G22.3210-001/G63.2170 Introduction to Cryptography	September 30, 2008
Lecture 5	
Lecturer: Yevgeniy Dodis	*Fall 2008*

In this lecture we formalize our understanding of *next-bit security* and its relationship to pseudorandomness. Namely, we prove that next-bit security implies a PRG. On our way to this proof we introduce the important cryptographic idea of *computational indistinguishability* and the related technique of the *hybrid argument*. Having proved that functions $G(x)$ and $G'(x)$ (which we introduced in the last lecture and which are defined here in Equations (1) and (2)) are next-bit secure and therefor PRG's, we show that Equation (1) can be used to construct a PRG without our having to decide a length in advance. We look at two specific examples of such PRG's: the Blum-Micali generator and the Blum-Blum-Shub generator. Next we examine the relationship between PRG's and OWF's and come to the startling conclusion that asserting the existence of one of these primitives is equivalent to asserting the existence of the other. Finally we introduce the important idea of *forward security* for a PRG and discuss the role of PRG's in real life.

1 Next-Bit Unpredictability and PRG's

Last lecture we used the concept of *hardcore bits* to construct public- and secret-key cryptosystems whose security was plausible but unclear. Both of our constructions used a OWP (possibly trapdoor) f and its hardcore bit h and considered iterating $f(x)$ (for a random $x \in \{0,1\}^k$) n times, and output the hardcore bits of $f^i(x)$ in reverse order. It particular, we considered two functions $G' : \{0,1\}^k \to \{0,1\}^n$ and $G : \{0,1\}^k \to \{0,1\}^{k+n}$ defined as

$$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \quad (1)$$

$$G(x) = f^n(x) \circ G'(x) = f^n(x) \circ h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \quad (2)$$

Intuitively and by the analogy with the S/Key system, these functions seem to satisfy the following notion of security which we now define formally.

Definition 1 [Next-bit Unpredictability] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) satisfies the *next-bit unpredictability* property if for every index $0 \le i \le p(k)$ and every PPT next-bit predictor P

$$\Pr\left(b = g_i \;\middle|\; x \stackrel{r}{\leftarrow} \{0,1\}^k,\ g = g_1 \ldots g_{p(k)} = G(x),\ b \leftarrow P(g_1 \ldots g_{i-1})\right) < \frac{1}{2} + \mathsf{negl}(k)$$

Namely, no predictor P can succeed in guessing g_i from $G_{i-1} = g_1 \ldots g_{i-1}$ significantly better than by flipping a coin. $\diamondsuit$

We now formally show that $G(x)$ (and hence $G'(x)$ as well) satisfies this property.

Lemma 1 *If f is a OWP, h is a hardcore bit of f and G is defined by Equation (2), then G is next-bit unpredictable.*

Proof: The proof is almost identical to the one we used for the S/Key system. So assume for some i and some PPT P our function G is not next-bit unpredictable. We notice that we must have $i > k$, since otherwise g_i is part of a truly random $f^n(x)$ (remember, x is random), and is independent of $G_i = g_1 \dots g_{i-1}$. Thus, assume $i = k+j$ where $j > 0$. Thus,

$$\Pr\left(P(f^n(x), h(f^{n-1}(x)), \dots, h(f^{n-j+1}(x))) = h(f^{n-j}(x)\right) > \frac{1}{2} + \epsilon$$

We now construct a predictor A which will compute the hardcore bit $h(\tilde{x})$ from $\tilde{y} = f(\tilde{x})$. As with S/key, A simply outputs

$$P(f^{j-1}(\tilde{y}), h(f^{j-2}(\tilde{y})), \dots, h(\tilde{y}))$$

with the hope that P computes $h(f^{-1}(\tilde{y})) = h(\tilde{x})$. The analysis that the advantage of A is ϵ is the same as with the S/key example, and uses the fact that f is a permutation. $\square$

Having formally verified this, we ask the same question as before. Do $G(x)$ and $G'(x)$ in Equation (1) and Equation (2) really satisfy their purpose of being "computational one-time pads"? Last lecture we also intuitively argued that this means that $G(x)$ (resp. $G'(x)$) should really look indistingushable from a truly random string of length $n+k$ (resp. n). We then formalized this property by defining the notion of a *pseudo-random generator*.

