0368.4162 Foundations of Cryptography November 16, 2017

Lecture 4: Hardcore Bits and the Goldreich-Levin Theorem

Lecturer: Nir Bitansky *Scribes: Gal Cohen, Shlomi Shamir*

1 Previously on Foundations of Crypto

We defined one-way functions (OWFs) and argued that they capture necessary properties for cryptography, and indeed OWFs will be necessary for basically all the primitives that we'll see in the course. The surprising part and one of the great achievements of modern cryptography is that *one-way functions are also sufficient!* at least for everything we've seen so far (and for more that's to come), which falls under the category of *secret-key crypto*.

Theorem 1.1 ([HILL99])**.** *OWFs imply PRGs!*

$$\boxed{\text{OWF}} \longrightarrow \boxed{\text{PRG}} \longrightarrow \boxed{\text{OEB Enc}} \longrightarrow \boxed{\text{Enc}}$$

The full proof of this amazing theorem is out of the scope of this course. Nevertheless, we'll develop some of the central concepts and tools that lead to this result (which will be interesting on their own), and use them to prove a weaker version.

2 Examples of OWFs

Before we go deeper into the theorem, let's try to get a sense of what one-way functions could look like. Indeed, OWFs can be based on a large variety of hard computational problems. For a comprehensive list of examples from different areas see this assay by Boaz Barak, where he makes the point that if you throw a rock at random you're likely to hit a OWF. Here, we'll give just three examples from different area:

- **Factoring:**
  - Instance: $N = pq$, for two n-bit prime numbers p, q sampled at random.
  - Solution: p.

  Today the best (classical) algorithms for this problem run in time $\approx 2^{n^{\Omega(1)}}$. The problem can be solved efficiently by quantum algorithms.

- **Subset Sum:**
  - Instance: $x_1, \ldots, x_n, T$, where $x_i \leftarrow [2^n]$, and $T = \sum_{i \in I} x_i$ for a random subset $I \leftarrow 2^{[n]}$.
  - Solution: I' such that $T = \sum_{i \in I'} x_i$.

  Today the best (classical) algorithms for this problem run in time $\approx 2^{\Omega(n)}$.

- **Learning Parity with Noise:**
  - Instance: $(a_1, \langle a_1, s\rangle + e_1), \ldots, (a_{2n}, \langle a_{2n}, s\rangle + e_{2n})$, where $s \leftarrow \mathbb{F}_2^n$, $a_i \leftarrow \mathbb{F}_2^n$, e_i is 1 w.p. 0.1.
  - Solution: s (uniquely defined with overwhelming probability).

  Today the best (classical) algorithms for this problem run in time $\approx 2^{\Omega(n)}$.

3 Toward Pseudorandom Generators from One-Way Functions

Let's start by developing some intuition on how we could conceivably transform the hardness given by a OWF f into randomness. Recall that all that we need to do is stretch a random seed s by a single bit. We are going to focus on an easier case where f is a (length-preserving) permutation $f : \{0,1\}^n \to \{0,1\}^n$. [1]

In what sense is it easier? it suggests the following natural direction to constructing a PRG:

$$PRG(s) = f(s), \text{"some bit } s_i \text{ that's hard to predict from } f(s)\text{"} \ .$$

Why should there be such a bit? well if we can predict all bits of s, then we can also invert $f(s)$. While this intuition turns out to be correct for some specific candidate functions f, it is not true in general. The main problem with this intuition is that one-wayness only guarantees that we cannot simultaneously predict all bits $s_1, \ldots, s_n$. This perhaps says that some individual bit cannot be predicted w.p. (very close to) 100%, but it may be that every individual bit s_i could be predicted w.p. 99%, thus the probability of predicting *all bits simultaneously* is still negligible (try to come up with an example).

It turns out, however, that this intuition can be made correct if we slightly generalize what we mean by "a bit of s".

Definition 3.1 (Hardcore Bit (HCB))**.** *A poly-time computable function $B : \{0,1\}^* \to \{0,1\}$ is a hardcore bit of f if*

$$f(U_n), B(U_n) \approx_c f(U_n), U_1' \ .^2$$

One could prove that being pseudorandom is equivalent to being unpredictable:

Definition 3.2 (Equivalent Definition)**.** *A poly-time computable function $B : \{0,1\}^* \to \{0,1\}$ is a hardcore bit of f if for every n.u. PPT A there exists a negligible μ such that:*

$$\Pr\left[A(f(U_n)) = B(U_n)\right] \leq \frac{1}{2} + \mu(n) \ .$$

Given a one-way permutation f with such a hardcore bit B, it is now clear how to get a PRG:

$$G(s) = f(s), B(s) \ .$$

So do all OWFs a HCB? Not exactly... but it turns out they do have a slightly more general version of a randomized HCB.

