6.875: Solutions for Homework 4

Yevgeniy Dodis

Problem 1: PRGs and Permutations

Let p be a polynomial, let $G : \{0,1\}^k \mapsto \{0,1\}^{p(k)}$ be a pseudo-random generator, and let $h : \{0,1\}^* \mapsto \{0,1\}^*$ be a function that is a permutation over $\{0,1\}^j$ when restricted to $\{0,1\}^j$, for all $j > 0$.

(a) Is it necessarily true that the distribution $h(G(s))$ (where s is chosen uniformly at random from $\{0,1\}^k$) is polynomially indistinguishable from the uniform distribution over $\{0,1\}^{p(k)}$? Is $h(G(s))$ necessarily a pseudo-random generator?

(b) Is it necessarily true that the distribution $G(h(s))$ (where s is chosen uniformly at random from $\{0,1\}^k$) is polynomially indistinguishable from the uniform distribution over $\{0,1\}^{p(k)}$? Is $G(h(s))$ necessarily a pseudo-random generator?

(c) Will any of yours answers to Parts (a) or (b) change if it is known that h is polynomial-time computable?

Solution: Let $U(i)$ denote the uniform distribution over $\{0,1\}^i$.

(a) It is *not* necessarily true that $h(G(s))$ is indistinguishable from $U(p(k))$, because no one said that h is polynomial time computable. In fact, h can undo all the "scrambling" done by G. For example, define $h(G(s)) = s'0^{p(k)-k}$ where s' is the lexicographically first preimage of $G(s)$ (since G does not need to be injective in general), and extend h arbitrarily for y not in the image of G such that h is a permutation over $\{0,1\}^{p(k)}$. Then $h(G(s))$ always has the last $p(k) - k$ bits equal to 0, which is very different from $U(p(k))$. This already implies that $h(G)$ is not necessarily a PRG.

(b) It is now true that $G(h(s))$ is polynomially indistinguishable from $U(k)$. Indeed, when s is chosen at random from $\{0,1\}^k$, $h(s)$ is also chosen at random from $\{0,1\}^k$ since h is a permutation. Therefore, the distribution $G(h(s))$ is identical to $G(s)$, and hence indistinguishable from the uniform. However, $G(h(s))$ need not be polynomial time computable if h is not polynomial time computable, so it is not necessarily a PRG.

(c) Now, $G(h(s))$ is a PRG: by part (b) and since it is polynomial time computable now. We also claim that $h(G(s))$ is a PRG now. It is polynomial time computable and we show next that $h(G(s))$ is indistinguishable from $U(p(k))$ when h is polynomial time computable, i.e. we can no longer "unscramble" G.

Assume not, i.e. there is a distinguisher A telling apart random $x = h(G(s))$ from random $y \in \{0,1\}^{p(k)}$. Here is a distinguisher B for G then. On input z, return $A(h(z))$. The statistical test is polynomial time, since h is polytime. When $z = G(s)$, $h(z) = h(G(s)) = x$, and when z is random over $\{0,1\}^{p(k)}$, then so is $y = h(z)$ as h is a permutation. So B has the same advantage as A.

Problem 2: Next Bit Unpredictability $\Longrightarrow$ Passing all Statistical Tests

In class we proved that if the output of a generator $G : \{0,1\}^k \mapsto \{0,1\}^n$ (here n is some polynomial of k) passes the next bit unpredictability test, then it passes any statistical test. The proof used a hybrid argument to show that if there was a polynomial time statistical test A that distinguishes a completely random string from the one generated by G, then the test could distinguish between a string in which the first i bits are from G and the rest random, and a string in which the first $(i+1)$ bits are from G and the rest random. To complete the proof, we then need to show how to use this to predict the next bit $(i+1)$ from the first i bits with probability non-negligibly better than $\frac{1}{2}$.

