Introduction to Complexity Theory*
Lecture 14: Pseudorandomness and Computational Difficulty

Notes taken by Moshe Lewenstein and Yehuda Lindell

April 14, 1999

Summary: In this lecture we continue our discussion of pseudorandomness and show a connection between pseudorandomness and computational difficulty. More specifically we show how the difficulty of inverting one-way functions may be utilized to obtain a pseudorandom generator. Finally, we state and prove that a hard-to-predict bit (called a hard-core) may be extracted from any one-way function. The hard-core is fundamental in our construction of a generator.

1 Introduction

The main theme of this lecture is the utilization of one-way functions in order to construct a pseudorandom generator. Intuitively, a one-way function is a function that is *easy* to compute and *hard* to invert. Generally, "easy" refers to polynomial time and "hard" to the fact that success in the *average case* requires more than polynomial time (for any polynomial). It is critical that the difficulty be in the average case and *not* in the worst case, as with $\mathcal{NP}$-Complete problems. This will become clear later.

How can one-way functions help us construct a pseudorandom generator? The answer lies in the property of unpredictability. This concept will be formalized in the coming lectures but for now we will discuss it informally. Assume that we have a string s and we begin to scan it in some given (computable) order. If at some stage we can predict the next bit with probability significantly greater than one half, then the string is clearly not random (because for a random string, each bit is chosen *independently* with probability exactly one half). On the other hand, it can be shown that if we *cannot* predict any "next" bit during our scan with success significantly more than $\frac{1}{2}$, then the string is pseudorandom.

In this light, the use of computationally difficult problems becomes clear. We rely on the difficulty of inverting a one-way function f. More specifically we show that there exists a function $b : \{0,1\}^n \rightarrow \{0,1\}$ such that given x it is easy to compute $b(x)$ yet given only $f(x)$ it is difficult. This function is formally called a hard-core of f. Now, although given $f(x)$, $b(x)$ is fully and deterministically defined, we have no way of finding or predicting its value. Therefore, the computational difficulty of finding $b(x)$ provides us with an unpredictable bit which forms the basis of our generator.

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

2 Definitions

We now recall the concepts necessary for this lecture.

Definition 1 Pseudorandom generators: *G is a* pseudorandom generator *if:*

1. *G operates in (deterministic) polynomial time*
2. *For every s,* $|G(s)| > |s|$ *(w.l.o.g., assume that* $\exists l(\cdot)$ *such that* $\forall s \in \{0,1\}^n$ $|G(s)| = l(n)$*).*
3. $\{G(U_n)\}$ *and* $\{U_{l(n)}\}$ *are probabilistic polynomial time indistinguishable (where* U_n *is the uniform distribution over* $\{0,1\}^n$*).*

Definition 2 One-way functions: *Let* $f : \{0,1\}^* \to \{0,1\}^*$ *be a length-preserving function (i.e.* $\forall x$ $|f(x)| = |x|$*). Then f is* one-way *if:*

1. *f is "easy" to compute. Formally, there exists a polynomial time algorithm A such that* $\forall x$ $A(x) = f(x)$*.*
2. *f is "hard" to invert in the average case. Formally, for every probabilistic polynomial time algorithm A, for every polynomial* $p(\cdot)$*, and for all sufficiently large n's,*

$$Prob[A(f(U_n)) \in f^{-1}f(U_n)] < \frac{1}{p(n)}$$

The above definition refers to a length-preserving function. This is a simplifying assumption we will use throughout this lecture, but it is generally not necessary as long as $|x| = poly(|f(x)|)$. The requirement that the lengths of x and $f(x)$ be polynomially related is necessary to ensure that the difficulty involved in inverting $f(x)$ is not because x is too long. Since the inverting algorithm must work in time polynomial in $|f(x)|$, if $|f(x)|$ is logarithmic in $|x|$, no algorithm can even write x. In this case there is no computational difficult in *inverting* f and the one-wayness is due to the above technicality. Assuming that f is length-preserving avoids this problem.