DEFINITION 2 [Pseudorandom Generator] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) is called a *pseudorandom number generator* (PRG) if

1. $p(k) > k$ (it should be stretching).
2. There exists no PPT distinguishing algorithm D which can tell $G(x)$ apart from a truly random string $R \in \{0,1\}^{p(k)}$. To define this formally, let 1 encode "pseudorandom" and 0 encode "random". Now we say that for any PPT D

$$|\Pr(\ D(G(x)) = 1 \mid x \stackrel{r}{\leftarrow} \{0,1\}^k\ ) - \Pr(\ D(R) = 1 \mid R \stackrel{r}{\leftarrow} \{0,1\}^{p(k)}\ )| < \mathsf{negl}(k)$$

$\diamondsuit$

Thus, a *pseudorandom number generator* (PRG) stretches a short random seed $x \in \{0,1\}^k$ into a longer output $G(x)$ of length $p(k) > k$ which nevertheless "looks" like a random $p(k)$-bit strings to any computationally bounded adversary. For clear reasons, we call the adversary D a *distinguisher*.

Now, rather than verifying directly if our G and G' are PRG's, we will prove a much more suprising result. Namely, we show that *any* G which satisfies next-bit unpredictability is a PRG.

Theorem 1 *If an arbitrary $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ is next-bit unpredictable, then G is a* PRG. *More quantitively, if some* PPT *distinguisher for G has advantage ϵ is telling $G(x)$ apart from a random R, then for some index $1 \le i \le p(k)$ there is a* PPT *predictor for G which has advantage at least $\epsilon/p(k)$.*

We notice that the converse (PRG implies next-bit unpredictability) is obvious. Indeed, if some P break next-bit unpredcitability of some G at some index i, here is a distinguisher $D(y_1 \dots y_{p(k)})$:

Let $g = P(y_1 \dots y_{i-1})$.
If $g = y_i$ output 1 ("pseudorandom"), esle output 0 ("random")

Indeed, by assumption, if $y = G(x)$, then $\Pr(D(y) = 1) \geq \frac{1}{2} + \epsilon$. On a random string $y = R$, clearly $\Pr(D(y) = 1) = \frac{1}{2}$, since there is no way $P(R_1 \dots R_{i-1})$ can predict a totally fresh and independent R_i.

We give the proof of Theorem 1 in Section 3. The proof uses an extremely important technique called a *hybrid argument*. However, it is a somewhat technical to understand right away. Therefore, we step aside and introduce several very important concepts that will (1) make the proof of Theorem 1 less mysterious; (2) explain better the definition of a PRG by introducing the general paradigm of *computational indistiguishability*; (3) make new definitions similar to that of a PRG very easy to express and unserstand; and (4) introduce the hybrid argument in its generality. We will return to our mainstream very shortly.

2 Computational Indistinguishability + Hybrid Argument

The definition of OWF's/OWP's/TDP's had the flavor that

"something is hard to *compute* precisely"

We saw that this alone is not sufficient for cryptographic applications. The definition of a hardcore bit and subsequently of the next-bit unpredictability were the first ones that said that

"something is hard to *predict* better than guessing"

Finally, the definition of a PRG took a next crycial step by saying that

"something is *computationally indistinguishable* from being random"

Not surprisingly, we will see many more cryptographic concepts of a similar flavor where more generally

"something is *computationally indistinguishable* from something else"

Intuitively, "something" will often be the cryptographic primitive we are considering, while "something else" is the ideal (and impossible to achieve/very expensive to compute) object we are trying to efficiently approximate. In order to save time in the future and to understand this concept better, we treat this paradigm in more detail.

Definition 3 Let k be the security parameter and $X = \{X^k\}$, $Y = \{Y^k\}$ be two ensembles of probability distributions where the descrytpion of X^k and Y^k are of polynomial length in k. We say that X and Y are *computationally indistinguishable*, denoted $X \approx Y$, if for any PPT algorithm D (called the *distinguisher*) we have that

$$|\Pr(D(X^k) = 1) - \Pr(D(Y^k) = 1)| < \mathsf{negl}(k)$$

where the probability is taken over the coin tosses of D and the random choices of X^k and Y^k. The absolute value above is called the *advantage* of D in distinguishing X from Y, denoted $\mathsf{Adv}_D(X,Y)$. $\diamondsuit$

Notice that in this terminology the definition of a PRG $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ reduces simply to saying that for a random $x \in \{0,1\}^k$ and $R \in \{0,1\}^{p(k)}$, we have

$$G(x) \approx R$$

We give very simple properties of computational indistinguishability.

