Lectures 11–12 - One Way Permutations, Goldreich Levin Theorem, Commitments

Boaz Barak

March 10, 2010

From time immemorial, humanity has gotten frequent, often cruel, reminders that many things are easier to do than to reverse. Leonid Levin

Reading Arora Barak chapter 9, Trevisan's lecture notes, Katz-Lindell Section 6.3 (Goldreich-Levin proof, definition of hardcore bits).

Quick review of probability Union bound, Chernoff bound, Chebychev bound.

Minimizing assumptions Up to now, we always assumed the following **PRG Axiom**: There exists a pseudorandom generator mapping $\{0,1\}^n$ to $\{0,1\}^{n+1}$.

In other words, we believe that there is an algorithm G such that the following task *cannot* be done: distinguish $G(U_n)$ from U_{n+1}.

Since we still don't have a *proof* that this cannot be done, our only evidence that a task is hard is that many people tried many approaches to solve it and didn't succeed.

The problem is that while people have been studying algorithms in one form or another for thousands of years, there hasn't been as much attention devoted to distinguishing the output of a function from the uniform distribution. Therefore, we want to have an assumption that a more natural task, such as computing a function, is hard to do.

Def We say that a function $g : \{0,1\}^* \rightarrow \{0,1\}^*$ is *hard to compute* (in the average case) if for every polynomial-time A, polynomially-bounded ϵ and large enough n

$$\Pr_{x \leftarrow_R \{0,1\}^n}[A(x) = g(x)] < \epsilon(n)$$

Claim: There exists a hard to compute function g such that g maps $\{0,1\}^n$ to $\{0,1\}^n$ for every n.

Proof: Just pick g at random. For every particular $2^{\sqrt{n}}$-time algorithm A, the expected number of inputs on which $A(x) = g(x)$ is one, and the probability that A computes g successfully on an at least $2^{-n/10}$ fraction of the total 2^n inputs can be shown to be less than $2^{-2^{-n/2}}$. But a $2^{\sqrt{n}}$ algorithm can be described by about $2^{\sqrt{n}} \ll 2^{n/2}$ bits and so the total number of such algorithms is much smaller than $2^{2^{n/2}}$. □

One-way permutation Of course the mere existence of a hard function is not useful for cryptography. But the following assumption will be useful:

The OWP Axiom: There exists a polynomial-time function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every n, f is a permutation over $\{0,1\}^n$ (i.e., maps $\{0,1\}^n$ to $\{0,1\}^n$ in a one-to-one and

onto way) and such that the function $g = f^{-1}$ is hard to compute. Such an f is called a *one-way permutation*.

An equivalent condition is that for every poly-time A, poly-bounded ϵ and large enough n

$$\Pr_{x \leftarrow_R \{0,1\}^n}[A(f(x)) = x] < \epsilon(n)$$

(can you see why these are equivalent?)

We will prove the following theorem:

Theorem 1. *The OWP Axiom implies the PRG Axiom.*

This places the PRG Axiom on a much more solid foundation, since (as alluded by Levin's quote), this is the kind of task people have tried and failed to do for centuries. (Note that in cryptography we actually put our failures to good use!)

One way functions It is known that the PRG Axiom is implied by an even weaker assumption - the existence of a *one way function*, defined as a polynomial-time function f (not necessarily a permutation) such that for every poly-time A, poly-bounded ϵ and large enough n,

$$\Pr_{x \leftarrow_R \{0,1\}^n}[A(f(x)) = w \text{ s.t. } f(w) = f(x)] \leq \epsilon(n)$$

The assumption that one-way functions exist is *minimal* for many cryptographic tasks. It can be shown that the existence of pseudorandom generators, encryptions with key shorter than message, message authentication code implies the existence of one-way functions.

Proof of Theorem 1 Theorem 1 will follow from the following two theorems:

Theorem 2 (Yao's Theorem)**.** *A distribution X over $\{0,1\}^m$ is pseudorandom if and only if it is* unpredictable*, where the latter means that for every $i \in [m]$, poly-time A and poly-bounded ϵ,*

$$\Pr_{x \leftarrow_R X}[A(x_1, \ldots, x_{i-1}) = x_i] \leq 1/2 + \epsilon(n)$$

Theorem 3 (Goldreich-Levin)**.** *Let f be a one-way permutation. Then the following distribution is unpredictable:*

$$f(x), r, \langle x, r \rangle$$

where $x, r \leftarrow_R \{0,1\}^n$ and $\langle x, r \rangle \stackrel{def}{=} \sum x_i r_i \pmod 2$.

