Chapter 9

Derandomization

Chapter Outline: Derandomization intro. Psudo random generators. Next-bit prediction and pseudo-random equivalence. Oneway functions. Weak-oneway implies strong oneway. Goldreich Levin hardcore bit. Constructing pseudo random generators using GL-hardcore bit.

AN INTRO TO DERANDOMIZATION IS REQUIRED HERE.

9.1 Pseudorandom Generators

Informally, a pseudorandom generator G is a polynomial time algorithm, that stretches a short random string called seed into a long pseudorandom string. G is a successful, if its output cannot be distinguished from truly random strings, by any efficient adversary. In other words, the output of G is computationally indistinguishable from truly random strings. We formalize the notion of computational indistinguishability in two models, and show their equivalence.

A *generator* is a polynomial-time computable, deterministic function $G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$ where $l(n) > n$. An *adversary* is a probabilistic polynomial time Turing machine.

Pseudorandom Generators: We call the first model of formalism as pseudorandom model. Here, we adopt the following convention. An adversary A, given an input string, outputs 1, if it it thinks its input is pseudorandom, and outputs if it thinks its input is truly random. With respect to a generator G, we define the the success of an adversary A, at a length n as:-

$$\delta_n^A = |\Pr_{x\in\{0,1\}^n}[A(G(x)) = 1] - \Pr_{y\in\{0,1\}^{l(n)}}[A(y) = 1]|$$

So δ_n^A is larger if the adversary is distinguishing well.

Definition (Pseudorandom Generator): We say that G beats A if $\forall$ polynomials $p(\cdot)$, $\forall$ sufficiently large n, we have $\delta_n^A < \frac{1}{p(n)}$. G is defined to be a pseudorandom generator if G

beats every adversary.

Remark: By the above definition, A beats G if there exists $p(\cdot)$ such that for infinitely many n we have $\delta_n^A > \frac{1}{p(n)}$.

Next bit unpredictability: In this second model, adversary A is given first i bits of the output of the generator. It has to predict the $(i+1)^{th}$ bit. For an adversary A, at a length, the success is measured as,

$$\beta_n^A = \Pr_{\substack{x \sim U_n \\ i \in_R [0..l(n)-1]}} [A(G(x)_{1,\ldots,i}) = G(x)_{i+1}] - \frac{1}{2}$$

Thus, we take average over all i. Any adversary can easily get a success of $1/2$ by simply guessing. So, we have subtracted $1/2$, to normalize.

Definition (Next-bit unpredictable generator): We say that G beats A if $\forall p(\cdot)$, $\forall$ sufficiently large n, we have $\beta n^A < \frac{1}{p(n)}$. G is defined to be a next-bit unpredictable if G beats every adversary.

The first model is due to Yao and the second is due to Blum and Micali. The equivalence was shown by Yao.

Theorem (Yao): G is pseudorandom if and only if G is next-bit predictable.
Proof: Equivalently, G is not pseudorandom if and only if G is not next-bit predicatable. So, we show that, some adversary beats G in the pseudorandom model if and only if some adversary beats G in the next bit model.

Part I: Suppose A is an adversary in the next bit model. We want to find an adversary A' that performs equally good in the pseudorandom model.

Fix n. A' takes as input a $l(n)$ bit string and has to predict whether its truly random or pseudorandom. A' uses it A as an oracle. A' on input $y_1, \ldots y_{l(n)}$, does the following:-

1. First choose a random $i \in [0..l(n)-1]$.
2. If $A(y_1, \ldots, y_i) = y_{i+1}$ then output 1, i.e., guess pseudorandom. Otherwise output 0, i.e., guess truly random.

