CS60084

Foundations of Cryptography

Hard Core predicates

Akshit Sharma(05CS3020) **Naresh Shenoy(05CS3015)**

Hard Core Predicates

Hard Core Predicate - A polynomial-time predicate b is called a hard-core of a function f if every efficient algorithm, given $f(x)$, can guess $b(x)$ with success probability that is only negligibly better than one-half.

Formally speaking, we define a hard-core predicate as follows:-

A polynomial-time-computable predicate $b : \{0,1\}^* \rightarrow \{0,1\}$ *is called a hard-core of a function* f *if for every probabilistic polynomial-time algorithm* A*, every positive polynomial* $p(.)$*, and all sufficiently large* n*,*

$$Pr[A(f(U_n)) = b(U_n)] < 1/2 + 1/p(n)$$

For example, the predicate $b(\sigma\alpha) = \sigma$ is a hard-core of the function $f(\sigma\alpha) = 0\alpha$, where $\sigma \in \{0,1\}$ and $\alpha \in \{0,1\}^*$. Hence, in this case the fact that b is a hard-core of the function f is due to the fact that f loses information (specifically, the first bit σ). On the other hand, in case f loses no information (i.e., f is one-to-one), hard-cores for f exist only if f is one-way.

Hard-core predicates for collections of one-way functions - They are also defined in an analogous way as follows:

A polynomial-time algorithm $B : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ *is called a hard-core of the one-way collection (I, D, F) if for every probabilistic polynomial-time algorithm* A'*, every positive polynomial* $p()$*, and all sufficiently large n's,*

$$Pr[A'(I_n, f_{I_n}, (X_n)) = B(I_n, X_n)] < 1/2 + 1/p(n)$$

where $I_n = I(1_n)$ *and* $X_n = D(I_n)$*.*

For example, the least significant bit is a hard-core for the RSA collection, provided that the RSA collection is one-way. Namely, assuming that the RSA collection is one-way, it is infeasible to guess (with success probability significantly greater than 1/2) the least significant bit of x from $RSA_{N,e}(x) = x^e mod N$.

Goldreich and Levin Theorem

Goldreich and Levin Theorem - Oded Goldreich and Leonid Levin (1989) showed how every one-way function can be trivially modified to obtain a one-way function that has a specific hard-core predicate. G-L theorem basically states that if there is a family of trapdoor permutations, then there is a family with a hard core predicate. The theorem is formally stated below:-

Let f be an arbitrary strong one-way function, and let f' be defined by $f'(x, r) = (f(x), r)$, where $|x| = |r|$. Let $b(x, r)$ denote the inner product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function f'.

Note that the theorem requires that f be strongly one-way and that the conclusion is false if f is only weakly one-way. We shall now prove the G-L theorem.

General Outline of all Proofs

The G-L Theorem says that the probability of computing $b(x, r)$ from $f'(X, r) = (f(x), r)$ should be more than $\frac{1}{2}$ by only a negligible quantity. We shall prove G-L theorem by contradiction. Thus we assume that b is not the hard core predicate of f'. This means that there exists a probabilistic polynomial time algorithm A that computes $b(x, r)$ from $f'(x, r)$ with a probability that is significantly greater than $\frac{1}{2}$.

Now how much this probability is greater than $\frac{1}{2}$ is left to our will. Obviously if we assume this probability is 1, the proof will be less involved but a weak proof. Then we will assume that this probability is negligibly more than $\frac{3}{4}$ to yield a more involved but a stronger proof. Finally, we assume the general case when this probability is slightly more than the $\frac{1}{2}$ to yield the proper proof.

After assuming that b is not the hardcore predicate of f', we shall show that easy to compute x from $f(x)$. This contradicts our assumption that f is one-way function hence contradicting the fact that a polynomial time algorithm A exists that can compute $b(x, r)$ from $f'(x, r)$ with probability significantly greater than $\frac{1}{2}$. This will prove that b is indeed a hardcore predicate of f'.