Theorems 2 and 3 together imply that if f is a one-way permutation then the function $x, r \mapsto f(x), r, \langle x, r \rangle$ is a pseudorandom generator mapping $\{0,1\}^{2n}$ to $\{0,1\}^{2n+1}$.

Proof of Theorem 2 One direction (pseudorandomness implies unpredictability) is easy and left as an exercise.

For the other direction, to show that if X is unpredictable then it is pseudorandom, we define the following $m+1$ hybrid distributions: H^i is the first i bits of X concatenated with $m-i$ uniform random bits.

It suffices to prove that for every i, $H^{i-1} \approx H^i$. We do this by reduction using the following claim:

Claim 4. *Suppose that there is an algorithm D such that*

$$\left|\Pr[D(H^i) = 1] - \Pr[D(H^{i-1}) = 1]\right| \geq \epsilon \tag{1}$$

then, there is an algorithm P with almost the same running time, such that

$$\Pr_{x \leftarrow_R X}[P(x_1, \ldots, x_{i-1}) = x_i] \geq 1/2 + \epsilon$$

The claim clearly proves the theorem (exercise).

Proof of Claim 4. We can drop without loss of generality the absolute value in (1), since if D satisfies this condition with a negative number inside the absolute value, then $\overline{D}$ will satisfy it with a positive number.

The algorithm P will do the following on input $x_1, \ldots, x_{i-1}$:

1. Guess a value b for x_i, and choose also $m - i$ random bits $y_{i+1}, \ldots, y_m$.
2. Let $z = D(x_1, \ldots, x_{i-1}, b, y_{i+1}, \ldots, y_m)$.
3. If $z = 1$ then output b; otherwise, output $1 - b$.

Analysis: The intuition behind the analysis is that if we guessed correctly then we're in H^i situation, where we're more likely to get the $z = 1$ output.

The actual analysis is the following:

Let p be the probability that $D(H^{i-1}) = 1$ and $p + \epsilon$ the probability that $D(H^i) = 1$.

We know that $\Pr[z = 1] = p$.

On the other hand we know that $\Pr[z = 1 | b = x_i] \geq p + \epsilon$.

This means that

$$p = \Pr[z = 1] = \tfrac{1}{2}\Pr[z = 1 | b = x_i] + \tfrac{1}{2}\Pr[z = 1 | b = 1 - x_i]$$

implying that $\Pr[z = 1 | b = (1 - x_i)] \leq p - \epsilon$.

So, the probability we output a correct answer is:

$$\tfrac{1}{2}\Pr[z = 1 | b = x_i] + \tfrac{1}{2}(1 - \Pr[z = 1 | b = 1 - x_i]) \geq \tfrac{1}{2}(p + \epsilon) + \tfrac{1}{2}(1 - p + \epsilon) = \tfrac{1}{2} + \epsilon$$

□

Proof of Theorem 3 Theorem 3 will follow from the following lemma (exercise):

Lemma 5. *There is a $\mathrm{poly}(n, 1/\epsilon)$-time algorithm that given oracle access to an oracle A that computes the function $r \mapsto \langle x, r \rangle$ with probability $1/2 + \epsilon$ over the choice of r, outputs x with probability at least $\left(\frac{\epsilon}{100n}\right)^2$.*

Proof of Lemma 5 The proof of Lemma 5 is a bit involved, so we will do it step by step.

The errorless case Suppose that we had a perfect oracle A that computed $r \mapsto \langle x, r\rangle$ with probability 1. Then, we could recover the first bit of x by outputting $A(r) \oplus A(r \oplus e^1)$ for some r (where e^1 is the vector with all zeroes except at the first location).

Note that

$$A(r) \oplus A(r \oplus e_1) = \langle x, r\rangle \oplus \langle x, r \oplus e^1\rangle = \sum x_i r_i + \sum x_i (r_i \oplus e_i^1) = \sum x_i r_i + \sum x_i r_i + \sum x_i e_i^1 = x_1$$

The small error case Suppose that the oracle A was correct with probability 0.9 over the choice of r. Then because for a random r, $r \oplus e^1$ is uniformly distributed, we can use the union bound to show that the probability we get an incorrect answer when asking $A(r)$ and $A(r \oplus 1)$ is at most 0.2. (Note that these questions are dependent but the union bound still works in this case.)