Suppose the success of A, at length n is, β_n^A.

$$Pr_{x \sim U_n}[A'(G(x)) = 1] = Pr_{\substack{x \sim U_n \\ i \in_R [0..l(n)-1]}} [A(G(x)_{1,\ldots,i}) = G(x)_{i+1}] = \beta_n^A + \frac{1}{2}$$

If the input to A' is truly random, irrespective of the i chosen by A', A can output y_{i+1} with probability only $1/2$. So,

$$Pr_{y \in \{0,1\}^{l(n)}}[A(y) = 1] = \frac{1}{2}$$

Thus,

$$\delta_n^{A'} = Pr_{x \in \{0,1\}^n}[A(G(x)) = 1] - Pr_{y \in \{0,1\}^{l(n)}}[A(y) = 1] = \beta_n^A + \frac{1}{2} - \frac{1}{2} = \beta_n^A$$

Thus, for all n, $\delta_n^{A'} = \beta_n^A$. If A can beat G in next-bit model, A' can beat G in the pseudorandom model.

Part II: Now suppose that A is an adversary for G in the pseudorandom random model, with success δ_n^A at length n. We construct an adversary A' in the next bit model.

Fix n. Let the input to A' be $b_1, \ldots, b_i$. A' has to predict the next bit b_{i+1}. A' does the following:-

1. Choose $l(n) - i$ random bits $r_{i+1}, \ldots, r_{l(n)}$ at random.
2. Compute $a = A(b_1, \ldots, b_i, r_{i+1}, \ldots, r_{l(n)})$.
3. If $a = 1$ output r_{i+1}, else output $\overline{r_{i+1}}$.

The intuition is that A outputs 1 more often when given all pseudorandom bits. And outputs 0 more often when given all truly random bits. We input A with a mix of i pseudorandom bits and $(l(n) - i)$ truly random bits. b_{i+1} is either r_{i+1} or its compliment. If $r_{i+1} = b_{i+1}$ then A gets one more pseudorandom random bit and is more likely to output 1. To put it in a reverse Bayesian way, if A outputs 1, we are more likely to have $r_{i+1} = b_{i+1}$. To get a formal proof, we first quantify the behavior of A when given a mix of pseudorandom and truly random bits. Suppose we input A with i pseudorandom bits and $l(n) - i$ truly random bits. The probability that A outputs 1, for such a mix, is denoted by p_i. Formally,

$$p_i = Pr_{\substack{x \sim U_n \\ r_{i+1}, \ldots, r_{l(n)} \in_R \{0,1\}}} [A(G(x)_{1,\ldots,i} \circ r_{i+1}, \ldots, r_{l(n)}) = 1$$

First observe that,

$$p_0 = Pr_{y \in_R \{0,1\}^{l(n)}}[A(y) = 1]$$
$$p_{l(n)} = Pr_{x \in_R \{0,1\}^n}[A(G(x)) = 1]$$

Thus, $p_{l(n)} - p_0 = \delta_n^A$.

We next compute the success of A' in terms of δ_n^A. For any $0 \leq i < l(n)$, we denote the success of A' as $\beta_{n,i}^{A'}$. We let $w = b_1, \ldots b_i, r_{i+1}, \ldots, r_{l(n)}$. Then,

$$\begin{aligned}
\beta_{n,i}^{A'} &= Pr[A'(b_1, \ldots b_i) = b_{i+1}] \\
&= Pr[A(w) = 1 \text{ and } r_{i+1} = b_{i+1}] + Pr[A(w) = 0 \text{ and } r_{i+1} \neq b_{i+1}] \\
&= Pr[r_{i+1} = b_{i+1}] \cdot Pr[A(w) = 1 | r_{i+1} = b_{i+1}] + Pr[r_{i+1} \neq b_{i+1} \cdot Pr[A(w) = 0 | r_{i+1} \neq b_{i+1}] \\
&= \frac{1}{2} \left(Pr[A(w) = 1 | r_{i+1} = b_{i+1}] + Pr[A(w) = 0 | r_{i+1} \neq b_{i+1}] \right)
\end{aligned}$$

