Lecture 4 - Computational Indistinguishability, Pseudorandom Generators

Boaz Barak

September 27, 2007

Computational Indistinguishability Recall that we defined that statistical distance of two distributions X and Y over $\{0,1\}^n$ is at most ϵ if

$$\left|\Pr[A(X)=1]-\Pr[A(Y)=1]\right| \leq \epsilon \quad \textbf{(*)}$$

for every $A:\{0,1\}^n \to \{0,1\}$.

We will say that X and Y are *computationally indistinguishable* if **(*)** holds for every polynomial-time A and polynomially bounded ϵ. More formally, we need to use asymptotic notation and so make the following definition:

Definition 1. Let $\{X_n\},\{Y_n\}$ be sequences of distributions with X_n, Y_n ranging over $\{0,1\}^{\ell(n)}$ for some $\ell(n)=n^{O(1)}$. $\{X_n\}$ and $\{Y_n\}$ are *computationally indistinguishable* (notation: $X_n \approx Y_n$) if for every polynomial-time A and polynomially-bounded ϵ, and sufficiently large n

$$\left|\Pr[A(X_n)=1]-\Pr[A(Y_n)=1]\right| \leq \epsilon(n)$$

(If we want to nitpick, we'd give 1^n as additional input to A to ensure it can run in poly(n)-time even if $\ell(n)$ is much smaller than n.)

We will often drop the subscript n when it's clear from the context. So, we'll say that two distributions X and Y are computationally indistinguishable when we really mean that they are part of two computationally indistinguishable sequences.

Some properties of computational indistinguishability: (Exercise)

- Obviously, if $X_n \approx Y_n$ then $Y_n \approx X_n$ (i.e., $\approx$ is symmetric).
- It's weaker than statistical indistinguishability: if $\Delta(X_n,Y_n) \leq \epsilon(n)$ for $\epsilon(n)=n^{-\omega(1)}$ then $X_n \approx Y_n$. In particular, if $X_n \equiv Y_n$ then $X_n \approx Y_n$ (i.e., $\approx$ is reflexive).
- It satisfies transitivity / triangle inequality: if $X_n \approx Y_n$ and $Y_n \approx Z_n$ then $X_n \approx Z_n$ (i.e., $\approx$ is transitive and hence is an equivalence relation).
- If $X_n \approx Y_n$ and f is a polynomial time computable function then $f(X_n) \approx f(Y_n)$.
- If $X_n \approx Y_n$ then for every $m < n$, the *truncation* of X_n to the first m bits is indistinguishable from the truncation of Y_n to the first m bits.

Proof of transitivity condition The way is to simply express $\Pr[A(X)=1]-\Pr[A(Z)=1]$ as $\Pr[A(X)=1]-\Pr[A(Y)=1]+\Pr[A(Y)=1]-\Pr[A(Z)=1]$, and then use the standard triangle inequality $|a+b| \leq |a|+|b|$ to conclude that if both $|\Pr[A(X)=1]-\Pr[A(Y)=1]| \leq \epsilon$ and $|\Pr[A(Y)=1]-\Pr[A(Z)=1]| \leq \epsilon$, then $|\Pr[A(X)=1]-\Pr[A(Z)=1]| \leq 2\epsilon$.

Polynomial transitivity / hybrid argument We can generalize this proof to say that if we have a polynomial number m of distributions $X^1, X^2, \ldots, X^m$ such that $X_i \approx X^{i+1}$ for every i, then $X^1 \approx X^m$: Simply express $\Pr[A(X^1) = 1] - \Pr[A(X^m) = 1]$ as

$$\Pr[A(X^1) = 1] - \Pr[A(X^2) = 1] + \Pr[A(X^2) = 1] - \Pr[A(X^3) = 1] + \cdots + \Pr[A(X^{m-1}) = 1] - \Pr[A(X^m) = 1]$$

and conclude using the triangle inequality ($|\sum_{i=1}^m a_i| \leq \sum_{i=1}^m |a_i|$) that if $|\Pr[A(X^i) = 1] - \Pr[A(X^{i+1}) = 1]| \leq \epsilon$ for every i, then $|\Pr[A(X^1) = 1] - \Pr[A(X^m) = 1]| \leq m\epsilon$.

Security def in these notation For example, recall that last time we made the following definition:

Definition Let (E, D) be an encryption scheme that uses n-bit keys to *encrypt* $\ell(n)$-length messages. (E, D) is *computationally secure* if for every polynomial-time algorithm $A : \{0,1\}^* \to \{0,1\}$, polynomially bounded $\epsilon : \{0,1\}^* \to [0,1]$, n, and $x_0, x_1 \in \{0,1\}^{\ell(n)}$,

$$\left|\Pr[A(E_{U_n}(x_0)) = 1] - \Pr[A(E_{U_n}(x_1) = 1]\right| < \epsilon(n)$$

An equivalent way to say this is that (E, D) is computationally secure if $\mathsf{E}_{U_n}(x_0) \approx \mathsf{E}_{U_n}(x_1)$ for every two messages x_0, x_1.[1]

Pseudorandomness We say that a distribution $\{X_n\}$ is *pseudorandom* if it's computationally indistinguishable from the uniform distribution.

