CS 687 Introduction to Cryptography February 28, 2008

Lecture 11: Pseudorandom Functions

Instructor: Rafael Pass *Scribe: Tom Roeder*

1 Multi-Message Secure Encryption

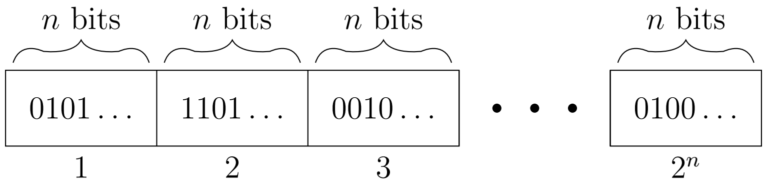
Recall that last time we began a discussion of multi-message secure encryption, and we determined that the scheme $\mathsf{Enc}_k(m) = m \oplus g(k)$, is not multi-message secure if g is a pseudorandom generator. In fact, it became clear that no deterministic encryption scheme can be multi-message secure, so we need to develop a probabilistic scheme.

One idea for such a scheme is to pick a random string r, then output $r||m \oplus f(r)$ for some function f. Ideally, we'd like the output of f to be a random string as well. One way to get such an f might be to have a long pseudorandom sequence of length on the order of $n2^n$. Then f could use r as an index into this sequence and return the n bits at r. But no pseudorandom generator can produce an exponential number of bits; our proof only held for pseudorandom generators with polynomial expansion.

If we were to use a pseudorandom generator, then r could be at most $O(\log n)$ bits long, so even if r is chosen randomly, we would end choosing two identical values of r with reasonable probability; this scheme would not be multi-message secure, though a stateful scheme that kept track of the values of r used could be.

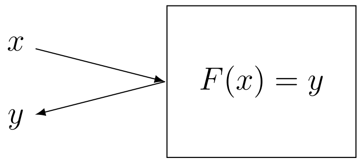
2 Random Functions

The scheme $r||m \oplus f(r)$ would be multi-message secure if f were a random function. Random functions can be described in one of two ways: a random function table or a machine that randomly chooses outputs given input and keeps track of its previous answers. The random function table can be viewed in a combinatorial manner as a long array that is stored by f. So, $f(x)$ returns the value at position nx.



Note that the description length of a random function is $n2^n$, so there are 2^{n2^n} random functions from $\{0,1\}^n \to \{0,1\}^n$. Let RF_n be the distribution that picks a function mapping $\{0,1\}^n \to \{0,1\}^n$ uniformly at random.

A computational description of a random function is as follows:



In this description, a random function is a machine that takes input x; if it has seen x before, then it looks up x, and outputs the same value as before. Otherwise, it chooses a new value y uniformly at random from $\{0,1\}^n$ and returns y. It then records that $F(x) = y$. These two descriptions give identical distributions of random functions.

But random functions have a long description length by definition, so f cannot be a random function in practice. Instead, we will define a new type of function that can be used in the place of a random function by any PPT algorithm.

3 Pseudorandom Functions

In the definition of pseudorandom functions, an adversary is an *oracle machine.* An oracle Turing machine M is a Turing machine that has been augmented with a component called an *oracle*: the oracle receives requests from M on a special tape and writes its responses to a tape in M.

Definition 1 $F = \{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s \in \{0,1\}^*}$ *is a* family of pseudorandom functions *if*

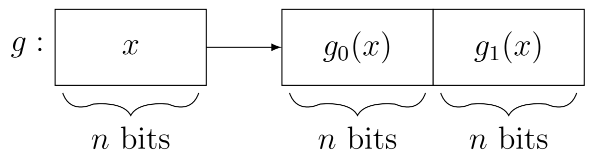
- *(Easy to compute): $f_s(x)$ is PPT computable (given s and x)*
- *(Pseudorandom): For all non-uniform PPT oracle machines $\mathcal{D}$ there is a negligible function ϵ such that*

$$\left| \Pr[s \leftarrow \{0,1\}^n : \mathcal{D}^{f_s(\cdot)}(1^n) = 1] - \Pr[F \leftarrow \mathsf{RF}_n : \mathcal{D}^{F(\cdot)}(1^n) = 1] \right| < \epsilon(n)$$

This definition differs critically from the definition of pseudorandom generators, since an adversary for a family of pseudorandom functions is allow to make many queries and see many values from the function before having to decide if the distribution is random or pseudorandom. So, any result that holds using a random function should also hold using a pseudorandom function, since otherwise a distinguisher exists that can distinguish random functions from pseudorandom functions.

Theorem 1 *If there is a pseudorandom generator, then there is a pseudorandom function.*

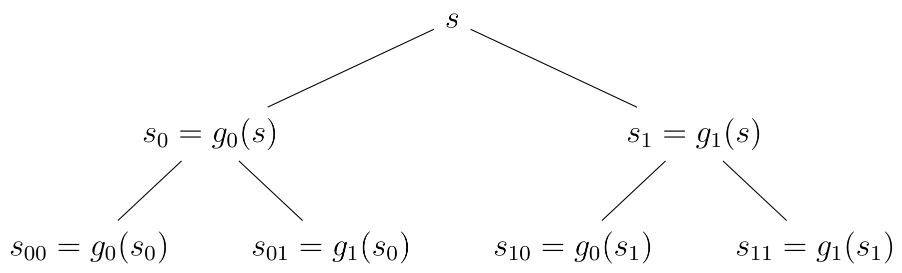
Proof. We have already shown that any pseudorandom generator g is sufficient to construct a pseudorandom generator g' that has polynomial expansion. So, without loss of generality, let g be a length-doubling pseudorandom generator.



