5 Pseudorandom Generators

We have already seen that randomness is essential for cryptographic security. Following Kerckhoff's principle, we assume that an adversary knows *everything* about our cryptographic algorithms except for the outcome of the internal random choices made when running the algorithms. Private randomness truly is the *only* resource honest users can leverage for security in the presence of an adversary.

One-time secrecy for encryption is a very demanding requirement, when we view randomness as a resource. One-time secrecy essentially requires one to use independent, uniformly random bits to mask every bit of plaintext that is sent. As a result, textbook one-time pad is usually grossly impractical for practical use.

In this chapter, we ask whether it might be beneficial to settle for a distribution of bits which is not uniform but merely "looks uniform enough." This is exactly the idea behind **pseudorandomness.** A pseudorandom distribution is not uniform in a strict mathematical sense, but is *indistinguishable* from the uniform distribution by polynomial-time algorithms (in the sense defined in the previous chapter).

Perhaps surprisingly, a relatively small amount of *truly uniform* bits can be used to generate a huge amount of *pseudorandom* bits. Pseudorandomness therefore leads to an entire new world of cryptographic possibilities, an exploration that we begin in this chapter.

5.1 Definition

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 λ uniform bits are used to *deterministically* (*i.e.*, without further use of randomness) produce $\lambda + \ell$ pseudorandom bits, for $\ell > 0$. The process that "extends" λ bits into $\lambda + \ell$ bits is called a pseudorandom generator. More formally:

Definition 5.1 (PRG security)

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

$\mathcal{L}^G_{\text{prg-real}}$
QUERY(): $s \leftarrow \{0,1\}^\lambda$ return $G(s)$

$\mathcal{L}^G_{\text{prg-rand}}$
QUERY(): $z \leftarrow \{0,1\}^{\lambda+\ell}$ return z

The value ℓ 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 $\approx$). That is, the output distribution of the PRG can never actually *be* uniform in a mathematical sense. Because it has postive 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 `0`s as `1`s? 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?[1]

  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 λ and output length 2λ). The PRG only has 2^λ possible inputs, and so there are at most only 2^λ possible outputs. Among all of $\{\texttt{0},\texttt{1}\}^{2\lambda}$, this is a miniscule fraction indeed. Almost all strings of length 2λ 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^λ of them are possible, which is certainly a lot from the perspective of a program who runs in polynomial time in λ. 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

[1] For a list of statistical tests of randomness that are actually used in practice, see http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf.

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 *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 $\mathcal{A}_t$ that has the value t hard-coded:*

$$\boxed{\begin{array}{l}\underline{\mathcal{A}_t:}\\ \quad z \leftarrow \text{QUERY}()\\ \quad \text{return } z \stackrel{?}{=} t\end{array}}.$$

What is the distinguishing advantage of $\mathcal{A}_t$?

We can see easily what happens when $\mathcal{A}_t$ is linked to $\mathcal{L}^G_{\text{prg-rand}}$. We get:

$$\Pr[\mathcal{A}_t \diamond \mathcal{L}^G_{\text{prg-rand}} \Rightarrow 1] = 1/2^{2\lambda}.$$

What happens when linked to $\mathcal{L}^G_{\text{prg-real}}$ 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 $\mathcal{L}^G_{\text{prg-real}}$.

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:

$$\Pr[\mathcal{A}_t \diamond \mathcal{L}^G_{\text{prg-real}} \Rightarrow 1] = 1/2^\lambda.$$

Hence, the distinguishing advantage of $\mathcal{A}_t$ is $\left|1/2^{2\lambda} - 1/2^\lambda\right| \leqslant 1/2^\lambda$. If t is not a possible output of G, then we have:

$$\Pr[\mathcal{A}_t \diamond \mathcal{L}^G_{\text{prg-real}} \Rightarrow 1] = 0.$$

Hence, the distinguishing advantage of $\mathcal{A}_t$ is $\left|1/2^{2\lambda} - 0\right| = 1/2^{2\lambda}$.