Lemma 2 *If $X \approx Y$ and g is polynomial time computable, then $g(X) \approx g(Y)$.*

Proof: Assuming a PPT distibguisher D for $g(X)$ and $g(Y)$, a PPT distinguisher $D'(z)$ for X and Y simply runs $D(g(z))$, which it can do since g is poly-time. Clearly, $\mathsf{Adv}_{D'}(X,Y) = \mathsf{Adv}_D(g(X), g(Y))$. $\square$

The next result, despite its simplicity is a foundation of a very powerful technique.

Lemma 3 *If $X \approx Y$ and $Y \approx Z$, then $X \approx Z$. More generally, if n is polynomial in k and $X_0 \approx X_1$, $X_1 \approx X_2$, $\ldots$, $X_{n-1} \approx X_n$, then $X_0 \approx X_n$. More quantitively, if some distinguisher D has $\mathsf{Adv}_D(X_0, X_n) = \epsilon$, then for some $1 \leq i < n$ we have that $\mathsf{Adv}_D(X_i, X_{i+1}) \geq \epsilon/n$.*

Proof: The proof is simple but is worth giving. We give it for the last quantitive version. Indeed, this implies the fact that $X_0 \approx X_n$, since if D has non-negligible advantage $\epsilon(k)$ on X_0 and X_n, then D has (still non-negligible as n is polynomial in k) advantage $\epsilon(k)/n$ on X_i and X_{i+1}, which is a contradiction.

To prove the result, let $p_i = \Pr(D(X_i) = 1)$. Thus, we assumed that $\mathsf{Adv}_D(X_0, X_n) = |p_n - p_0| \geq \epsilon$. But now we can use the following very simply algebra:

$$\begin{aligned}
\epsilon &\leq |p_n - p_0| \\
&= |(p_n - p_{n-1}) + (p_{n-1} - p_{n-2}) + \ldots + (p_2 - p_1) + (p_1 - p_0)| \\
&\leq |p_n - p_{n-1}| + |p_{n-1} - p_{n-2}| + \ldots + |p_2 - p_1| + |p_1 - p_0| \\
&= \sum_{i=0}^{n-1} |p_{i+1} - p_i|
\end{aligned}$$

Notice, we simply used algebraic manipulation and nothing else. However, now we see that for some index i, we have

$$|p_{i+1} - p_i| \quad \geq \quad \frac{\epsilon}{n}$$

$\square$

Despite its triviality, this lemma is very powerful in the following regard. Assume we wish to prove that $X \approx X'$, but X and X' look somewhat different on the first glance. Assume we can define (notice, its completely our choice!) $X_0 \ldots X_n$, where n is constant or even polynomial in k, s.t.

1. $X_0 = X$, $X_n = X'$.

2. For every $1 \le i < n$, $X_i \approx X_{i+1}$.

Then we conclude that $X \approx X'$. This simple technique is called the *hybrid argument*. The reason some people have difficulty in mastering this simple technique is the following. Usually X and X' are some natural distributions (say $G(x)$ and R, as in PRG example). However, the "intermediate" distributions are "unnatural", in a sense that they never come up in definitions and applications. In some sense, one wonders why a distinguisher D between natural X and X' should even work on these "meaningless" distributions X_i?

The answer is that D is simply an algorithm, so it expects some input. We are free to generate this input using any crazy experiment that we like. Of course, the behavior of D maybe crazy as well in this case. However, technically it has to produce some binary answer no matter how we generated the input. Of course a really malicious D may try to really do some crazy things if it *can tell* that we did something he does not expect (i.e., feed it X_i instead of X or X' as we were supposed to). But the point is that if for all i we have $X_i \approx X_{i+1}$, D really *cannot tell* that we generated the input according to some meaningless distributions.

As a simple example, we prove the following very useful theorem about PRG's which we call the composition theorem. Now you will see how simple the proof becomes: compare it with a direct proof!

