CS 600.442 – Modern Cryptography September 26; 2016

Lecture 5: Pseudorandomness - III

Instructor: Abhishek Jain *Scribe: Aarushi Goel*

1 Recall

PRG with 1-bit stretch can be constructed using hard-core predicates. PRG with poly stretch can be constructed using PRG with 1-bit stretch. In order to generate exponentially long pseudorandom strings we need an efficient indexing mechanism (functions) that can index exponentially long pseudorandom strings. We will do this by using the notion of Pseudorandom functions. But before that let us recall the definition of random functions.

Random Functions:

- Total number of random functions $F : \{0,1\}^n \to \{0,1\}^n$ mapping n-bits to n-bits: $(Size\ Of\ Codomain)^{Size\ Of\ Domain} = 2^{n2^n} = 2^{n2^n}$
- Ways to define random functions:
  1. A random function F from n-bits to n-bits is a function selected uniformly at random from all possible 2^{n2^n} functions that map n-bits to n-bits.
  2. Using a randomized algorithm to describe the function. The randomized program M that describes a random function F emulates a book keeping mechanism. It stores a table T that is initially empty. Given an input x, it checks if the entry corresponding to x already exists in T. If not, it creates a new entry in T, else it outputs the already existing entry. M's output distribution is identical to that of F.
- Truly random functions are huge random objects that cannot be efficiently stored (using either of the 2 methods). But if polynomial number of queries are made to M, then it can emulate random functions in efficient time and space.

2 Pseudorandom Functions (PRF)

PRFs look like random functions, but need only polynomial number of bits to be described and emulated. What does "look like" really mean. Let us use computational indistinguishability to define it. But the question that arises here is that, can the distinguisher, D be given the entire description of a random function or a PRF.

Issue: Since the description of a random function is of exponential size, D might not even be able to read the input efficiently in case of random function and can easy tell the difference just be looking at the size of input.

Suggested Solution What if D is not given the description of functions as input, but is instead allowed to only query the functions on inputs of its choice, and view the output. But the question here is whether the entire implementation of the PRF is hidden from the distinguisher or only a part of it.

Keeping the entire description of PRF secret from D, is similar to providing security by obscurity which in general is not a good idea according to Kerchoff's principle. Therefore we use the notion

of keyed functions. It is better to define PRFs as keyed functions. So only the key remains secret while the PRF evaluation algorithm can be made public. This is in accordance with Kerchoff's principle. The security of PRF is defined via game base definition.

2.1 Security of PRF via Game Based Definition

There are 2 players a Challenger Ch and an Adversary/Distinguisher D

Ch	**D**
$b \xleftarrow{\$} \{0,1\}$	
if $b == 0$: Ch implements F as a PRF else: Ch implements F as a random function	
	send queries $x_1, x_2, ...$ to Ch one-by-one.
correctly replies $F(x_1), F(x_2), ...$	
	outputs his guess b' i.e, whether F is random or a PRF.

If $b == b'$, D wins the game. Intuitively, we want that the adversary wins with only ($\frac{1}{2}$ + negligible) probability

Definition 1 (Pseudorandom Functions) *A family $\{F_k\}_{k\in\{0,1\}^n}$ of functions, where $F_k : \{0,1\}^n \to \{0,1\}^n$ for all k is pseudorandom if, it is:*

- ***Easy to Compute:*** *There is an efficient algorithm M such that $\forall k, x : M(k, x) = F_k(x)$.*
- ***Hard to Distinguish:*** *For every non-uniform PPT D, there exists a negligible function ν such that $\forall n \in N$:*

$$|Pr[D \text{ wins } GuessGame] - 1/2| \leq \nu(n)$$

$GuessGame(1^n)$ that incorporates D, can be formally defined as follows:

- The game chooses a key for PRF and a random bit b.
- It runs D, answering every query x as follows:
  1. If $b == 0$: answer using PRF. Output $F_k(x)$.
  2. If $b == 1$: answer using random F. Keep a table T for all previous queries, if $T[x]$ exists, output $T[x]$. Else, choose a random n-bit string y, $T[x] = y$, output y.
- Game stops, when D halts and outputs a bit b'.

All queries in the game are answered using the same function (either PRF or random F). If $b' == b$, D wins the *GuessGame*.

Remark. We are ensuring that the challenger is efficient by implementing the random function using the second method. It is important to construct challengers that are efficient, because while building reductions, the reduction acts as the challenger. If the challenger is not efficient, it would lead to the construction of an inefficient reduction. Since we only consider n.u. PPT adversaries, an inefficient reduction would not be able to give us a contradiction.

