Lecture Notes 9

22 Proving a Predicate is Hard-Core

We continue the proof of Lemma 3 of section 21. Recall that $f$ is a strongly one-way and length preserving function, and that

$$
g(x, r) \overset{\Delta}{=} (f(x), r)$$
$$b(x, r) \overset{\Delta}{=} x \cdot r \mod 2.
$$

We showed in Lemma 2 that $g$ is strongly one-way. We want to show that $b$ is a hard core for $g$.

Assume to the contrary that $b$ is not a hard core for $g$. Then there exists a p.p.t. algorithm $G$ that predicts $b$ with non-negligible advantage $\varepsilon_G(n)$. Thus, there is a polynomial $p(\cdot)$ such that

$$\Pr[G(f(U_n), X_n) = b(U_n, X_n)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

for all $n$ in an infinite set $N$.

To complete the proof of Lemma 3 we will do two things:

1. We will construct an algorithm $A$ that on input $y$ attempts to invert $f$.
2. We will show that, for some polynomial $q(\cdot)$, $A$ has success probability greater than $\frac{1}{q(n)}$ at inverting $f$ on length $n$ inputs for all sufficiently large $n$ in the infinite set $N$. This contradicts the assumption that $f$ is strongly one-way.

We construct $A$ now and defer the analysis of its success probability to the next lecture.

22.1 Using $G$ to invert $f$

How can $G$ help us to invert $f$? A priori it doesn’t seem to be much help to have a predictor for only a single predicate when our algorithm $A$ is supposed to correctly output a length-$n$ string in $f^{-1}(y)$. It’s not at all obvious how to use $G$, and several tricks are involved.

Using $b$ to extract bits of $x$: Let $e^i$ be the unit vector with a 1-bit in position $i$ and 0’s elsewhere. As before, we treat strings in $\{0, 1\}^*$ as bit-vectors, so $x_i$ is the $i^{th}$ bit of $x$. It follows from the definition of Boolean dot product that $b(x, e^i) = x \cdot e^i \mod 2 = x_i$.

Fact

$$b(x, r) \oplus b(x, r \oplus e^i) = x_i.$$  (2)
This follows because
\[
\begin{align*}
  b(x,r) \oplus b(x,s) &= \left( \bigoplus_{j=1}^{n} x_j \cdot r_j \right) \oplus \left( \bigoplus_{j=1}^{n} x_j \cdot s_j \right) \\
  &= \bigoplus_{j=1}^{n} (x_j \cdot (r_j \oplus s_j)) \\
  &= b(x, r \oplus s)
\end{align*}
\]
Taking \( s = r \oplus e^i \), equation 3 gives
\[
\begin{align*}
  b(x,r) \oplus b(x,r \oplus e^i) &= b(x,r \oplus (r \oplus e^i)) = b(x,e^i) = x_i
\end{align*}
\] as desired.

**First idea for \( A \):** For each \( i = 1, \ldots, n \), use \( G \) to guess \( b(x,r) \) and \( b(x,r \oplus e^i) \) for random \( r \), then use equation 4 to guess \( x_i \), i.e., guess \( x_i = G(y,r) \oplus G(y,r \oplus e^i) \). Repeat this procedure to obtain polynomially many guesses of \( x_i \) and choose the majority value. Return \( x = x_1 \ldots x_n \) as the guess for \( f^{-1}(y) \).

If \( G \) is correct about \( b \) on both calls, then \( x_i \) is correct. As long the probability that both calls on \( G \) are correct is greater than \( \frac{1}{2} + \frac{1}{\text{poly}} \), repetition will amplify this probability to be sufficiently close to 1 so that the probability of all \( n \) bits of \( x \) being correct is also close to 1. Unfortunately, this is only the case when \( G \)'s success probability is \( \frac{3}{4} + \frac{1}{\text{poly}} \), that is, its advantage is a little bit greater than \( \frac{1}{4} \). All we know about our \( G \) is that it satisfies inequality 1, so its advantage is only \( \frac{1}{\text{poly}} \).

