CS 276: Cryptography	**Spring 2006**
Lecture 4 — January 26, 2006	
Lecturer: David Wagner	*Scribe: David Turner*

1 Making big PRGs from little PRGs

Suppose that we need a PRG that stretches the input length from k bits to $l(k)$ bits, where $l : \mathbb{N} \rightarrow \mathbb{N}$ is a polynomial stretch function. That is, we want a PRG $G : \{0,1\}^k \rightarrow \{0,1\}^{l(k)}$. Constructing such a PRG may be difficult to accomplish in practice. Fortunately, there is a simple method for constructing larger PRGs from smaller ones.

1.1 A Simple Construction

Let $G : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ be a (t, ϵ)-secure PRG, which is depicted below in Figure 1.

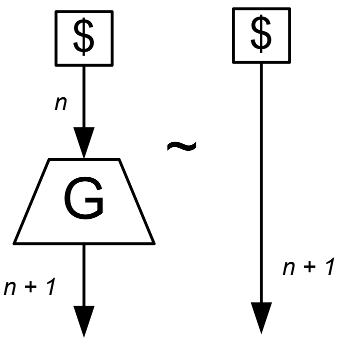

Figure 1: The output of G is computationally indistinguishable from the uniform distribution on $n + 1$ bits.

At first glance, G appears to be useless because it stretches the input length by 1 bit. Using G, we can actually construct a PRG $H : \{0,1\}^n \rightarrow \{0,1\}^{n+2}$ that stretches the input length by two bits. If $k \xleftarrow{\$} \{0,1\}^n$, then let the output of $G(k)$ be $g_1 g_2 \ldots g_n g_{n+1}$, where g_i denotes the ith bit of $G(k)$. Finally, we define $H(k)$, depicted in Figure 2(a), to equal $G(g_1 g_2 \ldots g_n) \circ g_{n+1}$, where the symbol $\circ$ denotes concatenation.

Now we shall sketch a proof that $H(k) \sim Z$, where $k \xleftarrow{\$} \{0,1\}^n$ and $Z \xleftarrow{\$} \{0,1\}^{n+2}$. We do this by using a hybrid argument as we make a series of small changes to go from (a) to (e) in Figure 2:

1. To go from (a) to (b), we can apply Figure 1 and use the data processing inequality.
2. The distribution of outputs for (b) and (c) are perfectly indistinguishable.
3. To go from (c) to (d), we can once again apply Figure 1. Because Y is independent of both X and X', we have $(X, Y) \sim (X', Y)$ by applying the data processing inequality[1].

[1]The last problem on Homework 1 considers the case when Y is not necessarily independent of both X and X'.

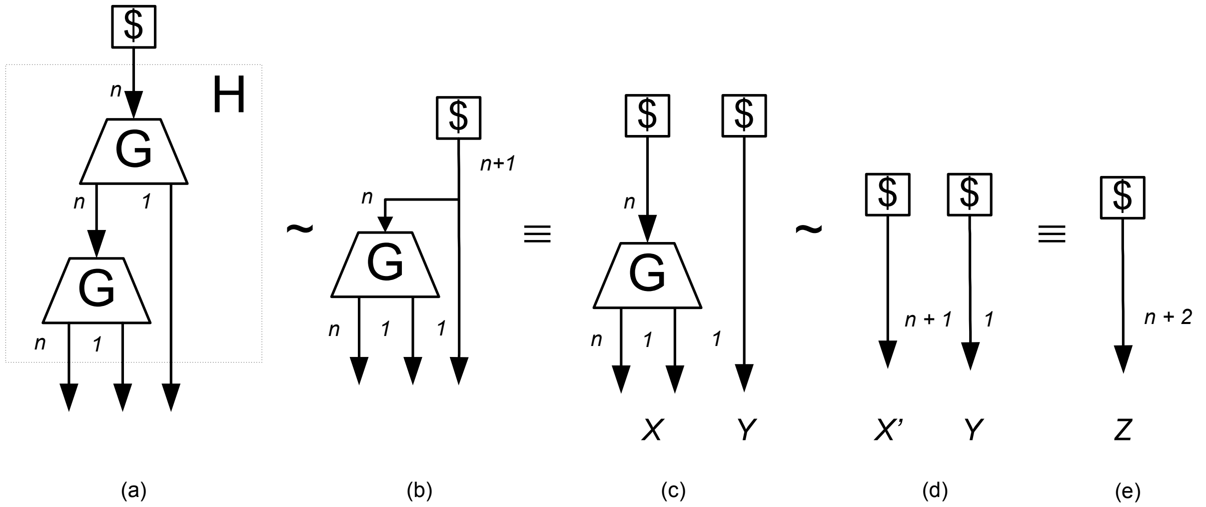

Figure 2: A hybrid argument showing that the output of H is computationally indistinguishable from Z.