2.2 PRF with 1-bit input

Intuitively, PRFs with 1- bit input, can be constructed using PRGs. Since PRFs are keyed functions, the key of PRF can be used as the random seed for PRG. We can construct a PRF $F_k : \{0,1\} \rightarrow \{0,1\}^n$ using a length doubling PRG, G as follows:

- Let G be a length doubling PRG such that, $G(s) = y_0||y_1$, where $|y_0| = |y_1| = n$.
- For PRF, set $s = k$, and $F_k(0) = y_0$, $F_k(1) = y_1$

The security of PRFs constructed like this, can be argued by the security of PRGs.
Proof. Since this PRF is directly using the output of PRG, we can say that if there exists a distinguisher A that can efficiently distinguish between a random F and this PRF, then another distinguisher B can be constructed using A, that will be able to distinguish between PRG and a random number generator in the following way:

1. B gets a 2n- bit input y which is either a random string or the output of G for some s.
2. The adversary A, which is a distinguisher for the PRF, is allowed to query on inputs of its choice. So if A sends a query for input "0", B sends the first n-bits of y to A. On input query "1", B sends the last n-bits of y to A.
3. Based on the queries and their responses, A either outputs "1" (if the function is a PRF) or "0" (if the function is random).
4. B outputs "0" if the output of A was "0", and it outputs "1", if the output of B was "1".

Since the distinguisher A can distinguish between a random F and PRF with noticeable probability. From the above construction, it follows that B can also distinguish between a random string and the output of a PRG with noticeable probability. But, we know that no such distinguisher for PRGs exists, therefore, no such distinguisher for this PRF can exist. ■

2.3 PRF with n-bit input

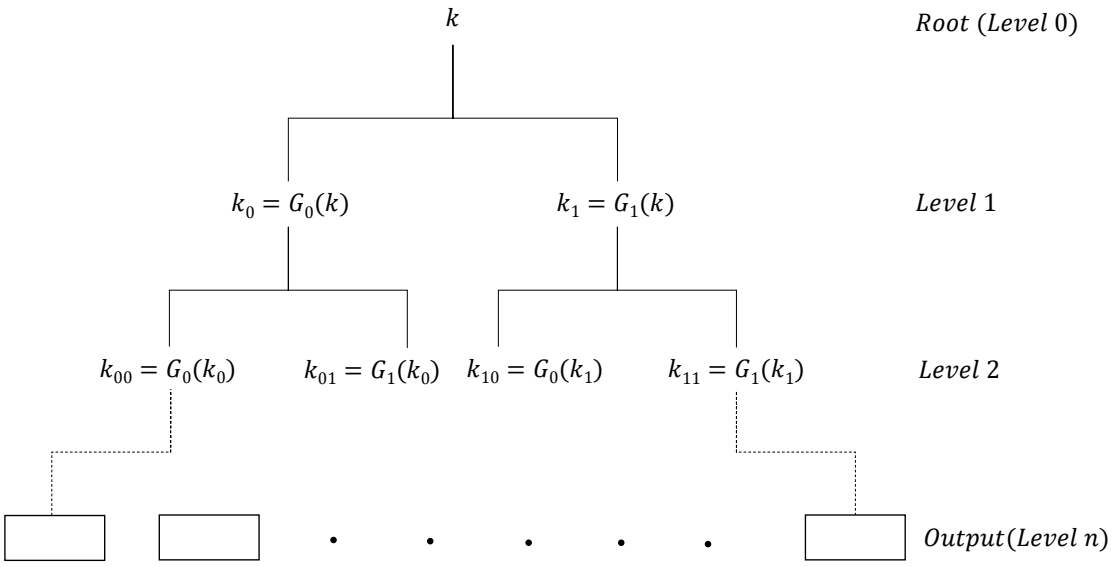
Total number of possible n-bit inputs are exponential. Since PRGs only stretch to poly(n) bits, a single PRG with poly stretch can only be used for constructing PRFs with $log(n)$ bit input. F_k: $\{0,1\}^{log(n)} \rightarrow \{0,1\}^n$ can be constructed using $G(s) = y_1||y_2||...||y_{L=poly(n)}$, where $|y_i|$ = n-bits.
However, For n-bit input, the "double and choose" policy used for constructing PRFs with 1-bit input can be used multiple times, repeatedly.

Theorem 1 (Goldreich-Goldwasser-Micali (GGM)) *If pseudorandom generators exist, then pseudorandom functions exist.*

Construction:
For a length doubling PRG G, we define G_0 and G_1 as: $G(s) = G_0(s)||G_1(s)$.
For n-bit input $x = x_1x_2...x_n$,

$$F_k(x) = G_{x_n}(G_{x_{n-1}}(.....(G_{x_1}(k))..)$$

It is convenient to think of the construction of F_k as a binary tree of size 2^n.
k denotes the root (chosen randomly). While $k_{x_1....x_l}$, denotes the the leaves of the tree at level l.



At level l, there are 2^l nodes, one for each path, denoted by $k_{x_1....x_l}$. The evaluation of function F_k on an input string $x_1x_2....x_n$ can be thought of as a walk down the branches of the tree up till the leaf nodes. Starting from the root, based on x_0, either the path traverses through the left branch or the right branch. Subsequently, the choice of branches is based on individual bits of the input string. Every input would correspond to a unique path and a unique leaf node, since atleast one bit (and correspondingly, one edge in the tree) would be different.

Efficiency: Considering the efficiency of this construction, there is no need to store this tree, the output values can be computed on the fly. To compute an output value,
RunningTime($F_k(x)$) = $n\times$ RunningTime($G(.)$) = $n \times poly(n) = poly'(n)$.

