**CS 600.442 – Modern Cryptography** 09/21/16

# Lecture 4: Pseudorandomness (Part II)

*Instructor: Abhishek Jain* *Scribe: Yeon Woo Kim*

## 1 Next-bit Unpredictability ⇒ Pseudorandomness

**Theorem 1 (Completeness of Next-bit Test)** *If $\{X_n\}$ is next-bit unpredictable then $\{X_n\}$ is pseudorandom.*

**Proof. of Theorem 1.**
FSOC, suppose $\exists$ a n.u. PPT distinguisher $D$, and a polynomial $p(\cdot)$ s.t. for infinitely many $n \in \mathbb{N}$, $D$ distinguishes $X_n$ and $U_{l(n)}$ with probability $\frac{1}{p(n)}$. Let $\mathcal{A}$ be a machine that predicts the next bit of $X_n$ for every $n$. A sequence of *hybrid distributions* is defined as:

$$H_n^i = \{x \leftarrow X_n : u \leftarrow U_{\ell(n)} : x_0 x_1 ... x_i u_{i+1} u_{i+2} ... u_{\ell(n)}\}$$

Note that the first hybrid $H_n^0$ is the uniform distribution $U_{\ell(n)}$, and the last hybrid $H_n^{\ell(n)}$ is the distribution $X_n$. Thus, $D$ distinguishes between $H_n^0$ and $H_n^{\ell(n)}$ with probability $\frac{1}{p(n)}$. By hybrid lemma, $\exists$ some $i \in [0, \ell(n)]$ s.t. $D$ distinguishes between $H_n^i$ and $H_n^{i+1}$ with probability $\frac{1}{p(n)\ell(n)}$. The only difference between $H^{i+1}$ and $H^i$ is that in $H^{i+1}$, $(i+1)^{th}$ bit is $x_{i+1}$, and in $H^i$, it is $u_{i+1}$. Thus, given only $x_1 ... x_i$, $D$ can distinguish $x_{i+1}$ from the random set. The distribution $\tilde{H}_n^i$ is defined as:

$$\tilde{H}_n^i = \{x \leftarrow X_n : u \leftarrow U_{\ell(n)} : x_0 x_1 ... x_{i-1} \bar{x_i} u_{i+1} u_{i+2} ... u_{\ell(n)}\} \text{ where } \bar{x_i} = 1 - x_i$$

Since $H_n^i$ can be sampled from either $H_n^{i+1}$ or $\tilde{H}_n^{i+1}$ with equal probabilities,

$$\begin{aligned}
&\left|\Pr\left[t \leftarrow H_n^{i+1} : D(t) = 1\right] - \Pr[t \leftarrow H_n^i : D(t) = 1]\right| \\
=&\left|\Pr\left[t \leftarrow H_n^{i+1} : D(t) = 1\right] - \left(\frac{1}{2}\Pr\left[t \leftarrow H_n^{i+1} : D(t) = 1\right] + \frac{1}{2}\Pr\left[t \leftarrow \tilde{H}_n^{i+1} : D(t) = 1\right]\right)\right| \\
=&\frac{1}{2}\left|\Pr\left[t \leftarrow H_n^{i+1} : D(t) = 1\right] - \Pr\left[t \leftarrow \tilde{H}_n^{i+1} : D(t) = 1\right]\right|.
\end{aligned}$$

The observation that $D$ distinguishes $H_n^i$ and $H_n^{i+1}$ with probability $\frac{1}{p(n)\ell(n)}$ implies that $D$ distinguishes $H_n^{i+1}$ and $\tilde{H}_n^{i+1}$ with probability $\frac{2}{p(n)\ell(n)}$. Therefore, by the prediction lemma, $\exists$ a machine $\mathcal{A}$ s.t.

