1 Recap: Pseudorandom Generators

1. It is possible to construct a hard-core predicate for any one-way function. Let $f : \{0,1\}^* \to \{0,1\}^*$ be any one-way function (or permutation). We define $f'(r, x) = (r, f(x))$ for $|r| = |x|$. Then f' is also a one way function (or permutation, respectively), and $h(r, x) = \langle r, x\rangle \bmod 2$ is a hard core predicate for f'.

2. A pseudorandom generator exists under the assumption that a one-way permutation exists. Formally, if f is a OWP and h is a hard-core predicate for f, then $G(s) = (f(s), h(s))$ is a PRG with output length $\ell(n) = n + 1$.

3. If there exists a PRG $G(s) = (f(s), h(s))$ with output length $\ell(n) = n + 1$, then

$$G'(s) = (h(s), h(f(s)), h(f^{(2)}(s)), \cdots, h(f^{(m-1)}(s)))$$

is a PRG of output length m for any $m = \mathrm{poly}(|s|)$.

While it is well-known that the existence of a PRG implies that a OWF must exist, is the converse also true? That is, does the existence of a one-way function also imply that a pseudorandom generator must exist? In fact, this turns out to be true as well. Hastad, Impagliazzo, Levin, and Luby established in 1989 that a PRG can be constructed from any OWF. Their construction is much more complicated than the one for OWPs, because it must address the issue that the OWF f may be very "unstructured," and thus the distribution of $f(x)$ may be very different from uniform, even when x is uniform. The HILL PRG construction utilizes a random seed of length n^{10} for a OWF of input length n. The details of the construction are beyond the scope of this class.

2 Pseudorandom Functions

2.1 Preliminary Concepts

Having already developed a precise definition for a pseudorandom *string* of bits, a natural extension is, what would a random *function* look like?

A function from $\{0,1\}^n$ to $\{0,1\}$ is given by specifying an output bit for every one of its inputs, of which there are 2^n. Therefore, the set of *all* functions from $\{0,1\}^n$ to $\{0,1\}$ contains exactly 2^{2^n} functions; a "random function" (with this domain and range) is a uniformly choice from this set. Such a function can also be viewed as a uniformly random 2^n-bit string, which simply lists all the function's outputs. However, stated this way, it is impossible to even look at the entire string efficiently (in $\mathrm{poly}(n)$ time). Therefore, we define a model in which we give *oracle* access to a function.

Writing $\mathcal{A}^f$ signifies that $\mathcal{A}$ has query access to f, i.e., $\mathcal{A}$ can (adaptively) query the oracle on any input x and receive the output $f(x)$. However, $\mathcal{A}$ only has a "*black-box*" (input/output) view of f, without any knowledge of how the function f is evaluated.

Definition 2.1 (Oracle indistinguishability)**.** Let $\mathcal{O} = \{O_n\}$ and $\mathcal{O}' = \{O'_n\}$ be ensembles of probability distributions over functions from $\{0,1\}^{\ell_1(n)}$ to $\{0,1\}^{\ell_2(n)}$, for some $\ell_1(n), \ell_2(n) = \mathrm{poly}(n)$. We say that $\mathcal{O} \overset{c}{\approx} \mathcal{O}'$ if, for all nuppt distinguishers $\mathcal{D}$,

$$\mathbf{Adv}_{\mathcal{O},\mathcal{O}'}(\mathcal{D}) := \left| \Pr_{f \leftarrow O_n}[\mathcal{D}^f(1^n) = 1] - \Pr_{f \leftarrow O'_n}[\mathcal{D}^f(1^n) = 1] \right| = \mathrm{negl}(n).$$

Naturally, we say that $\mathcal{O} = \{O_n\}$ is *pseudorandom* if

$$\mathcal{O} \overset{c}{\approx} \left\{ U\Big(\{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)} \Big) \right\},$$

i.e., if no efficient adversary can distinguish (given only oracle access) between a function sampled according to O_n, and a uniformly random function, with more than negligible advantage.

Definition 2.2 (PRF Family)**.** A family $\left\{ f_s : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)} \right\}_{s \in \{0,1\}^n}$ is a *pseudorandom function family* if it is:

- *Efficiently computable*: there exists a deterministic polynomial-time algorithm F such that $F(s,x) = f_s(x)$ for all $s \in \{0,1\}^n$ and $x \in \{0,1\}^{\ell_1(n)}$.
- *Pseudorandom*: $\{U(\{f_s\})\}$ is pseudorandom.

Having developed a precise definition of a pseudorandom family of functions, the natural questions arises: Does such a primitive even exist? And under what assumptions?

Notice that if $\ell_1(n) = O(\log n)$, all the outputs values of a function $f : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)}$ can be written down as a string of exactly $2^{\ell_1(n)} \cdot \ell_2(n) = \mathrm{poly}(n)$ bits. Moreover, all the function values can be queried in polynomial time, given oracle access. Therefore, a PRF family with $O(\log n)$-length input may be seen as a PRG, and vice-versa. But do there exist PRF families with longer input lengths — say, n?