Therefore, if we choose a random r, then with probability at least 0.8, $A(r) \oplus A(r \oplus e^1)$ will give us the first bit of x, and we can amplify this probability to $1/(10n)$ by making $10 \log n$ repetitions and taking the majority vote. In this way, we can recover *all* of the bits of x with high probability.

This will work as long as A is correct with probability more than $\frac{3}{4}$, but when A is correct with probability, say, 0.7, this analysis doesn't help us at all. The union bound will only say that we get the right value with probability at least 0.4 - worse than random guessing!

The full case The full case is when the oracle A is only guaranteed to be correct with probability $1/2 + \epsilon$.

Proof of Lemma 5.

Review: the low error case Recall that we said that if $\Pr_r[A(r) = \langle x, r\rangle] \geq 0.9$, then we can recover the i^{th} bit of x by choosing $r^1, \ldots, r^K$ at random ($K \geq 1000 \log n$ will do) and taking the majority of $A(r^1) \oplus A(r^1 \oplus e^i), \ldots, A(r^K) \oplus A(r^K \oplus e^i)$.

The analysis of this uses the following facts:

1. If r is chosen uniformly at random then $r \oplus e^i$ is also uniformly distributed.
2. Therefore, $\Pr[A(r) \neq \langle x, r\rangle] \leq 0.1$ and $\Pr[A(r \oplus e^i) \neq \langle x, r \oplus e^i\rangle] \leq 0.1$, implying by the union bound that $\Pr[A(r) = \langle x, r\rangle$ AND $A(r \oplus e^i) = \langle x, r \oplus e^i\rangle] \geq 0.8$. Thus, with probability at least 0.8, $A(r) \oplus A(r \oplus e^i) = x_i$.
3. Using the Chernoff bound, if we repeat this for K independently chosen random $r^1, \ldots, r^K$ then the probability that the majority of the values $A(r^j) \oplus A(r^j \oplus e^i)$ will be different from x_i is at most $2^{-K/1000}$.

   The reason is that the Chernoff bound guarantees that if $X_1, \ldots, X_K$ are independent random 0/1 variables with $\Pr[X_j] = p$, then

   $$\Pr\left[\left|\sum_j X_j - pK\right| > \epsilon pK\right] \leq 2^{-\epsilon^2 pK/5}$$

   Letting X_j be the random variable that is equal to 1 if both $A(r^j)$ and $A(r^j \oplus e^i)$ are correct we get the result.

4. This means that if we choose $K > 10^4 \log n$, then the probability we get the correct value for the i^{th} bit is at least $1 - \frac{1}{10n}$. Using the union bound, this means that with probability at least 0.9 we get the correct value for *all* of the bits.

Extending the analysis to the higher error case Suppose now that $A(r)$ is only correct with probability $1/2 + \epsilon$. In this case we can no longer argue that with probability better than $1/2$, both $A(r)$ and $A(r \oplus e^i)$ are correct. However, note the following (seemingly useless) observation:

If someone gave us the values of $z_1 = \langle x, r^1 \rangle, \ldots, z_K = \langle x, r^K \rangle$ for $K = \frac{100}{\epsilon^n} \log n$ randomly chosen strings $r^1, \ldots, r^K$ then we could run the algorithm above to deduce all the bits of x. The reason is that since $\Pr[A(r \oplus e^i) = \langle x, r \oplus e^i \rangle] \geq 1/2 + \epsilon$, the Chernoff bound implies that the i^{th} bit of z is equal to the majority of $z_j \oplus A(r^j \oplus e^i)$ with probability at least $1 - \frac{1}{10n}$.

Using pairwise independence Another observation is that we could still run the same algorithm if someone gave us the values of $z_1 = \langle x, r^1 \rangle, \ldots, z_K = \langle x, r^K \rangle$ for $K = \frac{10n}{\epsilon^2}$ strings that are chosen from a *pairwise independent distribution.*

By pairwise independent we mean that each r^j is has the uniform distribution and for every $i \neq j$, the random variables r^i and r^j are independent, *but* it's not necessarily the case that for a triple $i, j\ell$, the random variables r^i, r^j, r^ℓ are independent.