The first quantity inside the parenthesis, is easy to compute. When $r_{i+1} = b_{i+1}$, we are giving one more pseudorandom bit to A. So,

$$Pr[A(w) = 1 | r_{i+1} = b_{i+1}] = p_{i+1}$$

. To compute the second quantity:-

$$\begin{aligned} p_i &= Pr[A(b_1, \ldots, b_i, r_{i+1}, \ldots, r_{l(n)}) = 1] \\ &= Pr[r_{i+1} = b_{i+1}] \times Pr[A(w) = 1 | r_{i+1} = b_{i+1}] + Pr[r_{i+1} \neq b_{i+1} \times Pr[A(w) = 1 | r_{i+1} \neq b_{i+1}] \\ &= \frac{1}{2} p_{i+1} + \frac{1}{2} Pr[A(w) = 1 | r_{i+1} \neq b_{i+1}] \end{aligned}$$

So,

$$Pr[A(w) = 0 | r_{i+1} \neq b_{i+1}] = p_{i+1} - 2p_i + 1$$

Substituting in the equation for $\beta_{n,i}^{A'}$, we get,

$$\beta_{n,i}^{A'} = \frac{1}{2}(p_{i+1} + p_{i+1} - 2p_i + 1) = \frac{1}{2} + p_{i+1} - p_i$$

Now we can compute the success $\beta_n^{A'}$ of A' when i is chosen at random.

$$\begin{aligned} \beta_n^{A'} &= \sum_{i=0}^{l(n)} \frac{1}{l(n)} \beta_{n,i}^{A'} \\ &= \sum_{i=0}^{l(n)} \frac{1}{l(n)} \left(\frac{1}{2} + p_{i+1} - p_i \right) \\ &= \frac{1}{2} + \frac{1}{l(n)} \sum_{i=0}^{l(n)} p_{i+1} - p_i \\ &= \frac{1}{2} + \frac{1}{l(n)} (p_n - p_0) \\ &= \frac{1}{2} + \frac{\delta_n^A}{l(n)} \end{aligned}$$

The success probability of A' is polynomially related to that of A. So, if A beats G in the pseudorandom model, A' beats G in the next-bit model. ♣

9.2 One-way Functions

In this lecture, we define weak and strong one-way functions. Informally, a function f is one-way, if it's easy to compute $f(x)$ given x, but hard to compute x given $f(x)$. The notion of weak one-way requires that any probabilistic polynomial time algorithm that tries to invert f, fails for at least a significant fraction of inputs. The notion of strong one-way requires that any probabilistic polynomial time algorithm fails to invert f, except for an insignificant fraction of inputs. We make these notions precise and then show that existence of weak one-way function implies the existence of strong one-way functions.

Definition (weak one way function): A function $f : \{0,1\}^n \rightarrow \{0,1\}^{p(n)}$ is called weak one way function, if there exist an $l > 0$ such that, for all probabilistic polynomial time algorithms A, for sufficiently large n, A fails to invert f fails in, at least, $1/n^l$ fraction of the inputs. Formally,

$$Pr_{x \sim U_n} [f(A(f(x)) \neq f(x)] \geq \frac{1}{n^l}$$

Definition (strong one way function): A function $f : \{0,1\}^n \rightarrow \{0,1\}^p(n)$, is called strong one way function, if for all probabilistic polynomial time algorithm A, for all polynomials p, for sufficiently large n,

$$Pr_{x \sim U_n} [f(A(f(x)) = f(x)] \leq \frac{1}{p(n)}$$

In other words, A can invert f in only a negligible fraction of inputs.

Theorem (Yao): Strong one-way functions exist if weak one-way functions exist. **Proof:** Assume that $f : \{0,1\}^n \rightarrow \{0,1\}^{p(n)}$ is a weak one-way function. We construct a strong one-way function $F : \{0,1\}^{m \times n} \rightarrow \{0,1\}^{m \times p(n)}$. m is a parameter polynomial in n and we will fix it later. F is defined to be

$$F(x_1, x_2 \ldots x_m) =< f(x_1), f(x_2), \ldots f(x_m) >$$

The claim is that F is strongly one-way. The intuition is as follows:- Suppose an algorithm tries to invert F in the naive way, by inverting each component of its input. It will have a success probability of $\left(1 - \frac{1}{n^l}\right)$ in inverting each individual component. So it will have a success probability $\leq \left(1 - \frac{1}{n^l}\right)^m$ in inverting F. If we choose, $m = n^{l+1}$, this probability will be $\leq e^{-m}$. We next prove the claim formally.