4. Finally, in (d) and (e), $X' \circ Y$ is perfectly indistinguishable from Z.

5. Thus, H is a $(t - t', 2\epsilon)$-PRG for some appropriate choice of t' that results from the use of the data processing inequality, and the 2ϵ comes from the triangle inequality. □

This construction can be easily generalized to the case where n bits of input must be stretched to $n + k$ bits of output where $k > 0$. Thus, we have arrived at the following theorem whose proof we omit, but it can be derived from the above construction.

Theorem. *There exist polynomially secure PRGs* $\mathcal{G} = \{G^i : \{0,1\}^i \to \{0,1\}^{i+1} \mid i \in \mathbb{N}\}$ *with stretch function* $l(i) = i+1$ *if and only if for all polynomial stretch functions* $l'(i)$*, there exist polynomially secure PRGs* $\mathcal{H} = \{H^i : \{0,1\}^i \to \{0,1\}^{l'(i)} \mid i \in \mathbb{N}\}$.

2 The Goldreich-Goldwasser-Micali Construction

A major drawback of the construction in Section 1.1 for stretching n bits of input to $n + k$ bits of output is that it requires $\mathcal{O}(n + k)$ time to access any particular bit of output. We now describe the Goldreich-Goldwasser-Micali (GGM) construction, which provides logarithmic time to access the bits of output.

2.1 The Construction

For convenience, let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG. By using G, we can construct a PRG $H_d : \{0,1\}^n \to \{0,1\}^{2^d n}$ where $d \in \mathbb{N}$. The idea is to construct a complete binary tree of height d such that each node is represented by G and that at the dth level, all 2^{d-1} nodes exist. Each node outputs $2n$ bits and sends the first n bits to its left child and the last n bits to its right child. Figure 3(a) shows the GGM construction for H_3. By using a hybrid argument similar to the one used in Section 1.1, one can show that if $k \xleftarrow{\$} \{0,1\}^n$ and $U_d \xleftarrow{\$} \{0,1\}^{2^d n}$, then $H_d(k) \sim U_d$. Finally, to access any bit of output would take $\mathcal{O}(d + n)$ time.

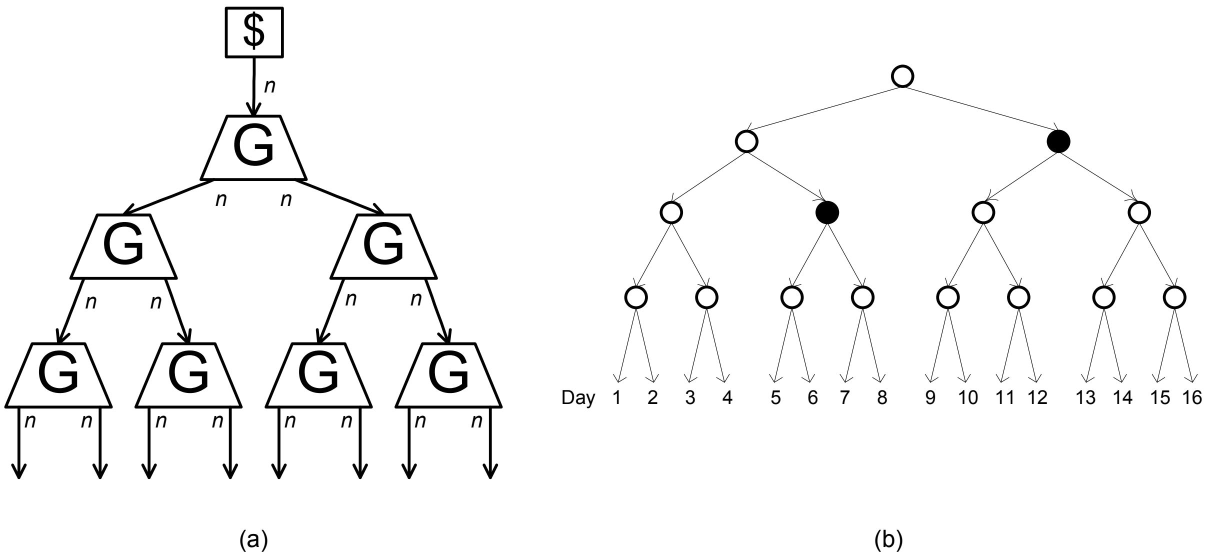

Figure 3: The GGM construction.