Denote by G_m the first m bits of $G(x)$ (where x is a random seed), and by R_m a sequence of m random bit chosen from the uniform distribution. Assume without loss of generality that $\Pr[A(G_i R_{n-i}) = 0] = p$, and that $\Pr[A(G_{i+1} R_{n-i-1}) = 0] = p + \frac{1}{k^c}$ for some $c > 0$ (that is, we are assuming w.l.o.g. that A outputs 0 more often when the $(i+1)$-st bit is from G). First, some shortcuts in notation. Whenever we will run A, the first i bits come from the generator, last $(n-i-1)$ bits are totally random, i.e. only the $(i+1)$-st bit is different. So I denote by $A(b)$, $b \in \{0,1\}$, running $A(G_i b R_{n-i-1})$. Note, that when running $A(b)$ several times, we always leave the *same prefix* G_i that was given to us at the beginning, but always put brand new random bits in the last $(n-i-1)$ positions. I denote by $r \in \{0,1\}$ a random bit and by g – the $(i+1)$-st bit of G, where the seed is chosen at random. In this problem, we will analyze the behavior of several tests B that will try to contradict the next bit test, i.e. to predict g given G_i.

Let $x = \Pr[g = 0]$ (where the probability is over random seed). Introduce the following "irreducible" probabilities:

$$\beta_{jk} := \Pr[A(j) = 0 \mid g = k], \qquad j,k \in \{0,1\} \tag{1}$$

The reason that this probabilities are important is that we will have to analyze $\Pr[B = g]$, and therefore, will have to immediately condition on the value of g, i.e. $g = 0$ or $g = 1$. And since B will always run A, the needed probability will indeed be some function of x and β_{jk}'s. We note that all 4 probabilities in (1) are generally different. Indeed, conditioning on a particular setting of g skews the distribution of the first i bits g_i. Many of you made "simplifying assumptions" by wrongly assuming that $\beta_{j0} = \beta_{j1}$. Let us start by expressing our given probabilities in terms of "irreducible" ones.

$$\begin{aligned} p &= \Pr[A(r) = 0] = \frac{1}{2}(\Pr[A(0) = 0] + \Pr[A(1) = 0]) = \frac{1}{2}(x\beta_{00} + (1-x)\beta_{01} + x\beta_{10} + (1-x)\beta_{11}) \\ p + \frac{1}{k^c} &= \Pr[A(g) = 0] = x\beta_{00} + (1-x)\beta_{11} \end{aligned}$$

Subtracting the first equation from the second, we get the main equality that we will use:

$$\frac{1}{k^c} = \frac{1}{2}[x(\beta_{00} - \beta_{10}) + (1-x)(\beta_{11} - \beta_{01})] \tag{2}$$

First, let us analyze the *correct* tests B, which are tests (b) and (c). Then we will give a common counter-example for tests (a) and (d).

(b) *Run $A(0)$ and call the output a_0. Then run $A(1)$ and call the output a_1. If $a_0 = a_1$, choose the output to be 0 or 1 randomly (with probability $\frac{1}{2}$). Otherwise, output the bit b for which $a_b = 0$ (that is, if $a_0 = 0$ output 0, and if $a_1 = 0$ output 1).*

Intuitively, the test makes sense intuitively. When $a_0 = a_1$, we did not learn much. If they are different, the one giving 0 is likely to be g since the generator gives 0 more often by assumption. However, the analysis is tricky. Here is the correct formal argument (we write $B = c$ meaning B

outputs c given the first i bits of G). First, using $\Pr[A \wedge B] + \Pr[A \wedge \bar{B}] = \Pr[A]$, let us compute