Theorem 2 (Composition of PRG's) *If $G_1 : \{0,1\}^k \to \{0,1\}^{p(k)}$ and $G_2 : \{0,1\}^{p(k)} \to \{0,1\}^{q(k)}$ are two PRG's, then their composition $G : \{0,1\}^k \to \{0,1\}^{q(k)}$, defined as $G(x) = G_2(G_1(x))$, is also a PRG.*

Proof: We know that $G_1(x) \approx r$ and $G_2(r) \approx R$, where $x \in \{0,1\}^k$, $r \in \{0,1\}^{p(k)}$ and $R \in \{0,1\}^{q(k)}$ are all random in their domains. We have to show that $G(x) = G_2(G_1(x)) \approx R$. We use a hybrid argument and define an intermediate distribution $G_2(r)$. First, since G_2 is polynomial time and $G_1(x) \approx r$ (as G_1 is a PRG), then by Lemma 2 we have $G_2(G_1(x)) \approx G_2(r)$. Combining with $G_2(r) \approx R$ (as G_2 is a PRG), we use Lemma 3 (i.e., the hybrid argument) to conclude that $G_2(G_1(x)) \approx R$, i.e. that G a PRG. $\Box$

Finally, so far we said that $X_0 \approx X_1$ if no distingusiher D can "behave noticeably differently" when given a sample of X_0 as opposed to a sample of X_1. Here is anollowing equivalent view of this fact, stating that D can behave differently on X_0 and X_1 only if it effectively can tell whether or not it is given a sample of X_0 as opposed to sample of X_1 with probability noticeably different from 1/2.

Lemma 4 *$X_0 \approx X_1$ if and only if, for any efficient distingusisher D,*

$$\left| \Pr(D(Z) = b \mid b \xleftarrow{r} \{0,1\}, Z \xleftarrow{r} X_b) - \frac{1}{2} \right| \le \mathsf{negl}(k)$$

Proof: We have

$$\begin{aligned} \left| \Pr(D(Z) = b \mid b \xleftarrow{r} \{0,1\}, Z \xleftarrow{r} X_b) - \frac{1}{2} \right| &= \frac{1}{2} \cdot |\Pr(D(X_1) = 1) + \Pr(D(X_0) = 0) - 1| \\ &= \frac{1}{2} \cdot |\Pr(D(X_1) = 1) - \Pr(D(X_0) = 1)| \end{aligned}$$

$\square$

In the sequel we will interchangeably use both of these equivalent formulations of indistinguishability.

3 Next-Bit $\Rightarrow$ PRG (Proof of Theorem 1)

Before proving this result, let us introduce some useful notation. Assume the input x is chosen at random from $\{0,1\}^k$. Let $n = p(k)$, $G(x) = g_1 \dots g_n$ be the output of G, and $G_i = g_1 \dots g_i$ be the first i bits of $G(x)$. We also denote by R_s a truly random string of length s. We will also often omit the concatenation sign (i.e., write $G_i R_{n-i}$ in place of $G_i \circ R_{n-i}$).

We will split the proof into two steps. The first step uses the hybrid argument to reduce our problem to showing indistinguishability of n pairs of distributions, each related to some specific output bit $1 \le i \le n$ of G. Later we will show that each of these pairs is indeed indistinguishable by using the unpredictability of the corresponding bit i of G.

3.1 Stage 1: Hybrid Argument

Let us see what we have to show. We have to show that $G(x) \approx R$, which in our notation means $G_n \approx R_n$. We use the hybrid argument with the following intermediate distributions:

$$X_0 = R_n,\ X_1 = G_1 R_{n-1}\ , \dots,\ X_i = G_i R_{n-i}\ , \dots,\ X_{n-1} = G_{n-1} R_1\ ,\ X_n = G_n$$