Definition 2. A polynomial-time-computable deterministic function G mapping n bit strings into $\ell(n)$ bit strings for $\ell(n) \geq n$ is called a *pseudorandom generator* if $G(U_n) \approx U_{\ell(n)}$. The function $\ell(n)$ is called the *stretch* of the pseudorandom generator.

It's trivial to construct a pseudorandom generator with $\ell(n) = n$. Also, because of the truncation property, a pseudorandom generator with stretch $\ell(n)$ trivially yields a pseudoarndom generator with stretch $\ell'(n)$ for every $\ell'(n) < \ell(n)$. We are now ready to state our axiom:

The PRG Axiom: There exists a pseudorandom generator with stretch $\ell(n) = n + 1$.

Our main Theorem for today will be

Main Theorem: If the PRG Axiom is true then for every constant c, there exists a computationally secure encryption scheme with message length $\ell(n) = n^c$.

It will follow from the following two theorems:

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 $\ell(n) = n^c$.*

Theorem 2. *If there exists a pseudorandom generator with stretch $\ell(n)$ then there exists a computationally secure encryption scheme with message length $\ell(n)$.*

Proof of Theorem 2 There is a pretty natural construction of a private key encryption with key length $<$ message length using a pseudorandom generator.

Let G be the pseudorandom generator mapping n bit strings to $\ell(n)$ bit strings.

[1] More formally we'd say that for every two sequences of messages $\{x_0^n\}$ and $\{x_1^n\}$, where $x_0^n, x_1^n \in \{0,1\}^{\ell(n)}$, the two sequences $\{\mathsf{E}_{U_n}(x_0^n)\}$ and $\{\mathsf{E}_{U_n}(x_1^n)\}$ are computationally indistinguishable.

The encryption scheme will be the following: $\mathsf{E}_k(x) = x \oplus \mathsf{G}(k)$, $\mathsf{D}_k(y) = y \oplus \mathsf{G}(k)$. That is, we use a *pseudorandom* pad instead of a random pad in the One-Time-Pad scheme. Intuitively this should be secure since using a pseudorandom string instead of random should be good enough for all practical purposes. However, relying on intuition is very dangerous in cryptography, and so we need to verify this with a proof. Fortunately, this time the intuition holds:

Claim 2.1. *For every message x, the distribution $\mathsf{E}_{U_n}(x)$ is pseudorandom.*

The claim implies that for every pair of messages x^0, x^1, we have that $\mathsf{E}_{U_n}(x^0) \approx U_{\ell(n)}$, and $U_{\ell(n)} \approx \mathsf{E}_{U_n}(x^1)$. Hence $\mathsf{E}_{U_n}(x^0) \approx \mathsf{E}_{U_n}(x^1)$.

Proof of claim. Assume, for the sake of contradiction, that there exists a polynomial-time A such that

$$\left|\Pr[A(\mathsf{G}(U_n) \oplus x) = 1] - \Pr[A(U_m) = 1]\right| \geq \epsilon \tag{1}$$

(where $m = \ell(n)$).

We'll construct an algorithm $B : \{0,1\}^m \to \{0,1\}$ which will contradict the security of G. We define B as follows: $B(y) = A(y \oplus x)$. Note that this means that $A(z) = B(z \oplus x)$. The running time of B is the same as the running time of A, but (1) means that

$$\left|\Pr[B(\mathsf{G}(U_n)) = 1] - \Pr[B(U_m \oplus x) = 1]\right| \geq \epsilon \tag{2}$$

but since $U_m \oplus x \equiv U_m$, (2) implies that B contradicts the fact that G is a pseudorandom generator. □

Proof of Theorem 1 Assume that we have a pseudornadom generator pmG 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 pmG.

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

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

$j \leftarrow 0$
$x^{(0)} \leftarrow x$
`while` $j < \ell(n)$`:`
    $j \leftarrow j + 1$
    $x^{(j)} \leftarrow \mathsf{G}'_n(x^{(j-1)}_{[1...n]})$
    `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 $x \leftarrow_{\mathrm{R}} \{0,1\}^n$:

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

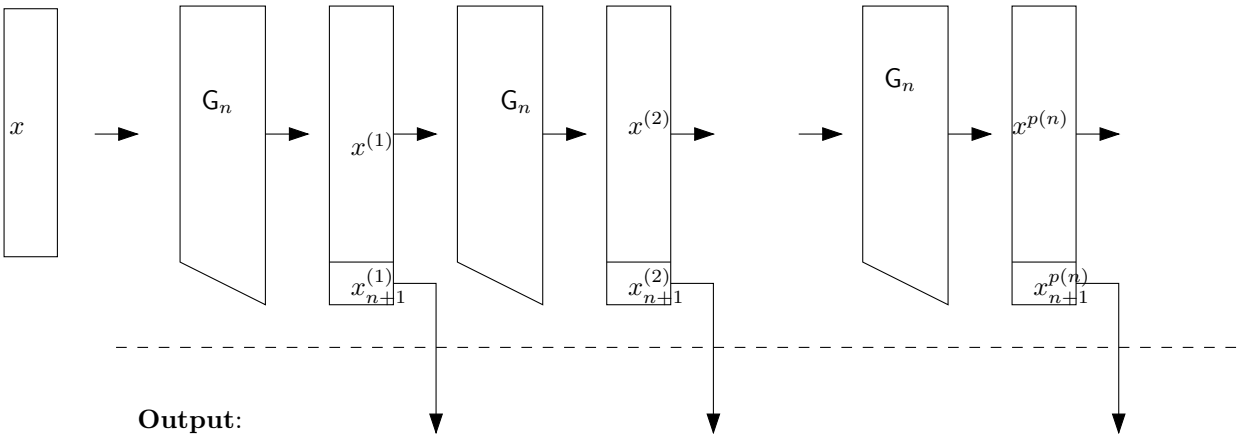

Figure 1: Extending output of pseudorandom genertator

$j \leftarrow 0$
$x^{(0)} \leftarrow x$
`while` $j < \ell(n)$`:`
    $y \leftarrow \mathsf{pmG}(x^{(j-1)})$
    $x^{(j+1)} \leftarrow y_{[1..n]}$
    `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 2.2. *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:

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

But since $\mathsf{pmG}(U_n) \approx U_{n+1}$, it follows that $f(\mathsf{pmG}(U_n)) \approx f(U_{n+1})$ 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 6.4.2 of the Katz-Lindell book.

Candidates for pseudorandom generators: One justification to The PRG Axiom is that candidate functions that we conjecture to be pseudorandom generators (even if we can't prove it). Below are two such candidate functions. We'll see more such candidates later on in the course.

RC4 RC4 was invented by Ron Rivest for RSA. Its design is a trade secret and so the actual algorithm is not supposed to be known (security by obscurity). Nevertheless the code was obtained by reverse engineering and leaked to the cyberpunks mailing list. Even though RC4 is widely used including in the WEP and WPA protocols for wireless networks (IEEE 802.11) and the SSL protocol, several weaknesses were found in it and so it can *not* be considered a secure pseudorandom generator. (And so in fact it's not a good candidate, however I present it here since it is widely used and illustrates principles that are utilized in more secure candidates for pseudorandom generators.)

A byte is a number from 0 to 255 (or equivalently, a string in $\{0,1\}^8$). The input to the pseudorandom generator is a permutation $S : \mathbb{Z}_{256} \to \mathbb{Z}_{256}$ (i.e., a 256 long array `S[]` such that `S[i]`$\neq$`S[j]` for every `i`$\neq$`j`). The output is `m` bytes (where we can control the value of `m` to be as large as we want). The following is the pseudocode for RC4:

```
i := 0
j := 0
num_outputted = 0;
while num_outputted < m:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    num_outputted := num_outputted + 1
    output S[(S[i] + S[j]) mod 256]
```

We see that RC4 expands $\log(256!)$ bits which is roughly $8 \cdot 256 = 2048$ bits into an arbitrary large `m` number of bits. However, in most current applications people desire an input much smaller than 2048 and so there's a separate pseudorandom generator (called the *key scheduling algorithm* or KSA) that takes an input of size ℓ bits, for $40 \leq \ell \leq 128$, and outputs an initial permutation S. The page `http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html` (written by my friend Itsik Mantin) is a good source of information on RC4 and the attacks on it.

Blum-Blum-Shub The Blum-Blum-Shub generator is even simpler than RC4 but it is much less efficient. However it has the advantage that we can relate its security to a well known problem. Assuming factoring a random n bit integer[2] cannot be done in polynomial-time with polynomially-bounded success probability, this pseudorandom generator will be secure. The input is a number N (of length n bits) and X where $1 \leq X < N$.[3] The output will be `m` bits where again we can choose `m` to be as large as we want. The pseudocode is as follows:

```
num_outputted = 0;
while num_outputted < m:
    X := X*X mod N
    num_outputted := num_outputted + 1
    output least-significant-bit(X)
```

[2]Random here means that we choose random primes p and q of length $n/2$ bits, where for technical reasons we require that their remainder modulu 4 is 3, and let $n = p \cdot q$.

[3]Actually X should satisfy $gcd(X, N) = 1$ but this will happen with overwhelmingly high probability for a random X.

We'll prove that a variant of this generator is as secure as factoring later in the course.