$$\begin{aligned}
\Pr[B=0 \mid g=0] &= \frac{1}{2}(\Pr[A(0)=A(1)=0 \mid g=0] + \Pr[A(0)=A(1)=1 \mid g=0]) + \\
&\quad \Pr[A(0)=0 \wedge A(1)=1 \mid g=0] \\
&= \frac{1}{2}(\Pr[A(0)=0 \wedge A(1)=0 \mid g=0] + \Pr[A(0)=0 \wedge A(1)=1 \mid g=0]) + \\
&\quad \frac{1}{2}(\Pr[A(0)=1 \wedge A(1)=1 \mid g=0] + \Pr[A(0)=0 \wedge A(1)=1 \mid g=0]) \\
&= \frac{1}{2}(\Pr[A(0)=0 \mid g=0] + \Pr[A(1)=1 \mid g=0]) \stackrel{(1)}{=} \frac{1}{2}(\beta_{00} + (1-\beta_{10})) \\
&= \frac{1}{2} + \frac{1}{2}(\beta_{00} - \beta_{10}). \\
\Pr[B=1 \mid g=1] &= \frac{1}{2}(\Pr[A(0)=A(1)=0 \mid g=1] + \Pr[A(0)=A(1)=1 \mid g=1]) + \\
&\quad \Pr[A(0)=1 \wedge A(1)=0 \mid g=1] \\
&= \frac{1}{2}(\Pr[A(0)=0 \wedge A(1)=0 \mid g=1] + \Pr[A(0)=1 \wedge A(1)=0 \mid g=1]) + \\
&\quad \frac{1}{2}(\Pr[A(0)=1 \wedge A(1)=1 \mid g=1] + \Pr[A(0)=1 \wedge A(1)=0 \mid g=1]) \\
&= \frac{1}{2}(\Pr[A(1)=0 \mid g=1] + \Pr[A(0)=1 \mid g=1]) \stackrel{(1)}{=} \frac{1}{2}(\beta_{11} + (1-\beta_{01})) \\
&= \frac{1}{2} + \frac{1}{2}(\beta_{11} - \beta_{01}).
\end{aligned}$$

Hence, the probability that B is correct is

$$\begin{aligned}
\Pr[B=g] &= x\Pr[B=0 \mid g=0] + (1-x)\Pr[B=1 \mid g=1] \\
&= x\left(\frac{1}{2} + \frac{1}{2}(\beta_{00} - \beta_{10})\right) + (1-x)\left(\frac{1}{2} + \frac{1}{2}(\beta_{11} - \beta_{01})\right) \\
&= \frac{1}{2} + \frac{1}{2}[x(\beta_{00} - \beta_{10}) + (1-x)(\beta_{11} - \beta_{01})] \\
&\stackrel{(2)}{=} \frac{1}{2} + \frac{1}{k^c}
\end{aligned}$$

(c) *Run $A(r)$. If the answer is 0, output r. If the answer is 1, output the negation of that bit.*

Intuitively, $A(r)$ *a-priori* outputs 0 less often than $A(g)$. Thus, if $A(r)$ returned 0, this gives us some *a-posteriori* indication that $r = g$. Formally, the probability of correctness for a fixed r is:

$$\begin{aligned}
\Pr[B=g \mid r=0] &= x\Pr[A(0)=0 \mid g=0] + (1-x)\Pr[A(0)=1 \mid g=1] \stackrel{(1)}{=} x\beta_{00} + (1-x)(1-\beta_{01}) \\
&= (1-x) + x\beta_{00} - (1-x)\beta_{01}. \\
\Pr[B=g \mid r=1] &= x\Pr[A(1)=1 \mid g=0] + (1-x)\Pr[A(1)=0 \mid g=1] \stackrel{(1)}{=} x(1-\beta_{10}) + (1-x)\beta_{11} \\
&= x - x\beta_{10} + (1-x)\beta_{11}.
\end{aligned}$$

Hence, conditioning on random bit r, the correctness probability is

$$\Pr[B=g] = \frac{1}{2}\Pr[B=g \mid r=0] + \frac{1}{2}\Pr[B=g \mid r=1]$$

$$
\begin{aligned}
&= \frac{1}{2}(x + (1-x) + x\beta_{00} - (1-x)\beta_{01} - x\beta_{10} + (1-x)\beta_{11}) \\
&= \frac{1}{2} + \frac{1}{2}[x(\beta_{00} - \beta_{10}) + (1-x)(\beta_{11} - \beta_{01})] \\
&\overset{(2)}{=} \frac{1}{2} + \frac{1}{k^c}
\end{aligned}
$$

