus a unique leaf node, since at least one bit (and correspondingly, one edge in the tree) would be different.

Efficiency. One of the requirements for a PRF (Definition 24) is that it must be efficient. While the tree itself is exponentially sized, there is no need to store the entire tree and the output value can be computed on the fly. From our description of F_k above, we note that we execute G n-times, one for each bit of the input. Therefore,

$$\text{RunningTime}(F_k(x)) = n \cdot \text{ RunningTime}(G(\cdot)) = n \cdot \mathsf{poly}(n) = \mathsf{poly}'(n),$$

satisfying the efficiency requirement.

Security. The natural intuition is to prove using Hybrid arguments. The first idea to construct hybrids, is to replace each node in the tree, from the output of a PRG to a random string one by one as was done in the proof of Lemma 12. But since there are exponential number of nodes in the tree, this would result in exponential number of hybrids. Unfortunately, the Hybrid Lemma (Lemma 9) only works for a polynomial number of hybrids. To overcome this barrier, we make use of the following interesting observation: any PPT adversary can only make polynomial queries to F_k. By construction, each query corresponds to a unique path (of n nodes), therefore with a single query the adversary learns some information about these n nodes. Does it learn information about any of the other nodes? Since any node in the tree is half the output of a PRG, the query could conceivably leak information about the siblings of each of these n nodes - corresponding to the *other half* of the PRG output. Therefore, with a single query, the adversary learns some information about $2n$ nodes. By making $\mathsf{poly}(n)$ many queries, it therefore learns information about at most $2n \cdot \mathsf{poly}(n) = \mathsf{poly}(n)$ nodes. Therefore, to argue security we only need to change polynomially many nodes in the tree, since the adversary is oblivious to all other nodes that are *untouched* by its queries to F_k. We achieve this by using 2 layers of nested hybrids.

Make sure you understand this observation; and why the siblings are inlcuded.

Let's look at the **first level of hybrids**.

H_0 :Level 0 nodes are random strings
Level $i > 0$ nodes are pseudorandom strings
H_1 :Level 0,1 nodes are random strings
Level $i > 1$ nodes are pseudorandom strings
$\vdots$
H_i :Level $\leq i$ nodes are random strings
Level $> i$ nodes are pseudorandom strings
$\vdots$
H_n :Nodes at all levels are random strings

First, note that when we say that the nodes of level i are random or pseudorandom,we mean all nodes of level i that are affected by the adversaries queries that are random or pseudorandom respectively. Next, note the hybrid H_0 corresponds to the adversary's view of the actual PRF construction, while H_n corresponds to the view of a random function (since the outputs are simply random strings).

Let us assume that the constructed function F_k is distinguishable from a truly random function F, then by Definition 24 there exists an adversary $\mathcal{D}_{\mathsf{PRF}}$ that can distinguish the two cases with noticeable probability $\varepsilon(n)$. From the construction of our hybrids, this corresponds to $\mathcal{D}_{\mathsf{PRF}}$ distinguishing between hybrids H_0 and H_n with the same probability. Then by the Hybrid Lemma (Lemma 9), there must exist $i \in [n]$ such that $\mathcal{D}_{\mathsf{PRF}}$ distinguishes between hybrids H_i and H_{i+1} with probability $\geq \frac{\varepsilon(n)}{n}$. Note, that from our description of the hybrids, the only difference between H_i and H_{i+1} is that:

- in H_i, nodes at level $i+1$ are pseudorandom strings.
- while in H_{i+1}, nodes at level $i+1$ are random strings.

Since there are multiple nodes at level $i+1$ that are affected by the adversary's queries, we need to create another set of hybrids between H_i and H_{i+1}. To create these hybrids, we only need to replace the nodes that are affected by the queries of the adversary. Since the number of queries are polynomial, changing polynomial number of nodes is sufficient from adversary's point of view.

Let S be the set of nodes at level i that are affected by the adversary/distinguisher's input queries. We know that $|S| = \mathsf{poly}(n)$. We now define the **second level of hybrids below**, where we assume that all node in S are in lexicographic

order: $\forall j \in [|S|]$

$\mathsf{H}_{i,j}$:same as H_i, except that all nodes at level $i+1$ that are children of nodes $\leq j$, are random strings.

From the description above, we see that $\mathsf{H}_{i,0}$ corresponds to the hybrid H_i while $\mathsf{H}_{i,|S|}$ corresponds to the string H_{i+1}.