Definition 3.3 (Randomized Hardcore Bit (RHCB))**.** *A poly-time computable function $B : \{0,1\}^* \to \{0,1\}$ is randomized hardcore bit of f if*

$$f(U_n), B(U_n; U_n'), U_n' \approx_c f(U_n), U_1'', U_n' \ .$$

or equivalently, for every n.u. PPT A there exists a negligible μ such that:

$$\Pr\left[A(f(U_n), U_n') = B(U_n; U_n')\right] \leq \frac{1}{2} + \mu(n) \ .$$

Assuming one-way permutations, the above is good enough for PRGs.

Claim 3.4. *Let f be a one-way permutation and B a randomized hardcore bit for f, then*

$$G(s, s') := f(s), B(s, s'), s'$$

is a PRG.

Proof. Note that:

$$G(U_{2n}) \equiv f(U_n), B(U_n, U_n'), U_n' \approx_c f(U_n), U_1'', U_n' \equiv U_{2n+1} \ .$$

The left relation follows by the definition of G; the middle one follows by the security guarantee of hardcore predicates; and the right relation follows from the fact that $f(U_n) \equiv U_n$ for any permutation f.

This PRG is defined over inputs of length $2n$, but as in the case of OWFs, can be extended using padding. □

[1] For example, such one-way permutations can be based on the hardness of the discrete logarithm problem.

[2] Above U_n, U_1' are independent. We often use a different number of $'$s to indicate independent samples from same distribution.

Remark: If you've encountered the notion of randomness extractors, you can think of such a HCB as a strong computational extractor. Given $f(s)$, s still has some form of computational entropy (even if no information-theoretic entropy), and B extracts it. It does so using a public seed s', and pseudorandomness is guaranteed even given this seed in the clear.

4 The Goldreich-Levin Theorem

In '89 Oded Goldreich and Leonid Levin proved:

Theorem 4.1 ([GL89])**.** *There exists a randomized B that is hardcore for any OWF f.*

As we've already seen, such a theorem is proven by a reduction (from a distinguisher/predictor to an inverter). At the heart of this reduction, lies a beautiful algorithm that has subsequently had different important interpretations, and has given rise to foundational concepts in TOC (beyond crypto per se).

Proof. We define

$$B(x;r) = \langle x, r\rangle \mod 2 = \sum_i x_i r_i \mod 2 = \bigoplus_{i:r_i=1} x_i \ .$$

Fix any (w.l.o.g deterministic) adversary A that given $(f(x), r)$ predicts $B(x,r)$, with advantage $\varepsilon = \varepsilon(n)$:

$$\Pr_{x,r}[A(f(x),r) = \langle x, r\rangle \mod 2] \geq \frac{1}{2} + \varepsilon(n) \ .$$

We will construct an inverter A' for f relying on the following two claims.

Claim 4.2. *With probability $\varepsilon/2$ over a random $x \leftarrow \{0,1\}^n$:*

$$p_x := \Pr_r[A(f(x),r) = \langle x, r\rangle \mod 2] \geq \frac{1}{2} + \varepsilon/2 \ ,$$

that is, fixing x, A can predict $B(x,r)$ for a random r, with good advantage.

Claim 4.3 (The GL Decoding Algorithm)**.** *There exists an algorithm GL such that for any $x \in \{0,1\}^n$ and function $\tilde{L}_x : \{0,1\}^n \to \{0,1\}$*

$$\textit{if } \Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle \mod 2\right] \geq 1/2 + \delta \ , \textit{ then } \Pr\left[GL_{n,\delta}^{\tilde{L}_x(\cdot)} = x\right] \geq \Omega(\delta^2/n) \ .$$

The algorithm runs in time $\mathrm{poly}(n, \delta^{-1})$.