More graphically,

$$\begin{array}{ccccccccccc}
R_n & = & \boxed{r_1} & r_2 & \dots & r_{i-1} & r_i & r_{i+1} & \dots & r_{n-1} & r_n \\
G_1 R_{n-1} & = & \boxed{g_1} & r_2 & \dots & r_{i-1} & r_i & r_{i+1} & \dots & r_{n-1} & r_n \\
\vdots & & & & \vdots & & & & \vdots & & \\
G_{i-1} R_{n-i+1} & = & g_1 & g_2 & \dots & g_{i-1} & \boxed{r_i} & r_{i+1} & \dots & r_{n-1} & r_n \\
G_i R_{n-i} & = & g_1 & g_2 & \dots & g_{i-1} & \boxed{g_i} & r_{i+1} & \dots & r_{n-1} & r_n \\
\vdots & & & & \vdots & & & & \vdots & & \\
G_{n-1} R_1 & = & g_1 & g_2 & \dots & g_{i-1} & g_i & g_{i+1} & \dots & g_{n-1} & \boxed{r_n} \\
G_n & = & g_1 & g_2 & \dots & g_{i-1} & g_i & g_{i+1} & \dots & g_{n-1} & \boxed{g_n}
\end{array}$$

Since $n = p(k)$ is polynomial in k and $X_0 = R_n$, $X_n = G_n$, by the hybrid argument we only have to show that for every $1 \le i \le n$, we have $G_{i-1} R_{n-i+1} \approx G_i R_{n-i}$. We will do it in the next step, but notice (see the table above) how similar these two distributions are: they are only different in the i-st bit. In other words, both of them are of the form $G_{i-1} b R_{n-i}$, where b is either the "next bit" g_i or a truly random bit r_i. Not surprisingly, the fact that they are indistinguishable comes from the unpredictability of g_i given G_{i-1}. Looking ahead, G_{i-1} is the legal input to our next bit predictor P, while R_{n-i} can be easily sampled by the predictor P itself!

3.2 Stage 1.5: defining the predictor

To complete the proof, we have to show that $G_{i-1}R_{n-i+1} \approx G_iR_{n-i}$. For this it suffices to show that if there exists a PPT distinguisher A for the above two distributions (that has non-negligible advantage δ), then there exists a PPT next-bit predictor P for g_i given G_{i-1}, which would contradict the next-bit unpredictability for G. So assume such A exists. Assume without loss of generality that $\Pr[A(G_iR_{n-i}) = 0] = q$, and that $\Pr[A(G_{i+1}R_{n-i-1}) = 0] = q + \delta$ (that is, we are assuming w.l.o.g. that A outputs 0 more often when the i-th bit is from G rather than random; if not, simply rename q to $1-q$ and swap 0 and 1 in the output of A). Now, some shortcuts in notation.

Whenever we will run A, the first $(i-1)$ bits come from the generator, last $(n-i)$ bits are totally random, i.e. only the i-th bit is different. So we denote by $A(b)$, $b \in \{0,1\}$, running $A(G_{i-1}bR_{n-i})$. Note, that when running $A(b)$ several times, we always leave the *same prefix* G_{i-1} that was given to us at the beginning, but always put brand new random bits in the last $(n-i)$ positions. Now we denote by $r \in \{0,1\}$ a random bit (to represent r_i) and by $g = g_i$ — the i-th bit of G, where the seed is chosen at random. Hence, we know that

$$\Pr(A(g) = 0) - \Pr(A(r) = 0) \geq \delta$$

Now, let us recap where we stand. We are trying to build P that will guess g. P can run $A(0)$ or $A(1)$. P knows that $A(g)$ is more likely to be 0 than $A(r)$ (for a random bit r). So how can P predict g? It turns out that there are several ways that work. Here is one of them.

P picks a random r and runs $A(r)$. If the answer is 0, it seems likely that $g = r$, since $A(g)$ is more likely to be 0 than $A(r)$. So in this case P guesses that $g = r$ (i.e. outputs the value of r). If, on the other hand, $A(r)$ returns 1, it seems like it is more likely that g is the compliment of r, so we guess $g = 1 - r$. This is our entire predictor, and let us call its output bit B. We wish to show that $\Pr[B = g] \geq \frac{1}{2} + \delta$.

To put our intuition differently, $A(r)$ *a-priori* outputs 0 less often than $A(g)$. Thus, if $A(r)$ returned 0, this gives us some *a-posteriori* indication that $r = g$.

3.3 Stage 2: proving our predictor is good

Let us now show that P works. The proof is quite technical. Keep in mind though, that what we are doing is simply an exercise in probability, our intuition is already in place!