As we will see, there is no requirement that f be length-preserving in the hard-core theorem stated and proved in section 4. However, the exact construction of the pseudorandom generator in section 3 relies heavily upon the length-preserving property of the one-way function and the assumption that it is $1-1$. Other constructions exist for the more general case but are more complex.

Although the definition of a one-way function guarantees that it is difficult to find the entire string x given $f(x)$, it may be very easy to obtain some of the bits of x. For example, assuming that f is one-way, consider the function $f'(\sigma \cdot x) = \sigma \cdot f(x)$, where $\sigma \in \{0,1\}$. It is easy to see that f' is also one-way. This is rigorously shown by assuming that f' is *not* one-way and showing how f can be inverted using an algorithm which inverts f'. So we see that despite the fact that f' is one-way, the first bit of the input is completely revealed.

Therefore, in order to obtain unpredictability as desired, we need a specific bit based on x, which is provably hidden given $f(x)$. This bit is called a *hard-core* of f.

Reducibility Arguments: The above-mentioned method of proof (showing that $f'(\sigma \cdot x) = \sigma \cdot f(x)$ is one-way) is called a reduction. It is worthwhile discussing this technique as it will form the basis of many proofs that we will see. The context in which it appears is when we take a certain primitive and construct a new primitive based on it. For example, we will need it to prove our construction of a pseudorandom generator from a one-way function. Although it may be intuitively clear that a new construction is "correct" this sort of intuition may be misleading and so requires a sound proof. We do this by showing that if the newly constructed primitive is not secure, then an algorithm defying it may be used to defy the original primitive as well. More concretely, if our generator is distinguishable, then a distinguishing algorithm for it may be used to invert the one-way function used in the construction. We will see this type of argument many times in the coming lectures.

Definition 3 Hard-Core: *The function $b : \{0,1\}^* \to \{0,1\}$ is a* hard-core *of f if:*

1. *b is easy to compute (in the same sense as above),*
2. *For every probabilistic polynomial time algorithm A, for every polynomial $p(\cdot)$, and for all sufficiently large n's,*
$$Prob[A(f(U_n)) = b(U_n)] < \frac{1}{2} + \frac{1}{p(n)}$$

Note that it is trivial to predict b with probability $\frac{1}{2}$ by simply guessing. The above definition requires that we cannot do significantly better than this guess. A hard-core of a one-way function plays an important role in cryptographic applications (as all information is hidden). However, in our case, we will use the computational difficulty involved as the basis for our construction of a pseudorandom generator. This can be seen in the following way. Given $f(x)$, $b(x)$ is fully-defined, yet completely unpredictable to any polynomially bound algorithm. This extra bit of unpredictable information is what will supply the "stretch" effect necessary for the generator.

The following claim shows that a hard-core can only exist for a one-way function. This is intuitively obvious, since if f is $1-1$ but not one-way then we can invert it and compute b. This is formally shown below.

Claim 2.1 *If f is 1-1 and polynomial time computable, and b is a hard-core of f, then f is one-way.*

Proof: By contradiction, assume that b is a hard-core of f, yet f is not one-way. We now show how an algorithm inverting f can be used to predict $b(x)$ from $f(x)$. Note once again the reduction technique we use.

f is not one-way, therefore there exists a probabilistic polynomial time algorithm A and a polynomial $p(\cdot)$ such that for infinitely many n's, $Prob[A(f(x)) = x] \geq \frac{1}{p(n)}$ ($|x| = n$).

We now construct an algorithm A' for predicting b from f:

Input: y

- $x' \leftarrow A(y)$ (attempt to invert y, using A).
- If $f(x') = y$ then output $b(x')$
- Otherwise, output 0 or 1 with probability 1/2.

We now calculate the success probability of A'.

$$\begin{aligned} Prob[A'(f(x)) = b(x)] &= Prob[A(f(x)) = x]\cdot 1 + Prob[A(f(x)) \neq x]\cdot\frac{1}{2} \\ &\geq \frac{1}{p(n)}\cdot 1 + (1-\frac{1}{p(n)})\cdot\frac{1}{2} \quad = \quad \frac{1}{2} + \frac{1}{2p(n)} \end{aligned}$$

The success probability of A' is greater than or equal to $\frac{1}{2} + \frac{1}{2p(n)}$ for infinitely many n's, thereby contradicting the fact that b is a hard-core of f. ■

Comment: If f is not 1-1, then the above claim is false. Consider f such that $f(x) = 0^{|x|}$. Clearly $b(x) =$ "the 1st bit of x" is a hard-core. However this is because of information theoretic considerations rather than computational bounds. The function f defined here is *trivially* inverted by taking any arbitrary string of length $|x|$ as the preimage. However, $b(x)$ may clearly not be guessed with any probability better than $\frac{1}{2}$.