In either case, $\mathcal{A}_t$ has negligible ditsinguishing advantage. This merely shows that $\mathcal{A}_t$ (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.

5.2 Application: Shorter Keys in One-Time-Secret Encryption

PRGs essentially convert a smaller amount of uniform randomness into a larger amount of *good-enough-for-polynomial-time* randomness. So a natural first application of PRGs is to obtain one-time-secure encryption but with keys that are shorter than the plaintext. This is the first of many acts of crypto-magic which are impossible except when restricting our focus to polynomial-time adversaries.

The main idea is to hold a short key k, expand it to a longer string using a PRG G, and use the result as a one-time pad on the (longer) plaintext. More precisely, let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be a PRG, and define the following encryption scheme:

Construction 5.2 (Pseudo-OTP)

$\mathcal{K} = \{0,1\}^\lambda$
$\mathcal{M} = \{0,1\}^{\lambda+\ell}$
$\mathcal{C} = \{0,1\}^{\lambda+\ell}$

KeyGen:
$k \leftarrow \mathcal{K}$
return k

Enc(k,m):
return $G(k) \oplus m$

Dec(k,c):
return $G(k) \oplus c$

The resulting scheme will not have (perfect) one-time secrecy. That is, encryptions of m_L and m_R will not be identically distributed in general. However, the distributions will be *indistinguishable* if G is a secure PRG. The precise flavor of security obtained by this construction is the following.

Definition 5.3 *Let Σ be an encryption scheme, and let $\mathcal{L}^\Sigma_{\text{ots-L}}$ and $\mathcal{L}^\Sigma_{\text{ots-R}}$ be defined as in Definition 2.6 (and repeated below for convenience). Then Σ has* ***(computational) one-time secrecy*** *if $\mathcal{L}^\Sigma_{\text{ots-L}} \approx$*

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$. *That is, if for all polynomial-time distinguishers* $\mathcal{A}$, *we have* $\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-L}} = 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-R}} = 1]$.

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$

QUERY$(m_L, m_R \in \Sigma.\mathcal{M})$:
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m_L)$
return c

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$

QUERY$(m_L, m_R \in \Sigma.\mathcal{M})$:
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m_R)$
return c

Claim 5.4 *Let* pOTP$[G]$ *denote Construction 5.2 instantiated with PRG* G. *If* G *is a secure PRG then* pOTP$[G]$ *has computational one-time secrecy.*

Proof We must show that $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}} \approx \mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$. As usual, we will proceed using a sequence of hybrids that begins at $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}}$ and ends at $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$. For each hybrid library, we will demonstrate that it is indistinguishable from the previous one. Note that we are allowed to use the fact that G is a secure PRG. In practical terms, this means that if we can express some hybrid library in terms of $\mathcal{L}^{G}_{\text{prg-real}}$ (one of the libraries in the PRG security definition), we can replace it with its counterpart $\mathcal{L}^{G}_{\text{prg-rand}}$ (or vice-versa). The PRG security of G says that such a change will be indistinguishable.

$\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}}$:

$\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}}$

QUERY$(m_L, m_R \in \{0,1\}^{\lambda+\ell})$:
$k \leftarrow \{0,1\}^{\lambda}$
$c := G(k) \oplus m_L$
return c

The starting point is $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}}$, shown here with the details of pOTP$[G]$ filled in.

$\mathcal{L}_{\text{hyb-1}}$:

QUERY(m_L, m_R):
$z \leftarrow$ QUERY$'()$
$c := z \oplus m_L$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-real}}$

QUERY$'()$:
$s \leftarrow \{0,1\}^{\lambda}$
return $G(s)$

The first hybrid step is to factor out the computations involving G, in terms of the $\mathcal{L}^{G}_{\text{prg-real}}$ library.

