CS 830: Randomness and Complexity 02/08/05

Lecture 7: HardCore Bit

instructor: Jin-Yi Cai Scribe: Rakesh Kumar and Yunpeng Li

1 Introduction

Last time we defined the notion of a *pseudorandom generator* (henceforth referred to as PRGs) and showed that it is equivalent to the notion of *next bit unpredictability.* Today we will look at one approach to constructing PRGs from one way permutations. This uses the idea of "hardcore bit" originally proposed by Goldreich and Levin.

We start off with $f' : 2^n \to 2^n$ which is assumed to be a one way permutation. An example of a function which is conjectured to be a one-way permutation is the discrete log problem. Given a prime p, consider the multiplicative group $\mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$ and let g be a primitive root of $\mathbb{Z}_p^*$. Then the mapping $f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ given by $f(x) = g^x$ is believed to be a one way permutation. Observe that one direction is pretty easy, given x we can compute $f(x)$ easily by repeated squaring. But given $y \in \mathbb{Z}_p^*$ finding $f^{-1}(y)$ seems difficult. The naive way of trying all possibilities takes time $O(2^n)$ where $n = \log p$ is the input size.

A slightly better algorithm is the *baby step, giant step* algorithm. Here we compute two sets of values. First we compute $S_1 = \{g^0, g^{\sqrt{p}}, g^{2\sqrt{p}}, \cdots, g^{(\sqrt{p}-1)\sqrt{p}}\}$ and then we compute the set $S_2 = \{x, xg, xg^2, \cdots, xg^{\sqrt{p}-1}\}$. There has to be some collision between S_1 and S_2, i.e. for some setting of a and b we will have $xg^a = g^{b\sqrt{p}}$ which implies that $x = g^{b\sqrt{p}-a}$. The above algorithm runs in time $\tilde{O}(2^{\frac{n}{2}})$. Thus whereas computing g^x can be done in polynomial time via repeated squaring, the inverse problem of computing the discrete log, seems to be much harder.

We know that it is hard to invert a one-way function. A natural question to ask is whether it is hard to predict a particular bit of the input. However this is not true in general. For instance in the above one way function, it is hard to compute the discrete log of the input, it is easy to predict the least significant bit. The least significant bit can be computed via the legendre symbol. To see this observe that if y is a perfect square then $y = g^{2s}$ and $y^{\frac{p-1}{2}} \bmod p = g^{s(p-1)} \bmod p = (g^{p-1})^s \bmod p = 1^s \bmod p = 1 \bmod p$. On the other hand if y is not a perfect square then $y = g^{2s+1}$ and so $y^{\frac{p-1}{2}} \bmod p = g^{(2s+1)\frac{(p-1)}{2}} \bmod p = (g^{p-1})^s . g^{\frac{p-1}{2}} \bmod p = 1^s . -1 \bmod p = -1 \bmod p$ where we use the fact that since g is a primitive root hence $g^{\frac{p-1}{2}} = -1 \bmod p$ (This follows from the fact that if $g' = g^{\frac{p-1}{2}}$ then we have $g'^2 = 1 \bmod p$. Hence $g' = \pm 1 \bmod p$ and the fact that g generates $\mathbb{Z}_p^*$ implies that $g' \neq 1$.) Note that the least significant bit is 1 if y is not a perfect square and is 0 otherwise and so legendre symbol suffices for predicting the least significant bit. It turns out that predicting the most significant bit of the discrete log is hard if the discrete log problem is hard. In fact this was the motivation of the original paper by Blum and Micali.

In general it is feasible that there are one way functions for which predicting a particular bit is easy. The above discrete log problem is an example. More generally if $f : 2^n \rightarrow 2^n$ is a hard one way permutation then we can construct the function $f' : 2^n \times 2 \rightarrow 2^n \times 2$ which is defined as $f'(x, b) =< f(x), b >$. It can be seen that f' is a one way permutation given that f is a one way permutation and yet predicting the least significant bit of the inverse is trivial.

The above examples suggest that it is not hopeful to expect that for a general one way permutation predicting a particular bit position of the input is hard. We can hope that a predicting a random bit position might be hard but this has not been shown yet.

