Lecture Notes 14

35 Construction of a Pseudorandom Function Ensemble

Let $\ell(n) = n$. We show how to construct an efficiently computable ℓ-bit pseudorandom function ensemble, starting from a pseudorandom generator G with expansion factor $2n$.

Let s be an n-bit string, so $G(s)$ is a $2n$-bit string. Define $G_0(s)$ (resp. $G_1(s)$) to be the first (resp. last) n bits of $G(s)$. Define a function $f_s : \{0,1\}^n \to \{0,1\}^n$ as follows: For every $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ of length n, let

$$f_s(\sigma) = G_{\sigma_n}(\ldots(G_{\sigma_2}(G_{\sigma_1}(s)))\ldots).$$

The construction of f_s can be represented by a tree, the first three levels of which are shown in Figure 35.1. Each node on level k is labeled by an n-bit string s_τ, where τ is a string of length k. The left son of node s_τ is labeled by $s_{\tau 0} = G_0(s_\tau)$, and the right son is labeled by $s_{\tau 1} = G_1(s_\tau)$. The leaf label s_σ is the value of the function $f_s(\sigma)$.

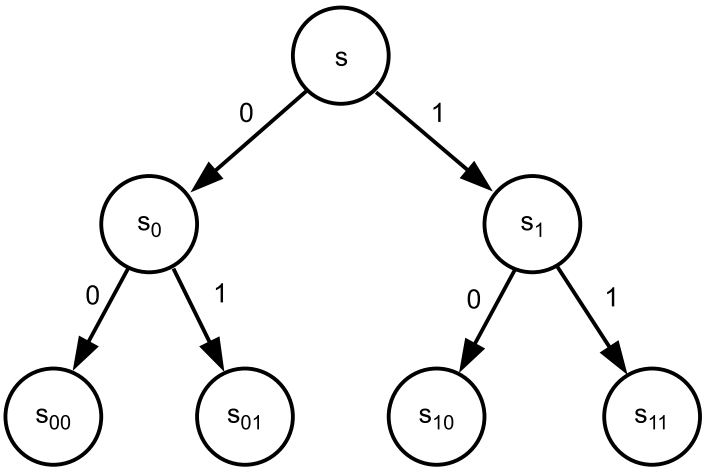

Figure 35.1: Tree representation of the function f_s.

Let F_n be the random variable that selects s uniformly from $\{0,1\}^n$, places it at the root of the tree, and then generates the labels of the remaining nodes as described above to obtain the resulting function f_s. Let $F = \{F_n\}_{n\in\mathbb{N}}$ be the function ensemble that corresponds to this random process.

Theorem 1 *If G is a pseudorandom generator, then F is an efficiently computable ensemble of pseudorandom functions.*

Proof: The proof is a hybrid construction. Fix a value for n.

The k^{th} hybrid H_n^k is a random variable ranging over functions that assign labels uniformly at the k^{th} level of the tree. In more detail, H_n^k selects s_τ uniformly from $\{0,1\}^n$ for each τ of length

k and places those 2^k values in the corresponding nodes of level k of the tree. It then generates the labels of the nodes below level k using the functions G_0 and G_1 as described above. The leaves of the tree define the resulting function, which in turn is uniquely determined by the level-k node labels $s_{\tau_1}, \ldots s_{\tau_{2^k}}$. We denote this function by

$$f_{s_{\tau_1},\ldots,s_{\tau_{2^k}}}(\sigma_1, \ldots, \sigma_n) = G_{\sigma_n}(\ldots(G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(s_{\sigma_1\ldots\sigma_k})))\ldots).$$

(Note that the nodes at levels less than k do not get labeled by this process, but only the leaves are relevant to the defined function.)

By this construction, we have $H_n^0 = F_n$, the original pseudorandom function construction, and $H_n^n = H_n$, the uniform distribution over functions $\{0,1\}^n \to \{0,1\}^n$.

Now, assume that F is not pseudorandom. Then there exists a probabilistic polynomial-time oracle machine M and a positive polynomial $p(\cdot)$ such that for infinitely many n,

$$\Delta(n) = |\Pr[M^{F_n}(1^n) = 1] - \Pr[M^{H_n}(1^n) = 1]| > \frac{1}{p(n)}.$$

Let $t(\cdot)$ be a polynomial bound on the running time of $M(1^n)$. It follows that $M(1^n)$ makes at most $t(n)$ queries of its oracle.

We construct a probabilistic polynomial-time distinguisher D that distinguishes $t(n)$ samples from $\{G(U_n)\}_{n\in\mathbb{N}}$ from $t(n)$ samples from $\{U_{2n}\}_{n\in\mathbb{N}}$ with non-negligible advantage.

Algorithm D on input $\alpha_1, \ldots, \alpha_{t(n)} \in \{0,1\}^{2n}$:
1. Select k uniformly from $\{0, 1, \ldots, n-1\}$.
2. Run $M(1^n)$ and output its result. Answer each of its oracle queries as described below.

