Pseudorandom Generators

Noah Stephens-Davidowitz

June 7, 2023

1 Pseudorandom generators

Random bits $\boldsymbol{X} \sim \{0,1\}^n$ are extremely useful in cryptography. For example, we have already seen how to use a shared random string $\boldsymbol{X} \sim \{0,1\}^n$ to encrypt a plaintext message $\boldsymbol{m} \in \{0,1\}^n$ with perfect, information-theoretic security, using Shannon's one-time pad. We were not particularly satisfied with the one-time pad, however, because it requires the key to be as long as the message (and because, if we were happy with the one-time pad, it would be more difficult to justify the previous few lectures, which are leading up to some quite magical results).

This leads naturally to the question of whether we can generate *pseudorandom* bits. Informally, we would like to generate a bit string $\boldsymbol{Y} \in \{0,1\}^m$ that "looks random" to an adversary. And, we would like to do so non-trivially—i.e., we would like to "put only n random bits in as a seed but get $m \gg n$ pseudorandom bits out." If we could do this, then we could at least use Shannon's one-time pad with a shorter key. In particular, we could use an n-bit key $\boldsymbol{X} \in \{0,1\}^n$ to generate a much longer pseudorandom key $\boldsymbol{Y} \in \{0,1\}^m$, which can then be used to send a much longer message or many short messages via the one-time pad. (This is still not a very good encryption scheme! For example, it only works for multiple messages if the sender and receiver both agree on the order of the messages, which is rather strange. We will see a more clever scheme in the next few lectures.)

1.1 Two (equivalent) notions of pseudorandomness

Informally, a *pseudorandom generator* (or PRG for short) is a function that takes as input n random *seed* bits and outputs $m > n$ "pseudorandom" bits—i.e., bits that "are as good as random." Let's try to make this formal.

Intuitively, a pseudorandom distribution over $\{0,1\}^m$ should have some of the properties of the uniform distribution over $\{0,1\}^m$. For example, suppose that $\boldsymbol{Y} \sim \{0,1\}^m$ is sampled uniformly at random. Then, if we look at any individual bit Y_i, we must have $\Pr[Y_i = 1] = 1/2$. A pseudorandom distribution should also have this property (or, well, we'll be okay with $\Pr[Y_i = 1] \le 1/2 + \mathrm{negl}(n)$), but it is clearly not enough on its own. E.g., we could take $\boldsymbol{Y} \sim \{0^m, 1^m\}$ to be the string that is all zeros with probability $1/2$ and all ones with probability $1/2$. It doesn't seem reasonable to call $\boldsymbol{Y}$ pseudorandom! For example, such a $\boldsymbol{Y}$ would not be useful as a key for Shannon's one-time pad...

What about the following stronger definition? For each i, suppose we ask that $\Pr[Y_i = 1 \mid Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}] = 1/2$ for any $y_1, \ldots, y_{i-1}$. I.e., no matter how we set the first $i-1$ bits, the ith bit is unbiased. This definition is certainly strong enough—in fact it's too strong! If $\boldsymbol{Y}$ satisfies this definition, then it must be uniformly random itself. (It's a nice exercise to prove this by induction, and I often give this for homework.) This makes our definition pretty useless, since it means that to sample such a distribution $\boldsymbol{Y}$ we need the seed length n to be at least m.

To make this idea non-trivial, we need to introduce the notion of a computationally bounded adversary—just like we introduced computationally bounded adversaries in order to move from Shannon's perfectly secure encryption schemes to semantically secure encryption schemes. So, suppose we try to make the above definition computational. We need some kind of computational test that's analogous to the information-theoretic statement that $\Pr[Y_i = 1 \mid Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}] = 1/2$ for any $y_1, \ldots, y_{i-1}$. The right notion here is *(un)predictability.* In particular, informally, we say that an adversary $\mathcal{A}$ *predicts* Y_i given $Y_1, \ldots, Y_{i-1}$, if $\Pr[\mathcal{A}(Y_1, \ldots, Y_{i-1}) = Y_i] \geq 1/2 + \varepsilon$ for some non-negligible advantage ε.