Following this chain of thought, we start with the simplest and least involved proof.

A small but weak proof

As a first proof, we take a weak case just to give a brief outline of how the actual proof proceeds. We assume that there exists a probabilistic polynomial time algorithm A that computes $b(x, r)$ from $f'(x, r)$ with a probability of 1. That is, we assume that there is a polynomial time algorithm A, that always correctly computes $b(x, r)$ given $f'(x, r) = (f(x), r)$. Now we shall show that easy to compute x from $f(x)$.

Let A be a PPT algorithm which computes the value of $b(x, r)$ from $f'(x, r) = (f(x), r)$ that is,

$$Pr_{\{x,r\} \to \{0,1\}^n}[A(f(x), r) = b(x, r)] = 1$$

Now we shall frame an experiment A', which invokes A for $i = 1, 2, ...n$. The arguments being passed to A are $f(x)$ and e_i where e_i denotes a string with the ith bit 1 and rest 0. Now, for each i, $A(f(x), e_i)$ will return $b(x, e_i)$ (because it does so with probability 1). But $b(x, e_i) = x_i$, that is the ith bit of x. Thus by running A for $i = 1, 2, ...n$, we can retrieve the entire x by executing A, n number of times. Since A is itself a polynomial time algorithm, A' is also a PTT algorithm. Thus we are able to extract x from $f(x)$ in polynomial time thus violating the fact that f is a strong one way function.
This contradicts our assumption that f is one-way function hence contradicting the fact that a polynomial time algorithm A exists that can compute $b(x, r)$ from $f'(x, r)$ with probability 1 (this is a weak case and not the real proof, it is just to give an idea of how the real proof will proceed). This proves that b is indeed a hardcore predicate of f'.

It should again be noted that we assumed the fact that A computes $b(x, r)$ from $f'(x, r)$ with probability of 1. This is not exactly the negation of the fact that b is a hardcore predicate of f'. We now present a more involved proof which deals with a more involved case.

A more involved but stronger proof

In this more involved proof, we will assume that there exists a probabilistic polynomial time algorithm A that computes $b(x, r)$ from $f'(x, r)$ with a probability that is negligibly greater than $\frac{3}{4}$. This is still a weak case compared to the real proof where we shall assume that it is slightly more than $\frac{1}{2}$. Thus here we assume that,

$$Pr_{\{x,r\} \to \{0,1\}^n}[A(f(x), r) = b(x, r)] \geq \frac{3}{4} + \in (n)$$

Before we proceed we will prove a few things.

Sub Proof 1: $b(x, r) \oplus b(x, r \oplus e_i) = x_i$
In the last proof, we observed that $b(x, e_i) = x_i$ since inner product of x with a

vector whose all bits are 0 except ith bit will give us the ith bit of x. This was important in obtaining x from $f(x)$ bit by bit.

An important property of the hard core predicate b is shown below:-

$$b(x,u) \oplus b(x,v) = b(x,u \oplus v)$$

It should be noted that $b(x,r) \oplus b(x,r \oplus e_i) = b(x, r \oplus (r \oplus e_i)) = b(x,(r \oplus r)e_i) = b(x,e_i) = x_i$. That is,

$$b(x,r) \oplus b(x,r \oplus e_i) = x_i$$

Sub Proof 2:

If $Pr_{\{x,r\} \to \{0,1\}^n}[A(f(x),r) = b(x,r)] \geq \frac{3}{4} + \in (n)$, then there exists a set $S_n \subseteq \{0,1\}^n$ of size at least $(\frac{\in(n)}{2}).2^n$, where for every $x \in S_n$,

$$Pr_{r \to \{0,1\}^n}[A(f(x),r) = b(x,r)] \geq \frac{3}{4} + \frac{\in(n)}{2}$$

The proof is given below.

