Notes for Lecture 3

1 Last Time

Last time, we first defined pseudorandom functions (PRFs) and pseudorandom generators (PRGs) and began the proof of showing how to construct a PRF from a PRG. The first step was showing how to create a length-doubling PRG from a 1-bit expanding PRG, which was done with a hybrid argument. We ended by giving a construction for using a length-doubling PRG to create a PRF.

2 Proof of Length-Doubling PRG $\rightarrow$ PRF

As a reminder, the construction for the PRF PRF is as follows. Let G be a length-doubling PRG and k be the key to the PRF, with length λ.

It takes the key k and applies G. The result is a 2λ-bit string. PRF splits the string in half into two λ-bit strings. Then it applies G separately to both halves, obtaining a 4λ-bit string. PRF splits this into 4 λ-bits strings, and applies G to each string. It continues in this way for $n(\lambda)$ steps, for any desired polynomial $n(\lambda)$.

The result is a $2^{n(\lambda)} \times \lambda$-bit string. PRF will interpret this string as $2^{n(\lambda)}$ separate λ-bit strings. The output of PRF on input x with be the xth string in this list. Therefore, PRF has inputs of length $n(\lambda)$ and outputs of length λ.

It will be useful to think of the PRF computation as a tree: at the root is the PRF key k. The children of a node containing x are the first and second half of $G(x)$. The tree has $n+1$ levels and $2n$ leaves. The leaves are the outputs of the PRF.

As described, PRF runs in time roughly $2^{n(\lambda)}$, which is exponential. However, we can use the fact that it is "locally computable", such that it runs in polynomial time to compute any particular leaf. Indeed, instead of computing every node of the tree, we only have to compute the nodes along the path from the root to the particular leaf that we want.

Thus the time to compute a specific output of PRF is only $n(\lambda) \times \text{Time}(G)$. These are both polynomial, so the entire run time is polynomial. Now we prove the security of PRF.

The first natural idea is to construct a hybrid for each of the PRGs in the tree, which each subsequent hybrid replacing one of the remaining PRGs with random. However, there are $2^{n(\lambda)} - 1$ such PRGs, and thus we would get an advantage of $\epsilon/(2^{n(\lambda)} - 1)$, which is negligible. Thus, we need something a bit more clever. The idea is to hybrid over levels instead, of which there are $n(\lambda)$, which is polynomial.

Theorem 1 *If G is a secure PRG, then the construction above is a secure PRF.*

For the sake of contradiction suppose there is an adversary A and non-negligible ϵ such that

$$\left| \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_1(A, \lambda)] \right| \geq \epsilon(\lambda)$$

Define hybrids H_i for $i \in \{0, \ldots, n\}$ as follows. Let H_0 be the entire tree, with a random λ-bit string as input. Note that this is the entire construction PRF. For H_1, we replace the top PRG of the tree with 2 uniformly random strings, which serve as input to the next layer. For H_2, we replace the next level of 2 PRGs with 4 uniformly random strings. Similarly define H_i up to H_n, and note that hybrid H_n is simply 2^n random strings, i.e. it is a uniformly random function.

Then we see that A distinguishes between H_0 and H_n, so there must exist some $i \in \{1, \ldots, n\}$ such that

$$\left| \Pr[1 \leftarrow H_{i-1}(A, \lambda)] - \Pr[1 \leftarrow H_i(A, \lambda)] \right| \geq \frac{\epsilon(\lambda)}{n(\lambda)}$$

However, this is not enough as a PRG adversary, since there is more than one PRG in each level. Thus, we will need to hybrid again between levels in order to isolate a single PRG output from random. However, we again cannot creat a sequence of hybrids for each PRG in the level, since this may still be exponential.

The idea is to note that adversaries must run in polynomial time, and thus can only make a polynomial number of queries. That is, A can make at most $q(\lambda)$ queries, where q is polynomial. Furthermore, because the tree is locally computable, A will only see the outputs of the PRGs in the tree that are along the path for a query.