To make this formal, we of course have to introduce asymptotics and a security parameter n. In fact, let's just define an unpredictable PRG, rather than bothering to define unpredictable distributions separately. (This definition is sometimes also called "next-bit unpredictability." Note that this is *not* the standard definition of a PRG, and not the one that we will typically use. Spoiler: We will, however, see that this definition is equivalent to the standard definition.)

Definition 1.1 (Unpredictable PRGs)**.** *A function $G : \{0,1\}^* \to \{0,1\}^*$ is an* unpredictable PRG *if the following hold.*

1. **Efficiently computable.** *There is a PPT algorithm $\mathcal{A}$ that computes $G(\boldsymbol{x})$ given $\boldsymbol{x}$.*
2. **Expanding.** *For every $n \in \mathbb{N}$, there is a fixed output length $m(n) > n$ such that $|G(\boldsymbol{x})| = m(n)$ for all $\boldsymbol{x} \in \{0,1\}^n$.*
3. **Unpredictable.** *For any PPT algorithm $\mathcal{B}$, there exists a negligible function $\varepsilon(n)$ such that for all positive integers n and all $1 \leq i \leq m(n)$, we have:*

$$\Pr_{\boldsymbol{x} \sim \{0,1\}^n}[\boldsymbol{Y} := G(\boldsymbol{x}),\ \mathcal{B}(1^n, (Y_1, \ldots, Y_{i-1})) = Y_i] \leq 1/2 + \varepsilon(n) .$$

We will see soon that this definition is quite useful, but if we want to use such a PRG to construct a pseudorandom one-time pad, we should worry that it's simply not strong enough. For example, we know that the last bit cannot be predicted given the first $m-1$ bits, but what if the *first* bit is "predictable" given the *last* $m-1$ bits? Such a distribution would not be useful for a one-time pad. (Why not?) Or, more generally, what if the ith bit can be "predicted" given *all* of the other bits, rather than just the $i-1$ before it? It seems strange for our notion of pseudorandomness to treat the ith bit differently from the $(i+1)$st bit.

And, even if no individual bit is predictable given all of the others, maybe there is some other statistic of our distribution that makes it insecure. Perhaps the standard deviation of $\boldsymbol{Y}$ viewed as an integer in binary is too small or perhaps $\boldsymbol{Y}$ fails some complicated statistical test. There's a whole field of statistics devoted to figuring stuff like this out! I think there's something called χ-squared test for example. I don't know what that is, but what if someone does a χ-squared test on our distribution? What if they do some other crazy statistical test that I've never heard of? Are we willing to bet that our simple definition above will defeat all of statistics?!

A much more robust definition of pseudorandomness (which is also the standard definition!) is the following. Intuitively, no polynomial-time test should distinguish $\boldsymbol{Y}$ from uniform bits $\boldsymbol{X} \sim \{0,1\}^m$.

Definition 1.2 (Strong PRGs (aka, PRGs))**.** *A function $G : \{0,1\}^* \to \{0,1\}^*$ is a* strong PRG *if the following hold.*

1. **Efficiently computable.** *There is a PPT algorithm $\mathcal{A}$ that computes $G(\boldsymbol{x})$ given $\boldsymbol{x}$.*
2. **Expanding.** *For every $n \in \mathbb{N}$, there is a fixed polynomial output length $m(n) > n$ such that $|G(\boldsymbol{x})| = m(n)$ for all $\boldsymbol{x} \in \{0,1\}^n$.*
3. **Pseudorandom.** *For all PPT algorithms $\mathcal{B}$, there exists a negligible function $\varepsilon(n)$ such that*

$$\Pr_{\boldsymbol{x}\sim\{0,1\}^n}[\mathcal{B}(1^n, G(\boldsymbol{x})) = 1] - \Pr_{\boldsymbol{y}\sim\{0,1\}^{m(n)}}[\mathcal{B}(1^n, \boldsymbol{y}) = 1] \le \varepsilon(n) \; .$$