$$\begin{aligned}
&Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r)] \\
&= Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r) \mid x \in S_n] Pr_{x \to \{0,1\}^n}[x \in S_n] + \\
&\quad Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r) \mid x \notin S_n] Pr_{x \to \{0,1\}^n}[x \in S_n] \\
&\leq Pr_{x \to \{0,1\}^n}[x \in S_n] + Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r) | x \in S_n]
\end{aligned}$$

$\therefore Pr_{x \to \{0,1\}^n}[x \in S_n] \geq \ Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r)] - Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r)]$

i.e. $Pr_{x \to \{0,1\}^n}[x \in S_n] \geq (\frac{3}{4} + \epsilon(n))(\frac{3}{4} + \epsilon(n)/2)$
i.e. $Pr_{x \to \{0,1\}^n}[x \in S_n] \geq \epsilon(n)/2$

Since the total size of the set $\{0,1\}^n$ is 2n, the size of the set S_n is $(\epsilon(n)/2).2n$ (in order to make the probability of selection of x from S_n equal to $\epsilon(n)/2$)

This completes the proof.

Sub Proof 3:

If $Pr_{x,r \to \{0,1\}^n}[A(f(x),r) = b(x,r)] \geq \frac{3}{4} + \epsilon(n)$ *, then there exists a set* $S_n \subseteq 0,1^n$ *of size at least* $(\epsilon(n)/2).2n$*, where for every* $x \in S_n$ *and every* i*, it holds that:*

$$Pr_{r \to \{0,1\}^n}[A(f(x),r) = b(x,r) \wedge A(f(x), r \oplus e_i) = b(x, r \oplus e_i] \geq \frac{1}{2} + \epsilon(n)$$

The proof is given below.

For every $x \in S_n$, $Pr_{r \to \{0,1\}^n}[A(f(x), r) \neq b(x, r)] < 1 - (\frac{3}{4} + \epsilon(n)/2)$. That is,

$$Pr_{r \to \{0,1\}^n}[A(f(x), r) \neq b(x, r)] < \tfrac{1}{4} - \epsilon(n)/2$$

For a fixed i, if r is uniformly distributed, then so is $r \oplus e_i$. Hence the similar result follows for $r \oplus e_i$ as well. That is,

$$Pr_{r \to \{0,1\}^n}[A(f(x), r) \neq b(x, r \oplus e_i)] < \tfrac{1}{4} - \epsilon(n)/2$$

The probability that either one of the two predicates are wrongly computed is given by the sum of the above two probabilities, since these are two independent events. That is,

$$Pr_{r \to \{0,1\}^n}[A(f(x), r) \neq b(x, r) \vee A(f(x), r \oplus e_i) \neq b(x, r \oplus e_i)] < 2(1/4 - \epsilon(n)/2)$$

i.e. $Pr_{r \to \{0,1\}^n}[A(f(x), r) \neq b(x, r) \vee A(f(x), r \oplus e_i) \neq b(x, r \oplus e_i)] < 1/2 - \epsilon(n)$

Thus,

$$Pr_{r \to \{0,1\}^n}[A(f(x), r) = b(x, r) \wedge A(f(x), r \oplus e_i) = b(x, r \oplus e_i)] \geq 1 - (1/2 + \epsilon(n))$$

i.e. $Pr_{r \to \{0,1\}^n}[A(f(x), r) = b(x, r) \wedge A(f(x), r \oplus e_i) = b(x, r \oplus e_i)] \geq 1/2 - \epsilon(n)$

This completes the third sub proof.

We now proceed back to the more involved proof. Remember, we already have a probabilistic polynomial time algorithm A that computes b(x,r) from f'(x,r) with a probability that is negligibly greater than $\frac{3}{4}$. That is,

$$Pr_{x, r \to \{0,1\}^n}[A(f(x), r) = b(x, r)] \geq \tfrac{3}{4} + \epsilon(n)$$

We now construct an algorithm A' which does the following for i = 1, 2,... n:

1) Choose a random $r \to \{0,1\}^n$ and guess that the value $x_i = A(f(x), r) \oplus A(f(x), r \oplus e_i)$.

