CS 830: Randomness and Complexity 02/17/05

Lecture 10: More on Pseudorandom Generators

Instructor: Jin-Yi Cai Scribe: Scott Diehl, Jeff Kinne

In the last lecture(s), we gave a pseudorandom generator (PRG) which, given a one-way function, is able to produce one additional pseudorandom bit. Today we show how to use such a PRG to obtain a generator which produces a larger number of pseudorandom bits.

We also introduce the BBS (Blum-Blum-Shub) generator as a more concrete example of a PRG.

1 PRG stretching construction

Given a one-way function f, we have already seen that we can create a PRG whose output contains one additional pseudorandom bit. This PRG T is defined as follows:

$$T(x, z) = (f(x), z, x \cdot z)$$

where $x \in \{0,1\}^n$, $z \in \{0,1\}^n$. It was proved that T is a PRG from $2n$ bits to $2n+1$ bits. We will use T in the construction that we present today to produce many random bits. The intuition is that rather than computing a single dot product using a vector z, we will compute many using a collection Z of vectors. We can think of Z as a matrix where each column can be thought of as a single vector z. The function that we would like to show is a PRG is:

$$R(x, Z) = (f(x), Z, x \cdot Z)$$

where $x \in \{0,1\}^n$, $Z \in \{0,1\}^{r \cdot n}$. Notice that $x \cdot Z \in \{0,1\}^r$.

The proof that R is a PRG involves a few steps. First, we show that if R is a not a PRG, then we can determine the inner product bits $x \cdot Z$ from $f(x)$ and Z with non-negligible advantage. We then use this fact to show that we can determine the single inner product bit of $x \cdot z$ with non-negligible advantage. Recall that we have seen in a previous lecture that if we can compute the bit $x \cdot z$ from $f(x)$ and z with non-negligible advantage, then we can compute x from $f(x)$. Therefore, given these two steps, the assumption that R is not a PRG leads to a contradiction with the fact that $f(x)$ is one-way, allowing us to conclude that R is a PRG when $f(x)$ is one-way. The rest of this lecture is devoted to proving these two key steps needed to establish this result.

In this lecture, we will argue in terms of uniform adversaries, but the arguments also work for non-uniform adversaries.

2 First step to PRG

We first show that if R is not a PRG, then we can determine the inner product bits with non-negligible advantage over a random choice.

Theorem 2.1. *Suppose there is an adversary A for R such that*

$$\delta(n) = \Pr[A(f(x), x, x \cdot Z) = 1] - \Pr[A(f(x), x, U) = 1]$$

where $U \in_u \{0,1\}^r$ and $\delta(n)$ is non-negligible.

Then there is a polynomial time computable function S that makes calls to A such that

$$\Pr[S^A(f(x), Z) = x \cdot Z] \geq \frac{1}{2^r} + \frac{\delta(n)}{2^r}.$$

Proof. We start by describing the algorithm S^A and then argue that it satisfies the thoerem.

> $S(Y, Z)$
> **Description:** Algorithm that uses A to compute $x \cdot Z$ non-negligible advantage when $Y = f(x)$. Used in proof of theorem.
> Pick $u, v \in_u \{0,1\}^r$.
> Let $a = A(Y, Z, u)$.
> **if** $a = 1$ **then return** u
> **else return** v

Let us denote the probability that A accepts inner product bits or truly random bits as

$$\begin{aligned}\mu_d &= \Pr[A(f(x), Z, x \cdot Z) = 1] \\ \mu_b &= \Pr[A(f(x), Z, U) = 1],\end{aligned}$$

so that by definition, $\delta(n) = \mu_d - \mu_b$. Now we calculate the probability that S^A succeeds.

$$\begin{aligned}
&\Pr[S(f(x), Z) = x \cdot Z] && (1)\\
&= \Pr[(A(f(x), Z, u) = 1 \wedge u = x \cdot Z) \vee (A(f(x), Z, u) = 0 \wedge v = x \cdot Z)] && (2)\\
&= \frac{1}{2^r}(\Pr[A(f(x), Z, u) = 1 | u = x \cdot Z] + \Pr[A(f(x), Z, u) = 0 | v = x \cdot z]) && (3)\\
&= \frac{1}{2^r}(\Pr[A(f(x), Z, x \cdot Z) = 1] + \Pr[A(f(x), Z, u) = 0]) && (4)\\
&= \frac{1}{2^r}(\mu_d + (1 - \mu_b)) && (5)\\
&= \frac{1}{2^r} + \frac{\delta(n)}{2^r} && (6)
\end{aligned}$$

