CSE 5852: Lecture 15

Prof. Benjamin Fuller
Scribe: Chao Shang

October 19, 2016

1 Review of Last Class

The last class introduced the expansion factor of pseudorandom generator(PRG), and demonstrated the existence of pseudorandom generators with arbitrary polynomial expansion factor. The proof is that the output of a generator result can be fed back in as input, as is shown in Figure 1.

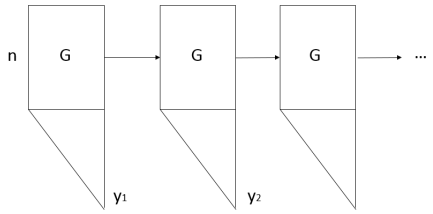

Figure 1: PRG-expansion

Then we have learned "what could we do with a truly random function?". There are two ways: 1) Create a $MAC(m)$ as $t = f(m)$; 2) Create an encryption scheme: $Enc(m)$: generate r, m, and send $c = (f(r) \oplus m, r)$.

Today we will introduce the definition of pseudorandom function and discuss how to construct pseudorandom functions from pseudorandom generators.

2 Pseudorandom Function

First, let's think about the goal of the function. We hope a pseudorandom function is an efficiently-computable keyed function that looks random to any polynomial-time distinguisher. That is no polynomial-time Attacker can tell if interacting with f or a random function.

The function f is a family of functions: $f : \{0,1\}^n \times \{0,1\}^n \to 0, 1^n$. We can consider there is no polynomial-time Attacker can tell if interacting with $f(s, .)$ or a random function.

Here we give the experiment of getting $f(s, .)$ and a random function.

1. $Exp - prf^{f,A}$
   (a) Select s from $\{0,1\}^k$
   (b) Repeat:
      1) Get x_i from A;
      2) Give $y_i = f(s, x_i)$.
   (c) Output A's bit
2. $Exp - r^A$
   (a) Initialize empty table
   (b) Repeat:
      1)Get x_i from A;
      2)Check table for x_i. If it exists, then return y;
      3) Generate random y_i ;
      4) Give y_i to A;
      5) Add(x_i, y_i) to table.
   (c) Output A's bit

Coming back to our discussion of pseudorandom functions, recall that we wish to construct a keyed function f chosen uniformly at randomly from a family $\mathcal{F}$ is indistinguishable from f chosen uniformly at random. Here we give the definition of PRF.

Definition 1. *A family f is a pseudorandom function, if for all probabilistic polynomial-time distinguishers A, there exists a negligible function ϵ such that:*

$$Pr[exp - prf^{f,A} = 1] - Pr[exp - r^A = 1] < \epsilon$$

Theorem 1. *Pseudorandom functions(PRFs) exist if and only if pseudorandom generators(PRGs) exist.*

Proof:
1. PRFs imply PRGs:
If we have a pseudorandom function:

$$f : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

We want to get: $G : \{0,1\}^k \to \{0,1\}^m$
Question: How to construct G?

$$G(s) = f(s, 0...0), f(s, 00...01), ..., f(s, 11...11)$$

2. PRGs imply PRFs:
First, we suppose we have the pseudorandom generator, as shown in Figure 2:

$$G : \{0,1\}^n \to \{0,1\}^{2n}$$

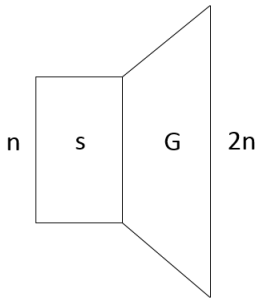

Figure 2: Pseudorandom generator

We'll start with the simpler case of constructing a function with one bit input:

$$f : \{0,1\}^n \to \{0,1\} \to \{0,1\}^n$$

Here is the approach to construct f:

$$f_s(0) = G_0(s)$$
$$f_s(1) = G_1(s)$$

Here G_0 is the first n bits of G and G_1 is the second n bits of G.

An alternate view of PRF is that a PRF is a PRG with exponential expansion where A can only look at some bits.

It seems like we could just use our previous result to expand the PRG to have exponential output length. There are two issues for this imagined giant PRG: 1) it does not have a security argument because the hybrid breaks down if there are super polynomial number of steps 2) it would take exponential time to compute. We need another strategy.

