Lecture 9 - One Way Permutations and Hardcore bits.

Boaz Barak

October 18, 2005

Admin Reading from Thursday, No office hours on Thursday. Public key cryptography will be introduced by Tal Rabin on Tuesday.

This lecture. Previously we saw that we can use Axiom 1 (existence of pseudorandom generators) to derive private key encryption schemes of various forms and strengths and message authentication codes.

In this lecture we will give a plausible computational assumption that implies Axiom 1, and some applications for this assumption. This assumption will serve as a useful introduction to the assumptions used for *public key cryptography*.

One way permutations. It is reasonable to assume that some computational processes are much easier to compute in one direction than in the other direction (consider the task of reconstructing a broken glass statue).

In computational terms this translates to the following conjecture / axiom / assumption:

Axiom 2 - One-way permutations. There exists a family of permutations $f = \{f_n\}$ where f_n is a permutation over $\{0,1\}^n$ such that f is computable in polynomial time, but for every polynomials $T(\cdot)$ and $\epsilon(\cdot)$ and large enough n, if A is a $T(n)$-sized circuit then

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

(Note that since we say that this is for *every polynomial* $T(\cdot)$ and $\epsilon(\cdot)$ this is the same as saying that there *exist super-polynomial* $T(\cdot)$ and $\epsilon(\cdot)$ satisfying this equation.)

We will discuss candidates for such functions later on but note that it does seem as a plausible assumptions, since it does not seem that for all permutations that are easy to compute in one direction, are also easy to compute in the other.

Note that there is a big difference between this assumption and the existence of *pseudorandom permutations*. The reason is that this collection $\{f_n\}$ is *unkeyed* and so *there is no key* that helps computing it in the "backwards" direction.

Applications of Axiom 2. Depending on time, we will show to applications of Axiom 2:

1. *Pseudorandom generators* Axiom 2 implies Axiom 1. That is, if there exist one-way permutations then there exist pseudorandom generators.
2. *Commitment schemes* Axiom 2 implies the existence of *commitment schemes*. These are a crypto tool that allows to capture another property of physical envelopes in the digital words. The ability to *commit* to some piece of data and hold it in a sealed envelope so

one hand nobody knows what you wrote down, but on the other hand you can't change the original value you wrote down to some different value.

Hard-core bits. Both applications depend crucially on the concept of a *hard-core bit.*

To illustrate this idea, let's think of an example. Suppose that $f(\cdot)$ is a one-way permutation. This means that given $y = f(x)$ for a random x, it's hard to compute x. Does this mean that it is hard given $f(x)$ to compute the first bit of x?

The answer is *no.* Just because it's hard to find x does not mean that it's hard to compute x_1. We can in fact prove this:

Suppose that $f(\cdot)$ is a one-way permutation over $\{0,1\}^n$. Then, the following function f' is also a one-way permutation over $\{0,1\}^{2n}$: for $x \in \{0,1\}^{2n}$ let x^1 denote the first half of x and x^2 the second half. Then $x = x^1 \circ x^2$ and define $f'(x^1 c \circ x^2) = x^1 \circ f(x^2)$.

Now we make two claims:

1. f' is a one-way permutation. (a) it is obviously still a permutation and (b) if there is an A' that for random x^1 and x^2 computes $x_1 \circ x_2$ from $x^1 \circ f(x^2)$ then we can convert it into A that inverts $f(\cdot)$ with the same probability: on input y, A will choose x^1 at random, feed $x^1 \circ y$ to A', and if A' returns $x^1 \circ x^2$ then A will output x^2.

This example shows that if $f(\cdot)$ is a one-way permutation that does not guarantee that the first bit (and equivalently any other bit of the input) is hard to compute from the output.

However, if such a bit is hard to compute we say that it is a *hard core* bit. That is, we say that the first bit is hard core if for every polynomial-sized A and polynomial ϵ

$$\Pr_{x \leftarrow_{\mathrm{R}} \{0,1\}^n}[A(f(x)) = x_1] \leq \frac{1}{2} + \epsilon$$

More generally, we can define this for every boolean function h of the input (not just a function that outputs a particular bit).

Definition 1 (Hard-core bits.)**.** Let $f = \{f_n\}$ be a one-way permutation. Let $h : \{0,1\}^* \rightarrow \{0,1\}$ be a polynomial-time computable Boolean function. We say that h is a *hard-core* for f if for every polynomials $T(\cdot)$ and $\epsilon(\cdot)$ and large enough n, if A is a $T(n)$-sized circuit then

$$\Pr_{x \leftarrow_{\mathrm{R}} \{0,1\}^n}[A(f(x)) = h(x)] < \frac{1}{2} + \epsilon(n)$$

Note that the hardcore is easy to compute from x but hard to compute from $f(x)$. There is an obvious "hard" way to compute $h(x)$ from y which is to compute x from y and then compute $h(x)$. However, as we saw it is not always clear that there isn't a faster way.

Every one-way permutation has a hard core. The important thing to know is that essentially every one-way permutation has a hard-core bit. More accurately, for every one-way permutation $f(\cdot)$ we can convert $f(\cdot)$ into a different one-way permutation $f'(\cdot)$ (similar to the way we had our counterexample above) such there is a simple polynomial-time computable $h(\cdot)$ that is a hard-core for the function $f'(\cdot)$.

Let's defer talking about the proof of this theorem (you'll prove parts of it in a guided way in the exercises) and from now on assume that we have both a one-way permutation $f(\cdot)$ and its hard-core function $h(\cdot)$.

Application 1: A pseudorandom generator.

Theorem 1. *Let f, h be a one-way permutation and its hardcore bit. Then the following is a pseudorandom generator $G : \{0,1\}^n \to \{0,1\}^{n+1}$: $G(x) = f(x) \circ h(x)$.*

Proof. Suppose we had C such that $|\Pr[C(U_{n+1}) = 1] - \Pr[C(G(U_n)) = 1]| > \epsilon$. We'll show how we can construct an algorithm A to compute the hardcore bit from y.

We can assume w.l.o.g that $\Pr[C(G(U_n)) = 1] > \Pr[C(U_{n+1}) = 1] + \epsilon$. Our algorithm A will do the following: on input y choose $b \leftarrow_{\mathrm{R}} \{0,1\}$ at random and run $C(y, b)$. If $C(y, b) = 1$ then output b otherwise, output $1 - b$.

First, note that $p \overset{\Delta}{=} \Pr[C(y, b) = 1] = \Pr[C(U_{n+1}) = 1]$. This because f is a permutation and so if x is chosen at random then $y = f(x)$ is also uniform.

Since with probability half $b = h(x)$ and with probability half $b = 1 - h(x) = \overline{h(x)}$, we get that $p = \frac{1}{2}\Pr[C(y, h(x)) = 1] + \frac{1}{2}\Pr[C(y, \overline{h(x)}) = 1]$. Since we know that $\Pr[C(y, h(x)) = 1] > p + \epsilon$ we get that $\Pr[C(y, \overline{h(x)}) = 1] < p + \epsilon$.

We get that with probability half, we choose $b = h(x)$ and then output b with probability at least $p + \epsilon$ and with probability half we choose $b = \overline{h(x)}$ and then output $\overline{b}$ with probability $1 - (p - \epsilon)$. Over all we output the right value with probability at least

$$\frac{1}{2}(p + \epsilon + 1 - p + \epsilon) \geq \frac{1}{2} + \epsilon$$

□

Application 2: 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 2 (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. $x =$ `'Fantasia'`), 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 2. *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}.

$\square$