By contradiction, assume that F is not strong one-way. Then there is an algorithm A can invert F on $\geq e^{-n} + \frac{1}{n^k}$ fraction of its inputs, for some k. We want to invert f on $> 1 - \frac{1}{n^l}$ fraction of our 2^n inputs.

Our algorithm A' to invert f is the following: Upon input $y = f(x)$, first randomly pick $i \in_R \{1, \ldots, m\}$. Then, randomly (uniformly) pick $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m \in \{0,1\}^n$. Construct,

$$Y = \langle f(x_1), \ldots, f(x_{i-1}), y, f(x_{i+1}), \ldots, f(y_m) \rangle$$

where $y = f(x)$ is in the ith position. Ask A to invert Y. Let it's output be $X' = (x'_1, x'_2, \ldots x'_m)$. If $y = f(x'_i)$, output x'_i. If A' succeeded in inverting F at Y, we would have succeeded in inverting y. We repeat the process some polynomial (to be fixed later) number of steps, to enhance our success probability. If we are unsuccessful in all these polynomial number of trials, we give up.

We now argue that our algorithm inverts f in more than $1/n^l$ fraction of its inputs. Define a bipartite graph of 2^n vertices on one side and $2^{m \cdot n}$ vertices on the other side. Strings in $\{0,1\}^n$ are the vertices in the left side L and strings in $\{0,1\}^{m \cdot n}$ are the vertices on the

right side R. Add an edge between x and $(x_1, \ldots, x_m)$ if $x = x_i$, for some i. The graph is a multigraph (meaning, there may be more than one edge between a pair of vertices). For a node x, we let $N(x)$ denote the multi-set of its neighbors. Let $G \subseteq R$ are the points at which A can invert F.

Now let's review our algorithm A'. Given $f(x)$, A' (implicitly) chooses a random element $r \in_R N(x)$, then computes $Y = F(r)$ and then asks A to find r. A' wants to find x=i^{th} component of r. If $N(x) \cap G$ is large, then with high probability Y will be in G. If $Y \in G$, then A' will invert Y successfully and hence A' will invert $f(x)$ successfully. We shall argue that for "large" fraction of x, $N(x) \cap G$ is "large".

To make the above argument formal, let's first state some simple facts. $|L| = 2^n$. $|R| = 2^{m \cdot n}$. The degree of each node in R is m. The degree of each node in L is $m2^{n(m-1)}$. Our bipartite graph is an "expander", in the following sense.

A bipartite graph has (ϵ, δ)-expansion property, if for all $S \subseteq L$, with $|S| \geq \epsilon|L|$, we have that $|N(S)| \geq (1 - \delta)|R|$. In other words, for any small set in L, its neighbor set is large. Our graph has good expansion properties.

Let $S \subseteq L$ be of size $|S| \geq \epsilon|L|$. What's the size of $N(S)$? Pick a random $Y \in R$. Let $Y = (x_1, x_2, \ldots x_m)$. What's the probability that $Y \notin N(S)$? So,

$$Pr[Y \notin N(S)] = Pr[x_1 \notin S, \ldots x_m \notin S] \leq (1 - \epsilon)^m$$

We want this probability to exponentially small. So, choose $m = n/\epsilon$. Then,

$$Pr[Y \notin S] \leq (1 - \epsilon)^m \leq e^{-n}$$

And

$$|N(S)| \geq (1 - e^{-n})|R|$$

We shall fix the parameters now. Fix:-

$$\epsilon_0 = \frac{1}{n^l}; m = \frac{n}{\epsilon_0} = n^{l+1}; \delta_0 = e^{-n}$$

Thus, our graph has (ϵ_0, δ_0)-expansion property.