Then we define f_s as follows to be a pseudorandom function:

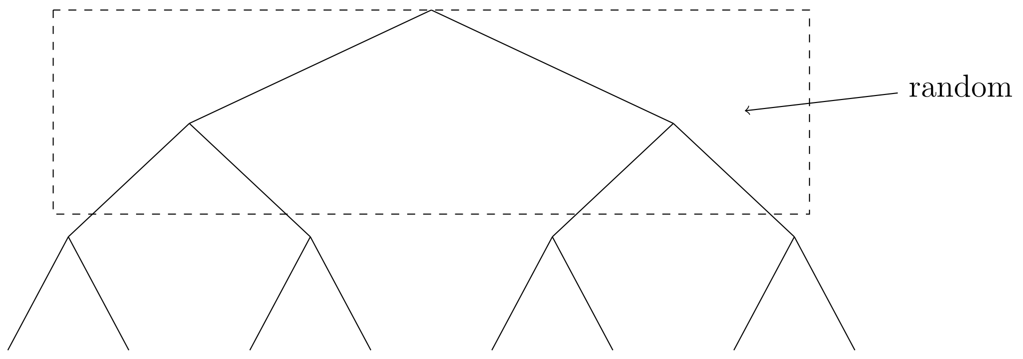
$$f_s(b_1 b_2 \dots b_n) = g_{b_n}(g_{b_{n-1}}(\cdots(g_{b_1}(s))\cdots))$$

f keeps only one side of the pseudorandom generator at each of n iterations. Thus, the possible outputs of f for a given input form a tree; the first three levels are shown in the following diagram. The leaves of the tree are the output of f.



The intuition about why f is a pseudorandom function is that a tree of height n contains 2^n leaves, so exponentially many values can be indexed by a single function with n bits of input. Thus, each unique input to f takes a unique path through the tree. The output of f is the output of a pseudorandom generator on a random string, so is random.

One approach to the proof is to look at the leaves of the tree. Build a sequence of hybrids by successively replacing each leaf with a random distribution. This approach does not work, since the hybrid lemma does not apply when there are exponentially many hybrids. Instead, we form hybrids by replacing successive levels of the tree: hybrid HF_n^i is formed by picking all levels through the ith uniformly at random, then applying the tree construction as before.



Note that $\mathsf{HF}_n^1 = \{s \leftarrow \{0,1\}^n : f_s(\cdot)\}$ (picking only the seed at random), which is the distribution defined originally. Further, $\mathsf{HF}_n^n = \mathsf{RF}_n$ (picking the leaves at random).

Thus, if $\mathcal{D}$ can distinguish $F \leftarrow \mathsf{RF}_n$ and f_s for a randomly chosen s, then $\mathcal{D}$ distinguishes $F_1 \leftarrow \mathsf{HF}_n^1$ and $F_n \leftarrow \mathsf{HF}_n^n$ with probability ϵ. Then the same argument as in the proof of the hybrid lemma applies, although the hybrid lemma itself is not defined for distributions of functions. The proof is left as an exercise for the reader; it shows that there is some i such that $\mathcal{D}$ distinguishes HF_n^i and HF_n^{i+1} with probability ϵ/n.

The difference between HF_n^i and HF_n^{i+1} is that level $i+1$ in HF_n^1 is $g(U_n)$, whereas in HF_n^{i+1}, level $i+1$ is U_n. Afterwards, both distributions continue to use g to construct the tree.

To finish the proof, we will construct one more set of hybrid distributions. Recall that there is some polynomial $p(n)$ such that the number of queries made by $\mathcal{D}$ is bounded by $p(n)$. So, we can now apply the first hybrid idea suggested above: define hybrid HHF_n^j that picks F from HF_n^i, and answer the first j new queries using F, then answer the remaining queries using HF_n^{i+1}.

But now there are only $p(n)$ hybrids, so the hybrid lemma applies, and $\mathcal{D}$ can distinguish HHF_n^j and HHF_n^{j+1} for some j with probability $\epsilon/(np(n))$. But HHF_n^j and HHF_n^{j+1} differ only in that HHF_n^{j+1} answers its $j+1$st query with the output of a pseudorandom generator on a randomly chosen value, whereas HHF_n^j answers its $j+1$st query with a randomly chosen value. The fact that $\mathcal{D}$ can distinguish these queries with inverse polynomial probability contradicts the claim that g is a pseudorandom generator.[1] ■

Note that the existence of pseudorandom functions shows that there are efficiently computable functions that cannot be learned by any polynomial-time machine. Further, we have seen that the existence of weak one-way functions implies the existence of strong one-way functions, which implies the existence of pseudorandom generators and hence pseudorandom functions. So, if factoring is hard, then there are concepts that are efficiently computable but cannot be learned.

[1] To be completely precise, we must also show that we can simulate HF_n^i, but it can be shown that choosing random values is identically distributed to HF_n^i; this is the same as the claim above that the combinatorial and the computational views of random functions are equivalent.