2 Goldreich Levin Hardcore bit

Let $f : 2^n \rightarrow 2^n$ be a one way permutation. We define the following function $g : 2^n \times 2^n \rightarrow 2^{2n+1}$ as $g(x, y) = (f(x), y, x.y)$ where $x, y \in 2^n$ and $x.y = \sum_{i=1}^{n} x_i y_i \bmod 2$.
The bit $x.y$ is called the "hardcore bit" and we will show that it is hard to predict $x.y$ given $f(x)$ and y. More specifically we will show that if there is a slight advantage in predicting $x.y$ (say $\frac{1}{2} + \epsilon$ where $\epsilon = \frac{1}{n^c}$) then we would be able to invert f. First we consider the simpler case when we have a significant advantage in predicting the hardcore bit ($\geq \frac{3}{4} + \epsilon$.) Next we will consider the more general scenario where we only have a slight advantage of predicting the hardcore bit($\geq \frac{1}{2} + \epsilon$.) Note that in both cases we assume that $\epsilon = \frac{1}{n^c}$.

2.1 A simpler scenario

Let us first consider the case when we have a significant advantage in predicting $x.y$. Let A be an adversary such that $\Pr_{x,y}[A(f(x), y) = x.y] \geq \frac{3}{4} + \epsilon$. We will construct an oracle adversary S^A which will invert f. On input $f(X)$, S^A first randomly generates a $Y \in 2^n$. It then uses A to invert $(f(X), Y)$ and $(f(X), Y')$ where $Y' = Y + e_1$ where the addition represents bitwise addition.Observe that since Y is uniformly distributed hence Y' is also uniformly distributed and so
$\Pr[A(f(X), Y) = X.Y] \geq \frac{3}{4} + \epsilon$ and
$\Pr[A(f(X), Y') = X.Y'] \geq \frac{3}{4} + \epsilon$.
Hence the probability that both A correctly predicts both $X.Y$ and $X.Y'$ is $\geq \frac{1}{2} + 2\epsilon$. Note however that $X.Y + X.Y' = X.Y + X.(Y + e_1) = 2X.Y + x_1 = x_1$ and so with probability $\geq \frac{1}{2} + 2\epsilon$ we get a correct estimate for x_1.

Now we can use the usual voting scheme to amplify the probability. Run the above protocol $\frac{n}{\epsilon^2}$ times and output the majority vote as our prediction for x_1. The probability that our prediction is wrong can be bounded by Chernoff's bound to be less than $e^{-2\frac{(2\epsilon n)^2}{n}} < e^{-n}$.

We run the protocol for each bit position in turn and output our predictions. The probability that any predicted bit value is wrong is atmost ne^{-n} and so with very high probability we can invert $f(X)$. This contradicts the fact that f was supposed to be a one way function.

2.2 More general scenario

Now let us tackle the more general scenario when the adversary A is only assumed to have a slight advantage in predicting the hardcore bit.
More specifically we assume that $\Pr_{x,y}[A(f(x), y) = x.y] \geq \frac{1}{2} + \epsilon$. Again we assume that $\epsilon = \frac{1}{n^c}$.
Note that we cannot use the method of the previous section here. In particular we cannot hope to generate a random vector $Y \in 2^n$ and then run the adversary on $(f(X), Y)$ and $(f(X), Y + e_1)$ since we won't have a good probability that both the answers are returned correctly.

The key idea of the new protocol is that if we know the inner product bit $X.Y_1$ and we know the inner product bit $X.Y_2$, then we also know the inner product bits $X.(\alpha Y_1 + \beta Y_2)$ where $\alpha, \beta \in \{0, 1\}$. This example might not seem very interesting. After all we already know $X.(0Y_1 + 0Y_2)$ and thus we only come to know of one more inner product bit, namely $X.(Y_1 + Y_2)$. However the key thing is to observe that this method scales exponentially.
More specifically if $Y_1, \cdots, Y_l$ are l vectors in $\{0, 1\}^n$, then if we know the inner product of X with each one of these, then we can infer the inner product of X with any linear combination of $Y_1, \cdots Y_n$. So for $\alpha_1, \alpha_2 \cdots \alpha_l \in \{0, 1\}$ we can infer the inner product $X.(\sum_{i=1}^{l} \alpha_i Y_i)$.