We assumed that the adversary A invert F in $e^{-n} + \frac{1}{n^k}$ fraction of the inputs. Let $\alpha = \frac{1}{n^k}$. Then $|G| \geq (\delta_0 + \alpha)|R|$.

Let $S \subseteq L$, be any set with $|S| \geq \epsilon_0|L|$. As our graph has (ϵ_0, δ_0)-expansion property, $|N(S)| \geq (1 - \delta_0)|R|$. So,

$$|G \cap N(S)| = |G| - |G \cap \overline{N(S)}| \geq |G| - |\overline{N(S)}| \geq (\alpha + \delta_0)|R| - \delta_0|R| = \alpha|R|$$

So $|G \cap N(S)| \geq \alpha|R|$. The left degree of the graph is $m2^{n(m-1)}$. So, the total number of edges incident on S is $|S|m2^{n(m-1)}$. How many of these are good? Meaning, how many have the other vertex in G? From each one of these vertices in $|G \cap N(S)|$, at least one edge goes

to S. Thus, the number of good edges incident on S is at least $\alpha|R|$. What's the average number of good edges incident on S?

$$\frac{\#\text{good edges}}{\text{total \# of edges}} \geq \frac{\alpha|R|}{Sm2^{n(m-1)}} = \frac{\alpha 2^{n \cdot m}}{Sm2^{n(m-1)}} \geq \frac{\alpha}{m}\frac{2^{nm}}{2^n 2^{n(m-1)}} = \frac{\alpha}{m}$$

So, for any set S of size at least $\epsilon_0|L|$, the fraction good edges is more than $\frac{\alpha}{m}$.

Now consider the set,

$$S = \left\{ x \in L \;\middle|\; \frac{|\{(x,g) \mid g \in G\}|}{|N(x)|} < \frac{\alpha}{m} \right\}$$

For this set, the fraction of good edges is $< \frac{\alpha}{m}$. So its size should be $< \epsilon_0|L|$. So, the set

$$D = \left\{ x \in L \;\middle|\; \frac{|\{(x,g) \mid g \in G\}|}{\deg(x) = |N(x)|} \geq \frac{\alpha}{m} \right\}$$

has cardinality $|D| \geq (1-\epsilon_0)|L|$.

We are given $f(x)$ as input to invert. We randomly pick a neighbor r of x, compute $Y = F(r)$, use A to invert Y and get x. Suppose $x \in D$. Then, with probability $\geq \frac{\alpha}{m}$, r will be in G and we will be successful. If we repeat the process $\frac{m}{\alpha}$, we will be successful. The main point is that α was assumed to be $1/n^k$, a inverse polynomial. So m/α is polynomial. As $|D| \geq (1-\epsilon_0)|R|$, $Pr_x[x \in D] \geq (1-\epsilon_0) = 1 - \frac{1}{n^l}$. Thus, we will invert f with probability $\geq 1 - \frac{1}{n^l}$. This contradicts the assumption that f is $1/n^l$-hard. ♣

9.3 Goldreich-Levin Hardcore Bit

Let f be a strong one-way permutation. A hardcore predicate for f is a boolean function g such that, it's hard to compute $g(X,Y)$, given $f(X)$ and Y, Formally, for any probabilistic polynomial time algorithm, for all polynomials p, for sufficiently large n,

$$Pr_{x,y \in_R \{0,1\}^n}[A(f(x),y) = g(x,y)] \leq \frac{1}{2} + \frac{1}{p(n)}$$

We show that such predicates can be constructed for any strong one-way permutation f. The construction is due to Goldreich and Levin. Define $g(X,Y)$ to be the inner product $X \cdot Y$ of X and Y. That is, if $X = x_1 x_2 \ldots x_n$ and $Y = y_1 y_2 \ldots y_n$, then, $X \cdot Y = \sum x_i y_i$, where the summation is over mod 2.