The idea is that the oracle queries should be answered according to a tree that is dynamically constructed during the execution of D. Initially, all nodes are unlabeled. For each query $q_i = \sigma_1 \ldots \sigma_n$, the nodes at levels $> k$ that lie on the path q_i receive labels, as well as the brother of the level-$(k+1)$ node that lies on that path. Once a node is labeled, it retains that label throughout the duration of the run.

The labeling in turn works as follows. If the level-$(k+1)$ node $(\sigma_1 \ldots \sigma_k\sigma_{k+1})$ has not yet been labeled, it and its brother are labeled using input string α_i as follows:

$$s_{\sigma_1\ldots\sigma_k 0} = P_0(\alpha_i)$$
$$s_{\sigma_1\ldots\sigma_k 1} = P_1(\alpha_i),$$

where for any string y of length $2n$, $P_0(y)$ is the left half of y and $P_1(y)$ is the right half of y.[1] The nodes below $(\sigma_1 \ldots \sigma_{k+1})$ are labeled as usual using G, i.e.,

$$s_{\sigma_1\ldots\sigma_{k+1}\sigma_{k+2}\ldots\sigma_j} = G_{\sigma_j}(\ldots(G_{\sigma_{k+2}}(s_{\sigma_1\ldots\sigma_{k+1}}))\ldots)$$

for each $j = k+2, \ldots, n$.[2]

Observe that when the inputs to D are $t(n)$ samples from U_{2n}, then $P_0(\alpha_i)$ and $P_1(\alpha_i)$ are both uniformly distributed. This means that whenever a node on level-$(k+1)$ receives a label, the label is uniformly distributed. Hence, D behaves as M would given oracle H_n^{k+1}.

When the inputs to D are $t(n)$ samples from $G(U_n)$, then the nodes on level-$(k+1)$ are labeled according to $G_0(s)$ and $G_1(s)$ for randomly chosen s. That is, if $\alpha_i = G(U_n)$, then there is some s

[1] By this notation, we could have defined $G_0(s) = P_0(G(s))$ and $G_1(s) = P_1(G(s))$.

[2] To implement D so that it runs in polynomial time requires a polynomial size data structure for representing the non-empty portion of the tree. This is easily done, and we omit details.

such that $\alpha_i = G(s)$. The construction sets $s_{\tau 0} = P_o(\alpha_i) = G_0(s)$ and $s_{\tau 1} = P_o(\alpha_i) = G_1(s)$, so D behaves as M would given oracle H_n^k. Note that D does not actually label node τ with s, nor is it able to compute s from α_i, but the nodes below τ are labeled just as if node τ were labeled by s.

The following claims relate the success probability of D to the probability that M outputs 1. In both claims, K is the random value chosen in step 1 of Algorithm D.

Claim 1 $\Pr[D(G(U_n^{(1)}), \ldots, G(U_n^{(t(n))})) = 1 \mid K = k] = \Pr[M^{H_n^k}(^n) = 1]$.

Claim 2 $\Pr[D(U_{2n}^{(1)}, \ldots, U_{2n}^{(t(n))})) = 1 \mid K = k] = \Pr[M^{H_n^{k+1}}(1^n) = 1]$.

By the hybrid argument, since $\Delta(n) > \frac{1}{p(n)}$, it follows that

$$|\Pr[M^{H_n^k}(^n) = 1] - \Pr[M^{H_n^{k+1}}(1^n) = 1]| \geq \frac{\Delta(n)}{n} > \frac{1}{n \cdot p(n)}$$

for some k. Since also $\Pr[K = k] = \frac{1}{n}$, we have from claims 1 and 2 that

$$|\Pr[D(G(U_n^{(1)}), \ldots, G(U_n^{(t(n))})) = 1] - \Pr[D(U_{2n}^{(1)}, \ldots, U_{2n}^{(t(n))})) = 1]| > \frac{1}{n^2 \cdot p(n)}.$$

Thus, D distinguishes with inverse polynomial advantage and polynomially many samples between $G(U_n)$ and U_{2n}. Since $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{2n}\}_{n\in\mathbb{N}}$ are both efficiently constructible ensembles, it follows from Theorem 3 in section 28.2 of lecture 11 (Theorem 3.2.6 of the textbook) that $G(U_n)$ and U_{2n} are distinguishable in polynomial time by a single sample with polynomial advantage. Hence, G is not pseudorandom, a contradiction.

We conclude that F is an efficiently computable ensemble of pseudorandom functions, as desired. ■

36 An Application of a Pseudorandom Function Ensemble

Here's a simple example from the textbook of an application of a pseudorandom function ensemble F. Suppose a a secret society wants a way to identify its members. Imagine each club member is given the secret seed s that defines a function $f_s = F_n$. Members are instructed never to divulge s. Rather, to authenticate someone claiming to be a member, ask them to tell you the value of the secret function f_s on a random challenge string x. You also compute $f_s(x)$ and accept the person as a member if the two values agree. Note that you are giving away very little information on each such authentication. It can be shown that an adversary succeeds with negligible probability, even after a polynomial number of attempts. If not, the adversary can be used to distinguish F from H. (See section 3.6.3 of the textbook for further details.)