2) Repeat the procedure for a large number of cases and return the majority as correct guess.

Since A' calls A only poly(n) number of times, and A is a polynomial time algorithm, it follows that A' is also a polynomial time algorithm.

Also note, we have already proved in sub proof (3) that on choosing a random string $r \to \{0,1\}^n$, the probability that both A(f(x), r) = b(x, r) and

$A(f(x), r \oplus e_i) = b(x, r \oplus e_i)$ is more than 1/2. Thus if we repeat the procedure many times and return the majority, we can be assured that A(f(x), r) = b(x, r) and $A(f(x), r \oplus e_i) = b(x, r \oplus e_i)$.

Also it was shown in sub proof (1) that $b(x, r) \oplus b(x, r \oplus e_i) = x_i$. Hence we can be sure that the PTT algorithm A' is able to return x in polynomial time.

This contradicts our assumption that f is one-way function hence contradicting the fact that a polynomial time algorithm A exists that can compute b(x,r) from f'(x,r) with probability more than $\frac{3}{4}$. This proves that b is indeed a hardcore predicate of f'.

We again note that we assumed the fact that A computes b(x,r) from f'(x,r) with probability more than $\frac{3}{4}$. This is not exactly the negation of the fact that b is a hardcore predicate of f'. We now present the proper proof.

The Real proof of G-L Theorem

The problem with the foregoing procedure is that it doubles the original error probability of algorithm A on inputs of the form (f (x),).What is required is an alternative way of using the algorithm A, a way that does not double the original error probability of A.

The key idea is to generate the r 's in a way that requires applying algorithm A only once per each r (and i), instead of twice. Specifically, we shall use A to obtain a "guess" for $b(x, r \oplus e_i)$ and obtain b(x, r) in a different way. The good news is that the error probability is no longer doubled, since we use A only to get a "guess" of $b(x, r \oplus e_i)$. The bad news is that we still need to know b(x, r), and it is not clear how we can know b(x, r) without applying A. The answer is that we can guess b(x, r) by ourselves. This is fine if we need to guess b(x, r) for only one r, but the problem is that we need to know (and hence guess) the values of b(x, r) for polynomially many r's. We generate these polynomially many r's such that, on one hand, they are "sufficiently random," whereas, on the other hand, we can guess all the b(x, r)'s with noticeable success probability. Specifically, generating the r's in a particular pairwise-independent manner will satisfy both requirements. We stress that in case we are successful (in our guesses for all the b(x, r)'s), we can retrieve x with high probability. Hence, we retrieve x with noticeable probability.

Before we proceed we set m=poly(n) and set $l = log_2(m + 1)$.

We then select $l = log_2(m + 1)$ strings in $\{0, 1\}^n$ and denote them by $s_1, s_2 ..., s_l$.

We then guess $b(x, s_1)$ through $b(x, s_l)$. Let us denote these guesses, which are uniformly (and independently) chosen in $\{0, 1\}$, by σ_1 through σ_l . Hence, the probability that all our guesses for the $b(x, s_i)$'s are correct is $2^{-l} = 1/poly(n)$.

Now we proceed with selection of r's. The different r's correspond to the different non-empty subsets of 1, 2, ... , l denoted by J. Specifically, we let $r^J = \oplus_{j \in J} s_j$. It is evident that the r^J's are pairwise independent, and each is uniformly distributed in $\{0, 1\}^n$. The key observation is that

$$b(x, r^J) = b(x, \oplus_{j \in J} s_j) = \oplus_{j \in J} b(x, s_j)$$

Hence, our guess for the $b(x, r^J)$'s is $\oplus_{j \in J} \sigma_j$, and with noticeable probability all our guesses are correct. Let us denote our guess for $b(x, r^J)$by ρ^J.

$$\rho^J = \oplus_{j \in J} \sigma_j$$

We now construct the PPT algorithm A' that will invert f(x). The algorithm A' is described below:

1. It uniformly and independently selects $s_1, s_2 ..., s_l \in \{0, 1\}^n$ and $\sigma_1, ..., \sigma_l \in \{0, 1\}$.
2. For every non-empty set $J \subseteq \{1, 2, .., l\}$, it computes a string $r^J \leftarrow \oplus_{j \in J} s_j$ and a bit $\rho^J \leftarrow \oplus_{j \in J} \sigma_j$.
3. For every $i \in 1, ..., n$ and every non-empty $J \subseteq 1, ..., l$, it computes,

$$x^J{}_i \leftarrow \rho^J \oplus A(f(x), r^J \oplus e_i)$$

4. For every $i \in 1, ..., n$, it sets x_i to be the majority of the $x^J{}_i$ values.
5. It outputs $x = x_1 ... x_n$.

Now to prove that A' is indeed able to invert f(x) in polynomial time, we need to provide a proof to a sub proof as below.

Chebyshev's Inequality

Let X be a random variable and let $\delta > 0$. Then,

$$Pr[|X - E(X)| \geq \delta] \leq \frac{Var(X)}{\delta^2}$$

Where, E(X) is the expected value of X and Var(X) is the variance of the random variable X.

Sub Proof 4:

For every $x \in S_n$ and every $1 \leq i \leq n$,

$$Pr[|J : b(x, r^J) \oplus A(f(x), r^J \oplus e_i) = x_i| > \tfrac{1}{2}.(2^l - 1)] > 1 - 1/(2n)$$

Where, $r^J = \oplus_{j \in J} s_j$ and the s_j 's are independently and uniformly chosen in $\{0, 1\}^n$.

The proof of this is given below. For every J, define a 0-1 random variable ζ^J such that ζ^J equals 1 if and only if,

$$b(x, r^J) \oplus A(f(x), r^J \oplus e_i) = x_i$$

Since $b(x, r^J) \oplus b(x, r^J \oplus e_i) = x_i$, it follows that $\zeta^J = 1$ if and only if,

$$A(f(x), r^J \oplus e_i) = b(x, r^J \oplus e_i)$$

Now, since $x \in S_n$, it follows that the probability of $\zeta^J = 1$ is at least $1/2 + \epsilon(n)/2$. Thus the expected value of the random variable $\Sigma_J(\zeta^J)$ is given by,

$$E(\Sigma_J(\zeta^J)) = (2^l 1).(1/2 + \epsilon(n)/2) = (1/2 + \epsilon(n)/2).m$$

(because $l = log_2(m + 1)$, so $(2^l - 1)$ can be replaced by m)

Also on replacing $(2^l - 1)$ by m in the proof, we see that we need to evaluate $P[\Sigma_J(\zeta^J) \leq m/2]$.

$$\begin{aligned} P[\Sigma_J(\zeta^J) \leq m/2] &\leq P[|\Sigma_J(\zeta^J)(1/2 + \epsilon(n)/2).m| \geq \epsilon(n).m/2] \\ &\leq P[|\Sigma_J(\zeta^J) - E(\Sigma_J(\zeta^J))| \geq \epsilon(n).m/2] \end{aligned} \quad (1)$$

Comparing this with the Chebyshev's Inequality, we see that,

$$P[|\Sigma_J(\zeta^J)E(\Sigma_J(\zeta^J))| \geq \epsilon(n).m/2] \leq Var(\Sigma_J(\zeta^J))/(\epsilon(n).m/2)^2 \quad (2)$$

Hence, combining (1) and (2), we get,

$$P[\Sigma_J(\zeta^J) \leq m/2] \leq Var(\Sigma_J(\zeta^J))/(\epsilon(n).m/2)^2 \quad (3)$$

Now we compute $Var(\Sigma_J(\zeta^J))$ as follows,

$$Var(\Sigma_J(\zeta^J)) = m.(1/2 + \epsilon(n)/2).(1/2 - \epsilon(n)/2) < m/4 \quad (4)$$

