5.1: Definition of Pseudorandom Generators

As mentioned above, a pseudorandom distribution “looks uniform” to all polynomial-time computations. We already know of a distribution that “looks uniform” — namely, the uniform distribution itself! A more interesting case is when \(\lambda\) uniform bits are used to deterministically (i.e., without further use of randomness) produce \(\lambda + \ell\) pseudorandom bits, for \(\lambda > 0\). The process that “extends” \(\lambda\) bits into \(\lambda + \ell\) bits is called a pseudorandom generator. More formally:

Definition \(\PageIndex{1}\): PRG Security

Let \(G : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda + \ell}\) be a deterministic function with \(\ell > 0\). We say that \(G\) is a **pseudorandom generator (PRG)** if \(\mathcal{L}_{\text{prg-real}}^G \equiv \mathcal{L}_{\text{prg-rand}}^G\), where:

\[
\begin{align*}
\mathcal{L}_{\text{prg-real}}^G & \equiv \mathcal{L}_{\text{prg-rand}}^G \\
\text{QUERY:} & \equiv \text{QUERY:} \\
\text{return } G(s) & \equiv \text{return } z
\end{align*}
\]

The value \(\ell\) is called the **stretch** of the PRG. The input to the PRG is typically called a **seed**.

Discussion:

- Is 0010110110 a random string? Is it pseudorandom? What about 0000000001? Do these questions make any sense?

  Randomness and pseudorandomness are not properties of individual strings, they are properties of the process used to generate the string. We will try to be precise about how we talk about these things (and you should too).
When we have a value $z = G(s)$ where $G$ is a PRG and $s$ is chosen uniformly, we can say that $z$ was “chosen pseudorandomly”, but not that $z$ “is pseudorandom”. On the other hand, a distribution can be described as “pseudorandom.” The same goes for describing a value as “chosen uniformly” and describing a distribution as “uniform.”

- Pseudorandomness can happen only in the computational setting, where we restrict focus to polynomial-time adversaries. The exercises ask you to prove that for all functions $G$ (with positive stretch), $\mathcal{L}^G_{\text{prg-real}} \not\equiv \mathcal{L}^G_{\text{prg-rand}}$ (note the use of $\equiv$ rather than $\not\equiv$). That is, the output distribution of the PRG can never actually be uniform in a mathematical sense. Because it has positive stretch, the best it can hope to be is pseudorandom.

- It’s sometimes convenient to think in terms of statistical tests. When given access to some data claiming to be uniformly generated, your first instinct is probably to perform a set of basic statistical tests: Are there roughly an equal number of 0s as 1s? Does the substring 01010 occur with roughly the frequency I would expect? If I interpret the string as a series of points in the unit square $\{(0,1)^2\}$, is it true that roughly $\pi/4$ of them are within Euclidean distance 1 of the origin?

The definition of pseudorandomness is kind of a “master” definition that encompasses all of these statistical tests and more. After all, what is a statistical test, but a polynomial-time procedure that obtains samples from a distribution and outputs a yes-or-no decision? Pseudorandomness implies that every statistical test will “accept” when given pseudorandomly generated inputs with essentially the same probability as when given uniformly sampled inputs.

- Consider the case of a length-doubling PRG (so $\ell = \lambda$); the PRG has input length $\lambda$ and output length $2\lambda$. The PRG only has $2^\lambda$ possible inputs, and so there are at most only $2^{2\lambda}$ possible outputs. Among all of $\{0,1\}^{2\lambda}$, this is a miniscule fraction indeed. Almost all strings of length $2\lambda$ are impossible outputs of $G$. So how can outputs of $G$ possibly “look uniform?”

The answer is that it’s not clear how to take advantage of this observation. While it is true that most strings (in a relative sense as a fraction of $2^{2\lambda}$) are impossible outputs of $G$, it is also true that $2^\lambda$ of them are possible, which is certainly a lot from the perspective of a program who runs in polynomial time in $\lambda$. Recall that the problem at hand is designing a distinguisher to behave as differently as possible in the presence of pseudorandom and uniform distributions. It is not enough to behave differently on just a few strings here and there — an individual string can contribute at most $1/2^\lambda$ to the final distinguishing advantage. A successful distinguisher must be able to recognize a huge number of outputs of $G$ so it can behave differently on them, but there are exponentially many $G$-outputs, and they may not follow any easy-to-describe pattern.