$$\Pr\left[b \leftarrow \{0,1\}; t \leftarrow H_n^{i+1,b} : D(t) = b\right] > \frac{1}{2} + \frac{1}{p(n)\ell(n)}$$

where $b = 1$ if $\mathcal{A}$ predicts a sample came from $H_n^{i+1}$ and $b = 0$ if it is from $\tilde{H}_n^{i+1}$. Then, we construct a machine $\mathcal{A}'$ that predicts the $(i+1)^{th}$ bit of the pseudorandom sequence, $x_{i+1}$. $\mathcal{A}'$ picks $\ell(n) - i$ random bits from $u_{i+1} ... u_{\ell(n)} \leftarrow U^{\ell(n)-1}$ and run $g \leftarrow \mathcal{A}(t_1 ... t_i u_{i+1} ... u_{\ell(n)})$. If $g = 1$ then $\mathcal{A}'$ outputs $u_{i+1}$, else $\mathcal{A}'$ outputs $\bar{u}_{i+1} = 1 - u_{i+1}$. Then,

$$\begin{aligned}
&\Pr\left[t \leftarrow X_n : \mathcal{A}'(1^n, t_1 ... t_i) = t_{i+1}\right] \\
&= \Pr\left[b \leftarrow \{0,1\}; t \leftarrow H_n^{i+1,b} : \mathcal{A}(t) = 1\right] > \frac{1}{2} + \frac{1}{p(n)\ell(n)}
\end{aligned}$$

which shows that $D$ correctly predicts the next bit with some noticeable probability. This is a contradiction. ■

## 2 Pseudorandom Generators (PRG)

We have defined the notion of pseudorandomness and next-bit unpredictability. Now, we turn to construct the definition of pseudorandom generators using the above theorem.

**Definition 1 (Pseudorandom Generator)** *A deterministic algorithm $G : \{0,1\}^n \to \{0,1\}^n$ is called a pseudorandom generator (PRG) if:*

1. *(efficiency): $G$ can be computed in polynomial time*
2. *(expansion): $|G(x)| > |x|$*
3. *$\{x \leftarrow \{0,1\}^n : G(x)\} \approx_c \{U_{\ell(n)}\}$ where $\ell(n) = |G(0^n)|$*

**Definition 2 (Stretch)** *The stretch of $G$ is defined as: $|G(x)| - |x|$*

We will first construct a PRG by 1-bit stretch. Then, we will show how to generically transform a PRG with 1-bit stretch into one that achieves polynomial-bit stretch.

## 3 PRG with 1-bit Stretch

We can think of an initial construction of PRG as $G(s) = f(s)\|h(s)$ where $f$ is a one-way function and $h$ is the hardcore predicate associated with $f$. However, while $h(s)$ is indeed unpredictable even given $f(s)$, this construction is not really a PRG because of the following reasons:

- $|f(s)|$ might be less than $|s|$.
- $f(x)$ may always start with a prefix, which is not random. Indeed, OWF doesn't promise random outputs.

To solve this problem, we will set $f$ to be a one-way *permutation* (OWP) over $\{0,1\}^n$. Now, we can address both of the above issues:

- Since $f$ is a permutation, the domain and range have the same number of bits, i.e., $|f(s)| = |s| = n$.
- $f(s)$ is uniformly random over $\{0,1\}^n$ if $s$ is randomly chosen. In particular:

$$\forall y : \Pr[f(s) = y] = \Pr[s = f^{-1}(y)] = 2^{-n}.$$

  Thus, $f(s)$ is uniform and cannot start with a fix value.

**Theorem 2 (PRG based on OWP)** *Let $f$ be a one-way permutation, and $h$ be a hard-core predicate for $f$. Then $G(s) = f(s)\|h(s)$ is a PRG with 1-bit stretch.*

**Proof.**
FSOC, suppose $\exists$ a n.u. PPT adversary $\mathcal{A}$ and a polynomial $p(n)$ s.t. $\forall n$, $\exists\ i$ s.t. $\mathcal{A}$ predicts the $i^{th}$ bit with non-negligible probability $\frac{1}{p(n)}$. Since $f$ is a permutation, the first $n$ bits of $G(s)$ are uniformly distributed. $\mathcal{A}$ must predict $(n+1)^{th}$ bit with advantage $\frac{1}{p(n)}$, i.e.

$$\Pr[\mathcal{A}(f(s)) = h(s)] > \frac{1}{2} + \frac{1}{p(n)}$$

which contradicts the assumption that $h(s)$ is hard-core for $f$. Thus, $G$ is a PRG. ■

# 4 PRG with Poly-Stretch

**Lemma 3** *Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. For any polynomial $l$, $G' : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is defined as:*

$$\begin{aligned} G'(s) &= b_1 ... b_{\ell(n)} \text{ where} \\ X_0 &\leftarrow s \\ X_{i+1} \| b_{i+1} &\leftarrow G(X_i) \end{aligned}$$

*Then, $G'$ is a PRG.*

**Proof.** We first establish some notation. Let $G'(s) = G^m(s)$, where

$$\begin{aligned} G^0(x) &= \epsilon \\ G^i(x) &= b \| G^{i-1}(x') \text{ where } x' \| b \leftarrow G(x) \end{aligned}$$

and $\epsilon$ denotes the empty string. FSOC, suppose $\exists$ a distinguisher $D$ and a polynomial $p(\cdot)$ s.t. $\forall n$, $D$ distinguishes $\{U_{m(n)}\}_n$ and $\{G'(U_n)\}_n$ with non-negligible probability $\frac{1}{p(n)}$.