There are two important properties of the vectors $Y_\alpha (= \sum_{i=1}^{l} \alpha_i Y_i)$.
Suppose that $Y_1, \cdots Y_l$ are uniformly distributed and independent vectors in $\{0, 1\}^n$.
For $\alpha \in \{0, 1\}^l \backslash 0^l, Y_\alpha$ is uniformly distributed in $\{0, 1\}^n$. This can be easily observed since if $\alpha \neq 0$ then it has atleast one bit which is 1 and let us consider the index k such that $\alpha_k = 1$ and $\forall j > k, \alpha_j = 0$.
Then $Y_\alpha = \sum_{i=1}^{l} \alpha_i Y_i = \sum_{i=1}^{k-1} \alpha_i Y_i + Y_k$
Now for any values of α_i and Y_i for $i < k$, $\sum_{i=1}^{k-1} \alpha_i Y_i$ is a fixed vector and since Y_k is uniformly distributed hence Y_α is uniformly distributed. Since the Y_α are uniformly distributed, hence the vectors $Y'_{\alpha_i} = Y_\alpha + e_i$ are also uniformly distributed.
Another important property of these variables are that even though in general they are highly correlated, they turn out to be pairwise independent. More specifically for $\alpha, \alpha' \in \{0, 1\}^l \backslash 0^l$, Y_α and $Y_{\alpha'}$ are pairwise independent. This is the same as arguing that for any fixed value of Y_α, $Y_{\alpha'}$ is uniformly distributed. The proof is essentially similar to one before, for showing that the Y_α are uniformly distributed. For $\alpha \neq \alpha'$ and $\alpha, \alpha' \neq 0^l$, there is a bit position say k such that $\alpha_k \neq \alpha'_k$. Wlog assume that $\alpha_k = 0$ and so if we consider any fixed setting of all the other $\alpha_j, j \neq k$ and all the $Y_j, j \neq k$, then we know that the value of one of the Y_α is fixed. But as Y_k is uniformly distributed hence $Y_{\alpha'}$ is also uniformly distributed independent of Y_α.

Now we are ready to specify our oracle adversary S^A. Intuitively the adversary does the following - it first guesses the inner product of X with some logarithmic number of vectors $Y_1, \cdots, Y_l$ which are uniformly generated. It can then compute the inner product of X with the Y_α vectors where $\alpha \in \{0, 1\}^l \backslash 0^l$. It then asks the oracle A about the inner product of X with $Y'_{\alpha_i} = Y_\alpha + e_i$ for various values of $\alpha \in \{0, 1\}^l \backslash 0^l$. It uses the majority vote to output a guess for the value of X_i. Since the Y_α are pairwise independent uniform random variables, hence by Chebychev's inequality we have a very small probability that the guess for any bit X_i is wrong. Formally the algorithm is the following

1. For every possible value of $(b_1, b_2, \cdots, b_l) \in \{0,1\}^l$
    2. For every bit position $i \in \{1, \cdots, n\}$
        3. Uniformly and independently pick vectors $Y_1, \cdots, Y_l \in \{0,1\}^n$
            4. For each $\alpha \in \{0,1\}^l \backslash 0^l$ let $Y'_{\alpha_i} = \sum_{j=1}^{l} \alpha_i Y_i + e_i$.
            5.The bit $A(f(x), Y'_{\alpha_i}) + Y_\alpha$ is a guess for x_i. Take the majority vote over the different α and output this as our guess for x_i
    6. Check if $f(x_1, x_2, \cdots, x_n) = f(X)$ and if so output $X = (x_1 x_2 \cdots x_n)$.

Let us now analyze the success probability of this algorithm. We will focus on the iteration when we have the correct guess for all the inner product bits $b_1, \cdots b_l$.
Let us bound the probability that our guess for the i^{th} bit is wrong. Then we will use the union bound to find the probability that our guess for any bit is wrong.
We define indication variables

$$Z_j = \begin{cases} 1 & \text{if the } j^{th} \text{ guess for } x_i \text{ was correct.} \\ 0 & \text{otherwise.} \end{cases}$$