Common Mistakes: None of you formally solved parts (b) and (c) correctly. The typical mistakes were ignoring conditioning and assuming independence incorrectly. In particular, most of you ignored conditioning on g when analyzing correctness, incorrectly making the computations much easier (some terms now cancel, etc.) As I already said, since we use the first bits of G when we run A, conditioning on g changes the distribution. Conditioning on r in part (c) does not skew the distribution, since r is an independently chosen bit, placed only in position $i+1$. In part (b), most of you assumed that the events $A(0) = j$ and $A(1) = k$ are independent. For example, you (sometimes implicitly) used manipulations like this: $\Pr[a_0 = 0 \wedge a_1 = 1 \mid C] = \Pr[a_0 = 0 \mid C] \Pr[a_1 = 1 \mid C]$. This is *not* true in general since when we run $A(0)$ and $A(1)$, we use the *same* prefix G_i in the first i bits (even though we choose independent last $(n-i-1)$ random bits). Hence, the trick we used in part (b) $(\Pr[A \wedge B] + \Pr[A \wedge \bar{B}] = \Pr[A])$ was more or less necessary.

Now, let us give a common counter-example to tests (a) and (d) due to the former TA of this course Tal Malkin.

(a) *Run $A(0)$ and call the output a_0. Then run $A(1)$ and call the output a_1. If $a_0 = a_1$ output 0, otherwise output 1.*

The test is wrong. the intuitive reason is that when $a_0 = a_1$, we did not learn anything in terms of what is g (both 0 and 1 gave the same result). When $a_0 \neq a_1$, we learned something, but we must examine *what we learned*. Instead, the test essentially outputs whether or not we learned something, but does not say what we learned, so it does not predict the next bit (only says whether or not we could have predicted it successfully).

Here is the example. Let G be such that its i-th bit g_i is random (0 or 1 w/pr. 1/2), while our bit $g = g_{i+1}$ is always the negation of g_i. Clearly G is not a PRG (and it can not be if we hope to give a counter-example). Here is a particular test A that will give a non-negligible difference between $A(r)$ and $A(g)$, so that all the preconditions are satisfied.

A looks at the $(i+1)$-st bit b given to it. If $b = 0$, then output 0 or 1 with probability $\frac{1}{2}$ each. Otherwise, it outputs the i-th bit it was given.

$A(r)$ outputs a *random bit*. Indeed, when $r = 0$, it does so by definition, and when $r = 1$, it outputs g_i which is a random bit by our assumption on G. Hence, $p = \Pr[A(r) = 0] = \frac{1}{2}$. However, $A(g) = A(g_{i+1})$ outputs a random bit only when $g_{i+1} = 0$ (which happens whenever $g_i = 1$, i.e. with probability $\frac{1}{2}$), but *always* outputs $g_i = 0$ whenever $g_{i+1} = 1$ (which happens with probability $\frac{1}{2}$). Thus, it outputs 0 with probability $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot 1 = \frac{3}{4}$, which is indeed greater than $p = \frac{1}{2}$. Formally,

$$
\begin{aligned}
\Pr[A(g_{i+1}) = 0] &= \Pr[g_{i+1} = 0] \Pr[A(0) = 0 \mid g_{i+1} = 0] + \Pr[g_{i+1} = 1] \Pr[A(1) = 0 \mid g_{i+1} = 1] \\
&= \Pr[g_i = 0] \frac{1}{2} + \Pr[g_i = 1] \Pr[A(1) = 0 \mid g_i = 0] = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot 1 = \frac{3}{4}
\end{aligned}
$$

So, A satisfies our precondition. We now show that B nevertheless does not succeed in predicting g correctly. Indeed, since $A(0)$ outputs a random bit, $a_0 = a_1$ with probability $\frac{1}{2}$ irrespective of the value of g. So B simply outputs a random bit, i.e. it is correct with probability $\frac{1}{2}$.