3 A Pseudorandom Generator based on a 1-1 One-Way Function

In this section we show a construction of a pseudorandom generator given a length-preserving 1-1 one-way function. The construction is based on a hard-core of the one-way function. In the next section, we show how to generically construct a hard-core of any one-way function.

We note that constructions of a pseudorandom generator exist based on *any* one-way function (not necessarily length-preserving and 1-1). However, the constructions and proofs in the more general case are long and complicated and we therefore bring this case only.

Theorem 4 *Let f be a length-preserving, 1-1 one-way function and let b be a hard-core of f. Then $G(s) = f(s)b(s)$ is a pseudorandom generator (stretching the input by one bit).*

Proof: We first note that as f is length-preserving and 1-1, $f(U_n)$ is distributed uniformly over $\{0,1\}^n$ and is therefore fully random. It remains to show that for $s \in_R \{0,1\}^n$, the combination of $f(s)$ and $b(s)$ together remains indistinguishable from U_{n+1}. Intuitively, as we cannot predict $b(s)$ from $f(s)$, the string looks random to any computationally bound algorithm.

We will now show how a successful distinguishing algorithm may be used to predict b from f. This then proves that no such distinguishing algorithm exists (because b is a hard-core of f). By contradiction, assume that there exists an algorithm A and a polynomial $p(\cdot)$ such that for infinitely many n's

$$|Prob[A(f(U_n)b(U_n)) = 1] - Prob[A(U_{n+1}) = 1]| \geq \frac{1}{p(n)}$$

As $f(U_n)$ is distributed uniformly, this is equivalent to A successfully distinguishing between $\{f(U_n)b(U_n)\}$ and $\{f(U_n)U_1\}$. It follows that A can distinguish between $X_1 = \{f(U_n)b(U_n)\}$ and $X_2 = \{f(U_n)\overline{b(U_n)}\}$.

Let X be the distribution created by uniformly choosing $\sigma \in_R \{1,2\}$ and then sampling from X_σ. Clearly X is identically distributed to $\{f(U_n)U_1\}$. Now:

$$Prob[A(X) = 1] = \frac{1}{2} \cdot Prob[A(X_1) = 1] \; + \; \frac{1}{2} \cdot Prob[A(X_2) = 1]$$

$$\Rightarrow Pr[A(X_2) = 1] = 2 \cdot Prob[A(X) = 1] \; - \; Prob[A(X_1) = 1]$$

Therefore:

$$| \; Prob[A(f(U_n)b(U_n)) = 1] \; - \; Prob[A(f(U_n)\overline{b(U_n)}) = 1] \; |$$

$$= | \; Prob[A(X_1) = 1] - Prob[A(X_2) = 1] \; |$$

$$= | \; Prob[A(X_1) = 1] - 2 \cdot Prob[A(X) = 1] + Prob[A(X_1) = 1] \; |$$

$$= |\ 2{\cdot}Prob[A(X_1)=1] - 2{\cdot}Prob[A(X)=1]\ |$$

$$= 2{\cdot}|\ Prob[A(f(U_n)b(U_n))=1] - Prob[A(U_{n+1})=1]\ | \geq \frac{2}{p(n)}$$

Assume, without loss of generality, that for infinitely many n's it holds that:

$$Prob[A(f(U_n)b(U_n))=1] - Prob[A(f(U_n)\overline{b(U_n)})=1] \geq \frac{2}{p(n)}$$

Otherwise we simply reverse the terms here and make the appropriate changes in the algorithm and the remainder of the proof (i.e. we change step 2 of the algorithm below to: If $A(y\cdot\sigma)=0$ then output σ).