Let us denote $v = 2^l - 1$. Now our algorithm finds the wrong value for x_i iff $\sum_{j=1}^{v} Z_j < \frac{v}{2}$. Thus Pr [algorithm does not find correct value for x_i] = Pr [$\sum_{j=1}^{v} Z_j < \frac{v}{2}$]. Now E [Z_j] $= \frac{1}{2} + \epsilon$ since the Y'_{α_i} are uniformly distributed. Also as argued before the Y'_{α_i} are pairwise independent and so we can use Chebychev's inequality.
$\Pr[\sum_{j=1}^{v} Z_j < \frac{v}{2}\ ] \leq \Pr[|\ \sum_{j=1}^{v} Z_j - \mathrm{E}\ [\sum_{j=1}^{v} Z_j]\ | \geq \epsilon v]$
$\leq \dfrac{(v/4)}{\epsilon^2 v^2}$
$\leq \dfrac{1}{4\epsilon^2 v}$
Here we use the fact that for pairwise independent variables the variance of the sum of variables is the sum of the individual variances, and the variance of a 0-1 random variable is atmost $\frac{1}{4}$.
Hence if we choose $v = \dfrac{n^2}{\epsilon^2}$ then Pr[algorithm does not find the correct value for x_i] $\leq \dfrac{1}{4n^2}$. Hence by the union bound, the probability that the algorithm finds the correct value for every bit x_i is $\geq 1 - \dfrac{1}{4n}$. Also observe that since $\epsilon = \dfrac{1}{n^c}$ hence $v = n^{O(1)}$ and so $l = O(\log n)$ and hence the whole algorithm S^A runs in polynomial time if A runs in polynomial time. This will contradict the fact that f is a one way function.

This proves that if $f : 2^n \to 2^n$ is a one way function, then the function $g : 2^{2n} \to 2^{2n+1}$ given by $g(x, y) = (f(x), y, x.y)$ is a next bit unpredictable PRG and so by the theorem done in the previous lecture, this is a PRG. In Luby's notes, he directly argues that if there is an adversary A such that $\delta(n) = \Pr_{X,Z}[A(f(X), Z, X.Z) = 1] - \Pr_{X,Z,B}[A(f(X), Z, B) = 1]$ then we can construct a S^A such that $\Pr_{X,Z}[S^A(f(X), Z) = X.Z] = \frac{1}{2} + \frac{\delta(n)}{2}$.

Note that a straightforward application of the theorem done last time merely suffices to give an advantage of $\frac{1}{2} + \frac{\delta(n)}{n}$ in predicting the inner product bit, but a more careful analysis yields that the advantage of prediction is infact $\frac{1}{2} + \delta(n)$ (this makes use of the observation that the adversary does not have any advantage in predicting the i^{th} bit of $g(x, y)$ for $i \leq 2n$ since the first $2n$ bits are from a totally uniform distribution.)

3 Some different ways of prediction

We end this lecture with specifying some different ways in which a predictor S^A can predict a bit using a distinguisher A.

- The first alternative is simply the following. We uniformly pick $u, v \in \{0,1\}$ independently. We run $A(f(X), u, Z)$. If the output is 1 we "like" u and output u.
  If the output is 0 we output v.
  This scheme can be shown to have an advantage of $\frac{1}{2} + \frac{\delta(n)}{2}$ where the distinguisher A has an advantage of $\delta(n)$.

- The second scheme flips a bit $u \in \{0,1\}$.
  It runs $A(f(X), u, Z)$ and if the result is 1 it outputs u , otherwise it outputs $\bar{u}$.
  This scheme can be shown to have an advantage of $\frac{1}{2} + \delta(n)$.

- The third scheme is arguably the simplest. It first runs $A(f(X), 0, Z)$ and then $A(f(X), 1, Z)$. Let the outputs be v_1 and v_2 respectively.
  It then outputs

$$u \text{ if } \begin{cases} v_1 = 0 \text{ and } v_2 = 0, \text{ or} \\ v_1 = 1 \text{ and } v_2 = 1 \end{cases}$$

$$1 \text{ if } v_1 = 0 \text{ and } v_2 = 1$$

$$0 \text{ if } v_1 = 1 \text{ and } v_2 = 0$$

  where $u \in \{0,1\}$ is a random bit. This method also has an advantage of $\frac{1}{2} + \delta(n)$.

In the next lecture we will show that the last two strategies mentioned above are in fact equivalent.