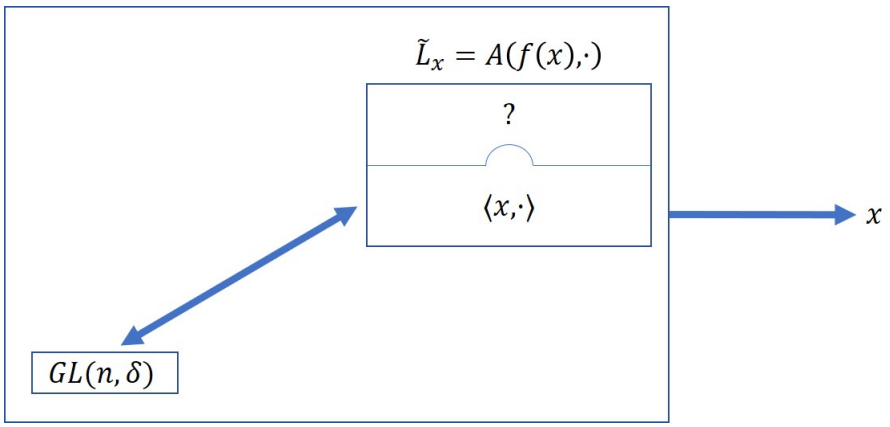

Figure 1: The Goldreich-Levin Algorithm

Indeed, given these two claims, we can easily get an inverter A' as follows:

- Given $f(x)$, "hope x is good" (occurs with probability $\varepsilon/2$).
- Apply the GL decoding procedure, with $\delta = \varepsilon/2$ to retrieve x from $\tilde{L}_x := A(f(x),\cdot)$ (succeed with probability $\Omega(\varepsilon^2/n)$).

Overall, this inverter succeeds with probability $\Omega(\varepsilon^3/n)$.

The core of the proof is the second claim, but for completeness, let us first prove the first claim.

Proof of Claim 4.2. By an averaging argument:

$$\begin{aligned}\frac{1}{2}+\varepsilon \le \mathbb{E}_x p_x &\overset{(1)}{=} \Pr_x\left[p_x < \frac{1+\varepsilon}{2}\right]\cdot \mathbb{E}\left[p_x \;\middle|\; p_x < \frac{1+\varepsilon}{2}\right] + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right]\cdot \mathbb{E}\left[p_x \;\middle|\; p_x \ge \frac{1+\varepsilon}{2}\right] \\ &\overset{(2)}{\le} \Pr_x\left[p_x < \frac{1+\varepsilon}{2}\right]\cdot \frac{1+\varepsilon}{2} + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right]\cdot 1 \le \frac{1+\varepsilon}{2} + \Pr_x\left[p_x \ge \frac{1+\varepsilon}{2}\right] .\end{aligned}$$

where (1) is by the law of total expectation and (2) is by monotonicity of expectation. □

We now move to the GL algorithm

Proof of Claim 4.3. Rather than giving the algorithm directly, we'll work our ways toward it through easier warmup steps. In what follows, inner products, additions, and multiplications will be $\bmod 2$ (and we will not write this explicitly every time).

Warmup 1: Imagine that $\tilde{L}_x(\cdot) \equiv \langle x,\cdot\rangle$. How would we learn say x_1 from the oracle? we can simply make the query $e_1 = (1,0,\ldots,0)$ and obtain exactly $\langle x, e_1\rangle = x_1$. We can learn any other x_i similarly.

Warmup 2: Imagine that

$$\Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle\right] \ge \frac{3}{4} + \delta \qquad \text{for } \delta \ge 0.01.$$

How would we learn x_1 now? We cannot simply apply the previous approach because the oracle may error on the specific query e_1. Still, the oracle is correct at many random points, and we'd like to take advantage of that. The idea is that we can embed the query e_1 into random queries. Specifically, we can choose a

random $r \leftarrow \{0,1\}^n$, and query both r and $r + e_1$. Note that if the oracle is correct on both then we can use the linearity of the inner product to learn:

$$\langle x, r + e_1 \rangle - \langle x, r \rangle = \langle x, e_1 \rangle = x_1 \ .$$

Now, what is the chance that it answers both correctly? Note, that each query *individually* is distributed uniformly at random. Thus, each one will be answered with probability at least $3/4 + \delta$ and by a union bound both will be answered correctly with probability at least $1/2 + 2\delta$.

We can now amplify this probability by repeating the experiment some number m times, and taking the majority. That is sample $r_1, \ldots, r_m$. For each i get a candidate

$$\tilde{L}_x(r_i + e_1) - \tilde{L}_x(r_i) = x_{1,i} \ ,$$

and output the majority bit as our x_1. How many times m we need to repeat?