(d) *Run $A(0)$ for polynomially many times (each time independently with new random bits), and count how many times A outputs 0. If this fraction is closer to $p + \frac{1}{k^c}$ than to p, then output 0, otherwise output 1.*

The test is kind of ridiculous, I see no reason why the estimated probability has anything to do with either p or $p + \frac{1}{k^c}$. The only thing I can say semi-formally is that if we let q be the estimated probability, then $\mathbf{E}[q] = \Pr[A(0) = 0] = x\beta_{00} + (1 - x)\beta_{01}$. Even if we assume that q always equals its expectation, I see no relation between it and p or $p + \frac{1}{k^c}$, as well as why comparison to any of these numbers tells us anything about g.

To prove it, the same example as in part (a) works. Indeed, $A(0)$ is simply a random bit, so our estimated probability of outputting 0 is very close to $\frac{1}{2}$, irrespective of the first i bits G_i. Since $p = \frac{1}{2}$, out fraction is almost always closer to p, so B outputs 1 with probability essentially 1. And this equals to g_i with probability $\frac{1}{2}$, since $g_{i+1} = \bar{g}_i$ and g_i is a random bit.

Problem 3: PRG $\Longrightarrow$ OWF

Let G be a pseudo-random generator. Show that G implies the existence of a length preserving one way function f. That is, construct a length preserving f using G, and prove that f is one way, using the fact that G is polynomially unpredictable.

Solution: Let $G : \{0,1\}^k \mapsto \{0,1\}^{2k}$ be our PRG. Define $f : \{0,1\}^{2k} \mapsto \{0,1\}^{2k}$ by $f(x, z) = G(x)$, where $|x| = |z| = k$, i.e. we simply ignore z (we need it so that f is length preserving). Assume we can invert f with a reasonable probability. Here is a statistical test for G Contradicting the fact it is a PRG. Given $2k$-bit string y (either random or $= G(x)$), find $f^{-1}(y) = (x', z')$ using the assumed inverter. Accept iff $y = G(x')$.

Here is the proof of correctness. If y is random, no matter what x' we try, there are only 2^k values of $G(x')$. Since there are 2^{2k} values of y, we accept with probability at most $2^k/2^{2k} = 1/2^k$. If $y = G(x)$, we will find x' s.t. $y = G(x) = G(x')$ whenever we succeed in inverting f, i.e. with probability at least $1/poly(k)$, in which case we will accept. So the advantage is at least $\frac{1}{poly(k)} - \frac{1}{2^k}$ which is non-negligible, a contradiction.

Common Mistakes: A very common solution many of you gave was to define $f : \{0,1\}^k \mapsto \{0,1\}^k$ by $f'(x) = [G(x)]_k$, where $[y]_k$ denotes the first k bits of y. I do not know how to show that f' is a OWF. In fact, I believe that this need not be true for some G, even though I can't show it either. The difference with the previous solution is that we are outputting only the part of the generator and it is very hard to argue anything based on that. Let me just indicate 2 pseudo-arguments "proving" that f' is a OWF.

"Assume we can invert $f'(x) =$ first k bits of $G(x)$. Here is the statistical test: given y of length $2k$ (either random, or $= G(x)$ for a random x), write $y = y_0 y_1$ (concatenation). Find $f'^{-1}(y_0) = x'$." Then there are 2 options to finish the pseudo-algorithm:
Option 1: "output 1 iff $y = G(x')$".
Option 2: "output 1 iff we succeed inverting, i.e. $y_0 = [G(x')]_k$".

Let me indicate why I do not know how to prove that any of these is a statistical test.

