Reading Project: Goldreich-Levin Theorem
Cryptography and Network Security

P. Hemanth Naidu

28 April 2023

0.1 Hardcore Predicate

- A hard core predicate for a OWF f is a function over its inputs x. The output of the function is a single bit (hardcore bit). The output bit can be easily computed given x. But the output bit is hard to compute given f(x).
- If it were easy to compute the hardcore bit from f(x), then an attacker who knows f(x) could learn something about x without having to invert the one-way function, which would weaken the security of the function.
- The reason this bit is called hardcore bit is because it is guaranteed hard to compute information about x. Learning the hardcore bit of x given the f(x) is as hard as inverting the function f(x) and learning x.

0.1.1 Definition

A predicate $h : \{0,1\}^* \rightarrow \{0,1\}$ is a hard-core predicate for f if h is efficiently computable given x and there exists a negligible function ν such that for every non-uniform PPT adversary A and $\forall n \in N$:

$$\Pr[x \leftarrow \{0,1\}^n A(1^n, f(x)) = h(x)] \leq \frac{1}{2} + \nu(n)$$

0.2 Goldreich-Levin Theorm

let f be a one way function, define function

$$g(x,r) = (f(x), r)$$

where $|x| = |r|$. Then g is one way function and

$$h(x,r) = \langle x, r \rangle$$

is a hard-core predicate for g.

0.3 Proof by reduction

We will show that if a non-uniform probabilistic polynomial-time (PPT) adversary A, given $(f(x), r)$, can compute $h(x,r)$ with significantly better probability than 1/2, then there exists a non-uniform PPT adversary B that inverts $f(x)$.

0.3.1 Main Challenge

Adversary A of the hard core predicate function h outputs only 1-bit. But, for the purpose of this proof, we need to build an inverting function B for OWF f that outputs all n-bits of the input x.

0.3.2 Warmup Proof 1

Assumption 1

Given one-way function (OWF) (or one-way permutation (OWP)) $g(x) = (f(x), r)$, adversary A always outputs $h(x, r)$ correctly with probability 1.

Building Inverter B

Since adversary A always computes $h(x, r)$ correctly, we can construct $(f(x), r)$ such that r has its ith bit set to 1 and all other bits are set to 0. Thus, we obtain the bit x_i as below.

Proof

$$\text{Compute: } x_i^* \leftarrow A(f(x), e_i) \text{ for every } i \in [n], \text{ where}$$

$$e_i = (0, ..., 0, 1, 0, ..., 0,)$$

$$\text{Output: } x^* = (x_1^{*} x_2^{*} ..., x_n^*)$$

0.3.3 Warmup Proof 2

Assumption 1

Given one-way function (OWF) (or one-way permutation (OWP)) $g(x) = (f(x), r)$, adversary A outputs $h(x, r)$ with probability $3/4 + \epsilon(n)$.

0.3.4 Main Challenge

Adversary may detect and ignore improper inputs - One example for improper input could be e_i in the previous case

Building Inverter B

Here, we split each query into two queries such that each query looks random individually thus not giving the attacker any opportunity to identify it as an improper input.

1. Let the random queries be $a \leftarrow A(f(x), e_i \oplus r)$ and $b \leftarrow A(f(x), r)$ for $r \leftarrow \{0, 1\}^n$.

2. Compute $c \leftarrow a \oplus b$ as a guess for x_i^*.

3. Repeat step 2 many times to get value of c agreed by majority for x_i

4. Output: $x^* = (x_1^{*} x_2^{*} ..., x_n^*)$

0.3.5 Proof

1. If both a and b are correct, then c = x_i because,

$$\begin{aligned} c &= a \oplus b \\ &= \langle x, e_i \oplus r_i \rangle \oplus \langle x, r_i \rangle \\ &= x \cdot (r + e_i) + x \cdot r \bmod 2 \\ &= x \cdot e_i \\ &= x_i \end{aligned}$$

2. **Claim:** $c = x_i$ with probability $1/2 + 2\epsilon$.
   By union bound, the probability for A being wrong about either a or b is at most:

$$\begin{aligned} \text{Prob} &= \left(\frac{1}{4} - \epsilon(n)\right) + \left(\frac{1}{4} - \epsilon(n)\right) \\ &= \frac{1}{2} - 2\epsilon(n) \end{aligned}$$

   So, both a and b can be correct with probability $\geq \frac{1}{2} + 2\epsilon$, which applies to c as well.

3. By **Chernoff Bound**, if we repeat computation of c $\frac{2n}{\epsilon(n)}$ times, the majority of c will be correct x_i^* with probability $1 - e^{-n}$.