It might help when reading this definition to interpret an output of 1 from $\mathcal{B}$ as a guess that its input is pseudorandom. In other words, the definition says that $\mathcal{B}$ "says $G(\boldsymbol{x})$ looks pseudorandom" essentially as often as she "says that the uniform distribution looks pseudorandom." So, evidently, $\mathcal{B}$ "can't tell the difference" between the two distributions. (It might be tempting to say something like "$\mathcal{B}$ is unlikely to say that $G(\boldsymbol{x})$ looks pseudorandom," so something like $\Pr[\mathcal{B}(1^n, G(\boldsymbol{x})) = 1] \le \varepsilon(n)$. Unfortunately, this is nonsense, because we of course have to worry about, e.g., the adversary $\mathcal{B}$ that just says that everything looks pseudorandom.)

1.2 Aside: Notes on a few different versions of "indistinguishability"

Our definition of pseudorandomness says that for every PPT $\mathcal{B}$,

$$\Pr[\mathcal{B}(G(\boldsymbol{x})) = 1] - \Pr[\mathcal{B}(\boldsymbol{y}) = 1]$$

is negligible. More generally, we say that two families of distributions $\boldsymbol{Y}_n$ and $\boldsymbol{X}_n$ (families of distributions and not just single distributions because we need a security parameter) are *computationally indistinguishable* (or just indistinguishable) if for any PPT $\mathcal{B}$,

$$\Pr[\mathcal{B}(1^n, \boldsymbol{X}_n) = 1] - \Pr[\mathcal{B}(1^n, \boldsymbol{Y}_n) = 1]$$

is negligible. Then, our pseudorandomness definition simply says that $G(\boldsymbol{x})$ is indistinguishable from the uniform distribution on m bits when $\boldsymbol{x} \sim \{0,1\}^n$. (This is sometimes written succinctly as

$$\boldsymbol{X}_n \approx_c \boldsymbol{Y}_n \; ,$$

where the symbol $\approx_c$ is read as "computationally indistinguishable." However, this notation is dangerous because it makes it very tempting to forget about the security parameter n entirely. We will therefore avoid using this notation for most of this course, until we see some very fancy security definitions in which this notation is really quite necessary.)

There are a few alternative forms of the above definition that are trivially equivalent, and we often switch between them. A common one is to say that $\boldsymbol{X}_n^{(0)}$ is computationally indistinguishable from $\boldsymbol{X}_n^{(1)}$ if for any PPT $\mathcal{A}$,

$$\Pr_{b\sim\{0,1\}}[b' \leftarrow \mathcal{A}(1^n, \boldsymbol{X}_n^{(b)}) \;:\; b' = b] \le 1/2 + \varepsilon(n)$$

for some negligible $\varepsilon(n)$. In other words, "$\mathcal{A}$ cannot guess whether it has seen $\boldsymbol{X}_n^{(0)}$ or $\boldsymbol{X}_n^{(1)}$ with non-negligible advantage over random guessing." We saw a definition like this in the context of semantic security, and it is a simple exercise to see that the two definitions are equivalent. (Make sure you can do this exercise!)

Another variant uses an absolute value. I.e., instead of saying that

$$\Pr[\mathcal{B}(1^n, \boldsymbol{X}_n) = 1] - \Pr[\mathcal{B}(1^n, \boldsymbol{Y}_n) = 1] \leq \varepsilon(n)$$

for some negligible $\varepsilon(n)$, we say that

$$\big|\Pr[\mathcal{B}(1^n, \boldsymbol{X}_n) = 1] - \Pr[\mathcal{B}(1^n, \boldsymbol{Y}_n) = 1]\big| \leq \varepsilon(n) \ .$$