- Option 1: If y is random, there is at most $2^k/2^{2k} = 1/2^k$ chance of succeeding since there are at most 2^k values of $G(x')$. Assume now $y = G(x)$. While we indeed found with non-negligible probability some x' s.t. $[G(x')]_k = [G(x)]_k$, we DO NOT know that the WHOLE $G(x') = G(x)$ (in particular, don't know that $x' = x$). In fact, it is easy to construct an example where this does not happen indeed. Let $G'(x) : \{0,1\}^{k/2}\{0,1\}^k$ be a good PRG. Define $G(x_1 \circ x_2) = G'(x_1) \circ G'(x_2)$. It is easy to see that G is a good PRG as well. However, for any valid prefix of $G(x)$ of length k, there are at least $2^{k/2}$ preimages, corresponding to a free choice of x_2. Hence, even though inverting algorithm succeeded to essentially invert $G'(x_1)$, the chance that x_2' it returns in fact equals the right x_2 is $2^{-k/2}$, so there is a negligible probability of success on both random and pseudo-random strings.

- Option 2: Now, if $y = G(x)$, we will succeed with non-negligible probability in inverting f' (i.e. we'll accept). However, if y (and thus y_0) are random, I do not see why this implies that my inverting algorithm won't do well when I give it y_0, i.e. k random bits. In fact, since G is a PRG, we should expect most of the k-bit strings be possible k-prefixes of $G(x)$, so it can very well be that majority of y_0 have preimages. In fact, if $[G]_k$ is a *permutation*, the test never distinguishes! In other words, I can no longer say in general that I will accept a random y with a very small probability.

Another common mistake was *incorrect* use of PRFs. People said: "PRG $\Rightarrow$ PRF, so take pseudo-random function $f : \{0,1\}^k \mapsto \{0,1\}^k$, and this is our OWF". This is wrong since there is no such thing as a pseudo-random function, there is a function *drawn at random from a family of pseudo-random functions.* In particular, any *particular* member of PRF family can be terrible, e.g. a constant function. This does not contradict much since the chance this function is selected, is 2^{-k}. So, you have to include the "seed" in your definition of OWF, so that when we pick random x from the domain of the function, we also pick a random seed. Once you do it, most PRF-based constructions work, but it is an unnecessary overkill since PRG itself works perfectly.

Problem 4: Family of Trapdoor Permutations $\Longrightarrow$ GM-secure encryption scheme

Show that the existence of a family of trapdoor permutations implies the existence of a GM-secure (e.g. under three-pass) encryption scheme.

Solution: The solution to that was given in class. Let me just refresh the idea. Let f be a trapdoor permutation (technically, we need family of trapdoor permutations, but I omit trivialities here). let $B(x)$ be a hardcore bit of f, i.e. $B(x)$ is easy to compute given x and hard – given $f(x)$ (if you wish, we can write $B(f(x))$ instead of $B(x)$, it is unimportant). We can assume w.l.o.g. that such B exists since otherwise we can use Goldreich-Levin hardcore bit $B(x, r) = x \cdot r \bmod 2$ for the trapdoor permutation $f'(x, r) = \langle f(x), r \rangle$.

To encrypt m, we pick a random x, construct a "one-time pad" $p(x) = B(x) \circ B(f(x)) \circ \ldots \circ B(f^{k-1}(x))$ and return $\langle m \oplus p(x), f^k(x) \rangle$. To decrypt, we can recover x from $f^k(x)$ using the trapdoor, and then reconstruct $p(x)$ and m. To show semantic security w.r.t. m_0 and m_1, we see that if an adversary can distinguish encryptions of m_0 and m_1, he can distinguish $p(x) \oplus m_0$ from $p(x) \oplus m_1$ only given $f^k(x)$. We saw in class that $p(x)$ is indistinguishable from the uniform distribution. And we note that for a random $y \in \{0,1\}^k$, $y \oplus m_0$ and $y \oplus m_1$ are random, so it is impossible to distinguish them. Thus, distinguishing $p(x) \oplus m_0$ from $p(x) \oplus m_1$ implies a statistical test distinguishing $p(x)$ from the uniform distribution. There is a problem, however, since we are *also* given $f^k(x)$. However, a careful repetition of the proofs that $p(x)$ passes next bit test and that next bit test implies all statistical tests shows that this knowledge does not help, since the algorithm contradicting that $B(x)$ is a hardcore bit can produce $f^k(x)$ on his own as well. I omit the details.