

Goldreich-Levin Hardcore Bit Theorem: Let f be a one way permutation. Define a function $b : \sum^{2^n} \rightarrow \sum$, called as hard-core bit as

$$b(x, r) = \langle x, r \rangle = \sum x_i r_i \pmod{2}.$$

If there is a circuit C such that

$$\Pr_{x, r \in U_n} [C[f(x), r] = \langle x, r \rangle] \geq \frac{1}{2} + \epsilon$$

then there exist circuit D of size

$$\text{poly}\left(|C|, \frac{1}{\epsilon}\right)$$

such that

$$\Pr_{x \in U_n} [D[f(x)] = x] \geq \frac{1}{2\epsilon}$$

Proof: Let f be a one way permutation.

First consider the following simple case. Assume that for every x , the following holds:

$$\Pr_r [C[f(x), r] = \langle x, r \rangle] = 1$$

Define e_i as string with 1 in i^{th} place and zero's all other places.

Observe that $\langle x, e_i \rangle = x_i$. Thus computing $\langle x, e_i \rangle$, $1 \leq i \leq n$, gives x . Since computes $\langle x, r \rangle$ correctly for every r , we can easily compute x .

Now let us consider the following case:

$$\forall x, \Pr_r [C[f(x), r] = \langle x, r \rangle] \geq \frac{3}{4} + \delta$$

Consider the following algorithm to compute f^{-1} .

Computing $f^{-1}(x)$

1. Input $f(x)$ [Goal: Compute x]
2. for $i = 1$ to n Do
 - (a) $j = 1$. Repeat following for m times
 - Randomly pick r
 - $b_j = C[f(x), r] \oplus C[f(x), r + e_i]$
 - $j++$

(b) $y_i = \text{Maj}\{b_1 \cdots b_m\}$.

3. Output $y_1 \dots y_n$

Observe that

$$x_i = \langle x, r \rangle \oplus \langle x, r + e_i \rangle$$

Let us compute the probability that $y_i = x_i$. Observe that when $C[\langle f(x), r \rangle]$ and $C[\langle f(x), r + e_i \rangle]$ are correct, then $C[\langle f(x), r \rangle] \oplus C[\langle f(x), r + e_i \rangle]$ equals x_i .

On a random r , the circuit computes C computes the value of $\langle x, r \rangle$ with probability at least $3/4 + \delta$. Thus the probability the C is wrong on of $\langle f(x), r \rangle$ and $\langle f(x), r + e_i \rangle$ is at most $1/2 - 2\delta$.

Thus $b_j = x_i$ with probability at least $1/2 + 2\delta$. Recall that $y_i = \text{Maj}\{b_j\}$ and b_j 's are independent, we obtain that $y_i = x_i$ with probability roughly $1 - 2e^{-\delta^2 m}$ (By Chernoff Bound).

Thus probability that $y \neq x$, is at most $2ne^{-\delta^2 m}$, by Union bound. If we set $m = O(n^2/\delta^2)$, this probability is at most $1/n$.

Now we move to the (almost) desired case:

Let us assume that for every x

$$\Pr_r[C(f(x), r) = \langle x, r \rangle] \geq \frac{1}{2} + \epsilon$$

Observe that this assumption is still stronger than our hypothesis. Here we are assuming that C computes $\langle x, r \rangle$ with probability $1/2 + \epsilon$ for every x . For now, we will proceed with this assumption.

Note that in the analysis of the earlier algorithm, it was crucial that the success probability of C is strictly bigger than $3/4$. This is because the strings r and $r + e_i$ are not independent and to obtain the probability that C is correct on both of them we used union bound.

Suppose, for a moment assume that we could compute the value of $\langle x, r \rangle$ without invoking C , and to compute $\langle x, r + e_i \rangle$, we invoke C . Then we do not need to use the union bound, and even if the success probability of C is $1/2 + \epsilon$, we are in a good shape.