We obtain at (2) simply expanding out the probability based off of the definition of S. (3) follows by conditioning on disjoint events and the fact that $\Pr[u = x \cdot Z] = \Pr[v = x \cdot Z] = \frac{1}{2^r}$ (u and v are uniform by definition, and $x \cdot Z$ is uniform because Z is uniform). To arrive at (4), we substitute the values of the conditioned random variables– we rewrite $\Pr[A(f(x), Z, u) = 1 | u = x \cdot Z]$ as $\Pr[A(f(x), Z, x \cdot Z) = 1 | u = x \cdot Z]$, and can then remove the condition since $A(f(x), Z, x \cdot Z)$ is independent of u. We do a similar substition for the second probability. The last few lines should be clear. □

Notice that the algorithm S used in the proof could be modified slightly. In S, if A returns 1 when given u, then we return u as the guess for $x \cdot Z$. If A returns 0 when given u, then we generate a new uniform random set of bits v and return that as our guess. Rather than picking a totally uniform v, it is more intuitive to pick v uniformly in $\{0,1\}^r - \{u\}$. The fact that A returned 0 when given u is evidence that u is not $x \cdot Z$, so we remove it from consideration. We now spell out the details of this alternate proof.

Alternate proof: We first formally describe the modified algorithm S'.

$S'(Y, Z)$
Description: Algorithm that uses A to compute $x \cdot Z$ with non-negligible advantage when $Y = f(x)$. Used in alternate proof of theorem.
Pick $u \in_u \{0,1\}^r$.
Pick $v \in_u \{0,1\}^r - \{u\}$.
Let $a = A(Y, Z, u)$.
if $a = 1$ **then return** u
else return v

Consider the probability that S' is correct:

$$
\begin{aligned}
&\Pr[S'(f(x), Z) = x \cdot Z] \\
&= \Pr[(A(f(x), Z, u) = 1 \wedge u = x \cdot Z) \vee (A(f(x), Z, u) = 0 \wedge v = x \cdot Z)] \\
&= \frac{1}{2^r}(\Pr[A(f(x), Z, u) = 1 | u = x \cdot Z]) + \Pr[A(f(x), Z, u) = 0 \wedge v = x \cdot Z]) \\
&= \frac{1}{2^r}\mu_d + \alpha
\end{aligned}
$$

where $\alpha = \Pr[A(f(x), Z, u) = 0 \wedge v = x \cdot z]$. Notice that

$$\alpha = \frac{1}{2^r} \Pr_{x,Z,u' \in \{0,1\}^r - \{v\}}[A(f(x), Z, u') = 0 | v = x \cdot Z]$$

by conditioning and assuming that we had picked $u \neq v$ rather than picking $v \neq u$ (it is equivalent in either case). We would like to relate α somehow to μ_b, and ultimately relate the entire thing to $\delta(n)$. Working in this direction, we have:

$$
\begin{aligned}
\mu_b &= \Pr[(A(f(x), Z, u) = 1] \\
&= \frac{1}{2^r} \Pr[A(f(x), Z, x \cdot Z) = 1] \\
&= \frac{2^r - 1}{2^r} \Pr[A(f(x), Z, u) = 1 | u \neq x \cdot Z] \\
&= \frac{2^r - 1}{2^r}(1 - \alpha)
\end{aligned}
$$

Combining this with what we had before, we have:

$$
\begin{aligned}
\Pr[S'(f(x), Z) = x \cdot Z] &= \frac{1}{2^r}\mu_d + \alpha \\
&= \frac{1}{2^r}\mu_d + \frac{1}{2^r}(1 - \frac{2^r \mu_d - \mu_b}{2^r - 1}) \\
&= \frac{1}{2^r} + (\frac{1}{2^r} + \frac{1}{2^r - 1})\mu_d - \frac{1}{2^r - 1}\mu_b \\
&= \frac{1}{2^r} + \frac{1}{2^r - 1}\mu_d - \frac{1}{2^r - 1}\mu_b \\
&= \frac{1}{2^r} + \frac{\delta(n)}{2^r - 1}
\end{aligned}
$$

$\square$

Therefore, by changing the algorithm from S to S', we can slightly improve our chances of getting $x \cdot Z$ correctly.