We now construct A' to predict $b(U_n)$ from $f(U_n)$. Intuitively, A' adds a random guess to the end of it's input y (where $y=f(x)$ for some x) and attempts to see if it guessed $b(f^{-1}(y))$ correctly based on A's response to this guess. The algorithm follows:

Input: y

1. Uniformly choose $\sigma \in_R \{0,1\}$
2. If $A(y\cdot\sigma)=1$ then output σ
3. Otherwise, output $\overline{\sigma}$

We now calculate the probability that A' successfully computes $b(f^{-1}(y))$. As σ is uniformly chosen from $\{0,1\}$ we have:

$$\begin{aligned} Prob[A'(f(U_n))=b(U_n)] &= \frac{1}{2}\cdot Prob[A'(f(U_n))=b(U_n)\ |\ \sigma=b(U_n)] \\ &+\frac{1}{2}\cdot Prob[A'(f(U_n))=b(U_n)\ |\ \sigma=\overline{b(U_n)}] \end{aligned}$$

Now, by the algorithm (see steps 2 and 3 respectively) we have:

$$Prob[A'(f(U_n))=b(U_n)\ |\ \sigma=b(U_n)] = Prob[A(f(U_n)b(U_n))=1]$$

and

$$\begin{aligned} Prob[A'(f(U_n))=b(U_n)\ |\ \sigma=\overline{b(U_n)}] &= Prob[A(f(U_n)\overline{b(U_n)})=0] \\ &= 1-Prob[A(f(U_n)\overline{b(U_n)})=1] \end{aligned}$$

By our contradiction hypothesis:

$$Prob[A(f(U_n)b(U_n))=1] - Prob[A(f(U_n)\overline{b(U_n)})=1] \geq \frac{2}{p(n)}$$

Therefore:

$Prob[A'(f(U_n))=b(U_n)]$

$$= \frac{1}{2}{\cdot}Prob[A(f(U_n)b(U_n))=1] + \frac{1}{2}{\cdot}(1-Prob[A(f(U_n)\overline{b(U_n)})=1])$$

$$= \frac{1}{2} + \frac{1}{2} \cdot (Prob[A(f(U_n)b(U_n)) = 1] - Prob[A(f(U_n)\overline{b(U_n)}) = 1]) \;\; \geq \;\; \frac{1}{2} + \frac{1}{p(n)}$$

which is in contradiction to the fact that b is a hard-core of f and cannot be predicted with non-negligible advantage over $\frac{1}{2}$. ■

We remind the reader that in the previous lecture we proved that a generator stretching the seed by even a single bit can be deterministically converted into a generator stretching the seed by any polynomial length. Therefore, it should not bother us that the above construction seems rather weak with respect to its "stretching capability".

At this stage it should be clear why it is crucial that the one-way function be hard to invert in the average case and not just in the worst case. If the function is invertible in the average case, then it is easy to distinguish between $\{f(U_n)b(U_n)\}$ and $\{U_{n+1}\}$ most of the time. This would clearly *not* be satisfactory for a pseudorandom generator.

4 A Hard-Core for Any One-Way Function

In this section we present a construction of a hard-core from any one-way function. Here there is no necessity that f be 1-1 or even length-preserving. As we have seen in the previous section, the existence of a hard-core is essential in our construction of a pseudorandom generator.

Theorem 5 *Let $f_0 : \{0,1\}^* \to \{0,1\}^*$ be a one-way function. Define $f(x,r) = (f_0(x), r)$ where $|x| = |r|$. Then $b(x,r) = \sum_{i=1}^{n} x_i r_i \bmod 2$ is a hard-core of f.*

Note that since f_0 is one-way, f is clearly one-way (trivially, any algorithm inverting f can be used to invert f_0).

Proof: Assume by contradiction, that there exists a probabilistic polynomial time algorithm A and a polynomial $p(\cdot)$ such that for infinitely many n's

$$Prob_{x,r}[A(f(x,r)) = b(x,r)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

where the probabilities are taken, uniformly and independently, over $x \in_R \{0,1\}^n$, $r \in_R \{0,1\}^n$ and coin tosses of A (if any).