Let $z = \Pr[g = 0]$ (where the probability is over random seed x). We introduce the following “irreducible” probabilities:

$$\beta_{jk} := \Pr[A(j) = 0 \mid g = k], \qquad j, k \in \{0,1\} \tag{3}$$

The reason that this probabilities are important is that we will have to analyze the expression $\Pr[P(G_{i-1}) = g]$, and therefore, will have to immediately condition on the value of g, i.e. $g = 0$ or $g = 1$. And since P runs A, the needed probability will indeed be some function of z and β_{jk}'s. We note that all 4 probabilities in (3) are generally different. Indeed, conditioning on a particular setting of g skews the distribution of the first $(i-1)$

bits G_{i-1}. We start by expressing our given probabilities in terms of "irreducible" ones (in both formulas in the last step we condition over $g = 0$ or $g = 1$):

$$\begin{aligned}
q &= \Pr[A(r) = 0] \\
&= \frac{1}{2}(\Pr[A(0) = 0] + \Pr[A(1) = 0]) \\
&\overset{(3)}{=} \frac{1}{2}(z\beta_{00} + (1-z)\beta_{01} + z\beta_{10} + (1-z)\beta_{11}) \\
q + \delta &= \Pr[A(g) = 0] \\
&\overset{(3)}{=} z\beta_{00} + (1-z)\beta_{11}
\end{aligned}$$

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

$$\delta = \frac{z(\beta_{00} - \beta_{10}) + (1-z)(\beta_{11} - \beta_{01})}{2} \tag{4}$$

Now, let us return to the analysis of P (recall, it chooses a random r, runs $A(r)$ and then decides if it output $B = r$ or $B = 1 - r$ depending on whether or not the answer is 0). The probabilities of P's success for a *fixed* $r = 0$ or $r = 1$ are:

$$\begin{aligned}
\Pr[B = g \mid r = 0] &= z \Pr[A(0) = 0 \mid g = 0] + (1-z)\Pr[A(0) = 1 \mid g = 1] \\
&\overset{(3)}{=} z\beta_{00} + (1-z)(1-\beta_{01}) \\
&= (1-z) + z\beta_{00} - (1-z)\beta_{01}. \\
\Pr[B = g \mid r = 1] &= z \Pr[A(1) = 1 \mid g = 0] + (1-z)\Pr[A(1) = 0 \mid g = 1] \\
&\overset{(3)}{=} z(1-\beta_{10}) + (1-z)\beta_{11} \\
&= z - z\beta_{10} + (1-z)\beta_{11}.
\end{aligned}$$

Hence, conditioning on random bit r, the overall probability of P's success is

$$\begin{aligned}
\Pr[B = g] &= \frac{1}{2}\Pr[B = g \mid r = 0] + \frac{1}{2}\Pr[B = g \mid r = 1] \\
&= \frac{1}{2}(z + (1-z) + z\beta_{00} - (1-z)\beta_{01} - z\beta_{10} + (1-z)\beta_{11}) \\
&= \frac{1}{2} + \frac{z(\beta_{00} - \beta_{10}) + (1-z)(\beta_{11} - \beta_{01})}{2} \\
&\overset{(4)}{=} \frac{1}{2} + \delta
\end{aligned}$$

This completes the proof. One remark is in place, though. Despite its technicality, the proof is quite intuitive. Unfortunately, it seems like the "right" expressions are magically appearing at the "right" places. This is just an illusion. There are several other intuitive predictors, and all come up to the same expressions. Unfortunately, having four probabilities β_{jk} indeed seems to be necessary.

4 Consequences

Having proven that next-bit unpredictability $\Rightarrow$ PRG, and using Lemma 1, we get

Corollary 5 *G' and G defined by Equation (1) and Equation (2) are* PRG*'s.*

Notice, however, that G' and G are very inconvenient to evaluate. Specifically, we (1) have to know n in advance, and (2) have to output the bits in reverse order, so that we have to wait for n steps before outputting the first bit. It would be much nicer if we could output the hardcore bits in the natural "forward order". But now, using Corollary 5, we can! Indeed, redefine

$$\begin{aligned} G'(x) &= h(x) \circ h(f(x)) \circ \ldots \circ h(f^{n-1}(x)) && (5) \\ G(x) = G'(x) \circ f^n(x) &= h(x) \circ h(f(x)) \circ \ldots \circ h(f^{n-1}(x)) \circ f^n(x) && (6) \end{aligned}$$