Proof Strategy:

The natural intuition is to prove using Hybrid arguments. The first idea to construct hybrids, is to replace each node in the tree, from the output of a PRG to a random string one by one. But since there are exponential number of nodes in the tree, this would result in exponential number of hybrids. While we know that the Hybrid Lemma, only works for polynomial number of hybrids. To overcome this, we use an interesting observation, which is that any PPT adversary is only allowed to make polynomial queries. Since each query corresponds to a unique path (of n nodes), total number of nodes affected by all queries = $n \times poly(n) = poly(n)$. Therefore, we only need to change polynomial number of nodes in the tree. We can achieve this by using 2 layers of nested hybrids.

Proof. Level 1 Hybrids:

H_0 : Level 0 is random, level $i > 0$ is pseudorandom. (actual PRF)
H_1 : Level 0,1 are random, level $i > 1$ is pseudorandom.
..

..
H_i : Levels $\leq i$ are random, levels $> i$ are pseudorandom.
..
..
H_n : all levels are random. (random function)

Remark. By saying that level i is random or pseudorandom, we mean that all the nodes at level i, that are affected by the adversary's queries are random or pseudorandom respectively. H_0 corresponds to a PRF, while H_n corresponds to a random function. if we assume that PRF and random function are distinguishable, then $\exists D$, such that it can differentiate between H_0 and H_n with noticable advantage $\epsilon(n)$, then by Hybrid Lemma, $\exists i \in [n]$, such that $\exists D'_{H_i,H_{i+1}}$ that distinguishes between H_i and H_{i+1} with advantage atleast $\frac{\epsilon(n)}{n}$. The only difference between H_i and H_{i+1} is that:
- in H_i, level $i+1$ is pseudorandom.
- in H_{i+1}, level $i+1$ is random.
We need to create another set of hybrids between H_i and H_{i+1}.

To create these hybrids, we only need to replace the nodes that are affected by the queries of the adversary. Since the number of queries are polynomial, changing polynomial number of nodes is sufficient from adversary's point of view. Let S be the set of nodes at level i that are affected by the adversary/distinguisher's input queries, i.e., $|S| = poly(n)$.

Level 2 Hybrids (assuming all nodes in S are in lexicographic order):
$H_{i,j\in|S|}$: Same as H_i, except that all nodes at level $i+1$, that are children of nodes $\leq j$, are random.
$H_{i,0} \equiv H_i$ (a dummy hybrid)
$H_{i,|S|} \equiv H_{i+1}$

Given $D'_{H_i,H_{i+1}}$, by Hybrid Lemma, $\exists j \in |S|$ and $\exists D''_{H_{i,j},H_{i,j+1}}$ such that, it distinguishes between $H_{i,j}$ and $H_{i,j+1}$ with advantage atleast $\frac{\epsilon(n)}{n\times|S|} \geq \frac{1}{poly(n)}$ (recall that $\epsilon(n)$ is noticeable (i.e., $\geq \frac{1}{poly(n)}$)). The only difference between $H_{i,j}$ and $H_{i,j+1}$ is that:
- in $H_{i,j}$, node j in level $i+1$ is pseudorandom.
- in $H_{i,j+1}$, node j in level $i+1$ is random.
If $D''_{H_{i,j},H_{i,j+1}}$ can distinguish between $H_{i,j}$ and $H_{i,j+1}$ with a noticeable advantage, then it is possible to construct another adversary D''' that can efficiently distinguish between PRG and a random number generator as follows:

1. D''' gets a 2n-bit input $y_0||y1$ ($|y_0| = |y_1|$), that is sampled either as a random string or as $G(s)$ for some random string s.

2. In his mind, D''' chooses random n-bit strings for nodes that are affected by the adversary's queries, up till level i and for nodes at level $i+1$ that are children of nodes $< j$. The children of node j are substituted by y_0 and y_1 respectively. The remaining nodes in the tree that are affected by the adversary's queries as computed using the chosen random values and y_0 and y_1. D''' replies to the queries of $D''_{H_{i,j},H_{i,j+1}}$ using this tree.

3. If y_0 and y_1 were pseudorandom the distribution of D''''s outputs would be similar to $H_{i,j}$

and they were random then, the output distribution would be similar to $H_{i,j+1}$.

4. If $D''_{H_{i,j},H_{i,j+1}}$ decides that the distribution of these nodes matches the distribution of $H_{i,j}$, then D''' decides that the input is pseudorandom, else if it matches the distribution of $H_{i,j+1}$, then D''' decides that the input is random.

According to the above construction, B can efficiently distinguish between a random string and the output of a PRG. Since we know that no such distinguisher for PRGs exists, therefore no $D''_{H_{i,j},H_{i,j+1}}$ can exist. From earlier definitions and assumptions, it follows that no distinguisher $D'_{H_i,H_{i+1}}$ for $H_i(H_{i,0})$ and $H_{i+1}(H_{i,|S|})$ can exist. And subsequently no D that distinguishes between a random function and a PRF can exist. Which implies that this construction of PRF is indistinguishable. ■