Lecture 10: 2/21/02

Last time we discussed the relation between One-Way Permutation (OWP) and pseudo-random generators (PRG): if a length-preserving OWP exists, we can use it in combination with its hardcore bit to build a PRG. In today's lecture, we will show that the same relation holds between One-way Functions (OWF) and PRG, so that if there exists a OWF a PRG must also exist. Having proved that, we will then show that the existence of a PRG implies the existence of a pseudo-random function(PRF)

1 One-Way Functions and PRG

In the last lecture we showed that following result first obtained in [HILL]:

Theorem 1 (OWP -> PRG)
Suppose $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way permutation, and $B : \{0,1\}^n \to \{0,1\}$ is a hardcore predicate on f. Then $G(x) = (f(x), B(x))$ is a PRG.

This theorem thus means that if a one-way permutation exists, there must also exist a pseudo-random generator. We will show that this theorem also holds when f is a one-way function that's not length preserving. First consider the case that that f is a injective OWF $f : \{0,1\}^k \to \{0,1\}^n$, with $n > k$. Define H to be a family of pairwise independent hash functions, so that every $h \in H$ satisfies the following:

$\forall x, y \in \{0,1\}^n$ and $\forall q, b \in \{0,1\}^m$ chosen randomly, $Pr[h(x) = q \text{ and } h(y) = b] = (\frac{1}{2^m})^2$

Intuitively, a pairwise independent hash function maps every distinct pair of input randomly and independently. An example of H is the set $\{h_{A,b}\}$, with $A \in \{0,1\}^{(n*m)}$, $b \in \{0,1\}^m$, and $h_{A,b}(x) = Ax \oplus b$. The following theorem can be used to construct a pseudo-random generator from a one-way function and a family of hash functions:

Theorem 2
Suppose $f : \{0,1\}^k \to \{0,1\}^n$ is an injective OWF, so that for any $y \in \{0,1\}^n$, $Pr[y = f(x)] <= 1/2^k$. Let H be a family of pairwise independent hash functions satisfying the properties above, with $h : \{0,1\}^n \to \{0,1\}^{(k-log(1/\epsilon))}$ for every $h \in H$. Then $(h, h(f(x)))$ is (∞, ϵ) indistinguishable from (h, r), where r is randomly chosen over $\{0,1\}^{(k-log(1/\epsilon))}$. Furthermore, the function g defined as $g(h, x) = (h, h(f(x)))$ is one-way.

To build a PRG based on this theorem, we first need to generalize the notion of hardcore predicates into hardcore functions: given $g : \{0,1\}^k \to \{0,1\}^n$ a one-way function, $B : \{0,1\}^n \to \{0,1\}^l$ is a *hardcore function* of g, if $(g(x), B(x))$ is indistinguishable from $(g(x), r)$, where r is random over $\{0,1\}^l$. By a generalization of the Goldreich-Levin Theorem we can show that there is a constant c such that every $(S, 1/S)$-one-way function can be modified into a one-way function admitting a hardcore function with $l = c \log S$.

A pseudo-random generator G then can be constructed given $h \in H$, the family of hash functions:

$G(h, x) =< h, h(f(x)), B(h, x) >$, where B is the hardcore function.

We have shown that a PRG can be constructed from an injective one-way function, but what about arbitrary OWFs? To show that any OWFs can be used to build a PRG, we first consider the case of regular length-preserving one-way functions. A function $f : \{0,1\}^n \to \{0,1\}^n$ is *regular* if every possible output has 2^k pre-images for some k; $f(x)$ has $2^{(n-k)}$ possible values, each taken with $1/2^{(n-k)}$ probability. Since f is not one-to-one, we cannot apply the same method for one-way permutations to get a pseudo-random generator. Instead, A PRG G can be constructed from a regular function using pairwise independent hash function:

$G(h_1, h_2, x) =< h_1, h_2, h_1(f(x)), h_2(x), b(x) >$, where $h_1 : \{0,1\}^n \to \{0,1\}^{(n-k)}$, $h_2 : \{0,1\}^n \to \{0,1\}^k$ are pairwise independent hash functions and $b()$ is a hard-core function.

Proof that G is indeed a pseudo-random generator can be found in [HILL].

Finally, given an arbitrary one-way function f, we can define $f'(x_1, \ldots, x_t) = f(x_1), \cdots, f(x_t)$, argue that f' is still one-way (which is trivial), prove that f' is "almost-regular" in the sense that almost all the elements in the range of f' have approximately the same number of pre-images (which is easy to prove for, say, $t = O(n^3)$), and finally generalize the previous result from regular to "almost-regular" one-way functions (which is the hardest step). Again, we refer to [HILL] for details.