$\mathcal{L}_{\text{hyb-2}}$:

QUERY(m_L, m_R):
$z \leftarrow$ QUERY$'()$
$c := z \oplus m_L$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-rand}}$

QUERY$'()$:
$z \leftarrow \{0,1\}^{\lambda+\ell}$
return z

From the PRG security of G, we may replace the instance of $\mathcal{L}^{G}_{\text{prg-real}}$ with $\mathcal{L}^{G}_{\text{prg-rand}}$. The resulting hybrid library $\mathcal{L}_{\text{hyb-2}}$ is indistinguishable from the previous one.

$\mathcal{L}_{\text{hyb-3}}$:

$\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$

$\underline{\text{QUERY}(m_L, m_R)}$:
$z \leftarrow \{0,1\}^{\lambda+\ell}$
$c := z \oplus m_L$
return c

A subroutine has been inlined. Note that the resulting library is precisely $\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$! Here, OTP is instantiated on plaintexts of size $\lambda + \ell$ (and the variable k has been renamed to z).

$\mathcal{L}_{\text{hyb-4}}$:

$\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$

$\underline{\text{QUERY}(m_L, m_R)}$:
$z \leftarrow \{0,1\}^{\lambda+\ell}$
$c := z \oplus m_R$
return c

The (perfect) one-time secrecy of OTP allows us to replace $\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$ with $\mathcal{L}^{\text{OTP}}_{\text{ots-R}}$; they are interchangeable.

The rest of the proof is essentially a "mirror image" of the previous steps, in which we perform the same steps but in reverse (and with m_R hanging around instead of m_L).

$\mathcal{L}_{\text{hyb-5}}$:

$\underline{\text{QUERY}(m_L, m_R)}$:
$z \leftarrow \text{QUERY}'()$
$c := z \oplus m_R$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-rand}}$

$\underline{\text{QUERY}'()}$:
$z \leftarrow \{0,1\}^{\lambda+\ell}$
return z

A statement has been factored out into a subroutine, which happens to exactly match $\mathcal{L}^{G}_{\text{prg-rand}}$.

$\mathcal{L}_{\text{hyb-6}}$:

$\underline{\text{QUERY}(m_L, m_R)}$:
$z \leftarrow \text{QUERY}'()$
$c := z \oplus m_R$
return c

$\diamond$

$\mathcal{L}^{G}_{\text{prg-real}}$

$\underline{\text{QUERY}'()}$:
$s \leftarrow \{0,1\}^{\lambda}$
return $G(s)$

From the PRG security of G, we can replace $\mathcal{L}^{G}_{\text{prg-rand}}$ with $\mathcal{L}^{G}_{\text{prg-real}}$. The resulting library is indistinguishable from the previous one.

$\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$:

$\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$

$\underline{\text{QUERY}(m_L, m_R)}$:
$k \leftarrow \{0,1\}^{\lambda}$
$c := G(k) \oplus m_R$
return c

A subroutine has been inlined. The result is $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$.

Summarizing, we showed a sequence of hybrid libraries satisfying the following:

$$\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}} \equiv \mathcal{L}_{\text{hyb-1}} \approx \mathcal{L}_{\text{hyb-2}} \equiv \mathcal{L}_{\text{hyb-3}} \equiv \mathcal{L}_{\text{hyb-4}} \equiv \mathcal{L}_{\text{hyb-5}} \approx \mathcal{L}_{\text{hyb-6}} \equiv \mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}.$$

Hence, $\mathcal{L}^{\text{pOTP}[G]}_{\text{ots-L}} \approx \mathcal{L}^{\text{pOTP}[G]}_{\text{ots-R}}$, and pOTP has (computational) one-time secrecy. ■

⋆ 5.3 Taking the Contrapositive Point-of-View

We just proved the statement "if G is a secure PRG, then pOTP[G] has one-time secrecy," but let's also think about the contrapositive of that statement:

> If the pOTP[G] scheme is **not** one-time secret, then G is **not** a secure PRG.