3 Final steps to PRG

We have shown that if the extra bits generated by R can be distinguished from uniform, then there is a procedure A' which with small probability actually produces $x \cdot Z$ from $(f(x), Z)$. We would like to show that we can use this small advantage to compute $x \cdot z$ from $(f(x), z)$ with a non-negligible advantage over a random coin flip.

The first idea we might have is to choose a random matrix Z and insert the vector z into a column of our choosing. We may then recover $x \cdot z$ by running the algorithm A' to discover $x \cdot Z$ and reporting the bit in the position corresponding to the column in which z was inserted. However, from the discussion in the previous section we can see that using this approach only succeeds with probability of about $\frac{1}{2^r} + \frac{\delta(n)}{2^r-1}$. Since we are only computing one bit, we would be much better off simply flipping a coin!

A better approach would be to choose a uniformly distributed Z in such a way that when A' successfully discovers all of the bits of $x \cdot Z$, we are able to recover the bit $x \cdot z$, but when it is unsuccessful, the answer given by the recovery process is similar to that of a coin flip. If we can find such an approach, we can see that it will have a correctness probability of around $\frac{1}{2} + \frac{\delta(n)}{2^r}$. We now formally describe this process.

Theorem 3.1. *Suppose there is an adversary A' such that*

$$\Pr[A'(f(x), Z) = x \cdot Z] = \frac{1}{2^r} + \frac{\delta(n)}{2^r - 1}$$

where $x \in \{0,1\}^n, Z \in \{0,1\}^{r \cdot n}$. Then there is a polynomial time computable function P such that

$$\Pr[P(f(x), z) = x \cdot z] \geq \frac{1}{2} + \frac{\delta(n)/2}{2^r - 1}.$$

Proof. To create Z as described above, we need to construct it so that there is a procedure which correctly outputs $x \cdot z$ on input $x \cdot Z$, but whose output is close to uniform when given any $b \neq x \cdot Z$. To do this, we pick a random column ℓ and insert z into this row, and fill the remaining rows with completely random vectors. Then we add a random linear combination of the $r-1$ random columns to column ℓ, and record the coefficients of this linear combination in a bit vector $\alpha \in \{0,1\}^r$ (setting $\alpha_\ell = 1$). Since adding this same linear combination again to column ℓ will yield z, we have that $z = Z \cdot \alpha$. Therefore, if A' successfully returns $x \cdot Z$, taking the inner product of this vector and α yields

$$(x \cdot Z)\alpha = x \cdot (Z \cdot \alpha) = x \cdot z,$$

which is exactly the bit we desire in this case.

However, on any vector $b \neq x \cdot Z$, what does the output bit look like? To answer this, note that for every α, Z is uniformly distributed. Therefore, Z is independent of α, which implies that α is independent of Z! Since the output b of A' depends only on x and Z, this indicates that $b \cdot \alpha$ will be very close to uniformly distributed (it is not exactly uniform since we know that $\alpha \neq 0$ from setting $\alpha_\ell = 1$) . Thus when A' gives the wrong value for $x \cdot Z$, we are not much worse off than flipping a coin, but we still gain an advantage from when it does give the correct value. Therefore, the Z constructed as above meets our needs.

We now give a more formal description of the algorithm P which constructs such a Z, queries A', and reconstructs the bit $x \cdot z$ from the answer $x \cdot Z$ and α.

