Lecture 5: Pseudorandomness - III

Going beyond Poly Stretch

- PRGs can only generate polynomially long pseudorandom strings
- Think: How to efficiently generate exponentially long pseudorandom strings?

Idea: Functions that index exponentially long pseudorandom strings

Random Functions

- How do we define a random function?
- Consider functions $F : \{0,1\}^n \to \{0,1\}^n$
- Think: How many such functions are there?
- Write F as a table:
  - first column has input strings from 0^n to 1^n;
  - against each input, second column has the function value
  - i.e., each row is of the form $(x, F(x))$
- The size of the table for $F = 2^n \times n = n2^n$
- Total number of functions mapping n bits to n bits $= 2^{n2^n}$

Random Functions

There are two ways to define a random function:

- **First method**: A random function F from n bits to n bits is a function selected *uniformly at random* from all 2^{n2^n} functions that map n bits to n bits
- **Second method:** Use a randomized algorithm to describe the function. Sometimes more convenient to use in proofs
  - randomized program M to implement a random function F
  - M keeps a table T that is initially empty.
  - on input x, M has not seen x before, choose a random string y and add the entry (x, y) to the table T
  - otherwise, if x is already in the table, M picks the entry corresponding to x from T, and outputs that
- M's output distribution identical to that of F.

Random Functions

- Truly random functions are huge random objects
- No matter which method we use, we cannot store the entire function efficiently
- With the second method, we can support **polynomial** calls to the function efficiently because M will only need polynomial space and time to store and query T
- Can we use some crypto magic to make a function F' so that:
  - it "looks like" a random function
  - but actually needs much fewer bits to describe/store/query?

Pseudorandom Functions (PRF)

- PRF looks like a random function, and needs polynomial bits to be described
- Think: What does "looks like" mean?
- First Idea: Use computational indistinguishability
  - Random Functions and PRFs are hard to tell apart efficiently
- Think: Should the distinguisher get the *description* of either a random function or a PRF?
- **Main Issue:** A random function is of exponential size
  - D can't even read the input efficiently
  - D can tell by looking at the size
- **Idea**: D can only *query* the function on inputs of its choice, and see the output.

Pseudorandom Functions

- Keep the description of PRF **secret** from D?
  - Security by obscurity not a good idea (Kerckoff's priniciple)
- Solution: PRF will be a keyed function. Only the key will be secret, and the PRF evaluation algorithm will be public
- **Security via a Game based definition**
  - Players: a **challenger** Ch and D. Ch is randomized and efficient
  - Game starts by Ch choosing a random bit b. If $b = 0$, Ch implements a random function, otherwise it implements a PRF
  - D send queries $x_1, x_2, \ldots$ to Ch, one-by-one
  - Ch answers by correctly replying $F(x_1), F(x_2), \ldots$
  - Finally, D outputs his guess b' (of F being random or PRF)
  - D *wins* if $b' = b$
- PRF Security: No D can win with probability better than $1/2$.

Pseudorandom Functions: Definition

Definition (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:

- **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 $\forall n \in \mathbb{N}$:

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

  where GuessGame is defined below

Pseudorandom Functions: Game Based Definition

GuessGame(1^n) incorporates D and proceeds as follows:

- The games choose a PRF key k and a random bit b.
- It runs D answering every query x as follows:
- If $b = 0$: (answer using PRF)
  - output $F_k(x)$
- If $b = 1$: (answer using a random F)
  - (keep a table T for previous answers)
  - if x is in T: return $T[x]$.
  - else: choose $y \leftarrow \{0,1\}^n$, $T[x] = y$, return y.
- Game stops when D halts. D outputs a bit b'

D **wins** GuessGame **if** $b' = b$**.**

Remark: note that for any b only one of the two functions is ever used.

Pseudorandom Functions (contd.)

- Think: How can we construct a PRF?
- Use PRG?
- **Simpler problem:** build PRF for just 1-bit inputs using PRG

From PRG to PRF with 1-bit input

- Let G be a length doubling PRG
- Want: $\{F_k\}$ such that $F_k : \{0,1\} \to \{0,1\}^n$
- G is length doubling, so let

$$G(s) = y_0 \| y_1$$

  where $|y_0| = |y_1| = n$

- PRF: Set $k = s$ and,

$$F_k(0) = y_0, \; F_k(1) = y_1$$

- Think: What about n-bit inputs?
  - Idea for 1-bit case: "double and choose"
  - For general case: Apply the "double and choose" idea repeatedly!

PRF from PRG

Theorem (Goldreich-Goldwasser-Micali (GGM))

If pseudorandom generators exist then pseudorandom functions exist

- **Notation:** define G_0 and G_1 as

$$G(s) = G_0(s) \| G_1(s)$$

  i.e., G_0 chooses left half of G and G_1 chooses right half

- Construction for n-bit inputs $x = x_1 x_2 \ldots x_n$

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

PRF from PRG (contd.)

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

- We can represent F_k as a binary tree of size 2^n
- The root corresponds to k
- Left and right child on level 1 and 2 are:

$$k_0 = G_0(k) \text{ and } k_1 = G_1(k)$$

- Second level children:

$$k_{00} = G_0(k_0),\ k_{01} = G_1(k_0),\ k_{10} = G_0(k_1),\ k_{11} = G_1(k_1)$$

- At level ℓ, 2^ℓ nodes, one for each path, denoted by $k_{x_1 \dots x_\ell}$

Proof Strategy

- Let's use Hybrid Arguments!
- Problem: If we replace each node in the tree one-by-one with random, then exponentially many hybrids. Hybrid lemma doesn't apply!
- **Observation:** Efficient adversary can only make polynomial queries
- Thus, only need to change polynomial number of nodes in the tree

Proof Strategy (contd.)

Two layers of hybrids:

- First, define hybrids over the n levels in the tree. For every i, H_i is such that the nodes up to level i are random, but the nodes below are pseudorandom.
- If H_1 and H_n are distinguishable with noticeable advantage, then use hybrid lemma to find level i s.t. H_i and H_{i+1} are also distinguishable with noticeable advantage
- Now, hybrid over the nodes in level $i+1$ that are "affected" by adversary's queries, replacing each node one by one with random
- Use hybrid lemma again to identify one node that is changed from pseudorandom to random and break PRG's security to get a contradiction

Proof Details

- Must make sure that all hybrids are implementable in polynomial time
- Will use two key points to ensure this:
  1. Adversary only makes polynomial number of queries
  2. A random function can be efficiently implemented (using second method) if the number of queries are polynomial
- Think: Formal proof?

Concluding Remarks

- **PRFs from concrete assumptions:** [Naor-Reingold97], [Banerjee-Peikert-Rosen12]
- **Constrained PRFs:** PRFs with "punctured" keys that are disabled on certain inputs [Boneh-Waters13, Kiayias-Papadopoulos-Triandopoulos-Zacharias13, Boyle-Goldwasser-Ivan14, Sahai-Waters14]
- **Related-key Security:** Evaluation of $F_s(x)$ does not help in predicting $F_{s'}(x)$ [Bellare-Cash10]
- **Key-homomorphic PRFs:** Given $f_s(x)$ and $f_{s'}(x)$, compute $f_{g(s,s')}(x)$ [Boneh-Lewi-Montgomery-Raghunathan13]