From the definition of a PRG, it is clear that the order of the output bits does not matter — a PRG remains pseudorandom no matter which order we output its bits. For the sake of exercise, let us show this formally for G' (similar argument obviously holds for G). Clearly, it suffices to show that

Lemma 6 *If $F(x) = g_1 \ldots g_n$ is a* PRG*, then so is $H(x) = g_n \ldots g_1$.*

Proof: Let $rev(g_1 \ldots g_n) = g_n \ldots g_1$. Since rev is poly-time computable, by Lemma 2 we have $H(x) \equiv rev(G(x)) \approx rev(R) \equiv R$, showing that H is a PRG. □

Theorem 3 *G' and G defined by Equation (5) and Equation (6) are* PRG*'s, provided f is a* OWP *and h is its hardcore bit.*

Notice, we can now evaluate G' and G much more efficiently. Simply keep state $s = f^i(x)$ after outputting i bits, then output bit $h(s)$ as the next bit, and update the state to $s = f(s)$. Moreover, to evaluate G' we do not even need to know n in advance! We can get as many bits out as we wish! On the other hand, the moment we are sure we do not need many more bits from G', we can simply output our current state $s = f^n(x)$ (for some n), and get the output of G instead. This will save us k evaluations of our OWP. However, it seems like using G' is still much better that using G since we can keep going forever (with G, we cannot go on with the current seed x the moment we reveal $f^n(x)$). The pseudo-code is summarized below.

Pick a random seed $x_1 \stackrel{r}{\leftarrow} \{0,1\}^k$;
repeat until no more bits are needed
  output next bit pseudorandom bit $b_i = h(x_i)$;
  update the seed $x_{i+1} := f(x_i)$;
  $i := i + 1$;
end repeat
If want the last chunk of k bits, output the current seed x_i;

5 Examples

We discuss two specific examples of pseudorandom generators induced from familiar OWP's.

5.1 Blum/Micali Pseudo-Random Generator

The Blum/Micali generator uses the candidate OWP $EXP_{p,g}(x)$ to generate a pseudo-random string. Namely, we recognize that if $x_{i+1} = g^{x_i} \bmod p$, the $MSB(x)$ is always hardcore.

5.2 Blum/Blum/Shub Pseudo-Random Generator

Next, we look at the Blum/Blum/Shub Generator, which uses the proposed OWF $SQ_n(x) = x^2 \bmod n$ where n is a Blum integer (ie the product of two primes p and q such that $p \equiv q \equiv 3 \bmod 4$. The restriction on n to be a Blum integer comes from the fact that $(x^2 \bmod n)$ becomes a *permutation* when restricted to the subgroup of quadratic residues QR_n of $\mathbb{Z}_n^*$. We mentioned that under the assumption that $SQ_n : QR_n \to QR_n$ is a OWP, the $LSB(x)$ is a hardcore bit for SQ_n, and this defined the Blum-Blum-Shub generator. Notice, $x_{i+1} = x^{2^i} \bmod n$. As you show in the homework, x_{i+1} is very easy to compute directly when given the factorization of n (i.e. without iterating SQ_n for i times). Also, each next bit requires only one modular multiplication. Finally, one can show that it is safe to use simultaneously even up to $\log k$ least significant bits of each x_i, making the BBS generator even more efficient.

By comparison we look at a linear congruential generator of the form based on $(f_n(x) = (ax + b) \bmod n)$, where a, b are chosen at random, which seems very similar but which in fact proves insecure. The contrast shows us the importance of building a general theory. We see that BBS is secure since $LSB(x)$ is a hardcore bit for SQ_n, while one of the reasons the once-popular linear congruential generator is insecure, is the fact that $LSB(x)$ is not its hardcore bit. Without this understanding, we would have hard time a-priori to say which generator is better.