Lemma 4.4 (Chernoff-Hoefding Bound)**.** *Let $X_1, \ldots, X_m \in \{0,1\}$ be independent experiments such that $X_i = 1$ with probability p, then*

$$\Pr_{X_1,\ldots,X_m}\left[\left|\frac{1}{m}\sum_{i\in[m]} X_i - p\right| \geq \Delta\right] \leq 2^{-\Delta^2 m} \ .$$

In our case, $p \geq 1/2 + 2\delta$, and we'd like to know how many times m to repeat to guarantee that the majority succeeded with high probability. Using the above tail bound, for $\Delta = 2\delta$, we can set $m = \delta^{-2}\log n$, which isn't too big since δ is constant. We will get an error with probability at most

$$2^{-(2\delta)^2\delta^{-2}\log n} = n^{-4} \ .$$

Thus, we can learn all n coordinates x_i w.p. at least $1 - n^{-3}$ (this follows by another union bound).

Warmup 3: Imagine that

$$\Pr_r\left[\tilde{L}_x(r) = \langle x, r\rangle\right] \geq \frac{1}{2} + \delta \qquad \text{for } \delta \geq 0.01.$$

We cannot apply the previous strategy, each individual answer may be incorrect w.p. 49%, and the probability that both are correct could be as low as 2%. The problem is that for each r, we make two correlated queries and have to absorb the error twice. Let's try to avoid it. For the time being, we'll use our imagination even further, and assume that we're given the following additional help for free. We're given samples $(r_1, \langle x, r_1\rangle), \ldots, (r_m, \langle x, r_m\rangle)$, where $r_1, \ldots, r_m$ are chosen independently at random, and as before $m = \delta^{-2}\log n = 10^4 \log n$. Can we solve the problem now?

Now, to learn x_1, we just need to ask our oracle one query $r_i + e_1$ for each i, and we know it's correct with probability 51%. Again, we can take majority of all the samples

$$\tilde{L}_x(r_i + e_1) - \langle x, r_i\rangle = x_{1,i}$$

and recover x_1 with probability $1 - n^{-4}$. In fact, we can use the same "helper samples" $(r_1, \langle x, r_1\rangle), \ldots, (r_m, \langle x, r_m\rangle)$ to recover each of the coordinates x_i, and as before take a union bound (the recovery procedure at different coordinates is not independent, but it doesn't have to be, it's enough that it succeeds with high enough probability).

But where will the helper samples come from? Well, how about we just guess them? That is, we'll sample the random r_i ourselves and guess the values $\langle x, r_i\rangle$. Our guess will hit the right value w.p. $2^{-m} = 2^{\delta^{-2}\log n} = 2^{-10^4 \log n} = n^{-10^4}$. This is still an inverse polynomial probability (admittedly, a ridiculously small one), which is enough for us.

The Actual Algorithm: We've imagined enough for one day, and now need to deal with the actual setting where the advantage δ is not constant but rather an arbitrarily small polynomial $1/p(n)$. In this case, we cannot just guess the helper values, as there are $2^{p^2(n)}$ such values.

The main idea is to choose $r_1, \ldots, r_m$ not completely at random, but in a correlated manner. On one-hand, they'll be correlated enough so that we can guess all $\langle x, r_i \rangle$ with reasonable probability. On the other hand, they will still be pseudorandom in some sufficient sense.

The algorithm $GL_{n,\delta}^{\tilde{L}_x(\cdot)}$ proceeds as follows:

1. Let $k = \log(2\delta^{-2}n + 1)$.
2. Sample $s_1, \ldots, s_k \leftarrow \{0,1\}^n$ (think about these as a seed for pseudorandomly generating the r's).
3. Sample $\sigma_1, \ldots, \sigma_k \leftarrow \{0,1\}$ (think about these as guesses for $\langle x, s_1 \rangle \ldots \langle x, s_k \rangle$).
4. For any non-empty subset $I \subseteq [k]$, let $r_I = \sum_{j \in I} s_j$ (these will be our pseudorandom r's).
5. Also, for each such I, let $\rho_I = \sum_{j \in I} \sigma_j$ (think about this as a derived guess for $\langle x, r_I \rangle$).
6. For each i, to learn x_i, for every I, obtain from the oracle $\tilde{L}_x(r_I + e_i)$, compute $x_{i,I} = \tilde{L}_x(r_I + e_i) - \rho_I$, and output the majority $maj\{x_{i,I}\}_I$.