In particular, on level i, only a polynomial number of PRGs are able to be simulated by the adversary, and thus we only need a polynomial number of hybrids across this level. Note that this assumes the adversary "commits" to certain nodes, but it is possible to make this work even for adaptive queries: you can create samples of PRG outputs beforehand, and use them as necessary.

Thus, we have obtained a PRG adversary B such that

$$\left| \Pr[1 \leftarrow B(G(x)) : x \leftarrow \{0,1\}^\lambda] - \Pr[1 \leftarrow B(y) : y \leftarrow \{0,1\}^{2\lambda}] \right| \geq \frac{\epsilon(\lambda)}{n(\lambda)q(\lambda)}$$

which is non-negligible because n and q are polynomial. This finishes the proof.

To combine everything together, what we have is the chain of constructions

$$\text{1-bit PRG} \rightarrow \text{length-doubling PRG} \rightarrow \text{PRF} \rightarrow \text{encryption}$$

3 OWP $\rightarrow$ 1-bit PRG

First we define one-way functions (OWFs) and one-way permutations (OWPs).

Definition 2 *A one-way function is a deterministic poly-time function $f : \{0,1\}^\lambda \rightarrow \{0,1\}^{n(\lambda)}$ such that for all PPT adversaries A, there exists a negligible ϵ such that*

$$\Pr[f(A(f(x))) = f(x) : x \leftarrow \{0,1\}^\lambda] < \epsilon(\lambda)$$

That is, given the OWF output of a random input x, it is difficult to find any pre-image of that output.

Definition 3 *A one-way permutation is a OWF that is bijective and has $n(\lambda) = \lambda$.*

Note that this means an adversary has to find the actual random input x, since this is the only pre-image.

It is known that OWFs can be used to create a PRG with a 1-bit stretch, but this is beyond the scope of this course. Instead, we will prove the specific case of using OWPs to create such a PRG. To do so, we define another notion of a "hardcore bit":

Definition 4 *Let $f : X \rightarrow Y$ be a OWF. A hardcore bit for f is a function $hc : X \rightarrow \{0,1\}$ such that for all PPT adversaries A, there exists a negligible ϵ such that*

$$\left| \Pr[1 \leftarrow A(f(x), hc(x)) : x \leftarrow X] - \Pr[1 \leftarrow A(f(x), b) : x \leftarrow X, b \leftarrow \{0,1\}] \right| \geq \epsilon(\lambda)$$

That is, the hardcore bit looks like a random bit, even when the adversary gets to see $f(x)$ for the same input x.

We first see that if f is a OWP and it has a hardcore bit hc, then $G(x) = (f(x), hc(x))$ is a PRG with 1-bit stretch. The 1-bit stretch is satisfied because f is a permutation, and security is immediately seen by the above definition of a hardcore bit. This is harder if f is only a OWF instead, since the size of the range is no longer guaranteed, but various tricks can be done to do this more general case as well.

Thus we see that it remains to show that any OWF has a hardcore bit.

Question from student: Would the XOR of all the bits in x and all the bits in $f(x)$ be hardcore for all OWPs?

Observation 1: XORing with the bits in $f(x)$ is redundant, since the adversary is given $f(x)$. Thus the adversary could just XOR $hc(x)$ with the bits of $f(x)$ to obtain the parity of only x.

Observation 2: For any fixed function hc, there exists a OWF/OWP f such that hc is not hardcore for f. This can be proved in general, but for parity of x in particular, for a OWF f we can define f' such that $f'(x) = f(x), hc(x)$. This is a OWF (otherwise it can be used to create an inverter for f by simply guessing the extra bit), and hc is not hardcore for f' (since $hc(x)$ is always equal to the last bit).

Now we return to the main theorem we want to prove, i.e. that any OWF has a hardcore bit.

Theorem 5 *(Goldreich-Levin). Let $F : \{0,1\}^n \to Y$ be a OWF. Define $F' : \{0,1\}^{2n} \to \{0,1\}^n \times Y$ such that $F'(x,r) = r, F(x)$. Then $\langle x, r\rangle = \sum x_i r_i \pmod 2$ is hardcore for F'.*

Note: F' is also a OWF, but this theorem does not give a hardcore bit for the original function F, but the idea remains.