The following claim shows that there are a significant number of x's for which A succeeds with non-negligible probability. We will then show how to invert f for these x's.

Claim 4.1 *Let $S_n \subseteq \{0,1\}^n$ be the set of all x's where $Prob_r[A(f(x,r)) = b(x,r)] \geq \frac{1}{2} + \frac{1}{2p(n)}$. Then, $Prob_x[x \in S_n] > \frac{1}{2p(n)}$.*

Proof: By a simple averaging argument. Assume by contradiction that $Prob_x[x \in S_n] \leq \frac{1}{2p(n)}$. Then:

$$Prob_{x,r}[A(f(x,r)) = b(x,r)] = Prob_{x,r}[A(f(x,r)) = b(x,r) \mid x \in S_n] \cdot Prob[x \in S_n]$$

$$+ Prob_{x,r}[A(f(x,r)) = b(x,r) \mid x \notin S_n] \cdot Prob[x \notin S_n]$$

Now, trivially both $Prob_{x,r}[A(f(x,r)) = b(x,r) \mid x \in S_n] \leq 1$ and $Prob[x \notin S_n] \leq 1$. Furthermore, by the contradiction hypothesis $Prob[x \in S_n] \leq \frac{1}{2p(n)}$. Finally, based on the definition of S_n, $Prob_{x,r}[A(f(x,r)) = b(x,r) \mid x \notin S_n] < \frac{1}{2} + \frac{1}{2p(n)}$. Putting all these together:

$$Prob_{x,r}[A(f(x,r)) = b(x,r)] < 1 \cdot \frac{1}{2p(n)} + (\frac{1}{2} + \frac{1}{2p(n)}) \cdot 1 = \frac{1}{2} + \frac{1}{p(n)}$$

which contradicts the assumption regarding the success probability of A. ■

It suffices to show that we can retrieve x from $f(x)$ for $x \in S_n$ with probability $\frac{1}{poly(n)}$ (because then we can invert any $f(x)$ with probability $\geq \frac{1}{2p(n)poly(n)} = \frac{1}{poly(n)}$). So, assume that for x: $Prob_r[A(f(x,r)) = b(x,r)] \geq \frac{1}{2} + \frac{1}{2p(n)}$ (as in the claim).

Denote $B_x(r) = A(f(x,r)$. Now x is fixed and $B_x(r)$ is a black-box returning $b(x,r)$ with probability $\geq \frac{1}{2} + \frac{1}{2p(n)}$. We use calls to B_x to retrieve x given $f(x)$. We also denote $\epsilon = \frac{1}{2p(n)}$.

As an exercise, assume that $Prob_r[B_x(r) = b(x,r)] > \frac{3}{4} + \epsilon$. There is no reason to assume that this is true but it will help us with the proof later on. Using this assumption, for every $i = 1, ..., n$ we show how to recover x_i (that is, the i'th bit of x).

Input: $y = f(x)$

1. Uniformly select $r \in_R \{0,1\}^n$
2. Compute $B_x(r) \oplus B_x(r \oplus e^i)$ where $e^i = (0^{i-1}10^{n-i})$ (1 in the i'th coordinate and 0 everywhere else).

Now, $Prob[B_x(r) \neq b(x,r)] < \frac{1}{4} - \epsilon$ by the hypothesis. Therefore, $Prob[B_x(r \oplus e^i) \neq b(x, r \oplus e^i)] < \frac{1}{4} - \epsilon$ (because $r \oplus e^i$ is also uniformly distributed). So, the probability that $B_x(r) = b(x,r)$ and $B_x(r \oplus e^i) = b(x, r \oplus e^i)$ is greater than $\frac{1}{2} + 2\epsilon$ (by summing the errors).
In this case: $B_x(r) \oplus B_x(r \oplus e^i) = b(x,r) \oplus b(x, r \oplus e^i)$. However,

$$b(x,r) \oplus b(x, r \oplus e^i) = \sum_{j=1}^{n} x_j r_j + \sum_{j=1}^{n} x_j (r_j + e^i_j) \ mod\ 2$$