This definition is nice because it is clearly symmetric in $\boldsymbol{X}_n$ and $\boldsymbol{Y}_n$—i.e., $\boldsymbol{X}_n$ is indistinguishable from $\boldsymbol{Y}_n$ if and only if $\boldsymbol{Y}_n$ is indistinguishable from $\boldsymbol{X}_n$. Fortunately, this definition is also equivalent. (To see this, simply notice that if $\Pr[\mathcal{B}(1^n, \boldsymbol{X}_n) = 1]$ were non-negligibly *smaller* than $\Pr[\mathcal{B}(1^n, \boldsymbol{Y}_n) = 1]$, then $\Pr[\overline{\mathcal{B}}(1^n, \boldsymbol{X}_n) = 1]$ would be non-negligibly larger than $\Pr[\overline{\mathcal{B}}(1^n, \boldsymbol{Y}_n) = 1]$, where $\overline{\mathcal{B}}$ is the algorithm that simply runs $\mathcal{B}$ and outputs the opposite of whatever $\mathcal{B}$ outputs. Of course, if $\mathcal{B}$ is a PPT algorithm, then so is $\overline{\mathcal{B}}$.)

1.3 Proof of equivalence via a hybrid argument

As it turns out, unpredictability and pseudorandomness are equivalent. The proof uses a very common technique called a hybrid argument, which will come up *repeatedly* in this course.

Theorem 1.3. *A function G is an unpredictable PRG if and only if it is a (strong) PRG.*

Proof. We need to prove two things. First, if G is predictable, then it is not a PRG. This direction is easy, so we leave it as an exercise. (In other words, if there exists a predictor, then there exists a distinguisher.)

Second, we need to show that if G is not a PRG, then it is predictable. Of course, we do this via a reduction. I.e., suppose that some PPT $\mathcal{B}$ has non-negligible advantage $\varepsilon(n)$ in distinguishing $G(\boldsymbol{x})$ from random. We claim that we can use this to construct a PPT algorithm $\mathcal{P}$ (for predictor) that can predict the next bit with non-negligible advantage.

Fix some security parameter n and let $m := m(n)$ and $\varepsilon := \varepsilon(n)$. Before we even describe the behavior of $\mathcal{P}$, we need to find the index i that $\mathcal{P}$ will predict. Notice that, even if $G(\boldsymbol{x})$ is not pseudorandom, *some* bits of $G(\boldsymbol{x})$ might still be unpredictable. E.g.. the last bit of $G(\boldsymbol{x})$ might be completely independent of all of the other bits. So, we need to somehow argue that at least one bit is predictable—we will not be able to argue that all bits are. We do this via a hybrid argument, like we saw in the previous lecture.

For any index $0 \leq i \leq m$, we wish to define $\boldsymbol{Y}^{(i)} \in \{0,1\}^m$ as the random variable whose "first i bits are pseudorandom and last $m - i$ bits are random." Formally, to sample $\boldsymbol{Y}^{(i)}$, we sample $\boldsymbol{x}^{(i)} \sim \{0,1\}^n$ and set the first i bits of $\boldsymbol{Y}^{(i)}$ to be the first i bits of $G(\boldsymbol{x}^{(i)})$. The last $m - i$ bits are uniformly random. These distributions are called *hybrid distributions*, since they are somehow mixtures (i.e., hybrids) of the purely random and purely pseudorandom distributions.

Let p_i be the probability that $\mathcal{B}$ outputs 1 on input $\boldsymbol{Y}^{(i)}$. We claim that there must exist an i such that $p_i - p_{i-1} \geq \varepsilon/m$. To see this, notice that $p_m - p_0 = \varepsilon$, since $\boldsymbol{Y}^{(m)}$ is a purely pseudorandom string and $\boldsymbol{Y}^{(0)}$ is a purely random string. By assumption, the probability that $\mathcal{B}$ outputs one on pseudorandom input is ε larger than the probability that it does so on random input. I.e. $p_m - p_0 = \varepsilon$. Then, since $p_m - p_0 = (p_1 - p_0) + (p_2 - p_1) + \cdots + (p_m - p_{m-1})$, at least one of these differences must be as large as the average, which is ε/m.

