4 Section Week 4 - Pseudorandomness and Computational Security

This lecture assumes knowledge of what a probability distribution is.

Motivation: "randomness is not free" (there's even work done on how to extract randomness from imperfect random sources). Thus we want to study computation through the lens of using less randomness (analogous to using less time or space).

Draw diagrams for extending a small amount of randomness to be used as input to a program that needs lots of randomness

4.1 Randomness is Measured Relative to the Observer

Another history lesson: How to generate cryptographically strong sequences of pseudo random bits (Blum Micali 1984)

"The randomness of an event is relative to a specific model of computation with a specified amount of computing resources."

They were not the first to think about pseudorandomness by any means; around this time lots of people were thinking about how to generate pseudo-random strings of bits.

Yao's foundational paper which is credited with the notion of computational indistinguishability. First, the definition of negligible:

A function $\epsilon(n)$ is negligible if for all polynomials p, $\epsilon(n) < \frac{1}{p(n)}$ for sufficiently large n. In other words, for all polynomials p, $\epsilon(n) = o(\frac{1}{p(n)})$. This definition is motivated by the Chernoff bound.

Now computational indistinguishability:

Given two distributions D and E with security parameter n, we say $D \approx E$ (computationally indistinguishable) if for every poly time algorithm A,

$$\left| \Pr_{x \sim D}[A(x) = 1] - \Pr_{x \sim E}[A(x) = 1] \right| < \epsilon(n)$$

For this lecture, D and E are distributions over $\{0,1\}^n$ which is the "security parameter."

Again tying back to the Chernoff bound, this means that running A on fresh x drawn from a distribution any polynomial times will not help, since you need exponential trials to distinguish D and E.

Theorem (Hybrid argument)**.** *Let $\ell = p(n)$ for some polynomial p, and consider distributions $D_0, D_1, \ldots D_\ell$ where $D_i \approx D_{i+1}$ for all i. Then $D_0 \approx D_\ell$.*

Proof. For any i, by the definition of computationally indistinguishable, for all poly time algorithms A,

$$\left| \Pr_{x \sim D_i}[A(x) = 1] - \Pr_{x \sim D_{i+1}}[A(x) = 1] \right| < \epsilon_i(n)$$

Pick ϵ to be the max of ϵ_i over all i. Now by the triangular inequality, for all i,

$$\left| \Pr_{x \sim D_i}[A(x) = 1] - \Pr_{x \sim D_{i+2}}[A(x) = 1] \right| < 2\epsilon(n)$$

and in general

$$\left| \Pr_{x \sim D_0}[A(x) = 1] - \Pr_{x \sim D_\ell}[A(x) = 1] \right| < \ell\epsilon(n)$$

Since $\ell = p(n)$ and $\epsilon(n) < \frac{1}{q(n)}$ for all polynomials q, it's also the case that $\epsilon(n) < \frac{1}{p(n)q(n)}$ for all polynomials q, so $\ell\epsilon(n)$ is still negligible. Thus, $D_0 \approx D_\ell$.

□

Hybrids (not just for distributions) are a very important idea in theoretical computer science and are still used in state of the art research in many different ways.

4.2 Psudorandom Generators

Now we are ready to tackle the definition of a pseudorandom generator. In english a PRG takes a short uniformly random string and outputs a longer string that looks uniformly random.

A PRG is a function $PRG : \{0,1\}^s \to \{0,1\}^{\ell(s)}$ such that $P(U_s) \approx U_{\ell(s)}$, where U_n is the uniform distribution on strings of length n. s is called the seed length.

Refer back to picture for motivation of a PRG; using a pseudorandom string instead of a random string only changes the behavior of the algorithm by ϵ.

We don't actually know if pseudorandom generators exist. In particular, the PRG conjecture is there exists a PRG mapping n bits to $n+1$ bits.

