University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 12

1 More on Improving the Stretch of a PRG

Two more comments on our result from last time which showed how to take a PRG G that stretches by one bit and construct from it a PRG H that stretches by two bits.

Fixed-length inputs to G. Our construction was $H(x) = G(G(x))$. This construction has the annoying property that G is being applied to inputs of different lengths. Even though this is technically fine (since G was defined for inputs in $\{0,1\}^*$) in practice we may be given a function G that only works for fixed input lengths (we will see that this is the case when we discuss block ciphers later in the semester). It may also be more efficient to consider G that operates on fixed-length inputs only.

We can modify the construction as follows: for any string x, let $x_{1\ldots\ell}$ denote the first ℓ bits of x and let x_i denote the i^{th} bit of x. If $|x| = k$, we can re-write the above construction as $H(x) = G(G(x)_{1\ldots(k+1)})$. (Since $|G(x)| = k+1$, we have $G(x)_{1\ldots(k+1)} = G(x)$.) But we can also consider the alternate construction $H'(x) = G(G(x)_{1\ldots k}) \circ G(x)_{k+1}$. Here, we first compute $y = G(x)$ and then run the first k bits of y through G a second time, but leave the last bit of $G(x)$ unchanged. It should be clear that the proof of security we gave in the last lecture would apply here also to show that H' is a PRG (see if you can give the necessary modifications to the proof!).

Stretching input by an arbitrary amount. It should also be clear that the construction above may be iterated (giving $H(x) = G(G(\cdots G(x)\cdots)))$ to obtain a PRG that stretches its input even more. A careful proof of security shows that G can be iterated any *polynomial* number of times and H will still be a PRG (it doesn't even make sense to iterate more than this, since in that case computing H requires exponential time).

Let's actually state this as a theorem for future reference.

Theorem 1 *If a PRG exists that stretches its input by one bit, then a PRG exists that stretches its input by any polynomial amount. (More formally, for any desired (polynomial-size) "stretch factor", we can construct a PRG having that stretch factor.)*

Coming full circle, note that this gives us the "ultimate" improvement over the one-time pad! If Alice and Bob share a reasonable number of bits (enough so that exhaustive search is infeasible; more formally, enough so that $\epsilon(k)$ — where k is the number of bits shared and $\epsilon(\cdot)$ is the function capturing the adversary's advantage in "breaking" the PRG — is small enough) then they can essentially encrypt an arbitrarily-long message, and do much, much better than the one-time pad.

Before we get too excited, though, let's notice that there is still plenty of work to be done. First, we need to construct a PRG that stretches its input by even one bit (we give such a construction below). But even once we do this, note that so far we only have secure encryption when a *single* message (albeit a long one) is sent. Obtaining security when multiple messages are sent (which corresponds to security against chosen-plaintext attacks) will require more work.

2 Constructing a PRG

As we mentioned above, although Theorem 1 shows that a PRG stretching by one bit is enough to construct a PRG stretching by many bits, and thus is enough to solve the problem of one-time encryption, we have not yet shown that PRGs stretching by one bit exist! This is the problem we deal with now.

To motivate the construction, let's take an informal detour and consider the property of *unpredictability.* We'll call a function $G : \{0,1\}^* \to \{0,1\}^*$ *unpredictable* if, for any ℓ, given $G(x)_{1\ldots\ell}$ (for a random x), it is "hard" to predict $G(x)_{\ell+1}$ with probability significantly better than $1/2$. In other words, given the first ℓ bits of $G(x)$, it is "hard" to predict the $(\ell+1)^{\text{th}}$ bit of $G(x)$ (one can think of this as a game, in which bits of $G(x)$ are revealed one-by-one and at some point you make a guess as to the next bit to be revealed). You should be able to take the above definition and make it more formal, in terms of PPT algorithms and negligible quantities.