We now have a specific bit i such that $\mathcal{B}$ distinguishes between the distribution $\boldsymbol{Y}^{(i-1)}$ with $i-1$ pseudorandom bits (with the rest random) and the distribution $\boldsymbol{Y}^{(i)}$ with i pseudorandom bits. It remains to construct $\mathcal{P}$ that predicts the ith bit.

Given the first i bits of a pseudorandom string $Y_1, \ldots, Y_{i-1} \in \{0,1\}$ (and 1^n), our predictor $\mathcal{P}$ behaves as follows. It samples $Y_i^*, \ldots, Y_m^* \sim \{0,1\}$ uniformly at random and runs $\mathcal{B}$ on input $(Y_1, \ldots, Y_{i-1}, Y_i^*, Y_{i+1}^*, \ldots, Y_m^*)$. If $\mathcal{B}$ outputs 1 (i.e., if it "says that the input looks pseudorandom"), then $\mathcal{P}$ guesses Y_i^*. Otherwise, it guesses $1 - Y_i^*$.

First, notice that $\mathcal{P}$ is efficient (assuming that $\mathcal{B}$ is efficient). Its running time is essentially the same as the running time of $\mathcal{B}$ on an m-bit string. Since $\mathcal{B}$ runs in polynomial time and m is polynomial in n, $\mathcal{P}$ runs in time polynomial in n, as needed. So, we only need to prove correctness.

Intuitively, this works because we know that $\mathcal{B}$ is more likely to output 1 when its ith bit is "correct." To make this formal, it helps to define p_i' as the probability that $\mathcal{B}$ outputs 1 when its input is the same as $\boldsymbol{Y}^{(i)}$ in all bits except for the ith, and has the ith bit flipped. I.e., p_i' is the probability that $\mathcal{B}$ outputs 1 when it is given "the wrong ith bit."[1] Notice that the probability that $\mathcal{P}$ guesses correctly is exactly

$$\begin{aligned}
&\Pr[\mathcal{B}(Y_1, \ldots, Y_{i-1}, Y_i^*, Y_{i+1}^*, \ldots, Y_m^*) = 1 \mid Y_i^* = Y_i]/2 \\
&\qquad + \Pr[\mathcal{B}(Y_1, \ldots, Y_{i-1}, Y_i^*, Y_{i+1}^*, \ldots, Y_m^*) = 0 \mid Y_i^* \neq Y_i]/2 \\
&\quad = p_i/2 + (1 - p_i')/2 \\
&\quad = 1/2 + (p_i - p_i')/2 \; .
\end{aligned}$$

We also have that $p_{i-1} = (p_i + p_i')/2$ (i.e., "$\mathcal{B}$'s behavior on a random bit is the average of its behavior on the right bit and on the wrong bit").

Therefore, $\mathcal{P}$'s advantage is

$$(p_i - p_i')/2 = p_i - p_{i-1} \geq \varepsilon/m \; .$$

Since ε is non-negligible and $m = \mathrm{poly}(n)$, this is also non-negligible, and we are done. □

2 A PRG from any one-way permutation with a hardcore predicate

Ideally, we would now like to show how to construct a pseudorandom generator (a rather strong object!) from any one-way function (a seemingly much weaker object!). This very impressive feat was was accomplished by Håstad, Impagliazzo, Levin, and Luby [HILL99], but their construction and proof are rather complicated. Instead, we allow ourselves an extra assumption and prove the result for one-way *permutations* instead. This simpler construction is due to Blum and Micali [BM84].