Again, given $\mathcal{D}_{\mathsf{PRF}}$ distinguishing between hybrids $\mathsf{H}_{i,0}$ and $\mathsf{H}_{i,|S|}$ (from above), by Hybrid Lemma there must exist $j \in [|S|]$ such that $\mathcal{D}_{\mathsf{PRF}}$ distinguishes between hybrids H_i and H_{i+1} with probability $\geq \frac{\varepsilon(n)}{n \cdot |S|} \geq \frac{1}{\mathsf{poly}(n)}$ (recall that $\varepsilon(n)$ is noticeable (i.e., $\geq 1/\mathsf{poly}(n)$)).

- in $\mathsf{H}_{i,j}$, node j in level $i+1$ is a pseudorandom string.
- while $\mathsf{H}_{i,j+1}$, node j in level $i+1$ is a random string.

Given $\mathcal{D}_{\mathsf{PRF}}$ that distinguishes between hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$, we will construct an adversary $\mathcal{D}_{\mathsf{PRG}}$ that can efficiently distinguish between the output of a PRG and a random string. See Figure 5.14 for the reduction. This is slightly more complex than the reductions you've seen so far. Primarily because $\mathcal{D}_{\mathsf{PRG}}$ has to simulate $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ relying only on the string $y_0||y_1$ to introduce the difference between the two hybrids. In order to do so, $\mathcal{D}_{\mathsf{PRG}}$ chooses random n-bit strings for nodes that are affected by the adversary's queries, up till level i and for nodes at level $i+1$ that are children of nodes $< j$. The children of node j are substituted by y_0 and y_1 respectively. The remaining nodes in the tree that are affected by the adversary's queries as computed using the chosen random values and y_0 and y_1. $\mathcal{D}_{\mathsf{PRG}}$ replies to the queries of $\mathcal{D}_{\mathsf{PRF}}$ using this tree.

Since the distinguisher $\mathcal{D}_{\mathsf{PRF}}$ can distinguish between hybrids $\mathsf{H}_{i,j}$ and $\mathsf{H}_{i,j+1}$ with noticeable probability. From the described reduction (Figure 5.12), it follows that $\mathcal{D}_{\mathsf{PRG}}$ can also distinguish between a random string and the output of the PRG with the same noticeable probability. But from the security of the PRG G, we know that no such distinguisher exists and therefore no such distinguisher for the $\mathcal{D}_{\mathsf{PRF}}$ can exist. Therefore, following through all our assumptions we conclude that the constructed PRF F_k must be secure. □

Chapter Notes

We would like to note some other interesting results pertaining to PRFs and extensions to the definition

Efficient PRFs from concrete assumptions: [NR97, BPR12].

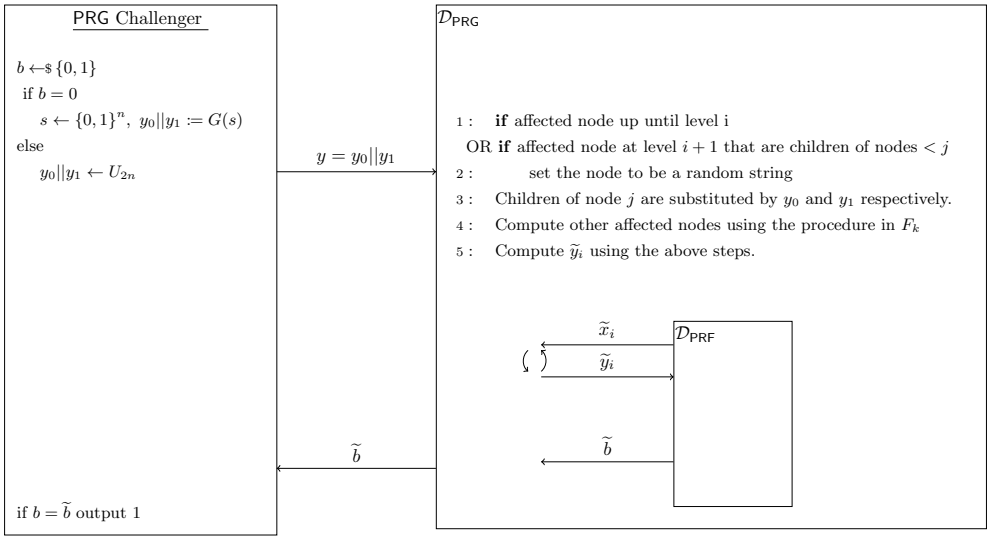


Figure 5.14: Security of 1-bit PRF

Constrained PRFs: PRFs with "punctured" keys that are disabled on certain inputs [BW13, KPTZ13, BGI14, SW14].

Related-key Security: Evaluation of $F_k(x)$ does not help in predicting $F_{k'}(x)$ [BC10].

Key-Homomorphic PRF: Given $F_k(x)$ and $F_{k'}(x)$ compute $F_{g(k,k')}(x)$ [BLMR13].