COM S 687 Introduction to Cryptography Sept 26, 2006

Lecture 10: Hard-core predicates

Instructor: Rafael Pass *Scribe: Ari Rabkin*

1 The Next-bit Test

Last time, we rushed through the proof that the next-bit test is a "complete" statistical test; that passing the next-bit test implies that a pseudorandom function will pass *any* statistical test. Now, we'll go through the proof again very carefully.

Definition 1 (Next-bit test) *An ensemble of probability distributions* $\{X_n\}$ *over* $\{0,1\}^{m(n)}$ *is said to pass the next bit test if* $\exists$ *a negligible function* $\epsilon(n)$ *so that* $\forall$ *nonuniform PPT* A *and* $\forall n \in \mathbb{N}$ *it holds that* $Pr[t \leftarrow X_n \; A(t_{0 \to i}) = t_{i+1}] \leq \frac{1}{2} + \epsilon(n)$

Note that the uniform distribution passes the next-bit test, since no algorithm can predict the next bit with probability better than $\frac{1}{2}$.

Proposition 1 (completeness of next-bit test) *If an ensemble of probability distributions* $\{X_n\}$ *over* $\{0,1\}^{m(n)}$ *passes the next-bit test, then* $\{X_n\} \approx \{t \leftarrow \{0,1\}^{m(n)}\}_{n \in \mathbb{N}}$

Proof. Assume for contradiction that that there exists a (nonuniform PPT) distinguisher D, and a polynomial p(n) so that for infinitely many $n \in \mathbb{N}$, D distinguishes X_n from U_m [the uniform random distribution] better than $\frac{1}{p(n)}$. That is,

$$|Pr\left[t \leftarrow X_n \; : \; D(t) = 1\right] - Pr\left[t \leftarrow \{0,1\}^n \; : \; D(t) = 1\right]| > \frac{1}{p(n)}$$

Without loss of generality, we can drop the absolute value: we can replace D with $D' = 1 - D$, reversing the sense of the test.

Now for the core of the proof: We're going to define *hybrid distributions* and then apply the polynomial jump lemma from last lecture. This will show us that there must be two adjacent hybrid distributions that we can distinguish with non-negligible probability. (More precisely, we can distinguish which of the two distributions a given string came from).

Define the hybrids as follows:

$$H^i = \{L \leftarrow X_n \; : \; R \leftarrow \{0,1\}^m \; : \; L_{0 \to i} || R_{i+1 \to m}\}$$

Note that $H^0 = U_m$ and $H^m = X_n$

It follows from the polynomial-jump lemma that for infinitely many $n \in \mathbb{N}$, an attacker able to tell H^0 and H^m apart, can also distinguish H^i and H^{i+1} for some I, with probability $\frac{\epsilon(n)}{m(n)}$.

$$\begin{aligned} H^i &= X_1 X_2 ... X_k U_{k+1} U_{k+2} \\ H^{i+1} &= X_1 X_2 ... X_k X_{k+1} U_{k+2} ... \end{aligned}$$

Therefore, it's possible to distinguish which of the two hybrid distributions a string came from looking only at bit $k+1$, the only bit that's different.

Define an $\tilde{H}^i = X_1 X_2 ... X_k \bar{X}_{k+1} U_{k+2}$. This is just H^{i+1} with bit $i+1$ flipped.

Now, half the time X_k corresponds to U_k, and so D can do no work; D can only infer anything when X_k and U_k differ. Well, X_k and $\bar{X}_k$ always differ, so D should be able to distinguish H^i and $\tilde{H}^i$ at least as efficiently as it distinguishes H^i and H^{i+1}. It follows that D also distinguishes strings from H^i and $\tilde{H}^i$ with probability at least $\frac{1}{p(n)m(n)}$.

We now show that for infinitely many n, there exists an algorithm A(y) and index i so that A predicts the i^{th} bit of X_n with probability $\frac{1}{2} + \frac{1}{2p(n)m(n)}$. The algorithm is as follows:

A(y): Let $r \leftarrow 0, 1^{m-i-2}$ and $b \leftarrow 0, 1$. If $D(y||b||r) = 1$ then output b, else output $1 - b$.

Now, we said that D can predict the i^{th} bit for somei, but is it the right one? Two answers: First, since we're in a nonuniform setting, we can have A pick the appropriate D. Second, we can guess randomly, paying an accuracy penalty of $\frac{1}{m}$ – in which case, A still succeeds with probability that's inverse-polynomial in m.