It is not hard to see, at least informally speaking, that G is a PRG if and only if G is unpredictable. Actually, showing that a PRG is unpredictable is relatively easy; it requires a bit more work to show that an unpredictable function is in fact a PRG. In any event, this notion of unpredictability might prompt us to try to tie in the computational hardness of one-way permutations in the following sort of construction: $G(x) = f(x) \circ x$. What is the intuition here? Well, let $|x| = k$. Certainly (since f is a permutation), you cannot predict the i^{th} bit of $G(x)$ given the first $i-1$ bits for $i \leq k$ (make sure you see why). Furthermore, even given the first k bits does not allow you to "predict" the rest of the output x; this seems to follow from the difficulty of inverting a one-way permutation.

Unfortunately, this naive approach is wrong. Clearly $G(x) = f(x) \circ x$ is *not* a PRG (can you construct and analyze an explicit algorithm that distinguishes the output of G from random?). And in some sense this is because unpredictability fails for, say, the final bit: given the first $2k-1$ bits, it is easy to predict the final bit because you just try both 0 and 1 and see which satisfies "f applied to the second half equals the first half". So we need to do a little better.

Actually, a little thought shows that unpredictability may even fail for the $(k+1)^{\text{th}}$ bit; that is, given $f(x)$ (i.e., the first k bits of $G(x)$) it may be "easy" to determine the first bit of x. For example, let f' be a one-way permutation and define f as: $f(b \circ x) = b \circ f'(x)$. You should be able to prove that f is a one-way permutation, yet clearly it is easy, given y, to determine the first bit of $f^{-1}(y)$.

For a more natural example of how one-way permutations do not necessarily hide information about the pre-image, consider the RSA permutation $f_{N,e}(x) = x^e \bmod N$. Since $\mathcal{J}_N(x) = \mathcal{J}_N(x^e)$ (when $\gcd(e, \varphi(N)) = 1$ as is true for the case of RSA), one can compute — given y — the value $\mathcal{J}_N(f_{N,e}^{-1}(y))$, and not all partial information about the pre-image of y is hidden.

Hopefully, this discussion has motivated the following definition:

Definition 1 *Let f be a one-way permutation and let $h : \{0,1\}^* \to \{0,1\}$ be an efficiently computable function. We say that h is a* hard-core bit *of f if the following holds: For all* PPT *algorithms A, the probability that A can correctly compute $h(x)$ given $f(x)$ is not significantly greater than $1/2$. More formally,*

$$\left|\Pr[x \leftarrow \{0,1\}^k; y = f(x) : A(y) = h(x)] - 1/2\right| = \epsilon(k),$$

for some negligible function $\epsilon(\cdot)$.

Do hard-core bits exist in general? The following important theorem (due to O. Goldreich and L. Levin) states that they do:

Theorem 2 *Every one-way permutation f has a hard-core bit h.*

(The theorem also shows how to go about finding — with high probability — a hard-core bit for any f, so it is a constructive result.) We will not prove the theorem this semester, but if you are interested check the references on the web page or take graduate crypto in the spring!

Our problem has now been simplified somewhat: how do we construct a PRG that stretches by one bit, given a one-way permutation f with hard-core bit h? Hopefully the discussion to this point makes the following construction more intuitive: define $G(x) = f(x) \circ h(x)$. Notice that, in terms of unpredictability, it almost immediately follows that G is unpredictable (letting $|x| = k$, the first k bits of G are certainly unpredictable, and unpredictability of the last bit follows from the definition of a hard-core bit!). However, since we did not prove here that unpredictability implies pseudorandomness, we give a direct proof here.