However, how can we compute $\langle x, r \rangle$ without invoking C ? The answer is try all possible choices. For each possible choice, run the above algorithm to compute y . Observe that once we have a candidate y , checking y is really an inverse of $f(x)$ is easy. The following algorithm use this idea.

1. Input y . Randomly pick $r_1 \cdots r_m$. For every m bit string $b_1 \cdots b_n$. Do the following:

- (a) Assume $b_j = \langle x, r_j \rangle$
- (b) For $i = 1$ to m Do
- (c) $t_i = \text{Maj}(C[f(x), r_j \oplus e_i] \oplus b_j)$
- (d) end for

(e) If $f(t_1 \cdots t_n) = y$, output $t_1 \cdots t_n$.

Let us analyze the probability that the above algorithm computes i^{th} bit of $f^{-1}(y)$ correctly.

There exist an iteration, when $b_1 \cdots b_m = \langle x, r_1 \rangle \cdots \langle x, r_m \rangle$. Consider this iteration. Let $z_j = 1$ if $C[y, r_j \oplus e_i] = \langle f^{-1}(y), r_j \oplus e_i \rangle$, else $z_j = 0$.

$$Pr[z_j = 1] \geq \frac{1}{2} + \epsilon$$

$$\begin{aligned} Pr[t_i \neq f^{-1}(y)_i] &= Pr[\text{Majority of } z_j \text{ 's are not } f^{-1}(y)_i] \\ &= Pr[\text{Majority of } z_j \text{'s are 0}] \\ &= Pr\left[\sum_{j=1}^m z_j < \frac{m}{2}\right] \\ &\leq 2e^{-\epsilon^2 m} \text{ (By Chernoff Bound)} \end{aligned}$$

Thus the probability that the above algorithm does not correctly compute i^{th} bit is at most $2e^{-\epsilon^2 m}$. By union bound, the probability that the above algorithm computes all bits of $f^{-1}(y)$ is at least $1 - 2e^{-\epsilon^2 m n}$. To make this probability non-negligible, we want $m = O\left(\frac{\log n}{\epsilon^2}\right)$. This implies that the running time of the above algorithm (hence the circuit size) at least $O\left(2^{\frac{1}{\epsilon}}\right)$. However, we want our algorithm to run in $poly\left(\frac{1}{\epsilon}\right)$. For this we have to reduce the number of random strings that are picked.

Observe that even if we use Chebyshev bound, the above analysis still works, the only difference is that we require $m \cong \frac{n^2}{\epsilon^2}$. Recall that even if the random variables z_j 's are pairwise independent, the Chebyshev bound is still applicable. In the above algorithm the z_j 's are completely independent as r_j 's are completely independent. If we make r_j 's to be pair-wise independent, then z_j 's will be pairwise independent. Since we only require pair-wise independence, we can generate the random strings using less randomness. The next algorithm precisely does this.

1. Input y .
2. Randomly pick $r_1 \cdots r_{\log m + 1}$
3. For every $\log m + 1$ bit string $b_1, \dots, b_{\log m + 1}$ Do
 - (a) Assume $\langle x, r_j \rangle = b_j$
 - (b) For every subset $S \subseteq \{1 \cdots \log m + 1\}$, let $R_S = \oplus_{i \in S} r_i$. Let $b_S = \oplus_{j \in S} b_j$.
 - (c) Set $t_i = \text{Maj}_S[C[y, R_S + e_i] \oplus b_S]$ for $1 \leq i \leq n$.
 - (d) If $f(t_1 \cdots t_n) = y$, output $t_1 \cdots t_n$.

For each $S \in \{1, \dots, \log m + 1\}$, define a random variable z_S as follows: $z_S = 1$ if $C(y, R_S + e_i) = \langle f^{-1}(y), R_S + e_i \rangle$. Now observe that z_S 's are pairwise independent random variables. Now using Chebyshev's inequality, we can show that the above algorithm inverts f with good probability. The running time is now $O(2^{\log m}) = poly\left(\frac{1}{\epsilon}\right)$.