How well does this work?

$$Pr[t \leftarrow X_n \; A(t_{1 \to i}) = t_{i+1}] = \frac{1}{2} Pr[t \leftarrow H_n^i \; : \; D(t) = 1] + \frac{1}{2} Pr[t \leftarrow \tilde{H}_n^i \; : \; D(t) \neq 1]$$

We can flip the sense of the test and rewrite the second term as: $\frac{1}{2}(1 - Pr[t \leftarrow \tilde{H}_n^i \; : \; D(t) = 1])$

At this point, we can do some algebra to get:

$$Pr[t \leftarrow X_n \; A(t_{1 \to i}) = t_{i+1}] = \frac{1}{2} + \frac{1}{2} Pr[t \leftarrow H_n^i \; : \; D(t) = 1] - Pr[t \leftarrow \tilde{H}_n^i \; : \; D(t) = 1])$$

As we showed, D distinguishes with probability at least $\frac{1}{p(n)m(n)}$, so, plugging in, we get

$$Pr[t \leftarrow X_n \; A(t_{1 \to i}) = t_{i+1}] = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{p(n)m(n)}$$

Thus, we are able to find bits of X_n with probability $\frac{1}{2} + \frac{1}{p(n)}$. So the presence of a distinguisher implies that X fails the next-bit test. By contradiction, any PRNG that passes the next-bit test passes any other poly-time test as well.

2 Constructing a PRNG

2.1 First attempt

The first attempt to provide a general construction of pseudorandom generators was by Shamir (he of the "S" in RSA). The construction is as follows:

Definition 2 (PRG-Shamir) *Start with a one-way permutation f, then define G_{Shamir} as follows: $G(s) = f^n(s)||f^{n-1}(s)||...||f(s)||s$*

The basic idea here is very similar to that of Lamport one-time passwords if you're familiar with those: iterate a one-way function, then output in reverse order. The insight behind the security of the scheme is that given some prefix of the random sequence, computing the next block is equivalent to inverting the one-way function f.

This scheme leads to unpredictable numbers, but not necessarily unpredictable *bits*; since some of the output bits of a one-way function may be predictable.

The reason we need f to be a permutation and not a general one-way function is two-fold. First, we need the domain and range to be the same number of bits. Second, and more importantly, we require that the output of $f^k(x)$ be uniformly distributed if x is. If f is a permutation, this is true, but it need not hold for a general one-way function.

There is an extension of this technique, though, that does what we want.

2.2 Hard-core bits

Definition 3 (Hard-core predicate) *A predicate $b : \{0,1\}^* \rightarrow \{0,1\}$ is a hard-core predicate for $f(x)$ if b is efficiently computable given x, and $\forall$ nonuniform PPT A, $\exists$ negligible ϵ so that $\forall k \in \mathbb{N} Pr[x \rightarrow \{0,1\}^k : A(1^k, f(x)) = b(x)] \leq \frac{1}{2} + \epsilon(n)$*

In other words, a hard-core predicate for a function can't be computed given the result of the function, but can be computed given the function's input.

The least significant bit of the RSA one-way function is known to be hardcore (under the RSA assumption). Formally, given n,e, and $1^e mod n$, there's no efficient algorithm to predict LSB(x) from $f_{RSA}(x)$.

Some other examples:

- The function $half_n(x)$ iff $0 \leq x \leq \frac{n}{2}$ is also hardcore for RSA, under the RSA assumption.
- The least significant bit is hardcore for Rabin under the factoring assumption.
- For *exp*, $half_{p-1}$ is a hardcore predicate.

2.3 Constructing the Generator

Now, how do we construct a PRNG from a one-way permutation? [Blum and Micali]

Proposition 2 *Let f be a one-way permutation, and b a hard-core predicate for f. Then $G(s) = f(s)||b(s)$ is a PRNG.*

Proof. Assume for contradiction that $\exists$ a nonuniform PPT A and polynomial $p(n)$ so that for infinitely many n, $\exists i$ so that A predicts the i^{th} bit with probability $\frac{1}{p(n)}$. The first n bits of $G(s)$ are a permutation of a uniform distribution, and are therefore also distributed uniformly at random. So A must predict bit $n+1$ with probability $\frac{1}{p(n)}$. Formally,

$$Pr[A(f(s)) = b(s)] > \frac{1}{2} + \frac{1}{p(n)}$$

This contradicts the assumption that b was hard-core. Therefore, G is a PRNG.

This construction only extends a k-bit seed to $k+1$ bits. Is that sufficient? It is. We can generate an infinite sequence very simply by saying that bit k of our sequence is just $G(f^k(s))$. That is, iterate the one-way permutation k times, and then apply the hard-core predicate.

Aside: This means that repeatedly squaring and outputting the least significant bit in a finite field is a secure PRNG, based on the Rabin one-way function. In fact, this function is actually used in practice. If you think back to the fifth question on homework 1, this shows that replacing ax with x^2 would have made the scheme secure. This "square and output least-significant-bit" scheme is known as the Blum-Blum-Schub PRNG.