0.4 Proof of GL Theorem

Given A such that:

$$\text{Pr}_{r,x}[A(f(x), r) = \langle x, r \rangle] \geq \frac{1}{2} + \epsilon$$

We will design an algorithm B for inverting f with probability more than $\epsilon/4$. To do this, let us first define a good set of x values. These are the x values for which A guesses the hardcore bit with better than 1/2 probability. Let G_d be the set of good values defined as follows:

$$Gd = \{x : \text{Pr}_{r,r'}[A(f(x), r) = \langle x, r' \rangle] \geq \frac{1}{2} + \frac{\epsilon}{2}\}$$

We claim that there are many good x values; more precisely:

$$\Pr_{\text{x}}[x \in \mathit{Good}] \geq \frac{\epsilon}{2}$$

Suppose that this is not true then , $\Pr_{\text{x,r}}[A(f(x), r) = \langle x, r' \rangle] < \frac{1}{2} + \epsilon$
Now we define adversary B which guesses b_1, b_2, ..., b_l for random values r_1, r_2, ..., r_l and then generates values b'_1, ..., b'_m and r'_1, ..., r'_m as we discussed above. B then uses them to guess bits of x one by one.
Suppose that the values B generates are correct hard core bits, i.e., $(b'_1, \ldots, b'_m)$ and $(r'_1, \ldots, r'_m)$ are such that $\langle x, r'_j \rangle = b'_j$. Then, B can use A to guess the hardcore bit for $r''_j = e_i \oplus r'_j$. It

can then recover a guess for x_i as we did in the warm up proof for the $3/4 + \epsilon$ case. Then, the guess for x_i is obtained as:

$$x_i^* = \text{majority bit in } \{x_{i,j}^*\}_{j=1}^n$$

We claim that if $m = \frac{2n}{\epsilon^2}$, then for every $x \in G_d$:

$$Pr[x_i^* \neq x_i] < \frac{1}{2n}$$

Keep an indicator variable y_j such that $y_j = 1$ if $x_{i,j} \neq x_i$. Let:

$$y = y_1 + y_2 + ... + y_m$$

Then, x_i^* is not correct if $y > m/2$. We apply Chebyshev for $x \in Gd$. Notice that for $x \in Gd$, each y_i is 1 with probability $p = \Pr[y_i = 1] = 1 - \left(\frac{1}{2} + \frac{\epsilon}{2}\right) = 1 - \frac{\epsilon}{2}$, and $E[y] = mp$ where $m = \frac{2n}{\epsilon^2}$. Let $\delta = \frac{1}{2} - p = \frac{\epsilon}{2}$. Then, using Chebyshev:

$$Pr[y > \frac{m}{2}] = \frac{1}{2n}$$

As claimed. Therefore, for any given x, by the above strategy B will get x_i wrong for any given $x \in G_d$ with at most $\frac{1}{2n}$ probability. By union bound, if B guesses each x_i one by one for each i to construct full x, the probability that x will not be correct is at most $n \times \frac{1}{2n} = \frac{1}{2}$. This gives us the following algorithm B for inverting $f(x)$ for a random x

0.5 Algorithm B to invert f

1. Pick random values $(r_1, \ldots, r_l)$ for $l = \log m + 1$ where $m = \frac{2n}{\epsilon^2}$.
2. Cycle through all possible values of $(b_1, \ldots, b_l)$ starting from $(0, 0, \ldots, 0)$ to $(1, 1, \ldots, 1)$ doing the following:
   (a) for $i = 1$ to n:
      i. Construct strings $(r_1', \ldots, r_m')$ and $(b_1', \ldots, b_m')$ using the independent set construction
      ii. for $j = 1$ to m: feed $e_j \oplus r_j'$ to A and get his answer, denoted: $b_j" = A(f(x), e_j \oplus r_j')$.
      iii. Compute $x_i^* =$ majority bit in $\{x_{i,j}^*\}_{j=1}^m$ where $x_{i,j}^* = b_j' \oplus b_j"$.
   (b) return x^* if $f(x^*) = z$ where $x^* = (x_1^*, \ldots, x_n^*)$.
3. Return `fail`. (i.e., no candidate x^* found so far).

0.6 Time complexity of B

- It is easy to check that B runs in polynomial time. We have already argued that if $x \in G_d$ then the probability that B is wrong about x^* is at most $\frac{1}{2n}$ provided that it starts with $(b_1, ..., b_l)$ that are correct hardcore bits corresponding to $(r_1, ..., r_l)$.

- Since B cycles through all possible values of $(b_1, ..., b_l)$, one of them would be correct.
- Therefore, when the loop in point 2 exits, the probability that B does not invert z for any $x \in G_d$ is at most $\frac{1}{2}$.
- Since x is chosen uniformly, it is in G_d with probability at least $\epsilon/2$ as we argued before. Therefore, B inverts f with probability at least $\epsilon/2 \cdot \frac{1}{2} = \epsilon/4$. This is a contradiction and proves the GL theorem.

0.7 $G(x,r) = (f(x), r, \langle x, r \rangle)$ is a PRG

A pseudorandom generator (PRG) is a function $G_{n,n+l} : \{0,1\}^n \to \{0,1\}^{n+l}$ such that, for $x \leftarrow \{0,1\}^n$, the output $G_{n,n+l}(x)$ looks like a random $(n+1)$-bit string.
A one-bit extension PRG has $l = 1$.
Suppose $f : \{0,1\}^n \to \{0,1\}^n$ is a OWP (i.e., f is a OWF and it is a bijection). Note that the mapping $(r, x) \to (r, f(x))$ is a bijection.
So, the output $(r, f(x))$ is a uniform distribution if $(r, x) \leftarrow \{0,1\}^{2n}$. Now, the output $(r, f(x), h(r, x))$ looks like a random $(2n+1)$-bit string if f is a OWP (because of Goldreich-Levin Hardcore Predicate result).
Consider the function $G_{2n,2n+1} : \{0,1\}^{2n} \to \{0,1\}^{2n+1}$ defined as follows: $G_{2n,2n+1}(r, x) = (r, f(x), h(r, x))$ This is a one-bit extension PRG if f is a OWP.

.