A permutation is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is length-preserving and bijective. I.e., for every $\boldsymbol{x} \in \{0,1\}^n$, there is a unique $\boldsymbol{y} \in \{0,1\}^n$ such that $f(\boldsymbol{x}) = \boldsymbol{y}$. The key property that we will use about permutations is that they map uniformly random n-bit strings to uniformly random

[1] I use this "the wrong bit" terminology very loosely. $\boldsymbol{Y}^{(i)}$ is a random variable, and its ith bit could be undetermined given the other bits. So, there's no formal notion of "the wrong bit." Still, I hope the terminology helps to give some intuition for why the reduction works without being confusing. E.g., here, talking about "the wrong bit" intuitively makes sense because, at least intuitively, we chose the index i specifically because the ith bit seems to "depend significantly on the previous bits" (formally, because $p_i - p_{i-1}$ is large). In general, I (along with many other authors) put words in "scare quotes" when they are meant to provide intuition but not to be interpreted formally.

n-bit strings. (In fact, this is true if and only if f is a permutation.) A *one-way* permutation is a one-way function that is also a permutation. We assume that the one-way permutation comes with a hardcore predicate $P : \{0,1\}^* \to \{0,1\}$. (In fact, we will not need one-wayness. We will only need the assumption that P is a hardcore predicate. However, if a permutation has a hardcore predicate, then it must be one way anyway.)

Blum and Micali's construction works as follows. Given $\boldsymbol{x} \in \{0,1\}^*$, the Blum-Micali PRG computes $\boldsymbol{s}_0 := \boldsymbol{x}, \boldsymbol{s}_1 := f(\boldsymbol{s}_0), \boldsymbol{s}_2 := f(\boldsymbol{s}_1), \boldsymbol{s}_3 := f(\boldsymbol{s}_2), \ldots, \boldsymbol{s}_{m(n)-1} := f(\boldsymbol{s}_{m(n)-2})$, and outputs $(P(\boldsymbol{s}_{m(n)-1}), \ldots, P(\boldsymbol{s}_0))$. I.e., it outputs the hardcore predicates of the $\boldsymbol{s}_i$ in reversed order. (The reversed order is simply more convenient for the proof. Obviously, if a string is pseudorandom, then the reversed string is also pseudorandom.)

Theorem 2.1. *If f is a permutation and b is a hardcore predicate for f, then the Blum-Micali construction is a (strong) PRG.*

Proof. By Theorem 1.3, it suffices to show that the output of the Blum-Micali PRG is unpredictable. (This is why it is convenient to output the bits in reversed order. The unpredictability definition treats a string and the reversed string differently.) Again, we do this by reduction. I.e., we assume that we have an efficient predictor $\mathcal{P}$ that takes as input $P(\boldsymbol{s}_{m(n)-1}), \ldots, P(\boldsymbol{s}_i)$ and outputs a prediction b^* with $\Pr[P(\boldsymbol{s}_{i-1}) = b^*] = 1/2 + \varepsilon(n)$ for some non-negligible ε, and we show how to use this to build an efficient adversary $\mathcal{A}$ that breaks the hardcore predicate P. I.e., $\mathcal{A}$ takes as input $f(\boldsymbol{x})$ for uniformly random $\boldsymbol{x} \in \{0,1\}^n$ and outputs $b(\boldsymbol{x})$ with probability $1/2 + \delta(n)$ for non-negligible $\delta(n)$. In fact, we will achieve $\delta(n) = \varepsilon(n)$.

So, given $\boldsymbol{s}'_i := f(\boldsymbol{x}) \in \{0,1\}^n$, $\mathcal{A}$ behaves as follows. It computes $\boldsymbol{s}'_{i+1} := f(\boldsymbol{s}_i), \ldots, \boldsymbol{s}'_{m(n)} := f(\boldsymbol{s}'_{m(n)-1})$. $\mathcal{A}$ then feeds the predicates $P(\boldsymbol{s}'_{m(n)}), \ldots, P(\boldsymbol{s}'_i)$ to $\mathcal{P}$ (together with 1^n). It then simply outputs $\mathcal{P}$'s guess b^*.

First, notice that $\mathcal{A}$ runs in polynomial time (assuming that $\mathcal{P}$ does). So, we only need to show that $\mathcal{A}$ has non-negligible advantage.