If the pOTP scheme is not secure, then there is some distinguisher $\mathcal{A}$ that can distinguish the two $\mathcal{L}_{\text{ots-}\star}$ libraries with better than negligible advantage. Knowing that such an $\mathcal{A}$ exists, can we indeed break the security of G?

Imagine going through the sequence of hybrid libraries with this hypothetical $\mathcal{A}$ as the calling program. We know that one of the steps of the proof must break down since $\mathcal{A}$ successfully distinguishes between the endpoints of the hybrid sequence. Some of the steps of the proof were unconditional; for example, factoring out and inlining subroutines *never* has an effect on the calling program. These steps of the proof always hold; they are the steps where we write $\mathcal{L}_{\text{hyb-}i} \equiv \mathcal{L}_{\text{hyb-}(i+1)}$.

The steps where we write $\mathcal{L}_{\text{hyb-}i} \approx \mathcal{L}_{\text{hyb-}(i+1)}$ are *conditional*. In our proof, the steps $\mathcal{L}_{\text{hyb-1}} \approx \mathcal{L}_{\text{hyb-2}}$ and $\mathcal{L}^{\text{OTP}}_{\text{ots-R}} \approx \mathcal{L}_{\text{hyb-4}}$ relied on G being a secure PRG. So if a hypothetical $\mathcal{A}$ was able to break the security of pOTP, then that same $\mathcal{A}$ *must* also successfully distinguish between $\mathcal{L}_{\text{hyb-1}}$ and $\mathcal{L}_{\text{hyb-2}}$, or between $\mathcal{L}_{\text{hyb-5}}$ and $\mathcal{L}_{\text{hyb-6}}$ — the only conditional steps of the proof.

Let's examine the two cases:

- Suppose the hypothetical $\mathcal{A}$ successfully distinguishes between $\mathcal{L}_{\text{hyb-1}}$ and $\mathcal{L}_{\text{hyb-2}}$. Let's recall what these libraries actually look like:

$$\mathcal{L}_{\text{hyb-1}} = \boxed{\begin{array}{l}\underline{\textsc{query}(m_L, m_R)\text{:}} \\ \quad z \leftarrow \textsc{query}'() \\ \quad c = z \oplus m_L \\ \quad \text{return } c\end{array}} \diamond \mathcal{L}^G_{\text{prg-real}};$$

$$\mathcal{L}_{\text{hyb-2}} = \boxed{\begin{array}{l}\underline{\textsc{query}(m_L, m_R)\text{:}} \\ \quad z \leftarrow \textsc{query}'() \\ \quad c = z \oplus m_L \\ \quad \text{return } c\end{array}} \diamond \mathcal{L}^G_{\text{prg-rand}}.$$

  Interestingly, these two libraries share a common component that is linked to either $\mathcal{L}_{\text{prg-real}}$ or $\mathcal{L}_{\text{prg-rand}}$. (This is no coincidence!) Let's call that common library $\mathcal{L}^*$ and write

$$\mathcal{L}_{\text{hyb-1}} = \mathcal{L}^* \diamond \mathcal{L}^G_{\text{prg-real}}; \qquad \mathcal{L}_{\text{hyb-2}} = \mathcal{L}^* \diamond \mathcal{L}^G_{\text{prg-rand}}.$$

  Since $\mathcal{A}$ successfully distinguishes between $\mathcal{L}_{\text{hyb-1}}$ and $\mathcal{L}_{\text{hyb-2}}$, the following advantage is non-negligible:

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}^* \diamond \mathcal{L}^G_{\text{prg-real}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}^* \diamond \mathcal{L}^G_{\text{prg-rand}} \Rightarrow 1]\right|.$$