> $P(Y, z)$
> **Description:** Algorithm that uses A' to compute $x \cdot z$ with non-negligible advantage when $Y = f(x)$. Used in proof of theorem.
> Pick $\ell, i \in_u \{1, ..., r\}$.
> Pick $\alpha_1, ..., \alpha_{i-1}, \alpha_{i+1}, ...\alpha_r \in_u \{0,1\}^r$.
> Let $\widehat{\alpha} = (\alpha_1, ..., \alpha_{i-1}, 1, \alpha_{i+1}, .., \alpha_r)$.
> Pick $z_1, ..., z_{\ell-1}, z_{\ell+1}, ..., z_r \in_u \{0,1\}^n$.
> Set $z_l = z \cdot \sum_{i=1, i\neq\ell}^{r} \alpha_i z_i$ (with everything done $\pmod 2$).
> Let Z be a $r \times n$ bit matrix with i^{th} column z_i.
> Run $A'(f(x), Z)$ to get bits $(b_1, ..., b_r)$.
> **return** $(b_1, ..., b_r) \cdot (\alpha_1, ..., \alpha_r)$.

Given the discussion above, we calculate the success probability of P by conditioning on the success of A'. When A' is successful, which happens with probability $\frac{1}{2^r} + \frac{\delta(n)}{2^r - 1}$, P succeeds with probability 1. However, when A' is not successful, P succeeds with probability $\frac{2^{r-1}-1}{2^r-1}$. To see this, suppose A' outputs $b \neq x \cdot Z$, so that P outputs the correct answer when $b \cdot \alpha = (x \cdot Z) \cdot \alpha$. Equivalently, we have the correct answer when $(b - x \cdot Z) \cdot \alpha = 0$ for $b - x \cdot Z \neq 0$. Since we choose α uniformly over all nonzero vectors, there are $2^{r-1} - 1$ correct answers out of the $2^r - 1$ possible α's. Using these ideas, the analysis follows simply:

$$\begin{aligned}
\Pr_{x,z,\alpha,Z}[P(f(x), z) = x \cdot z] &= \Pr[(b_1, ..., b_r) = x \cdot z] \Pr[P(f(x), z) = x \cdot z | (b_1, ..., b_r) = x \cdot z] + \\
&\quad \Pr[(b_1, ..., b_r) \neq x \cdot z] \Pr[P(f(x), z) = x \cdot z | (b_1, ..., b_r) \neq x \cdot z] \\
&= \Pr[(b_1, ..., b_r) = x \cdot z] \cdot 1 + \\
&\quad \Pr[(b_1, ..., b_r) \neq x \cdot z] \Pr_{\alpha}[\alpha \cdot (b_1, ..., b_r) = x \cdot z | (b_1, ..., b_r) \neq x \cdot z] \\
&= (\frac{1}{2^r} + \frac{\delta(n)}{2^r - 1}) + (1 - \frac{1}{2^r} - \frac{\delta(n)}{2^r - 1})(\frac{2^{r-1} - 1}{2^r - 1}) \\
&= \frac{1}{2} + \delta(n)(\frac{1}{2^r - 1} - \frac{2^{r-1} - 1}{(2^r - 1)^2}) \\
&= \frac{1}{2} + \delta(n)(\frac{(2^r - 1) - (2^{r-1} - 1)}{(2^r - 1)^2}) \\
&= \frac{1}{2} + \delta(n)(\frac{1}{2} \frac{2^r}{(2^r - 1)^2}) \\
&\geq \frac{1}{2} + \frac{\delta(n)/2}{2^r - 1}
\end{aligned}$$

□

Notice that for the small advantage of $\frac{1}{2^r}(1+\frac{\delta(n)}{2^r-1})$ to be non-negligible, we need that $r = O(\log n)$. By the above discussions, we can now state that R is a PRG when f is a one-way function.

COROLLARY 3.2. *If f is a one-way function, then R is a PRG. In other words, for any polynomially bounded adversary A,*

$$\delta(n) = |\Pr[A(R(x,Z)) = 1] - \Pr_{y \in_u \{0,1\}^{n+rn}}[A(y) = 1]|$$

is negligible for $r = O(\log n)$.

4 BBS generator

We now give a concrete example of a psuedorandom generator whose security is based on the complexity-theoretic assumption that the quadratic residue decision problem (QR) is one-way for composite n. Since QR can be shown to be as hard as factoring, it is likely that QR is hard.