$$= \sum_{j=1}^{n} x_j r_j + x_i + \sum_{j=1}^{n} x_j r_j \ mod\ 2 = x_i$$

So, if we repeat this $O(\frac{n}{\epsilon^2})$ times and take a majority vote, we obtain a correct answer with a very high probability (in the order of $1 - \frac{1}{2^n}$). This is true (but with probability $\geq 1 - \frac{1}{2n}$) even if the different executions are only pairwise independent (this can be derived using Chebyshev's inequality and is important later on). We do the same for all i's and in this way succeed in inverting $f(x)$ with high probability. Note that we can use the same set of random strings r for each i (the only difference is in obtaining $b(x, r \oplus e^i)$ each time).

This inversion algorithm worked based on the unreasonable assumption that $Prob_r[B_x(r) = b(x,r)] > \frac{3}{4} + \epsilon$. This was necessary as we needed to query B_x on two different points and therefore we had to sum the error probabilities. When the probability of success is only ϵ above $\frac{1}{2}$, the resulting error probability is far too high.

In order to solve this problem, remember that we are really interested in calculating $b(x, e^i) = x_i$. However, we cannot query $B_x(e^i)$ because we have no information on B_x's behaviour at any given point (the known probabilities are for $B_x(r)$ where r is uniformly distributed). Therefore we queried B_x at r and $r \oplus e^i$ (2 random points) and inferred x_i from the result.

We now show how to compute $b(x,r) \oplus b(x, r \oplus e^i)$ for $O(\frac{n}{\epsilon^2})$ pairwise independent r's. Based on what we have seen above, this is enough to invert $f(x)$. Our strategy is based on obtaining values of $b(x,r)$ for "free" (that is, without calling B_x). We do this by guessing what the value should be, but in such a way that the probability of being correct is non-negligible.

Let $l = \log_2(m+1)$ where $m = O(\frac{n}{\epsilon^2})$.
Let $s^1, ..., s^l \in_R \{0,1\}^n$ be l uniformly chosen n-bit strings.
Then for every non-empty $I \subseteq \{1, ..., l\}$, define: $r^I = \bigoplus_{i \in I} s^i$. Each r^I is an n-bit string constructed by xoring together the strings s^i indexed by I.

Each r^I is uniformly distributed as it is the xor of random strings. Moreover each pair is independent since for $I \neq J$, $\exists s^i$ s.t. w.l.o.g. $s^i \in I$ and $s^i \notin J$. Therefore, r^I given r^J is uniformly distributed based on the distribution of s^i.

Now, let us uniformly choose $\sigma^1, ..., \sigma^l \in \{0,1\}$. Assume that we were very lucky and that for every i, $\sigma^i = b(x, s^i)$ (in other words, we guessed $b(x,s^i)$ correctly every time). Note that this lucky event happens with the non-neglible probability of $\frac{1}{2^l} = \frac{\epsilon}{n^2} = \frac{1}{poly(n)}$. The following claim shows that in this lucky case we achieve our aim.

Claim 4.2 *Let $\tau^I = \bigoplus_{i \in I} \sigma^i$. Then, if for every i, $\sigma^i = b(x,s^i)$ then for every I, $\tau^I = b(x, r^I)$.*

Proof:

$$b(x, r^I) = b(x, \sum_{i \in I} s^i) = \sum_{j=1}^{n} x_j \cdot \sum_{i \in I} s^i_j = \sum_{i \in I} \sum_{j=1}^{n} x_j \cdot s^i_j = \sum_{i \in I} b(x, s^i) = \sum_{i \in I} \sigma^i = \tau^I$$

where all sums are mod 2. ■

The above claim shows that by correctly guessing $\log m$ values of σ^i we are able to derive the value of $b(x, \cdot)$ at m pairwise independent points. This is because under the assumption that we guessed the σ^i's correctly, each τ^I is exactly $b(x, r^I)$ where the r^I's are uniformly distributed and pairwise independent.

Note that since there are $2^l - 1 = m$ different subsets I, we have the necessary number of different points in order to obtain x_i. The algorithm uses these points in order to extract x_i instead of uniformly chosen r's. (An alternative strategy is not to guess the σ^i's but to try every possible combination of them. Since $2^l = m + 1$ which is polynomial, we can do this in the time available.)