We claim that g is a hardcore predicate for f. On the contrary, suppose there is an algorithm A that can compute $g(X,Y)$, given $f(X)$ and Y, with probability $\frac{1}{2} + \frac{1}{n^c}$. We show that, using A as a black-box, f can be inverted with high probability.

For two string $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_n$, we define $X + Y$ to be the bit-wise XOR of X and Y. That is, $X + Y = z_1 \ldots z_n$, where $z_i = x_i + y_i \pmod 2$. Define $\widehat{X} = \bar{x_1} x_2 \ldots x_n$,

the string X with first bit flipped. We have $X \cdot (Y+Z) = X \cdot Y + X \cdot Z$. So, $X \cdot Y + X \cdot \widehat{Y} = x_1$. This last property plays an important role in our construction. Our goal is to compute x_1 correctly, with high probability. Then use the same procedure to compute other bits of X, there by inverting f.

Let $m = O(\log n)$. We will fix its exact value later. Choose m strings $Y_1, Y_2, \ldots Y_m \in \{0,1\}^n$ at random. Let $\epsilon_1, \epsilon_2, \ldots, \epsilon_m$ be some fixed m bits, not all 0. Define $Y_{\epsilon_1\epsilon_2\ldots\epsilon_m} = \sum \epsilon_i Y_i$. The above one is a random variable. It will be uniformly distributed over $\{0,1\}^n$. Take any $\delta_1\delta_2\ldots\delta_m \in \{0,1\}^m - \{0\}^m$, such that $\epsilon_1\ldots\epsilon_m \neq \delta_1\ldots\delta_m$. For some i, $\epsilon_i = 1$ and $\delta_i = 0$ (or vice versa). Y_i is chosen uniformly from $\{0,1\}^m$ and it contributes only to $Y_{\epsilon_1\ldots\epsilon_m}$. So the random variables $Y_{\epsilon_1\ldots\epsilon_m}$ and $Y_{\delta_1\ldots\delta_m}$ are independent. Thus the set of $2^m - 1$ random variables $\{Y_{\epsilon_1\ldots\epsilon_m}\}$, for $\epsilon_1\ldots\epsilon_m \in \{0,1\}^m - \{0\}^m$, are pairwise independent.

Now we are ready to present our construction. We first choose $Y_1, Y_2, \ldots, Y_m \in_R \{0,1\}^n$. Then guess the values of $X \cdot Y_1, X \cdot Y_2, \ldots, X \cdot Y_m$ to be $b_1, b_2, \ldots, b_m$. Assume that we guessed correctly! Then, for any $\epsilon_1, \ldots, \epsilon_m$ we can compute $X \cdot Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}$ correctly:-

$$X \cdot Y_{\epsilon_1\epsilon_2\ldots\epsilon_m} = X \cdot (\epsilon_1 Y_1 + \cdots + \epsilon_m Y_m) = \sum \epsilon_i (X \cdot Y_i) = \sum \epsilon_i b_i$$

Fix some $\epsilon_1, \ldots, \epsilon_m$ and compute $X \cdot Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}$ as above. Then use the black-box A to compute $X \cdot \widehat{Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}}$. Assume that A computed this correctly. We can now compute the first bit x_1 of X, $x_1 = X \cdot Y_{\epsilon_1\epsilon_2\ldots\epsilon_m} + X \cdot \widehat{Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}}$. (Recall that for any Y, $x_1 = X \cdot Y + X \cdot \widehat{Y}$.) How to make the correct guesses for $b_1, \ldots, b_m$? As $m = O(\log n)$, total number of possible values for $b_1 \ldots b_m$ is polynomial in n. So we can run the above procedure for every possible sequence of guesses. And one of the guesses would be correct. To make the black-box work well with high probability, we run the above procedure over all possible (polynomial number of) $\epsilon_1, \ldots \epsilon_m$. Then, we take the majority. We would get x_1 this way. Similarly, compute other bits. Using Chebyshev inequality, we shall show that with high probability (over the random choices for $Y_1, Y_2, \ldots, Y_m$), we would have got the correct $x_1, x_2, \ldots, x_m$. We can verify whether we are correct by checking $f(x_1 x_2 \ldots x_m) = f(X)$. If we are not correct, we run the experiment again, by choosing at random another set of $Y_1, Y_2, \ldots, Y_m$.