2 Pseudo-random Functions

A pseudo-random function is "a family of functions with the property that the input-output behavior of a random instance of the family is computationally indistinguishable from that of a random function" [GB]:

Definition 1 *$F = \{f_s\}, f_s : \{0,1\}^n \to \{0,1\}^n$ is a family of (s, ϵ)* pseudo-random functions, *if for every adversary A running in time $\leq s$, $|Pr[A^{f_s}(x) = 1] - Pr[A^R(x) = 1]| \leq \epsilon$, where $R : \{0,1\}^n \to \{0,1\}^n$ is random.*

Intuitively, The above theorem states that given a function f, it should be hard for a adversary to tell from its input-output behavior whether f is chosen randomly or belongs to the family of PRF. The following theorem provides a way to construct PRFs from PRGs:

Theorem 3
If there is a pseudo-random generator G, then there is a pseudo-random function generator that generates families of functions that are PRF.

PROOF: The proof uses a binary tree construction. By results from the last lecture, we may assume without loss of generality that $G : \{0,1\}^n \to \{0,1\}^{2n}$. Let $G(x) = G1(x)||G_2(x)$, with $G_1(x), G_2(x) \in \{0,1\}^n$, and the key $s \in \{0,1\}^n$. We will define our pseudo-random function f_s with the behavior shown in figure 1. As Figure 1 shows, at each

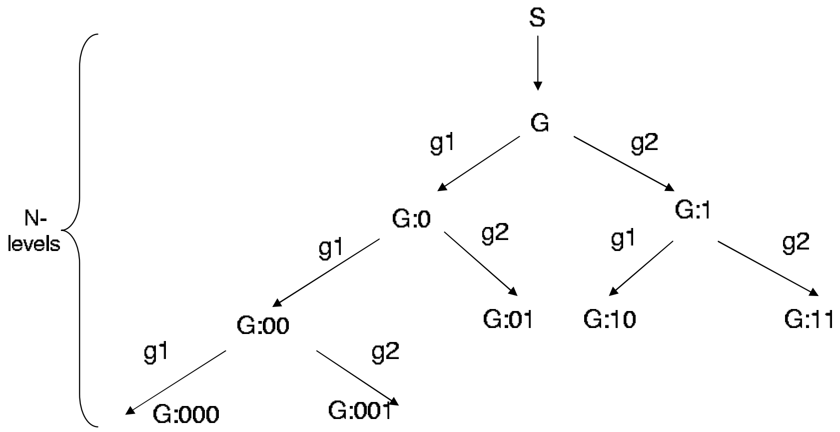

Figure 1: Construction of PRF from PRG: at each level the algorithm chooses the top or bottom half of the output of G based on the input x (the string after "G:"). The final output is the leaf of the tree

step the algorithm decides which output to use based on the current bit of x. Given n-bit strings x, Define $G_{x_i}(y)$ to output the top half of the bits of $G(y)$ (i.e., $G_1(y)$) if x_i (the ith bit of x) is 0, and the bottom half of the bits($G_2(y)$) if x_i is 1. Repeating this method for n(length of x) levels, the pseudo-random function then becomes

$$f_s(x) = G_{x_n}(G_{x_{n-1}}(...G_{x_1}(s)))$$

To see that the family of function $\{f_s\}$ defined above is indeed PRF, suppose instead that A is a good adversary running in time $\leq s$ against it, so $Pr[A^f s(x) = 1] - Pr[A^R(x) = 1] > \epsilon$. We will show that given such a A, we can build another adversary A' that can distinguish between the PRG G and a random function with high probability. We will use a hybrid argument (see Figure 2) for our proof. On the far left of Figure 2 is the binary tree representation of f_s, which we'll call H_0 since s (the 0th level) is random. Note that the first level of Ho involves running $G(s)$ to get two pseudo-random n-bit values and then use them as input to the next level; if we remove this level and give two random r_0, r_1 directly to the next level instead, we arrive at a new hybrid that we'll call $H1$ (everything on 1st level is random). Repeating this process, we get H_2 (4 random strings at level 2) H_3 (8 random string input at level 3), and so on until we reach H_n, at which point the whole binary tree is eliminated, and H_n simply becomes the random function R. Since we assume A has good probability in distinguishing between f_s (i.e., H_0) and R (H_n), it follows that A must be able to distinguish between H_i and H_{i+1} for some i. To put it formally,