The Actual Algorithm:

Input: y

1. Uniformly choose $s^1, ..., s^l \in_R \{0,1\}^n$ and $\sigma^1, ..., \sigma^l \in \{0,1\}$.
2. For every non-empty $I \subseteq \{1, ..., l\}$, define: $r^I = \bigoplus_{i \in I} s^i$ and compute $\tau^I = \bigoplus_{i \in I} \sigma^i$.
3. For every $i \in \{1, ..., n\}$ and non-empty $I \subseteq \{1, ..., l\}$ do
   - $v^I_i \ = \ \tau^i \oplus B_x(r^I \oplus e^i)$
   - Guess $x_i \ = \text{majority}_I\{v^I_i\}$

In order to calculate the probability that the above algorithm succeeds in inverting $f(x)$, assume that for every i, we guessed σ^i such that $\sigma^i = b(x, s^i)$ (in step 1). The probability of this event occurring is $\frac{1}{2^l} = \frac{1}{m+1}$ for $m = O(\frac{n}{\epsilon^2})$ and $\epsilon = \frac{1}{2p(n)}$. In other words, the probability that we are lucky is $\frac{1}{poly(n)}$.

Since we know that for every I, $B_x(r^I \oplus e^i)$ is correctly computed with a probability greater that $\frac{1}{2} + 2\epsilon$, it follows that $b(x, r^I) \oplus B_x(r^I \oplus e^i)$ is also correct with the same probability (as the σ^i's are correct). As previously mentioned, due to the pairwise independence of the events, the probability that we succeed in inverting $f(x)$ in this case is at least $1 - \frac{1}{2n}$ (this is proved using Chebyshev's inequality).

It is easy to see that the probability that we succeed in guessing all σ^i's correctly *and* then proceed to successfully invert f is the product of the above two probabilities, that is $\frac{1}{poly(n)}$.

We therefore conclude that the probability of successfully inverting f is non-negligible. This is in contradiction to the one-wayness of f. Therefore, b as defined is a hard-core of f. ■

Summary of the proof.

We began by assuming the existence of an algorithm A which predicts $b(x, r)$ from $f(x, r)$ with non-negligible success probability (over all choices of x, r and coin tosses of A). We then showed that there is a non-negligible set of x's for which A succeeds and that we proceed to attempt to invert $f(x)$ for these x's only. This enabled us to set x and focus only on the randomness in r.

In the next stage we described how we can obtain x, under the assumption that both $b(x, r)$ and $b(x, r \oplus e^i)$ can be computed correctly with probability $\geq \frac{1}{2} + \frac{1}{poly(n)}$. This was easily shown as $b(x, r) \oplus b(x, r \oplus e^i) = x_i$ and by repeating we can achieve a small enough error so that the probability that we succeed in obtaining x_i for all i is at least $\frac{1}{poly(n)}$.

Finally we showed how to achieve the previous assumption. This involved realizing that pairwise independence for different r's is enough and then showing how $poly(n)$ pairwise independent n-bit strings can be obtained using only $log(poly(n))$ n-bit random strings $s^1, ..., s^l$.

Based on this, we guess the value of $b(x, s^i)$ for each i. The critical point here is that the probability of guessing correctly for all i is $\frac{1}{poly(n)}$ and that the value of $b(x, \cdot)$ for all $poly(n)$ pairwise independent n-bit strings can be immediately derived from these guesses.

In short, we succeeded (with non-negligible probability) in guessing $b(x, \cdot)$ correctly for a polynomial number of pairwise independent strings. These strings are used as r for $b(x, r)$ in the inversion algorithm described in the middle stage. We compute $b(x, r \oplus e^i)$ using A. Assuming that we guessed correctly, we achieve the necessary success probability for computing both $b(x, r)$ and $b(x, r \oplus e^i)$. As we guess once for the entire inversion algorithm and this is independent of all that follows, we are able to extract x from $f(x)$ with overall probability of $\frac{1}{poly(n)}$.

This proves that no such algorithm A exists.