  But with a change of perspective, this means that $\mathcal{A} \diamond \mathcal{L}^*$ is a calling program that successfully distinguishes $\mathcal{L}^G_{\text{prg-real}}$ from $\mathcal{L}^G_{\text{prg-rand}}$. In other words, $A \diamond \mathcal{L}^*$ **breaks the PRG security of** G**!**

- Suppose the hypothetical $\mathcal{A}$ only distinguishes $\mathcal{L}_{\text{hyb-5}}$ from $\mathcal{L}_{\text{hyb-6}}$. Going through the reasoning above, we will reach a similar conclusion but with a different $\mathcal{L}^*$ than before (in fact, it will be mostly the same library but with m_L replaced with m_R).

So you can think of our security proof as a very roundabout way of saying the following:

> *If you give me an adversary/distinguisher $\mathcal{A}$ that breaks the one-time secrecy of* pOTP[G]*, then I can use it to build an adversary that breaks the PRG security of G. More specifically, G is guaranteed to be broken by (at least) one of the two distinguishers:*[2]

$$\mathcal{A} \diamond \boxed{\begin{array}{l}\underline{\text{QUERY}(m_L, m_R)\text{:}} \\ \quad z \leftarrow \text{QUERY}'() \\ \quad c = z \oplus m_L \\ \quad \text{return } c\end{array}} \qquad \textit{or} \qquad \mathcal{A} \diamond \boxed{\begin{array}{l}\underline{\text{QUERY}(m_L, m_R)\text{:}} \\ \quad z \leftarrow \text{QUERY}'() \\ \quad c = z \oplus m_R \\ \quad \text{return } c\end{array}}.$$

In fact, this would be the "traditional" method for proving security that you might see in many other cryptography texts. You would suppose you are given an adversary $\mathcal{A}$ breaking pOTP, then you would demonstrate why at least one of the adversaries given above breaks the PRG. Unfortunately, it is quite difficult to explain to students how one is supposed to *come up with* these PRG-adversaries in terms of the one-time-secrecy adversaries. For that reason, we will reason about proofs in the "if this is secure then that is secure" realm, rather than the contrapositive "if that is insecure then this is insecure."

5.4 Extending the Stretch of a PRG

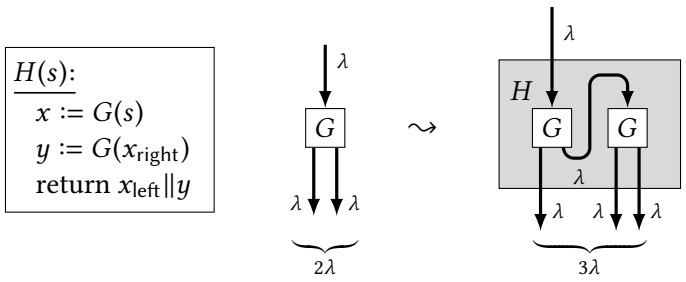
Recall that the *stretch* of a PRG is the amount by which the PRG's output length exceeds its input length. A PRG with very long stretch seems much more useful than one with small stretch. Is there a limit to the stretch of a PRG? Using only λ bits of true uniform randomness, can we generate 100λ, or even λ^3 pseudorandom bits?

In this section we will see that once you can extend a PRG a little bit, you can also extend it a lot. This is another magical feat that is possible with pseudorandomness but not with truly uniform distributions. We will demonstrate the concept by extending a PRG with stretch λ into one with stretch 2λ, but the idea can be used to increase the stretch of any PRG indefinitely (see the exercises).

Construction 5.5 (PRG feedback)

*Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a length-doubling PRG (*i.e.*, a PRG with stretch λ). When $x \in \{0,1\}^{2\lambda}$, we write x_{left} to denote the leftmost λ bits of x and x_{right} to denote the rightmost λ bits. Define the length-tripling function $H : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ as follows:*

[2] The statement is somewhat non-constructive, since we don't know for sure which of the two distinguishers will be the one that actually works. But a way around this is to consider a single distinguisher that flips a coin and acts like the first one with probability 1/2 and acts like the second one with probability 1/2.