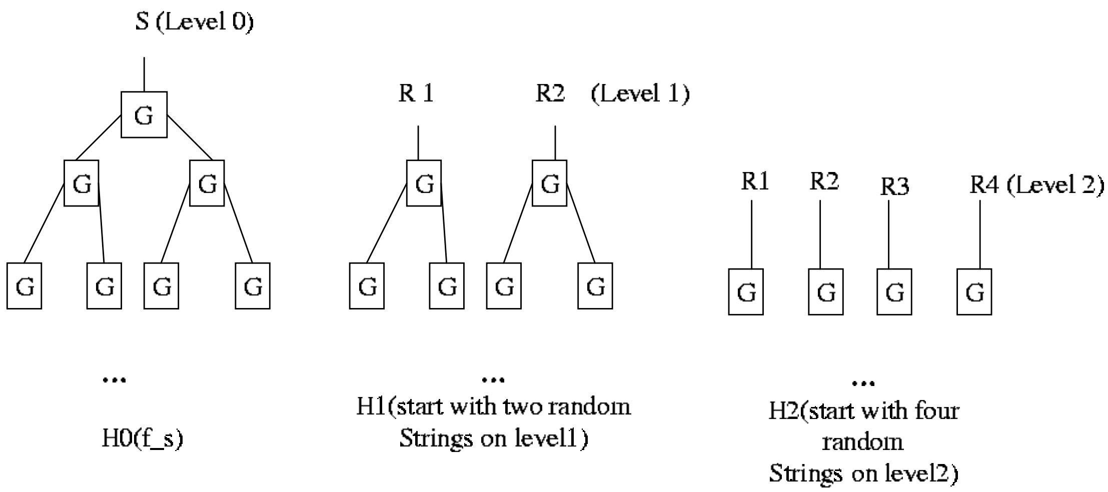

Figure 2: Examples of Hybrids

there exists $1 < i < n$ such that $Pr[A^{H_i} = 1] - Pr[A^{H_i+1} = 1] > \epsilon/n$

Also define l_x, where $1 <= |x| <= n$, to denote intermediate output of the binary tree on input x; thus for f_s, $l_{0x} = G_0(l_x)$ and $l_{1x} = G_1(l_x)$, with base cases $l_0 = G_0(s), l_1 = G_1(s)$. We'll now ready to build A' by simulating A. On input F, A' first generates a list of output $y_1, z_1, ..., y_s, z_s$ of F randomly, with $y_i, z_i \in \{0,1\}^n$ and s the upper-bound on the running time of A and thus the number of queries A can make. A' next builds a new hybrid H in the following way (see Figure 3):

1. let $H = H_0$
2. Simulate A on H; when A queries $H(x)$,
   (a) Find the longest common prefix p between x and x_i, where x_i is some value that H has been queried before
   (b) If the longest common prefix $p < i$, let u be the first i bits of x, take the next unused pair of (y_i, z_i), and assign $l_{u0} = y_i$ and $l_{u1} = z_i$. Compute $H(x)$ based on the new values of l_{u0} and l_{u1}.

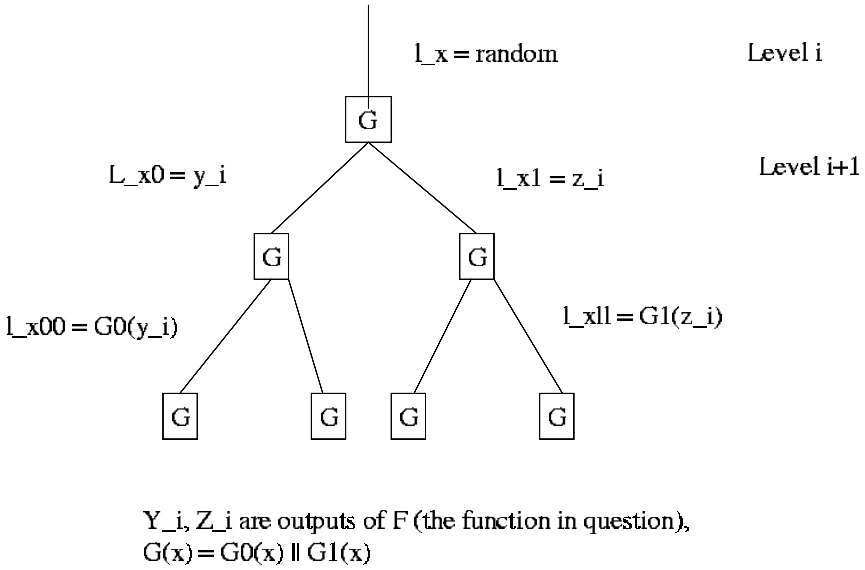

Figure 3: Example of How A' Constructs the Hybrid

(c) Otherwise, l_{u0} and l_{u1} must have already been assigned to some (y_i, z_i) previously, and we compute $H(x)$ directly

Note that the hybrid can be constructed in polynomail time because it's done vertically; we ignore any paths in the binary tree that A does not use, and A is allowed to see at most s paths since it can ask at most s queries. It's clear from the above construction that if y_i, z_i are random strings, the resulting H will be $H_i + 1$, since in H everything on the $i + 1$ level (that A queries) is now random. If y_i, z_i are instead strings generated by a PRG, then H is H_i, as the ith level is random but values on the $i + 1$ level are generated from a PRG. Since A is able to distinguish between H_i and H_{i+1}, A' can thus tell F is a pseudo-random generator or a function chosen randomly, a contradiction.

References

[HILL] Hastad and others. "A pseudorandom generator from any one-way function". SIAM Journal on Computing

[GB] Goldwasser S. and Bellare M. "Lecture notes on cryptography". http://www-cse.ucsd.edu/users/mihir/papers/gb.html.