Lecture 6 - Length extension, pseudorandom functions

Boaz Barak

February 17, 2010

Quick review The PRG Axiom, encryption schemes with key size $\ll$ message size.

From last time

> **Theorem 1.** *If there exists a pseudorandom generator with stretch $\ell(n) = n + 1$ then for every constant c, there exists a pseudorandom generator with stretch $\ell(n) = n^c$.*

Polynomial hybrid argument To show $X \approx Z$ it suffices to come up with $Y_0, Y_1, \ldots, Y_m$ such that $Y_0 = X$, $Y_m = Z$ and $Y_i \approx Y_{i+1}$.

Proof of Theorem 1 Assume that we have a pseudornadom generator G' mapping n bits to $n+1$ bits. We'll construct from it a pseudorandom generator G mapping n bits to $\ell(n)$ bits for every $\ell(n) = n^c$. The running time of G will be roughly $\ell(n)$ times the running time of G'.

The operation of G is as follows: (notation: for a string $x \in \{0,1\}^k$, and $i < j \leq k$, $x_{[i\ldots j]}$ is $x_i x_{i+1} \cdots x_j$)

Input: $x \in \{0,1\}^n$.

$j \leftarrow 0$
$x^{(0)} \leftarrow x$
`while` $j < \ell(n)$`:`
    $j \leftarrow j + 1$
    $x^{(j)} \leftarrow \mathsf{G}'_n(x^{(j-1)}_{[1\ldots n]})$
    `output` $x^{(j)}_{n+1}$

We define random variables $Y^{(0)}, \ldots, Y^{(m)}$ over $\{0,1\}^m$. Intuitively, $Y^{(i)}$ will correspond to running the pseudorandom generator from the i^{th} iteration onwards, starting from the uniform distribution U_{n+i}. More formally, $Y^{(i)}$ is obtained by concatenating a random i bit to the output of the following algorithm G^{m-i} on input $x \leftarrow_{\mathrm{R}} \{0,1\}^n$:

Algorithm G^{j_0} Input: $x \in \{0,1\}^n$.

$j \leftarrow j_0$
$x^{(j)} \leftarrow x$
`while` $j < \ell(n)$`:`
    $y \leftarrow \mathsf{G}'(x^{(j-1)})$
    $x^{(j+1)} \leftarrow y_{[1..n]}$

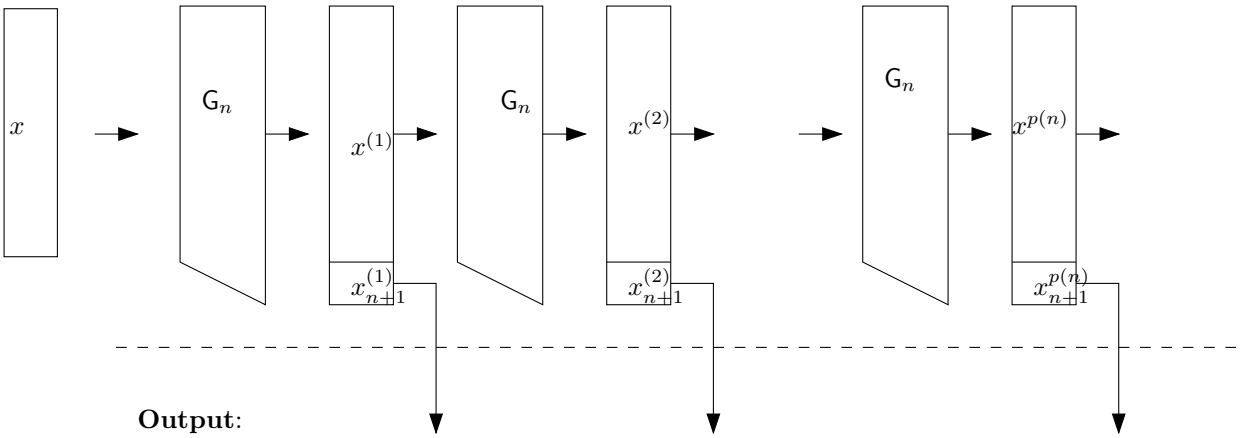

Figure 1: Extending output of pseudorandom genertator

`output` y_{n+1}
$j \leftarrow j + 1$

Note that $Y^{(0)} = \mathsf{G}(U_n)$ and $Y^{(m)} = U_m$. Therefore, the result will follow by showing that $Y^{(0)} \approx Y^{(m)}$. By the transitivity of computational indistinguishability, it suffices to prove:

Claim 2. *For every $i \in [m]$, $Y^{(i)} \approx Y^{(i+1)}$*

Proof. Note that $Y^{(i)} = U_i \mathsf{G}^{m-i}(U_n)$ and $Y^{(i+1)} = U_{i+1}\mathsf{G}^{m-i-1}(U_n)$. Both start with i random bits and so it suffices to show that $X = \mathsf{G}^{(m-i)}(U_n) \approx Y = U_1\mathsf{G}^{(m-i-1)}(U_n)$. Define $f : \{0,1\}^{n+1} \to \{0,1\}^{m-i}$ as follows: $f(y) = y_{n+1}\mathsf{G}^{(m-i-1)}(y_{[1..n]})$. Note that:

- $X = f(\mathsf{G}'(U_n))$.
- $Y = f(U_{n+1})$.