Specifically, let p and q be primes congruent to 3 (mod 4). Recall that the Legendre symbol $(\frac{x}{p})$ for a prime p is: 0 if $x|p$, 1 if x is a QR $\pmod p$, and -1 if x is a quadratic non-residue. For primes $p, q \equiv 3 \pmod 4$, the Legendre symbol of $-1 \pmod p$, $\pmod q$ is both -1. Let $n = pq$, and consider the field $\mathbb{Z}_n^*$. By the Chinese Remainder Theorem, we can think of every number x^2 as a pair of numbers $\pmod p$, $\pmod q$. Since each of these numbers has two square roots $\pm x$ $\pmod p$, $\pmod q$, this implies that x^2 has four square roots $\pmod n$.

Now consider the Jacobi symbol of $x \pmod n$, which can be computed as

$$(\frac{x}{n}) = (\frac{x}{p})(\frac{x}{q}).$$

This can also be computed efficiently without knowing the factors (via quadratic reciprocity). Notice that x has a square root $\pmod n$ if and only if it has a square root $\pmod p$ and $\pmod q$, namely when $(\frac{x}{p}) = (\frac{x}{q}) = 1$. Therefore, it is easy to decide the QR problem on half of the numbers $\pmod n$, namely, those with $(\frac{x}{n}) = -1$, since for this to happen, exactly one of $(\frac{x}{p})$ and $(\frac{x}{q})$ is -1 (and thus x is not a QR). However, it seems to be hard to distinguish between the two cases where $(\frac{x}{n}) = 1$, which are when $(\frac{x}{p}) = (\frac{x}{q}) = 1$ and x is a QR $\pmod n$, and when $(\frac{x}{p}) = (\frac{x}{q}) = -1$ and x is not a QR $\pmod n$. Thus, when we talk about the QR problem $\pmod{n = pq}$, we restrict to the set S of $x \in \mathbb{Z}_n^*$ with $(\frac{x}{n}) = 1$.

Now note that $x \in S$ if and only if $-x \in S$, since $1 = (\frac{-1}{n}) = (\frac{-1}{p})(\frac{-1}{q})$. We know that x and $-x$ have opposite parity and exactly one of them is a square. Therefore, we know that if x is a QR, then exactly one of its roots is a QR. We will use this when showing that the BBS generator is a PRG.

The BBS generator takes an input x and repeatedly squares it, taking the parity at each step as an output bit. Namely, given input x, the sequence

$$x_1 = x^2 \pmod n, x_2 = x_1^2 \pmod n, \ldots, x_i = x_{i-1}^2 \pmod n, \ldots$$

is generated, and the output string is $(x_1 \pmod 2, x_2 \pmod 2, \ldots, x_i \pmod 2, \ldots)$.

THEOREM 4.1. *Let p, q be primes $\equiv 3 \pmod 4$, and $n = pq$. If squaring $\pmod n$ is one-way, then the BBS generator is a PRG.*

Proof. We show that BBS is next-bit unpredictable. We have seen in a previous lecture that this implies BBS is a PRG.

We need to show how to use a next-bit predictor for the generator to determine if y is a QR $\pmod n$. Suppose we have a predictor A' which predicts the bit $x_i \pmod 2$ given input $(x_{i+1} \pmod 2, x_{i+2} \pmod 2, \ldots)$ with non-negligible advantage. Then given input y, we use the "sidling in" technique– generate the sequence $(y_{i+1} = y^2, y_{i+1} = y_{i+1}^2, \ldots)$ and give the parity bits of this sequence to the predictor A'. A', if successful, will return the parity of y_i, the square root of y_{i+1}. However, by how the BBS sequence is constructed, the y_i whose parity A' is predicting is itself a square. We are essentially trying to decide if either y or $-y$ is a square. We know that one of them is a square, and one of them is not, and that they have different parity. A' gives us the parity of the square root of y_{i+1} which is itself a square. After asking A' for the parity of y_i, we guess that y is a square only if this matches the parity of y (otherwise we guess y is not a QR). If A' has told us the correct parity of y_i, then we are correct with probability 1, and if it has lied, then we are incorrect with probability 1. Our success probability, then, is the same as the success probability of A', and that was assumed to be non-negligible. $\square$