Thus, combining (3) and (4), we have,

$$P[\Sigma_J(\zeta^J) \leq m/2] < (m/4)/(\epsilon(n).m/2)^2 = 1/(\epsilon^2(n).m) \quad (5)$$

Since we have taken m to be a polynomial in n. Let us assume,

$$m = 2npoly^2(n)$$

Or equivalently,

$$m = 2n/\epsilon^2(n) \quad (6)$$

Combining (5) and (6), we get finally,

$$P[\Sigma_J(\zeta^J) \leq m/2] < 1/(2n)$$

Hence we get,

$$P[\Sigma_J(\zeta^J) > m/2] \geq 1 - 1/(2n)$$

This completes the sub proof 4. We will now proceed with the actual and formal proof of the G-L theorem.

Sub proof 4 basically states that the probability that algorithm A' returns the correct value of x_i (that is, it returns the correct value of x_i for the majority of the ${x^J}_i$ values) is at least 1-1/(2n).

Hence the probability that A' makes an error in a particular x_i is at most 1/(2n). Now, A' will make an error in inverting f(x) if it makes an error in any of the n bits. Thus the probability that A' returns a wrong result for any of the n bits is at most n.(1/(2n)) = 1/2. Hence, probability that A' runs correctly for all n bits is at least 1/2.

Now, we have assumed till now that the initial l guesses for $\sigma 1, ..., \sigma l$ are correct. This is because we A' computes $\rho^J \oplus A(f(x), r^J \oplus e_i)$ assuming that $\rho^J \leftarrow \oplus_{j\in J}\sigma_j$ is the correct estimate for $b(f(x), r^J)$, where $r^J \leftarrow \oplus_{j\in J}s_j$. This in turn means that each of the $\sigma_1, ..., \sigma_l$ is a correct estimate for $b(x, s_1)...b(x, s_l)$. The probability of each of the randomly chosen $\sigma_1, ..., \sigma_l$ to be a correct estimate for $b(x, s_1)....b(x, s_l)$ is 2^{-l}. Thus for every $x \in S_n$, the algorithm A' is able to invert f(x) with a probability of

$$\begin{aligned}
&= \frac{1}{2}.2^{-l} \\
&= \frac{1}{2}.1/(m+1) \\
&= \frac{1}{2}.1/(2.n.p^2(n)+1)
\end{aligned}$$

(from(6))

Also, it is known that $Pr_x[x \in S_n] = \epsilon(n)/2$. Thus the probability that A' is able to invert f(x) is given by the following,

$$\begin{aligned}
&= \frac{1}{2}.1/(2.n.p^2(n)+1).\epsilon(n)/2 \\
&= \frac{1}{2}.1/(2.n.p^2(n)+1).1/(2.p(n)) \\
&= \frac{1}{4}.1/(2.n.p^3(n)+p(n))
\end{aligned}$$

Also, A' makes a polynomial number of calls to A, which is a PPT algorithm. Thus we can say that A' is able to invert f(x) in polynomial time and returns x with the probability computed above. Once again as before, this contradicts our assumption that f is one-way function hence contradicting the fact that a polynomial time algorithm A exists that can compute b(x,r) from f'(x,r) with probability more than 1/2. This proves that b is indeed a hardcore predicate of f'.

Note that this is a strong proof as it starts with

$$Pr_{x,r\rightarrow\{0,1\}^n}[A(f(x), r) = b(x, r)] < 1/2 + \epsilon(n)/2$$

And hence to contradict, it assumed the negative to be true, that is,

$$Pr_{x,r\to\{0,1\}^n}[A(f(x),r)=b(x,r)]\geq 1/2+\epsilon(n)/2$$

which was used for proving sub proof 4.

Hence this is a strong proof of the G-L theorem showing that b (as constructed in the theorem) is indeed the Hard Core predicate of f'(x, r) = (f(x), r), where f(x) is a strong one way function.