Now, here is an *almost correct* proof that $\mathcal{A}$ has advantage $\varepsilon(n)$. Let's imagine that we play an "honest" predicting game with $\mathcal{P}$. I.e., we first sample $\boldsymbol{s}_0 \in \{0,1\}^n$ uniformly at random, set $\boldsymbol{s}_1 := f(\boldsymbol{s}_0)$, etc., give $\mathcal{P}$ her input $P(\boldsymbol{s}_{m(n)-1}), \ldots, P(\boldsymbol{s}_i)$, and receive her prediction b^* for $P(\boldsymbol{s}_{i-1})$. (This next sentence is not quite correct. See if you can spot the mistake.) If we just rename $\boldsymbol{s}_{i-1}$ and call it $\boldsymbol{x}$, and if we just rename $\boldsymbol{s}_i, \ldots, \boldsymbol{s}_{m(n)-1}$ to $\boldsymbol{s}'_i, \ldots, \boldsymbol{s}'_{m(n)-1}$, then this is the same as the game above. So, clearly $b^* = P(\boldsymbol{x})$ if and only if $b^* = P(\boldsymbol{s}_{i-1})$, since $\boldsymbol{x} = \boldsymbol{s}_{i-1}$.

The issue with the above argument is that $\boldsymbol{x}$ and $\boldsymbol{s}_{i-1}$ are not the same. In particular, they were sampled in very different ways. E.g., for some functions f, we could have $\boldsymbol{s}_{i-1} = f^{i-1}(\boldsymbol{s}_0) = 0$ for all $\boldsymbol{s}_0 \in \{0,1\}^n$, which would clearly make the above argument invalid! To fix this, we need to show that the distribution of the *view* of $\mathcal{P}$ in the world in which we play the "honest" game is the same as the distribution of the view of $\mathcal{P}$ in the world in which $\mathcal{B}$ runs the above reduction using $\mathcal{P}$. (Here, the view of $\mathcal{P}$ is just her input. Sometimes, we will play more complicated interactive games with our adversaries, in which case the view is everything that the adversary sees.) Then, the probability that $\mathcal{P}$ outputs $b^* = P(\boldsymbol{x})$ in the honest game must equal the probability that $b^* = P(\boldsymbol{s}_{i-1})$ in the *simulated* game created by $\mathcal{B}$, since from the perspective of $\mathcal{P}$, there is no difference.

This is why we assumed that f is a permutation, which implies that the $\boldsymbol{s}_j$ are uniformly distributed for all j. (Recall that this was the key property of permutations that we said we would need.) In particular, $\boldsymbol{s}_{i-1}$ has the same distribution as $\boldsymbol{x}$, which means that from $\mathcal{P}$'s perspective, she really is just playing the "honest" predicting game. Therefore, $b^* = P(\boldsymbol{s}_{i-1})$ with probability $1/2 + \varepsilon(n)$, as needed. $\square$

3 The big picture and the proof that we're missing (actually, two proofs)

We would like to conclude from the above that the existence of a one-way function implies the existence of a PRG. Indeed, as we have mentioned a few times, this was proven in [HILL99]. However, there are two gaps in our proof.

The first gap is that we needed a one-way permutation, rather than a one-way function. This is a very big gap! (In fact, it is believed that one-way permutations are strictly stronger objects than one-way functions, in a certain precise sense.) However, we will simply ignore it in this class—except to say that this *can* be fixed. So, if you believe that one-way *functions* exist, then you must believe that PRGs exist, though we will not see the proof here.

Even assuming that we're okay with one-way permutations, the other issue is that we assumed not only the existence of a one-way permutation f, but also an associated hardcore predicate P! Luckily, Goldreich and Levin proved that any one-way permutation can be converted into a one-way permutation with a hardcore predicate [GL89].

References

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4), 1984. http://dx.doi.org/10.1137/0213053. 5

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989. 7

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4), 1999. 5, 7