We get a nice result for how to extend the output of a pseudorandom generator:

Suppose we have a $PRG : \{0,1\}^s \to \{0,1\}^{s+1}$. That is, it only extends the seed by a single bit. I claim that I can construct a $PRG' : \{0,1\}^s \to \{0,1\}^{\ell(s)}$ for any polynomial ℓ by using "bootstrapping."

The idea will be to apply PRG to itself $\ell(s)$ times. In particular, let the starting seed be s_0. Let $PRG(s_i) = x_i \circ s_{i+1}$, where $x_i \in \{0,1\}$, $s_i \in \{0,1\}^s$, and $i \in [\ell(n)]$. I claim that if we set $x_0, x_1, \ldots x_{\ell(n)}$ to be the output of PRG', then PRG' is a secure pseudorandom generator.

Draw bootstrapping diagram for how this works

Theorem. *Assuming PRG is a secure PRG, then PRG' is also a secure PRG.*

Proof. Let $x \in \{0,1\}^{\ell(s)}$ be the output of PRG'. We consider $\ell(s)$ hybrids

H_i, where in H_i we consider the string $u_0, u_1, \ldots u_i, x_{i+1}, \ldots x_{\ell(s)}$, where the u_j are uniformly random bits. We claim that $H_i \approx H_{i+1}$, and thus $U_{\ell(s)} \approx PRG'(U_s)$.

Assume for the sake of contradiction that there exists some algorithm EVE' for which H_i and H_{i+1} are not computationally indistinguishable, and thus EVE' breaks the security of PRG'. We construct an algorithm EVE which breaks the security of PRG using EVE'.

Draw reduction diagram

Suppose EVE is given some string $y \in \{0,1\}^{s+1}$ and is trying to determine if y came from the distribution $PRG(U_s)$ or U_{s+1}. EVE writes $y = b_{i+1} \circ s_{i+2}$, chooses random $u_0, \ldots u_i$, computes $x_{i+2}, \ldots x_{\ell(s)}$ from s_{i+2}, and passes $u_0, \ldots u_i, b_{i+1}, x_{i+2}, \ldots x_{\ell(s)}$ to EVE'.

If y came fom $PRG(U_s)$ then we are in hybrid H_i, and if y came from U_{s+1} then we are in hybrid H_{i+1}. Thus, if y came from $PRG(U_s)$, EVE has the same probability of outputting 1 as EVE' in H_i, and same for H_{i+1}. This means EVE can distinguish between $PRG(U_s)$ and U_{s+1} if EVE' can distinguish between H_i and H_{i+1} for any i. Thus, $H_0 \approx H_{\ell(s)}$ and PRG' is secure.

□

This theory is good and all and tells us that even extending by one bit of randomness is enough, but how do we actually construct PRGs?

4.3 Hardness Assumptions and Constructing PRGs

We observe that saying "no poly time algorithm can distinguish between random and pseudorandom" is roughly the same as saying "the problem of distinguishing between random and pseudorandom is hard". Thus, we construct PRGs with problems that are NP-hard, and thus we assume no poly time algorithm can distinguish.

Example: Blum Blum Shub. Pick some $P, Q \equiv 3 \mod 4$ and let $N = PQ$. Given X, outputs $X^2 \mod N$ and the least significant bit of X.

Conjecture. *Factoring randomly chosen integers of the form $N = PQ$ is hard for non-quantum algorithms.*

Example: subset sum problem, which is NP-complete. Given some seed of length k and output length n, we choose k constants $c_1, c_2, \ldots c_k$ of length n at random, but they are fixed and not considered secret. On input s we output

$$\sum_{i \in [k]} s_i c_i \mod 2^n$$

where multiplication and addition are arithmetic in this setting.

Theorem. *Given k non-negative intgers $c_1, c_2, \ldots c_k$ and a target sum W, the problem of deciding if there exists a subset $I \subseteq [k]$ s.t.*

$$\sum_{i \in I} c_i = W$$

is NP-complete.

Intuitively, the hardness comes from the requirement that the seed s is a boolean vector in an ring of size 2^n, and finding a small vector that satisfies a linear set of constraints is hard.