The reason we can still carry through the analysis is that if we define X_j to be the random variable that is 1 if $A(r^j \oplus e^1)$ is correct and 0 otherwise, then we know that $\mathbb{E}[X_j] \geq \frac{1}{2} + \epsilon$, and that the variables $X_1, \ldots, X_K$ are pairwise independent, and hence $Var(X_1 + \ldots + X_K) = Var(X_1) + \ldots + Var(X_K) \leq K$. (Note that $\mathbb{E}[X_1 + \ldots + X_k] = \sum_{j=1}^K \mathbb{E}[X_j] \geq (1/2 + \epsilon)K$.)

It follows that by the Chebychev Inequality

$$\Pr\Big[\text{majority value incorrect}\Big] \leq \Pr\left[\left|\sum_j X_j - \mathbb{E}[\sum_j X_j]\right| \geq \epsilon K = \epsilon\sqrt{K}\sqrt{K}\right] \leq \frac{\epsilon}{K}$$

Meaning that for $K > \frac{10n}{\epsilon^2}$, this probability is less than $\frac{1}{10n}$.

Getting these values How do we get these magical values $z_1, \ldots, z_K$? One way is to just guess them but this will be successful with probability 2^{-K} which is far too small.

The crucial observation is the following lemma:

Lemma 6. *Let $K = 2^k - 1$ and identify every number j between 1 and K with a non-empty subset S_j of $[k]$. Consider the following distribution $r^1, \ldots, r^K$ over $\{0,1\}^n$: first $s^1, \ldots, s^k$ are chosen independently at random in $\{0,1\}^n$, then we define $r^j = \sum_{i \in S_j} s_i$ (where the sum is done componentwise modulo 2).*

Then $r^1, \ldots, r^K$ are pairwise independent.

Once we have Lemma 6 we're done. The reason is that we can choose $k = \log(\frac{10n}{\epsilon^2}) + 1$ strings $s^1, \ldots, s^k$ at random and guess values $y_1, \ldots, y_k$, hoping that $y_i = \langle x, s^i \rangle$. We will be correct with probability $2^{-k} = \frac{\epsilon^2}{20n}$. Now, identifying the numbers between 1 and $K = \frac{10n}{\epsilon^2}$ with the non-empty subsets of $[k]$, define for every $j \in [K]$,

$$r^j = \sum_{i \in S_j} s^i$$

then we can set $\langle x, r^j \rangle = \sum_{i \in S_j} \langle x, s^i \rangle$ and hence we have a collection of K pairwise independent strings $r^1, \ldots, r^K$ for which we know the values $\langle x, r^j \rangle$ for every j!

Proof of Lemma 6 We need to show that for every $i \neq j$ and strings $z, w \in \{0,1\}^n$, $\Pr[r^i = z \text{ AND } r^j = w] = 2^{-2n}$.

In other words, we need to show that for every distinct pair of non-empty sets U, V

$$\Pr[\sum_{u \in U} s_u = z \text{ AND } \sum_{v \in V} s_v = w] = 2^{-2n}$$

We'll demonstrate this for the pair $U = 1, 2, 3$ and $V = 1, 2$. That is, we need to show that if we pick s_1, s_2, s_3 independently at random, then the probability that the following pair of equations are satisfied is exactly 2^{-2n}.

$$\begin{aligned} s_1 + s_2 + s_3 &= z \\ s_1 + s_2 &= w \end{aligned}$$

(If you know some linear algebra you can see this is the case because the two equations are linearly independent.)

Fix any choice for s_1. We will prove that there is a unique pair s_2, s_3 that satisfy

$$\begin{aligned} s_2 + s_3 &= z - s_1 \\ s_2 &= w - s_1 \end{aligned}$$

but this is immediate from the equations. □

Conclusion As a conclusion we get that the function $x, r \mapsto f(x) \| r \| \langle x, r \rangle$ is a pseudorandom generator.

Hard-core bits We can abstract the essence of the Goldreich-Levin theorem as follows: define a *hard-core* bit for a one-way function or permutation $g : \{0,1\}^* \to \{0,1\}^*$ to be a function $h : \{0,1\}^* \to \{0,1\}$ such that for every poly-time A and poly-bounded ϵ,

$$\Pr_{x \leftarrow_R \{0,1\}^n}[A(g(x)) = h(x)] \leq \tfrac{1}{2} + \epsilon(n)$$

The Goldreich-Levin Theorem says that if there exists a one-way permutation f, then there exists a different one-way permutation g (namely, $g(x, r) = f(x) \| r$) that has a hardcore bit h (namely, $h(x, r) = \langle x, r \rangle$). Thus it is often known as the theorem that every one-way permutation has a hardcore bit.