Claim 5.6 *If G is a secure length-doubling PRG, then H (defined above) is a secure length-tripling PRG.*

Proof We want to show that $\mathcal{L}^H_{\text{prg-real}} \approx \mathcal{L}^H_{\text{prg-rand}}$. As usual, we do so with a hybrid sequence. Since we assume that G is a secure PRG, we are allowed to use the fact that $\mathcal{L}^G_{\text{prg-real}} \approx \mathcal{L}^G_{\text{prg-rand}}$. In this proof, we will use the fact twice: once for each occurrence of G in the code of H.

$\mathcal{L}^H_{\text{prg-real}}$:

$\mathcal{L}^H_{\text{prg-real}}$
QUERY():
$s \leftarrow \{0,1\}^\lambda$
$x := G(s)$
$y := G(x_{\text{right}})$
return $x_{\text{left}} \| y$

The starting point is $\mathcal{L}^H_{\text{prg-real}}$, shown here with the details of H filled in.

QUERY():
$x :=$ QUERY′()
$y := G(x_{\text{right}})$
return $x_{\text{left}} \| y$
$\diamond$
$\mathcal{L}^G_{\text{prg-real}}$
QUERY′():
$s \leftarrow \{0,1\}^\lambda$
return $G(s)$

The first invocation of G has been factored out into a subroutine. The resulting hybrid library includes an instance of $\mathcal{L}^G_{\text{prg-real}}$.

QUERY():
$x :=$ QUERY′()
$y := G(x_{\text{right}})$
return $x_{\text{left}} \| y$
$\diamond$
$\mathcal{L}^G_{\text{prg-rand}}$
QUERY′():
$z \leftarrow \{0,1\}^{2\lambda}$
return z

From the PRG security of G, we can replace the instance of $\mathcal{L}^G_{\text{prg-real}}$ with $\mathcal{L}^G_{\text{prg-rand}}$. The resulting hybrid library is indistinguishable.

QUERY():
$x \leftarrow \{0,1\}^{2\lambda}$
$y := G(x_{\text{right}})$
return $x_{\text{left}} \| y$

A subroutine has been inlined.

```
QUERY():
  x_left ← {0,1}^λ
  x_right ← {0,1}^λ
  y := G(x_right)
  return x_left ‖ y
```

Choosing 2λ uniformly random bits and then splitting them into two halves has exactly the same effect as choosing λ uniformly random bits and independently choosing λ more.

```
QUERY():
  x_left ← {0,1}^λ
  y := QUERY'()
  return x_left ‖ y
```
$\diamond$
```
L^G_prg-real
QUERY'():
  s ← {0,1}^λ
  return G(s)
```

The remaining appearance of G has been factored out into a subroutine. Now $\mathcal{L}^G_{\text{prg-real}}$ makes its second appearance.

```
QUERY():
  x_left ← {0,1}^λ
  y := QUERY'()
  return x_left ‖ y
```
$\diamond$
```
L^G_prg-rand
QUERY'():
  z ← {0,1}^2λ
  return z
```

Again, the PRG security of G lets us replace $\mathcal{L}^G_{\text{prg-real}}$ with $\mathcal{L}^G_{\text{prg-rand}}$. The resulting hybrid library is indistinguishable.

```
QUERY():
  x_left ← {0,1}^λ
  y ← {0,1}^2λ
  return x_left ‖ y
```

A subroutine has been inlined.

$\mathcal{L}^H_{\text{prg-rand}}$:
```
L^H_prg-rand
QUERY():
  z ← {0,1}^3λ
  return z
```

Similar to above, concatenating λ uniform bits with 2λ independently uniform bits has the same effect as sampling 3λ uniform bits. The result of this change is $\mathcal{L}^H_{\text{prg-rand}}$.