Let G be a pseudorandom generator with expansion factor $l(n) = 2n$, and denote $G(s) = (G_0(s), G_1(s))$, where $|s| = |G_0(s)| = |G_1(s)| = n$; that is, the seed length is a $n, G_0(s)$ is the first half of the output and $G_1(s)$ is the second half of the output. Here we use $s_0 = G_0(s)$ and $s_1 = G_1(s)$.

Construction 1. *(Goldreich-Goldwasser-Micali Construction 1984)*

Let G be a deterministic function that maps inputs of length n into outputs of length $2n$. Denote by $G_0(s)$ the first n bits of G's output, and by $G_1(s)$ the second n bits of G's output. For every $s \in \{0,1\}^n$, define the function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ as:

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

where $x = \{x_0, ..., x_n\}$.

We will define our pseudo-random function with the behavior shown in Figure 3. As it shows, this construction can be viewed as a full binary tree of depth n. We can see that each step the algorithm decides which output to use based on the current bit of x. Repeating this method for n levels, we can get the pseudo-random function: $f(s,x) = G_{x_n}(G_{x_{n-1}}(...G_{x_1}(s)...))$

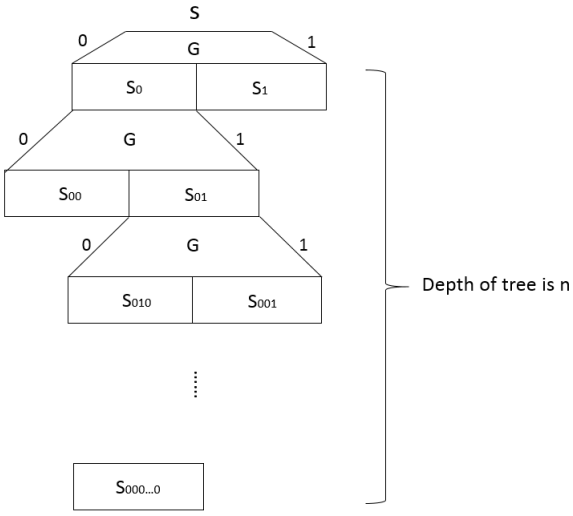

Figure 3: Pseudorandom function construction

The reason why we continue extending the construction is that if the intermediate result is pseudorandom, then it can be replaced by truly random strings, that can then be used as seeds to the pseudorandom generator once again.

The intuition behind the proof of security is shown in Figure 4. The idea is to perform a hybrid argument where each pseudorandom level is replaced by a truly random level. On the top of Figure 4 is the binary tree representation of f_s, which s in the 0th level is random. Note that the first level run G(s) to get two pseudorandom n-bit values and then use them as input to the next level; if we remove this level and give two random values directly to the next level instead, we arrive at a new hybrid. Repeating this process, we get the random strings at level $2, 3, .., n$. So in n level, it simply becomes the random function. Since we assume A has good probability in distinguishing between f and r, it follows that A must be able to distinguish between level i and level i+1 for some i. The only difference between hybrids i and $i+1$ is multiple copies of a PRG with independent seeds or multiple independent random strings. (The actual proof is significantly more complicated as the adversary is actively choosing the paths.)

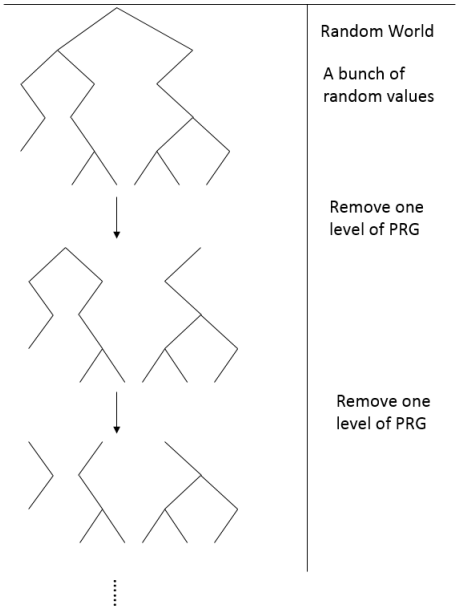

Figure 4: Examples of Hybrids