We next present the algorithm in full. The input is $f(X)$. It tries to compute X.

```
Choose Y_1, Y_2, ..., Y_m ∈_R {0,1}^n.
For all 2^m choices of b_1 ... b_m ∈ {0,1}^m do:-
    For 1 ≤ k ≤ n do:-
        For each ε_1 ... ε_m ∈ {0,1}^m − {0}^m do:-
            Use A to compute the the vote
                V_{ε_1,...,ε_m} = Σ_{i=1}^m ε_i b_i + A[f(X), Y_{ε_1 ε_2 ... ε_m} + 0^{k−1} 10^{n−k}]
        Set x_k to be the majority of the above 2^m − 1 votes.
    If f(x_1, ..., x_k) = f(X), output x_1 ... x_k and EXIT.
```

The above procedure runs in polynomial time, as $m = O(\log n)$. In the above procedure, as we run through all possible $b_1, \ldots, b_m$, in some iteration we would achieve $\forall i[b_i = X \cdot Y_i]$. Consider that particular iteration. What's the chance that we would get x_1 correctly? This

depends upon the what's the majority of the votes. Each $\epsilon_1, \ldots, \epsilon_m$ casts a vote for the value of x_1. The vote $V_{\epsilon_1,\ldots,\epsilon_m}$ will be correct, if A succeeds in computing $X \cdot \widehat{Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}}$ correctly. We already observed that the $2^m - 1$ random variables $Y_{\epsilon_1\epsilon_2\ldots\epsilon_m}$ are uniformly distributed over $\{0,1\}^n$ and they are pairwise independent. So, by our assumption that A can compute the inner product with probability $\geq \frac{1}{2} + \frac{1}{n^c}$, each vote will be correct with at least that much probability. We now apply Chebyshev over these $2^m - 1$ votes. Let χ be a random variable that counts the number of correct votes among these $2^m - 1$ votes. χ is a sum of $2^m - 1$ pairwise independent 0-1 random variables. Each of these individual random variable has a success probability, at least, $\frac{1}{2} + \frac{1}{n^c}$. Expectation of χ is

$$E[\chi] \geq (2^m - 1)\left(\frac{1}{2} + \frac{1}{n^c}\right)$$

Due to pairwise independence, var χ is the sum of variances of the individual 0-1 random variables. The variance of any 0-1 random variable is at most 1. So, var $\chi \leq 2^m - 1$. By Chebyshev,

$$\begin{aligned}
Pr[\text{ majority of } 2^m - 1 \text{ votes wrong }] &= Pr\left[\chi < \frac{2^m - 1}{2}\right] \\
&\leq Pr\left[|\chi - E[\chi]| > \frac{2^m - 1}{n^c}\right] \\
&\leq \frac{\text{var } \chi}{\left(\frac{1}{n^c}(2^m - 1)\right)^2} \\
&= \frac{(2^m - 1)n^{2c}}{(2^m - 1)^2} \\
&= \frac{n^{2c}}{2^m - 1}
\end{aligned}$$

Take $m \sim (2c \log n + 2)$. Then the probability that x_1 is not correct is at most $\frac{1}{n^2}$. The probability that any one of $x_1, x_2, \ldots, x_n$ is not correct is at most $\frac{1}{n}$. By repeating the above experiment, we can improve our success probability.

9.4 Construction of Pseudorandom Generators

Show how to construct PRG using gl-hardcore bit.