6 PRG's and OWF's

First, we notice that

Lemma 7 *The existence of a* PRG *which stretches* $\{0,1\}^k \to \{0,1\}^{k+1}$ *implies the existence of a* PRG *which stretches* $\{0,1\}^k \to \{0,1\}^{p(k)}$

For example, this follows from the repeated iteration of the composition theorem Theorem 2: simply call G repeatedly on its own output for $p(k)$ times (notice, the adversary's advantage increases by a polynomial factor $p(k)$, and hence remains negligible). Thus we see that the *assumption* that PRG exists is *universal* and the *actual expansion is unimportant.*

But now we can ask the question of finding the necessary and sufficient conditions for the existence of PRG's. We proved that OWP $\Rightarrow$ PRG. In your homework you show that PRG $\Rightarrow$ OWF. There is a very celebrated result (omitted due to its extremely complicated proof) which shows that in fact OWF $\Rightarrow$ PRG. This gives us the dramatic conclusion that

Theorem 4 OWF $\iff$ PRG

This is extremely interesting because we see that two of the main primitives that we have introduced (namely OWF and PRG) are in fact equivalent despite their disparate appearances.

7 Forward Security

Next we introduce the important notion of *forward security*. First, recall the iterative construction of $G'(x)$ in Equation (5). Every time we output the next bit $b_i = h(x_i)$, we also update our state to $x_{i+1} = f(x_i)$. We also noticed that at every moment of time n, we can simply output $x_{n+1} = f^n(x)$ and still get a secure PRG $G(x)$. To put it in different context, our current state x_{n+1} looks completely uncorrelated with the previously output bits $G'(x)$. Namely, *even if we manage to loose or expose our current state, all the previously output bits remain pseudorandom*!

This is exactly the idea of forward-secure PRG's formally defined below.

Definition 4 [Forward Secure PRG] A forward-secure PRG with block length $t(k)$ is a poly-time computable function $F : \{0,1\}^k \to \{0,1\}^k \times \{0,1\}^{t(k)}$, which on input s_i — the current state at period $i \geq 1$ — outputs a pair (s_{i+1}, b_i), where s_{i+1} is the next state, and b_i are the next $t(k)$ pseudorandom bits. We denote by $N(s)$ the next-state function, by $B(s)$ the next $t(k)$ pseudorandom bits output, by $F_i(s_1) = B(s_1)B(s_2)\ldots B(s_i)$ the pseudorandom bits output so far, and by R_i — a random string of length $\{0,1\}^{i \cdot t(k)}$. We require for any $i < \mathsf{poly}(k)$ that when the initial state s_1 is chosen at random from $\{0,1\}^k$, we have

$$(F_i(s_1), s_{i+1}) \approx (R_i, s_{i+1})$$

$\diamondsuit$

For example, when used for symmetric key encryption, a forward-secure generator implies that loosing the current key leaves all the previous "one-time pad" encryptions secure.

We notice that our generic PRG G' from Equation (5) and its efficient implementation naturally leads to a forward-secure generator with block length 1: $F(s) = (f(s), h(s))$, i.e. the next state is $f(s)$ and the next bit is $h(s)$. The proof of forward security immediately follows from the fact that G from Equation (6) is a PRG (check this formally as an exercise).

Finally, we notice that one can also build a forward-secure PRG with any polynomial block length $t(k)$ from any regular PRG $G : \{0,1\}^k \to \{0,1\}^{t(k)+k}$. If we let $G(s) = G_1(s) \circ G_2(s)$, where $|G_1(s)| = |s| = k$, then G by itself is a forward-secure generator $G(s) = (G_1(s), G_2(s))$. Namely, we use $G_1(s)$ as the next state, and $G_2(s)$ as the $t(k)$ bits we output.

Theorem 5 *Forward-secure* PRG*'s with any polynomial block length* $1 \leq t(k) < \mathsf{poly}(k)$ *exist* $\iff$ *regular* PRG*'s exist (* $\iff$ OWF*'s exist).*

We leave the proof of this fact as an exercise.

8 PRG's in Our Lives

As a final note we would like to emphasize how common (and important!) PRG's are in real life. In computing, most requests for "random" sequence in fact access a *pseudorandom* sequence. Indeed, to uise randomness in most computer languages we first make a call to the function randomize, which initializes some PRG using only a small (and hopefully) truly random seed. All the subsequent random calls are in fact deterministic uses of a PRG, which outputs the next "random-looking sequence" (and possibly updates its state). Hence, since what we usually take for random is in fact some deterministic sequence generated by a PRG, we see that it is very important to understand what constitutes a good PRG. Which is exactly what we have spent the past two weeks investigating.

Finally, we recap our discussion of PRG's by reminding that they can be used without much harm in any realistic (i.e., *efficient*) application which expects truly random bits.