For the sake of contradiction assume there is a PPT adversary A that breaks the hardcore bit, i.e. that there exists non-negligible ϵ such that

$$\left| \Pr[A(r, F(x), \langle r, x\rangle) = 1] - \Pr[A(r, F(x), b) = 1] \right| \geq \epsilon(n)$$

where randomness is taken over $r \in \{0,1\}^n$, $x \in \{0,1\}^n$, and $b \in \{0,1\}$.

Without loss of generality we take away the absolute values, and then we can construct a PPT B such that

$$\Pr[B(r, F(x)) = \langle r, x\rangle] \geq \frac{1}{2} + \epsilon(n)$$

To do so, $B(r, y)$ just runs $A(r, y, b)$ where b is chosen randomly. If A outputs 1 then B outputs b, otherwise if A outputs 0 then B outputs $1 - b$. The intuition is that A outputs 1 with slightly higher probability if the guess is correct, and B is using A to test if the random b is correct.

We now prove the theorem by using B to break the security of the OWF F. We successively decrease the values of ϵ, thereby decreasing the strength of B and making the problem harder.

Case 1: $\epsilon = 1/2$. Then we have that $B(r, F(x)) = \langle r, x\rangle$ always. Let e_i be the ith "standard basis" bitstring, i.e. with 0s everywhere but the ith bit. Then we see that $B(e_i, F(x)) = \langle e_i, x\rangle = x_i$, and thus running over all e_i will completely recover x.

What if $\epsilon < 1/2$? We see that B could possibly fail on one of the e_i, so we cannot use fixed inputs.

Case 2: $\epsilon = 1/2 - 1/2n$. Now instead we choose uniformly random bitstrings $r_1, \ldots r_n$ to calculate $\langle r_i, x \rangle$. For each r_i there is a $1 - 1/2n$ chance of success, i.e. $1/2n$ chance of failure, so by union bound there is a $1/2$ chance that they all succeed. Assuming success, we can then invert to recover x again.

What if $\epsilon < 1/2 - 1/2n$? We see that there is high probability that at least one of the evaluations of B is incorrect, but it is not known which ones are incorrect. Presumably this makes recovering x impossible, so we need to work a bit harder.

Case 3: $\epsilon = 1/4 + \gamma$, where γ is non-negligible. Using the Markov Inequality, we claim that

$$\Pr_{x \in \{0,1\}^n} \left[\Pr_{r \in \{0,1\}^n} [B(r, F(x)) = \langle r, x \rangle] \geq \frac{3}{4} + \frac{\gamma}{2} \right] \geq \frac{\gamma}{2}$$

The idea is that the two probabilities sum to the total probability $3/4 + \gamma$ of B being correct.

Call x *"good"* if $\Pr_{r \in \{0,1\}^n} [B(r, F(x)) = \langle r, x \rangle] \geq \frac{3}{4} + \frac{\gamma}{2}$ for that choice of x. The above says that the probability of a random x being good is at least $\gamma/2$, which in particular is non-negligible.

Now assume that x is good. Let $H(r) = B(r, F(x))$ and consider again the "standard basis" bitstrings e_i. For each i, choose a random string $r \in \{0,1\}^n$ and run $H(r) \oplus H(r \oplus e_i)$. Note that $r \oplus e_i$ is random because r is random, and thus both inputs to H are individually random.

Also note that if both evaluations of H are correct, then we have that

$$H(r) \oplus H(r \oplus e_i) = \langle r, x \rangle \oplus \langle r \oplus e_i, x \rangle = \langle e_i, x \rangle = x_i$$

which is what we want.

We can then prove (left as an exercise) that

$$\Pr[H(r) \oplus H(r \oplus e_i) = x_i] \geq \frac{1}{2} + \gamma$$

Then our algorithm would be: for each i, compute many guesses for the value of x_i and choose the majority. The intuition is that the majority is correct with high probability, since each iteration has slightly higher than $1/2$ probability of being correct.

4 Next time

Next time we consider when ϵ is simply non-negligible to prove the general theorem.