Commitment Schemes One use that we may like for a digital envelope is the ability to commit in advance to some value. For example, suppose I bet you a million dollar that I can predict the winner of American Idol. Now I don't want to tell you my prediction since you'd have considerable financial incentive to try to effect the competition's outcome. On the hand, you'd probably want me to *commit* in advance to my prediction (i.e., you won't be too happy with a protocol where after the results are known I'd tell you whether or not this was the winner I predicted.)

In the physical world, we might try to solve this problem by me writing the prediction in an envelope and putting the envelope in a safe (ideally, guarded by both of us). The digital analog for that is a *commitment*.

Definition 7 (Commitment schemes)**.** A *commitment scheme* Com is an *unkeyed* function that takes two inputs: a plaintext $x \in \{0,1\}^\ell$ and randomness r (chosen in $\{0,1\}^n$). The idea is that to commit to the winner I let x be my prediction (e.g. $x =$`'Siobhan'`), choose $r \leftarrow_{\mathrm{R}} \{0,1\}^n$ and publish $y = \mathsf{Com}(x,r)$. Later to prove I predicted x, I will publish x and r.

A commitment scheme should satisfy the following two properties:

Hiding / Secrecy / Indistinguishability For every $x, x' \in \{0,1\}^\ell$, $\mathsf{Com}(x, U_n)$ is computationally indistinguishable from $\mathsf{Com}(x', U_n)$. (Note this is the same as the indistinguishability property for encryption scheme, and implies that given $y = \mathsf{Com}(x, U_n)$ an adversary can't learn any new information about x.)

Binding For every y there exists at most a *single* x such that $y = \mathsf{Com}(x,r)$ for some $r \in \{0,1\}^n$. (This implies that it is not possible to come up with two different pairs x, r and x', r' with $x \neq x'$ that yield y.)

Why not encryption? You might be wondering why do we need to use a new primitive: why don't I simply *encrypt* the plaintext and give you the encryption. The problem with this approach is that an encryption does not necessarily bind me to a single value. As an example, consider the one-time-pad encryption: I can give you a random string y. Then, if the winner is Fantasia I will give you $x =$`'Fantasia'`, $k = x \oplus y$ and claim that initially I encrypted x with the key k to get y. If the winner is Eva I will give you $x' =$`'Eva'`, $k' = x' \oplus y$ and claim I initially encrypted x' with the key k' to get y. You have no way to dispute this claim.

Another application. Another, perhaps more plausible application for commitment schemes is to arrange close bids. Suppose I am a government agency that wants to award a contract to the lowest bidder. One way to arrange this is to have all bidders send their bids to the agency, but then perhaps an unscrupulous worker can leak the bid of one company to a different company. Instead, all bidders can send a *commitment* to their bid to the agency, and only after all bids have been received will they send the randomness needed to open the commitment.

We will see more applications for commitment schemes later in the course.

Constructing commitments The first observation is that to construct a commitment to strings of length ℓ, it is enough to construct a commitment to single bits. The reason is if I have a single-bit commitment then to commit to a string $x = x_1 \cdots x_\ell$ I will simply commit to each bit separately (using of course independent randomness for each bit). The security of this scheme is left as an exercise.

Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $h : \{0,1\}^n \to \{0,1\}$ be a hard-core bit for $f(\cdot)$. To commit to a bit b, I will choose $r \leftarrow_{\mathrm{R}} \{0,1\}^n$, and let $\mathsf{Com}(b,r) = f(r), h(r) \oplus b$.

Theorem 8. *The function* $\mathsf{Com}(b,r) = f(r), h(r) \oplus b$ *is a secure commitment scheme.*

Proof. (The following proof is a bit sketchy, and it's a good exercise for you to fill in the details.)

Binding Given $y = y', c$ there is a single r such that $y' = f(r)$. Thus, this r determines completely whether y is a commitment to 0 (in which case $c = h(r)$) or a commitment to 1 (in which case $c = \overline{h(r)}$).

Hiding We need to prove that $f(r), h(r) \oplus 0$ is indistinguishable from $f(r), h(r) \oplus 1$. However, $f(r), h(r)$ is indistinguishable from U_{n+1} and U_{n+1} with the last bit flipped is the same distribution as U_{n+1}.

□

Coin tossing over the phone