But since $\mathsf{G}'(U_n) \approx U_{n+1}$, it follows that $f(\mathsf{G}'(U_n)) \approx f(U_{n+1})$ for every polynomial-time computable function f. □

Note: This proof technique — proving that two distributions X and Y are indistinguishable by presenting *intermediate* distributions $X^{(0)}, \ldots, X^{(m)}$ with $X^{(0)} = X$ and $X^{(m)} = Y$ and the showing that $X^{(i)}$ is indistinguishable from $X^{(i+1)}$ — is called the *hybrid* technique, and is a very important technique in cryptographic proofs. I recommend that you also review this proof in Section 3.4 of the Boneh-Shoup book (note that they start with a warmup case, showing that if G is a pseudorandom generator then the function $G'(x_1, \ldots, x_k) = G(x_1)G(x_2)\cdots G(x_k)$ is also a pseudorandom generator).

Pseudorandom functions A random function $F(\cdot)$ from n bits to n bits can be thought of as the following process: for each one of its possible 2^n inputs x, choose a random n-bit string to be $F(x)$. This means that we need $2^n \cdot n$ coins (which is *a lot*) to choose a random function. We also need about that much size to store it.

We see that a function that can be described in n bits is very far from being a random function. Nevertheless we'll show that under our Axiom, there exists a *pseudorandom* function collection

that can be described and computed with poly(n) bits but is indistinguishable from a random function.

We let $\mathcal{F} = \{f_s\}_{s\in\{0,1\}^*}$ be a *collection of functions*. Suppose that $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$ (this is not important and we can generalize the definition and constructions to different input and output lengths). We say that the collection is *efficiently computable* if the mapping $s, x \mapsto f_s(x)$ is computable in polynomial time. Fix an efficiently computable collection and consider the following two games:

Game 1:

- s is chosen at random in $\{0,1\}^n$.
- Adversary gets black-box access to the function $f_s(\cdot)$ for as long as it wishes (but less than its poly(n) running time).
- Adversary outputs a bit $v \in \{0,1\}$.

Game 2:

- A random function $F : \{0,1\}^n \to \{0,1\}^n$ is chosen.
- Adversary gets black-box access to the function $F(\cdot)$ for as long as it wishes (but less than its poly(n) running time).
- Adversary outputs a bit $v \in \{0,1\}$.

$\mathcal{F}$ is a *pseudorandom function* (PRF) ensemble, if for every poly-time Adv and poly-bounded $\epsilon : \mathbb{N} \to [0,1]$, and large enough n

$$\Big|\Pr[\mathsf{Adv} \text{ outputs 1 in Game 1}] - \Pr[\mathsf{Adv} \text{ outputs 1 in Game 2}]\Big| < \epsilon(n)$$

GGM result Intuitively, it is not at all clear that such functions should exist. However, it was proven by Goldreich Goldwasser and Micali that if PRG exist then so do PRFs. (The other direction is pretty easy — can you see why?)

Construction of PRFs As you can see on the web page, there are several candidate constructions for PRFs. However, for us the important thing is that we don't need to introduce a new axiom, since we can construct them directly from ordinary PRG. That is, we have the following theorem:

Theorem 3. *IF the PRG Axiom is true, then there exist pseudorandom functions.*

Proving Theorem 3 The best way to think about the construction is the following. Suppose that you have a PRG $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{2n}$. Construct a depth n full binary tree, which you label as follows: the root is labeled with a string s (the seed of the function). For each non-leaf node labeled v, the two children are labeled with $\mathsf{G}_0(v) \triangleq \mathsf{G}(v)_{[1\ldots n]}$ and $\mathsf{G}_1(v) \triangleq \mathsf{G}(V)_{[n+1\ldots 2n]}$.

We have 2^n leaves and we can identify each one of them with a string in $\{0,1\}^n$ in the natural way (the string depicts the path from the root to the leaf with 0 meaning take the left child and 1 meaning take the right child). We define $f_s(x)$ to be the label of the leaf corresponding to x.

Although the full tree is of exponential size to compute $f_s(x)$ we only need to follow an n-long path from the root to the leaf and so it is computable in polynomial time.

Another way to state this definition is that $f_s(x)$ is defined to be

$$\mathsf{G}_{x_n}(\mathsf{G}_{x_{n-1}}(\cdots \mathsf{G}_{x_1}(s))\cdots)$$

Analysis We now show the following result:

Lemma 4. *If* $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{2n}$ *is a pseudorandom generator, then the construction described above is a PRF collection.*

Proof. Suppose for the sake of contradiction that there is an T-time adversary Adv that manages to distinguish between access to $f_s(\cdot)$ and access to a random function with probability at least ϵ. We'll convert it to a T' adversary that manages to distinguish between $\mathsf{G}(U_n)$ and U_{2n} with probability at least ϵ', for T' and ϵ' polynomially related to T, ϵ.