Through this sequence of hybrid libraries, we showed that:

$$\mathcal{L}^H_{\text{prg-real}} \equiv \mathcal{L}_{\text{hyb-1}} \approx \mathcal{L}_{\text{hyb-2}} \equiv \mathcal{L}_{\text{hyb-3}} \equiv \mathcal{L}_{\text{hyb-4}} \equiv \mathcal{L}_{\text{hyb-5}} \approx \mathcal{L}_{\text{hyb-6}} \equiv \mathcal{L}_{\text{hyb-7}} \equiv \mathcal{L}^H_{\text{prg-rand}}.$$

Hence, H is a secure PRG. ■

Exercises

5.1. Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$. Define $\text{im}(G) = \{y \in \{0,1\}^{\lambda+\ell} \mid \exists s : G(s) = y\}$, the set of possible outputs of G. For simplicity, assume that G is injective (i.e., 1-to-1).

(a) What is $|\text{im}(G)|$, as a function of λ and ℓ?

(b) A string y is chosen randomly from $\{0,1\}^{\lambda+\ell}$. What is the probability that $y \in \text{im}(G)$, expressed as a function of λ and ℓ?

5.2. Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be an injective PRG. Consider the following distinguisher:

$\mathcal{A}$
$x := \text{QUERY}()$
for all $s' \in \{0,1\}^\lambda$:
  if $G(s') = x$ then return 1
return 0

(a) What is the advantage of $\mathcal{A}$ in distinguishing $\mathcal{L}^G_{\text{prg-real}}$ and $\mathcal{L}^G_{\text{prg-rand}}$? Is it neglgible?

(b) Does this contradict the fact that G is a PRG? Why or why not?

(c) What happens to the advantage if G is not injective?

5.3. Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be an injective PRG, and consider the following distinguisher:

$\mathcal{A}$
$x := \text{QUERY}()$
$s' \leftarrow \{0,1\}^\lambda$
return $G(s') \stackrel{?}{=} x$

What is the advantage of $\mathcal{A}$ in distinguishing $\mathcal{L}^G_{\text{prg-real}}$ from $\mathcal{L}^G_{\text{prg-rand}}$?

Hint: When computing $\Pr[\mathcal{A} \diamond \mathcal{L}^G_{\text{prg-rand}}$ outputs 1], separate the probabilities based on whether $x \in \text{im}(G)$ or not. ($\text{im}(G)$ is defined in a previous problem)

5.4. For any PRG $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ there will be many strings in $\{0,1\}^{\lambda+\ell}$ that are impossible to get as output of G. Let S be any such set of impossible G-outputs, and consider the following adversary that has S hard-coded:

$\mathcal{A}$
$x := \text{QUERY}()$
return $x \stackrel{?}{\in} S$

What is the advantage of $\mathcal{A}$ in distinguishing $\mathcal{L}^G_{\text{prg-real}}$ from $\mathcal{L}^G_{\text{prg-rand}}$? Why does an adversary like this one not automatically break every PRG?

5.5. Show that the scheme from Section 5.2 does not have *perfect* one-time secrecy, by showing that there must exist two messages m_1 and m_2 whose ciphertext distributions differ.

Hint: There must exist strings $s_1, s_2 \in \{0,1\}^{2\lambda}$ where $s_1 \in \text{im}(G)$, and $s_2 \notin \text{im}(G)$. Use these two strings to find two messages m_1 and m_2 whose ciphertext distributions assign different probabilities to s_1 and s_2. Note that it is legitimate for an attacker to "know" s_1 and s_2, as these are properties of G alone, and independent of the random choices made when executing the scheme.