Question 1. Let $\{f_s\}$ be a pseudorandom function family. Is the family $\{g_s\}$ where $g_s(x) = f_s(x) \| 0$ also necessarily pseudorandom?

2.2 Constructing PRFs

Theorem 2.3. *If a pseudorandom generator exists (i.e., if a one-way function exists), then a pseudorandom function family exists for any $\ell_1(n), \ell_2(n) = \mathrm{poly}(n)$.*

At first glance, this theorem may seem completely absurd. The number of functions in the family $\{f_s\}$ with a seed length $|s| = n$ is at most 2^n, whereas the total number of functions overall (even with just one-bit outputs) is at least 2^{2^n}. Therefore, our function family is $\approx 2^{-2^n}$-sparse, i.e., the family $\{f_s\}$ makes up only a *doubly exponentially* small subset of the entire space of functions.

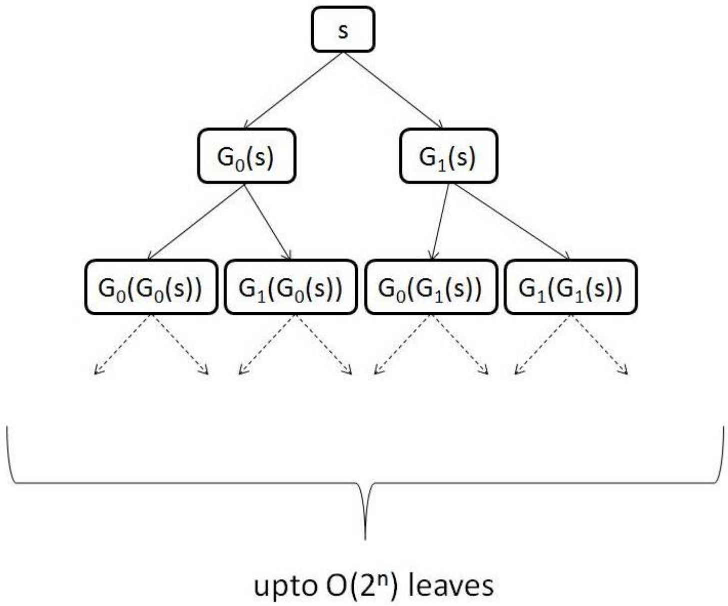
Proof of Theorem 2.3. For simplicity, we prove the theorem for $\ell_1(n) = \ell_2(n) = n$; extending to other values is straightforward.

Our objective is to "stretch" an n-bit uniformly random string to produce an exponential (at least 2^n) number of "random-looking" strings. Assume without loss of generality that G is a PRG with output length $\ell(n) = 2n$. The basic idea is to view the output of G as two length-n pseudorandom strings, which can be used recursively as inputs to G to generate an exponential number of strings.

Formally, view G as a pair of length-preserving functions G_0, G_1 (i.e., $|G_0(s)| = |G_1(s)| = |s|$), where

$$G(s) = G_0(s) \mid G_1(s).$$

The idea behind the PRF construction is that the function $f_s(x)$ computes a path, specified by the bits of x starting from the root seed s, as shown below:



Formally, we define the function $f_s(\cdot)$ as

$$f_s(x) = G_{x_n}(\cdots G_{x_2}(G_{x_1}(s))\cdots).$$

Why might we expect f_s to "look random," for a uniformly random (secret) seed s? Intuitively, $G_0(s)$ and $G_1(s)$ "look like" independent uniform n-bit strings, and we might expect this pseudorandomness to propagate downward through the layers of the tree. Let us try to prove this rigorously.

Attempt 1: Design a sequence of hybrid experiments where each leaf of the tree is successively replaced by its "ideal" form, i.e., with a uniform n-bit string. Clearly, the 0th hybrid corresponds to the "real" tree construction, and the 2^nth corresponds to a truly random function. However, this approach is flawed, as it requires 2^n hybrid steps. (As an exercise, show that the hybrid lemma is *false*, in general, for an exponential number of hybrid steps.)

Attempt 2: Successively replace each *layer* of the tree with ideal uniform, independent entries (all at once). Thus, H_0 corresponds to the real tree construction, and H_n corresponds to a truly random function. Note that we now have only n hybrid steps.

More formally, we describe hybrid distributions defining (a distribution over functions) f as follows:

- H_0 is the real tree construction, with a uniformly random root s, and $f(x) = G_{x_n}(\cdots G_{x_1}(s)\cdots)$.
- For $i \in [n]$, H_i is the tree construction, but using uniformly random seeds across the ith layer of the tree. Formally, $f(x) = G_{x_n}(\cdots G_{x_{i+1}}(s_{x_i \cdots x_1})$, where the seeds s_y are uniformly random and independent for each $y \in \{0,1\}^i$.