**Second idea for \( A \):** Same as first idea, except instead of using \( G \) to guess \( b(x,r) \), we just use a random bit \( \hat{b}(r) \). Thus, we guess that \( x_i = \hat{b}(r) \oplus G(y,r \oplus e^i) \). This of course doesn’t work since the probability of being correct is exactly \( \frac{1}{2} \) and we get no advantage. But if we somehow had an oracle that would give us the correct value for \( \hat{b}(r) = b(x,r) \), then this idea would indeed work since then the advantage at guessing \( x_i \) would be the same as \( G \)'s advantage at guessing \( b \).

**Third idea for \( A \):** We generate a small \( O(\log n) \)-sized set of random strings \( R_0 \) and corresponding guesses for \( \hat{b}(r) \) for each \( r \in R_0 \). From \( R_0 \), we deterministically generate a polynomial set of strings \( R \) and corresponding values for \( b(r) \), \( r \in R \). By the way the construction will work, if \( \hat{b}(r) \) is correct for all \( r \in R_0 \), then \( \hat{b}(r) \) is correct for all \( r \in R \). Since the number of strings in \( R_0 \) for which we require correct guesses is only \( O(\log n) \), it follows that the probability of all \( O(\log n) \) guesses being correct is at least \( \frac{1}{\text{poly}} \) since \( 2^{O(\log n)} = \text{poly}(n) \).

The strings in \( R \) are uniformly distributed over \( \{0,1\}^n \), but they are only pairwise independent. However, this turns out to be sufficient for the construction to work.

**How to generate \( R_0 \) and \( R \):** Let \( \ell = \lceil \log_2(2n \cdot p(n)^2 + 1) \rceil \).

1. Select \( s^1, \ldots, s^\ell \in \{0,1\}^n \) uniformly and independently. Select \( \sigma^1, \ldots, \sigma^\ell \in \{0,1\} \) uniformly and independently. Let \( R_0 = \{ s^1, \ldots, s^\ell \} \) and \( \hat{b}(s^i) = \sigma^i, i = 1, \ldots, \ell \).

2. Let \( \mathcal{J} \) be the family of non-empty subsets of \( \{1, 2, \ldots, \ell \} \). For all sets \( J \in \mathcal{J} \), compute
\[
  r^J = \bigoplus_{j \in J} s^j \quad \text{and} \quad \rho^J = \bigoplus_{j \in J} \sigma^j.
\]
Let \( R = \{ r^J \mid J \in \mathcal{J} \} \), and \( \hat{b}(r^J) = \rho^J \) for all \( J \in \mathcal{J} \).

Note that \( r^{\{i\}} = s^i \) and \( \rho^{\{i\}} = \sigma^i \), so indeed \( R \supseteq R_0 \).

**Fact** If \( b(x, s^j) = \sigma^j \) for all \( j \in \{1, \ldots, \ell\} \), then \( b(x, r^J) = \rho^J \) for all \( J \in \mathcal{J} \).

This follows because

\[
 b(x, r^J) = b(x, \bigoplus_{j \in J} s^j) = \bigoplus_{j \in J} b(x, s^j) = \bigoplus_{j \in J} \sigma^j = \rho^J.
\]

The first equality is from the definition of \( r^J \), the second comes from repeated use of equation 3, the third uses the assumption that \( \sigma^j \) is correct for all \( j \in \{1, \ldots, \ell\} \), and the final equality is from the definition of \( \rho^J \).

### 22.2 The complete algorithm for \( A \)

Here is the complete algorithm for \( A \). On input \( y \), let \( n = |y| \) and \( \ell = \lceil \log_2(2n \cdot p(n)^2 + 1) \rceil \).

1. Uniformly and independently, select \( s^1, \ldots, s^\ell \in \{0, 1\}^n \) and \( \sigma^1, \ldots, \sigma^\ell \in \{0, 1\} \).

2. For all non-empty \( J \subseteq \{1, 2, \ldots, \ell\} \), compute

\[
 r^J = \bigoplus_{j \in J} s^j \quad \text{and} \quad \rho^J = \bigoplus_{j \in J} \sigma^j.
\]

3. For all \( i \in \{1, \ldots, n\} \), for all non-empty \( J \subseteq \{1, \ldots, \ell\} \), compute

\[
 z_i^J = \rho^J \oplus G(y, r^J \oplus e^i).
\]

4. For all \( i \in \{1, \ldots, n\} \), let \( z_i = \text{majority}_J \{ z_i^J \} \).

5. Output \( z = z_1 \ldots z_n \).