Let's analyze the algorithm. First note that we guess all σ_j as $\langle x, s_j \rangle$, with probability $2^{-k} = \Omega(\delta^2 n^{-1})$. Whenever this happens, we also derive correct guesses ρ_I. That is

$$\rho_I = \sum_{j \in I} \sigma_j = \sum_{j \in I} \langle x, s_j \rangle = \langle x, \sum_{j \in I} s_j \rangle = \langle x, r_I \rangle \ .$$

From hereon, let us assume that we indeed guessed correctly, and examine the probability that we recover x. Let us focus on x_1 (the analysis for any other coordinate is the same). We need to make sure that the majority of answers $\tilde{L}_x(r_I + e_1)$ is indeed $\langle x, r_I + e_1 \rangle$.

First, note that each r_I is individually distributed uniformly at random, and thus also $r_I + e_1$. This means that the oracle answers correctly w.p. $1/2 + \delta$. However, now we cannot apply a Chernoff bound because the queries across different I's are not independent. Nevertheless, they are *pairwise independent*, meaning that for each two I, I' the variables $r_I, r_{I'}$ are independent.

Claim 4.5. $\{r_I\}_{\emptyset \neq I \subseteq [k]}$ *are pairwise independent.*

Proof Sketch. Let $I, I' \subseteq [k]$ s.t $I \neq I'$. Then there exists $m \in [k]$ such that $m \in I \triangle I'$. Assume w.l.o.g $m \in I$. Now, s_m and s_i are independent for any $i \in I'$, and thus s_m and $\sum_{i \in I'} s_i = r_{I'}$ are independent. Since s_m is included in sum representing r_I, we have that $r_I, r_{I'}$ are independent. □

We can now use a weaker tail bound for pairwise independent variables.

Lemma 4.6 (Chebyshev Bound (Special Case))**.** *Let $X_1, \ldots, X_m \in \{0,1\}$ be pairwise independent experiments such that $X_i = 1$ with probability p, then*

$$\Pr_{X_1,\ldots,X_m}\left[\left|\frac{1}{m}\sum_{i \in [m]} X_i - p\right| \geq \Delta\right] \leq \frac{1}{\Delta^2 m} \ .$$

In our setting, $p \geq 1/2 + \delta$, and we can set $\Delta = \delta$ and $m = 2^k - 1 = 2\delta^{-2}n$. Applying the inequality, we deduce that $\tilde{L}_x(r_I + e_1)$ will agree with $\langle x, r_I + e_1 \rangle$ for a majority of the sets I, except with probability $1/2n$. By a union bound, this will happen for all coordinates i, with probability at least $1/2$. If in addition, we guessed the right values $\sigma_1, \ldots, \sigma_k$, which happens w.p. $\Omega(\delta^2/n)$, then the algorithm succeeds.

Overall, the algorithm succeeds with probability $\frac{1}{2} \cdot \Omega(\delta^2/n)$. □

This concludes the proof that B is a (randomized) hardcore bit for any OWF. □

Reinterpreting The GL Algorithm. The GL algorithm has been given different interpretations and had a major impact on TOC, beyond cryptography. We mention two such central interpretations:

1. **List Decoding:** A central concept in coding theory, and more generally in TOC, is that of *list decoding*. Here we want to give a meaningful decoding guarantee even if the number of errors exceed the unique decoding bound. GL is exactly such an algorithm for the Hadamard code (where each codeword has the form $(\langle x, r\rangle \mid r \in \{0,1\}^n)$). The distance of this code is $1/2$, meaning that we can uniquely decode within a ball of radius $1/4$. The algorithm shows that we can even decode in a ball of radius $1/2 - \delta$, but rather than a unique codeword, we get a list of at most $O(\delta^{-2}n)$ codewords (The Johnson bound actually says that there will only be $O(\delta^{-2})$ such words).

2. **Learning:** GL solves a variant of learning parity with noise problem we discussed earlier, where instead of getting random noisy equations, the learner has the freedom to choose the equations. In particular, it could choose them to be correlated, which as we see makes a huge difference.

   Another interpretation is that the GL algorithm allows for any boolean function to learn a list of its δ-heavy Fourier coefficients in the Hadamard Fourier basis.

References

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32, 1989.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.