Theorem 3 *Let f be a one-way permutation with hard-core bit h. Then $G(x) = f(x) \circ h(x)$ is a PRG.*

Proof Our proof will follow the by-now standard format:

1. Assume toward a contradiction that G is not a PRG.
2. This means that there is a PPT algorithm A for which:
$$\left|\Pr[x \leftarrow \{0,1\}^k; y = G(x) : A(y) = 1] - \Pr[y \leftarrow \{0,1\}^{k+1} : A(y) = 1]\right| = \delta(k), \quad (1)$$
where $\delta(\cdot)$ is not negligible.
3. We use A to construct a PPT algorithm A' which "breaks" the hard-core bit h of f (in the appropriate sense).

We consruct A' as follows; recall that A' gets input y and wants to compute $h(f^{-1}(y))$: $A'(y)$ picks a random bit b and runs $A(y \circ b)$. If A outputs 1 (representing a guess by A that the string $y \circ b$ is an output of G), then A' outputs b as its guess for $h(f^{-1}(y))$. If A outputs 0, then A' outputs $\bar{b}$.

Let's first re-write (1). We have (this derivation is presented in full detail, so make sure you understand each step):

$$\begin{aligned}
&\Pr[y \leftarrow \{0,1\}^{k+1} : A(y) = 1] \\
&\quad = \Pr[y' \leftarrow \{0,1\}^k ; b \leftarrow \{0,1\} : A(y' \circ b) = 1] \\
&\quad = \Pr[x' \leftarrow \{0,1\}^k ; b \leftarrow \{0,1\} ; y' = f(x') : A(y' \circ b) = 1] \\
&\quad = \Pr[x' \leftarrow \{0,1\}^k ; b \leftarrow \{0,1\} ; y' = f(x') : A(y' \circ b) = 1 \mid b = h(x')] \cdot 1/2 \\
&\qquad + \Pr[x' \leftarrow \{0,1\}^k ; b \leftarrow \{0,1\} ; y' = f(x') : A(y' \circ b) = 1 \mid b \neq h(x')] \cdot 1/2 \\
&\quad = 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = G(x) : A(y) = 1] \\
&\qquad + 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ \overline{h(x)}) = 1].
\end{aligned}$$

Plugging this in to (1) gives:

$$\begin{aligned}
\delta(k) \quad = \quad & \Big| 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = G(x) : A(y) = 1] \\
& - 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ \overline{h(x)}) = 1] \Big| . \qquad (2)
\end{aligned}$$

Now, we want to know the probability that A' correctly computes $h(f^{-1}(y))$. Using the definition of algorithm A' (and conditioning on whether the b that is chosen is equal to $h(f^{-1}(x))$ or not), we have:

$$\begin{aligned}
&\Big| \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A'(y) = h(x)] - 1/2 \Big| \\
&\quad = \Big| \Pr[x \leftarrow \{0,1\}^k ; y = f(x) ; b \leftarrow \{0,1\} : A(y \circ b) = 1 \mid b = h(x)] \cdot \Pr[b = h(x)] \\
&\qquad + \Pr[x \leftarrow \{0,1\}^k ; y = f(x) ; b \leftarrow \{0,1\} : A(y \circ b) = 0 \mid b \neq h(x)] \cdot \Pr[b \neq h(x)] - 1/2 \Big| \\
&\quad = \Big| 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ h(x)) = 1] \\
&\qquad + 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ \overline{h(x)}) = 0] - 1/2 \Big| \\
&\quad = \Big| 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ h(x)) = 1] \\
&\qquad + 1/2 \left(1 - \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ \overline{h(x)}) = 1]\right) - 1/2 \Big| \\
&\quad = \Big| 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = G(x) : A(y) = 1] \\
&\qquad - 1/2 \cdot \Pr[x \leftarrow \{0,1\}^k ; y = f(x) : A(y \circ \overline{h(x)}) = 1] \Big| .
\end{aligned}$$

And this is exactly $\delta(k)$ (cf. (2)). Thus, if A can "break" G (i.e., $\delta(k)$ is not negligible) then A' can correctly compute $h(x)$ given $f(x)$ with probability non-negligibly different from 1/2. And this is our desired contradiction, because h is a hard-core bit of f. ■