Below is an example that explores these ideas in more detail.

Example \ref{PageIndex{1}}:

Let $G$ be a length-doubling PRG as above, let $t$ be an arbitrary string $t \in \{0,1\}^{2\lambda}$, and consider the following distinguisher $?_t$ that has the value $t$ hard-coded:

\[
\underline{\text{\texttt{A}\_}\_t}: \text{\texttt{QUERY()}} \\
\text{return } z \overset{?}{=} t
\]

What is the distinguishing advantage of $?_t$?

We can see easily what happens when $?_t$ is linked to $G$. We get:
What happens when linked to $\mathscr{L}_{\text{prg-real}}$ depends on whether $t$ is a possible output of $G$, which is a simple property of $G$. We always allow distinguishers to depend arbitrarily on $G$. In particular, it is fine for a distinguisher to “know” whether its hard-coded value $t$ is a possible output of $G$. What the distinguisher “doesn’t know” is which library it is linked to, and the value of $s$ that was chosen in the case that it is linked to $\mathscr{L}_{\text{prg-real}}$.

Suppose for simplicity that $G$ is injective (the exercises explore what happens when $G$ is not). If $t$ is a possible output of $G$, then there is exactly one choice of $s$ such that $G(s) = t$, so we have:

$$\Pr[\mathscr{A}_t \diamondsuit \mathscr{L}_{\text{prg-rand}} | G \Rightarrow 1] = \frac{1}{2^{2\lambda}}$$

Hence, the distinguishing advantage of $\mathscr{A}_t$ is $|\frac{1}{2^{2\lambda}} - \frac{1}{2^\lambda}| = \frac{1}{2^\lambda}$.

If $t$ is not a possible output of $G$, then we have:

$$\Pr[\mathscr{A}_t \diamondsuit \mathscr{L}_{\text{prg-rand}} | G \Rightarrow 0] = 0$$

Hence, the distinguishing advantage of $\mathscr{A}_t$ is $|\frac{1}{2^{2\lambda}} - 0| = \frac{1}{2^{2\lambda}}$.

In either case, $\mathscr{A}_t$ has negligible distinguishing advantage. This merely shows that $\mathscr{A}_t$ (for any hard-coded $t$) is not a particularly helpful distinguisher for any PRG. Of course, any candidate PRG might be insecure because of other distinguishers, but this example should serve to illustrate that PRG security is at least compatible with the fact that some strings are impossible outputs of the PRG.

### Related Concept: Random Number Generation

The definition of a PRG includes a uniformly sampled seed. In practice, this PRG seed has to come from somewhere. Generally a source of “randomness” is provided by the hardware or operating system, and the process that generates these random bits is (confusingly) called a random number generator (RNG).

In this course we won’t cover low-level random number generation, but merely point out what makes it different than the PRGs that we study:

- The job of a PRG is to take a small amount of “ideal” (in other words, uniform) randomness and extend it.
- By contrast, an RNG usually takes many inputs over time and maintains an internal state. These inputs are often from physical/hardware sources. While these inputs are “noisy” in some sense, it is hard to imagine that they would be statistically uniform. So the job of the RNG is to “refine” (sometimes many) sources of noisy data into uniform outputs.

### Perspective on this Chapter

PRGs are a fundamental cryptographic building block that can be used to construct more interesting things. But you are unlikely to ever find yourself designing your own PRG or building something from a PRG that couldn’t be built from some higher-level primitive instead. For that reason, we will not discuss specific PRGs or how they are designed in practice. Rather, the purpose of this chapter is to build your understanding of the concepts (like pseudorandomness, indistinguishability, proof techniques) that will be necessary in the rest of the class.