5.6. (a) Let f be any function. Show that the following function G is **not** a secure PRg (even if f is!). Describe a successful distinguisher and explicitly compute its advantage:

```
G(s):
  return s||f(s)
```

(b) Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ be a candidate PRG. Suppose there is a polynomial-time algorithm V with the property that it inverts G with non-negligible probability. That is,

$$\Pr_{s \leftarrow \{0,1\}^\lambda}\left[V(G(s)) = s\right] \text{ is non-negligible.}$$

Show that if an algorithm V exists with this property, then G is not a secure PRG. In other words, construct a distinguisher contradicting the PRG-security of G and show that it achieves non-negligible distinguishing advantage.

Note: Don't assume anything about the output of V other than the property shown above. In particular, V might very frequently output the "wrong" thing.

5.7. In the "PRG feedback" construction H in Section 5.4, there are two calls to G. The security proof applies the PRG security rule to both of them, starting with the first. Describe what happens when you try to apply the PRG security of G to these two calls in the opposite order. Does the proof still work, or must it be in the order that was presented?

5.8. Let $\ell' > \ell > 0$. Extend the "PRG feedback" construction to transform any PRG of stretch ℓ into a PRG of stretch ℓ'. Formally define the new PRG and prove its security using the security of the underlying PRG.

5.9. Let $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a secure length-tripling PRG. For each function below, state whether it is also a secure PRG. If the function is a secure PRG, give a proof. If not, then describe a successful distinguisher and explicitly compute its advantage.

(a)
```
H(s):
  x := G(s)
  y := first λ bits of x
  z := last λ bits of x
  return G(y)||G(z)
```

(b)
```
H(s):
  x := G(s)
  y := first 2λ bits of x
  return y
```

(c)
```
H(s):
  x := G(s)
  y := G(s)
  return x||y
```

(d)
```
H(s):
  x := G(s)
  y := G(0^λ)
  return x||y
```

(e)
```
H(s):
  x := G(s)
  y := G(0^λ)
  return x ⊕ y
```

(f)
```
// H : {0,1}^{2λ} → {0,1}^{3λ}

H(s):
  x := G(s_left)
  y := G(s_right)
  return x ⊕ y
```

(g)
```
// H : {0,1}^{2λ} → {0,1}^{6λ}

H(s):
  x := G(s_left)
  y := G(s_right)
  return x‖y
```

5.10. Let $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a secure length-tripling PRG. Prove that each of the following functions is also a secure PRG:

(a)
```
// H : {0,1}^{2λ} → {0,1}^{4λ}

H(s):
  y := G(s_right)
  return s_left‖y
```

Note that H includes half of its input directly in the output. How do you reconcile this fact with the conclusion of Exercise 5.6(b)?

(b)
```
// H : {0,1}^{2λ} → {0,1}^{3λ}

H(s):
  return G(s_left)
```

5.11. A frequently asked question in cryptography forums is whether it's possible to determine which PRG implementation was used by looking at output samples.

Let G_1 and G_2 be two PRGs with matching input/output lengths. Define two libraries $\mathcal{L}^{G_1}_{\text{which-prg}}$ and $\mathcal{L}^{G_2}_{\text{which-prg}}$ as follows:

$\mathcal{L}^{G_1}_{\text{which-prg}}$
```
QUERY():
  s ← {0,1}^λ
  return G_1(s)
```

$\mathcal{L}^{G_2}_{\text{which-prg}}$
```
QUERY():
  s ← {0,1}^λ
  return G_2(s)
```

Prove that if G_1 and G_2 are both secure PRGs, then $\mathcal{L}^{G_1}_{\text{which-prg}} \approx \mathcal{L}^{G_2}_{\text{which-prg}}$ — that is, it is infeasible to distinguish which PRG was used simply by receiving output samples.

★ 5.12. Prove that if PRGs exist, then P ≠ NP.

Hint: $\{y \mid \exists s : G(s) = y\} \in \text{NP}$.

Note: This implies that $\mathcal{L}^{G}_{\text{prg-real}} \not\equiv \mathcal{L}^{G}_{\text{prg-rand}}$ for all G. That is, there can be no PRG that is secure against computationally unbounded distinguishers.