As a warm-up, we first show that $H_0 \overset{c}{\approx} H_1$ (in the sense of oracle indistinguishability) assuming that G is a PRG. To prove this, we need to construct a simulator $\mathcal{S}$ that emulates one of H_0 or H_1 (as oracles), depending on whether its input is $G(U_n)$ or U_{2n}. That is, the simulator should use its input to answer arbitrary queries. The simulator $\mathcal{S}$ works as follows: given $(z_0, z_1) \in \{0,1\}^{2n}$, it answers each query x by returning $G_{x_n}(\cdots G_{x_2}(z_{x_1})\cdots)$.

It is easy to check that the simulator emulates the desired hybrids. First, if $(z_0, z_1) = (G_0(s), G_1(s))$ for $s \leftarrow U_n$, then $\mathcal{S}(z_0, z_1)$ answers each query $x \in \{0,1\}^n$ as

$$G_{x_n}(\cdots G_{x_2}(G_{x_1}(s))\cdots) = f_s(x),$$

exactly as in H_0. Similarly, if $(z_0, z_1) \leftarrow (U_n, U_n)$, then $\mathcal{S}$ answers each query exactly as in H_1. Now because $G(U_n) \stackrel{c}{\approx} U_{2n}$ and $\mathcal{S}$ is efficient, by the hybrid lemma we conclude that $H_0 \stackrel{c}{\approx} H_1$.

Unfortunately, this approach does not seem to scale too well when we go down to the deeper layers of the tree, because the simulator $\mathcal{S}$ would need to take as input an exponential number of input strings. However, we can make two observations:

- In H_i, all the subtrees growing from the ith level are *symmetric*, i.e., they are identically distributed and independent.
- The polynomial-time distinguisher $\mathcal{D}$ attacking the PRF can make only a *polynomial* number of queries to its oracle.

The key point is that the simulator then only needs to simulate $q(n) = \mathrm{poly}(n)$ number of subtrees in order to answer all the queries of the distinguisher correctly.

For the hybrids H_{i-1} and H_i, Algorithm 2.2 defines a simulator that takes $q(n) = \mathrm{poly}(n)$ pairs of n-bit strings.

Algorithm 1 Simulator $\mathcal{S}_i$ for emulating either H_{i-1} or H_i.

Input: $(z_0^1, z_1^1), \ldots, (z_0^q, z_1^q) \in \{0,1\}^{2n}$ for some large enough $q(n) = \mathrm{poly}(n)$

$j \leftarrow 1$
while there is a query $x \in \{0,1\}^n$ to answer **do**
  if prefix $x_1 \cdots x_i$ is not yet associated with any k **then**
    associate j to $x_1 \cdots x_i$
    $j \leftarrow j + 1$
  end if
  look up the k associated with prefix $x_1 \cdots x_i$
  answer $G_{x_n}(\cdots G_{x_{i+1}}(z_{x_i}^k)\cdots)$
end while

We analyze the behavior of $\mathcal{S}_i$. Suppose that the distinguisher $\mathcal{D}$ (making queries to $\mathcal{S}_i$) makes at most q queries, so the counter j never "overflows." Now, if each of the pairs $(z_0^j, z_1^j) \leftarrow G(U_n^j)$ are independent pseudorandom strings, then $\mathcal{S}_i$ answers each query x by $G_{x_n}(\cdots(G_{x_{i+1}}(G_{x_i}(U_n^k))\cdots)$, for a distinct k associated uniquely with the i-bit prefix of x. By construction, $\mathcal{S}_i$ therefore emulates H_{i-1} exactly. Similarly, if the $(z_0^j, z_1^j) \leftarrow U_{2n}^j$ are uniformly random and independent, then $\mathcal{S}_i$ simulates H_i.

At this point, we would like to conclude that $H_{i-1} \stackrel{c}{\approx} H_i$, but can we? To do so using the hybrid lemma, we would need to show that the two types of inputs to $\mathcal{S}_i$ (namely, a sequence of $q = \mathrm{poly}(n)$ *independent pairs* (z_0, z_1) each drawn from either $G(U_n)$ or U_{2n}) are indistinguishable. This can be shown via a straightforward hybrid argument, using the hypothesis that G is a PRG, and is left as an exercise. □

2.3 Consequences for (Un)Learnability

A family of functions is said to be *learnable* if any member of the family can be reconstructed efficiently (i.e., as code), given oracle access to the function. In this sense, a PRF family is *completely unlearnable*, in that no efficient adversary can determine *anything* about the values of the function (given oracle access) on any of the unqueried points. As a consequence, if a class of functions is expressive enough to "contain" a PRF family, then this class is unlearnable. E.g., under standard assumptions, the class NC^1 can implement PRFs, hence it is unlearnable.

Answers

Question 1. Let $\{f_s\}$ be a pseudorandom function family. Is the family $\{g_s\}$ where $g_s(x) = f_s(x)\|0$ also necessarily pseudorandom?

Answer. No.