Let $H_n^i = U_{m(n)-i} \| G^i(U_n)$ be the hybrid distributions for $i = 1, ..., m(n)$. Then, $H_n^0 = U_{m(n)}$ and $H_n^{m(n)} = G^{m(n)}(U_n)$ and $D$ distinguishes $H_n^0$ and $H_n^{m(n)}$ with probability $\frac{1}{p(n)}$. By the hybrid lemma, $\forall n$, $\exists i$ s.t. $D$ distinguishes $H_n^i$ and $H_n^{i+1}$ with probability $\frac{1}{m(n)p(n)}$. Then,

$$\begin{aligned} H_n^i &= U_{m-i} \| G^i(U_n) \\ &= U_{m-i-1} \| U_1 \| G^i(U_n) \\ H_n^{i+1} &= U_{m-i-1} \| G^{i+1}(U_n) \\ &= U_{m-i-1} \| b \| G^i(x) \text{ where } x \| b \leftarrow G(U_n) \end{aligned}$$

Suppose $\exists$ a n.u. PPT $M(y)$ that outputs from the following experiment:

$$\begin{aligned} &b_{prev} \leftarrow U_{m-i-1} \\ &b \leftarrow y_1 \\ &b_{next} \leftarrow G^i(y_2 ... y_{n+1}) \\ &\text{Output } b_{prev} \| b \| b_{next} \end{aligned}$$

The output of $D$ is distributed identically to the output of $H_n^i$ if the input was sampled from $U_{n+1}$ and $H_n^{i+1}$ if it was from $G(U_n)$. Since $\{U_{n+1}\}_n \approx \{G(U_n)\}_n$ with advantage $\frac{1}{p(n)\ell(n)}$ and $G$ runs in polynomial time, $\{H_n^i\}_n \approx \{H_n^{i+1}\}_n$, which is a contradiction. ■

# 5 Going beyond Poly Stretch

PRGs can only generate polynomially long pseudorandom strings. What if we want exponentially long pseudorandom strings? How can we efficiently generate them? One way to do this is by using functions that index exponentially long pseudorandom strings.

Towards that end, let us start by defining a random function? Consider a function $f : \{0,1\}^n \to \{0,1\}^n$. If we write $f$ as a table, where first column has input strings from $0^n$ to $1^n$ and the second column has the function value against each input, each row of the table is of the form $(x, f(x))$. Then, the size of the table is $2^n \times n = n2^n$. Thus, the total number of functions that map $n$ bits to $n$ bits is $2^{n2^n}$.

To define a random function, we can use one of the two methods:

1. Select a random function $F$ *uniformly at random* from all $2^{n2^n}$ functions that map $n$ bits to $n$ bits

2. Use a randomized algorithm to describe the function

   - A randomized program $M$ keeps a table $T$ (initially empty) to implement a random function $F$
   - On input $x$,
     - if $x$ is not in the table, choose a random string $y$ and add the entry $(x, y)$ to $T$
     - otherwise, $M$ picks $(x, y)$ corresponding to $x$ from $T$, and outputs the entry
   - The distribution of $M$'s output is identical to that of $F$.

However, truly random functions are huge random objects. Neither of the methods allows us to store the entire function efficiently. But with the second method, $M$ will only need polynomial space and time to store and query $T$, if one makes **polynomial** calls to the random function.

**Pseudorandom Functions (PRF): Intuition.** PRF looks like a random function and is described in polynomial bits. At first, it seems like a good idea to use computational indistinguishability to make PRF "look like" a Random Function. However, the main issue with this idea is that a random function is of an exponential size. If $D$ can't even read the input efficiently, then it can distinguish between PRFs and RFs by looking at its input size, and computational indistinguishability is violated. One way to solve this issue is to allow $D$ to only *query* the function on inputs of its choice, and let it see the output. We'll formalize this idea in the next lecture.