Without loss of generality. We're going to make some modifications to the behavior of Adv which will not change its distinguishing probability and not add too much to its running time but will make our life a little easier. Since such modifications can be made, we can just *assume* that Adv is already of the modified form. That is, we assume the following about Adv:

- It makes exactly T queries: if it makes less, we'll change it to ask "meaningless" queries.
- It never asks the same question twice: we can modify Adv to keep track of all the responses it received from its oracle and whenever it wants to get an answer for a query it already asked, it can use that table.

We now consider the interaction of Adv with an oracle computing $f_s(\cdot)$. The algorithm we specified for f_s is a stateless algorithm that given s and x computes $f_s(x)$ without relying on any precomputed information. However, we can implement the oracle in any way we want as long as it still computes $f_s(\cdot)$. Thus, we'll implement it in the following way:

Description of the $f_s(\cdot)$ oracle. The oracle will build keep the binary tree we described above. Of course it cannot keep the entire tree, but it will build it and maintain it in response to each query of Adv.

- Initially the tree contains only the root which is labeled with s.
- Whenever Adv makes a query for $f_s(x)$, the oracle will look at the path from the leaf x to the root. Let v be the lowest point in the path which is already computed. The oracle will compute all the values along the path from v to x and store the labels, finally returning the label of x.

Note: Whenever the oracle invokes G on a label x of an internal (non-leaf) node v, it will label the children of v with $x_0 = \mathsf{G}_0(x)$ and $x_1 = \mathsf{G}_1(x)$ and *erase* the label of v. Note that this is OK since the oracle will never need to use these values again. Also note that the oracle needs to make at most $M = T \cdot n$ invocations of G during the entire process.

The hybrids. We are going to use a hybrid argument to prove that the interaction of Adv with this oracle is indistinguishable from an interaction with a random function. For $i = 0, \ldots, M$ we define the hybrid H^i in the following way:

This is the adversary's view in an interaction with the oracle *except* that for the first i times when the oracle is supposed to invoke G to label the two children of some node v labeled x, the oracle does *not* do this but rather does a "fake invocation": instead of labeling v's children with $(x_0, x_1) = \mathsf{G}(x)$ it chooses x_0, x_1 at random from $\{0,1\}^n$ and labels the two children with x_0, x_1, erasing the label of v.

Clearly H^0 is equal to the adversary's view when interacting with f_s while H^M is equal to the adversary's view when interacting with a random function.

Thus, we only need to prove that H^i is indistinguishable from H^{i-1}. However, this follows from the fact that G is a pseudorandom generator.

Proof of indistinguishability of H^i and H^{i-1}. We'll make the following modification to the operation of the oracle in H^i: in the first i "fake invocations" of G, when the oracle chooses at random x_1 and x_2 and uses these to label the nodes of v, it will do something a bit different: it will erase the label of v but use a "lazy" evaluation: it will mark the children of v as "to be chosen at random" and will choose each of these labels at random only when it will be needed at a future time. (Note that typically the label for one of the children will be needed in the next step, but the label for the other child may only be required to answer a future query or perhaps never). Even the root s is not chosen initially but rather is initiated with the "to be chosen at random" label. Note that for the first i "fake invocations" whenever the value for an internal node is used then it is immediately deleted, and so in the first i steps all the internal nodes are either untouched or marked "to be chosen at random". The important observation is that all this is only about the oracle's internal computation and has no effect on the view of the adversary. (Also, the oracle can stop being lazy and choose values for some of the nodes without any effect on the view.)

We'll now prove the indistinguishability. Suppose we had a distinguisher C between H^i and H^{i-1}. Then, we'll build a distinguisher C' for the G in the following way:

Input: $y \in \{0,1\}^{2n}$ (y either comes from U_{2n} or from $\mathsf{G}(U_n)$)

Operation: Run the oracle as usual. However when getting to the i^{th} "fake invocation". In this invocation it is supposed to take an internal node v which is marked "to be chosen at random", and choose a random value x for it. In the hybrid H^{i-1} the oracle chooses $(x_0, x_1) = \mathsf{G}(x)$ and uses that to label v's children, then erasing x. In the hybrid H^i the oracle chooses x_0 and x_1 at random. Our distinguisher will simply let $(x_0, x_1) = y$ and use this as the labeling.

It is clear that if $y \sim \mathsf{G}(U_n)$ then we get H^{i-1} and if $y \sim U_{2n}$ we get H^i. Therefore the success of C' in distinguishing $\mathsf{G}(U_n)$ and U_{2n} equals the success of C in distinguishing H^{i-1} and H^i. Since C' is only polynomially slower than C we're done. □

□

See also Section 4.6 of Boneh-Shoup for this proof.