2.2 Application: Key Management

Let us suppose that you would like to watch your favorite cable channel CryptoTV. Everyday, CryptoTV encrypts their broadcast with a different key. Thus, in order to get the key, subscribers pay for each day they want to view the broadcast. Unfortunately, distributing these keys daily to several million subscribers is impractical. In order to reduce this bandwidth cost, we can actually use the GGM construction. For example, consider a sixteen day period and that a subscriber wants to watch CryptoTV on days four through sixteen. By using the GGM construction to create H_4, as shown in Figure 3(b), the broadcaster would distribute the key for day four as well as the inputs to the two black nodes because the subscriber can extract the keys for days five through sixteen.

3 Pseudorandom Functions

Let us further examine the GGM construction by defining a function f that can access the leafs in H_d. More formally, $f : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ is defined by $f(k,l) =$ value of leaf l in $H_d(k)$. Observe that l is a bit string encoding of the path to take from the root node to the leaf node where 0 means take the left branch and 1 means take the right branch[2]. Next, let $r \xleftarrow{\$} \mathsf{Functions}(\{0,1\}^d \to \{0,1\}^n)$, where $\mathsf{Functions}(\{0,1\}^d \to \{0,1\}^n)$ denotes the set of all functions with domain $\{0,1\}^d$ and range $\{0,1\}^n$. Finally, define $f_k : \{0,1\}^d \to \{0,1\}^n$ by $f_k(\cdot) = f(k,\cdot)$, where $k \xleftarrow{\$} \{0,1\}^n$. The GGM construction appears to have the property that $\forall l_1,\ldots,l_q \in \{0,1\}^d, (f_k(l_1), f_k(l_2), \ldots, f_k(l_q)) \sim (r(l_1), r(l_2), \ldots, r(l_q))$ and that f_k appears to be indistinguishable from r. Thus, f is called a pseudorandom function (PRF).

Before we formalize the notion of a PRF, let us gain some intuition by considering the following game. Let $F : \mathsf{Keys} \times \mathcal{X} \to \mathcal{Y}$ be some fixed function where Keys denotes the set of all keys, and $\mathcal{X}$ and $\mathcal{Y}$ are finite sets. Now at the beginning of the game, there are two black box functions with domain $\mathcal{X}$ and range $\mathcal{Y}$. One black box function is $F_K : \mathcal{X} \to \mathcal{Y}$, where $K \xleftarrow{\$} \mathsf{Keys}$, and it is defined by $F_K(x) = F(K,x)$, where $x \in \mathcal{X}$. The other black box function is $R \xleftarrow{\$} \mathsf{Functions}(\mathcal{X} \to \mathcal{Y})$. You are given one of these black

[2]This encoding is also used in Huffman codes.

box functions with probability $\frac{1}{2}$, and you are allowed to query the black box function with as many different inputs as you wish. To win the game, you must correctly guess which black box function you have. This leads to the following formal definitions.

Definition 1. *For all probabilistic polynomial time (PPT) adversaries A, its advantage at distinguishing F_K from R, where $K \xleftarrow{\$} \mathsf{Keys}$ and $R \xleftarrow{\$} \mathsf{Functions}(\mathcal{X} \to \mathcal{Y})$, is defined to be*

$$\mathsf{Adv}\ A = \left| \Pr\left[K \xleftarrow{\$} \mathsf{Keys}\ ; A^{F_K} = 1\right] - \Pr\left[R \xleftarrow{\$} \mathsf{Functions}(\mathcal{X} \to \mathcal{Y})\ ; A^{R} = 1\right] \right|.$$

If g is a function, then the notation A^g means that A has oracle access to g. Moreover, A is allowed to query the oracle with any input x, and then the oracle outputs $g(x)$ in a single step of computation.

Definition 2. *F is a said to be a (t, q, ϵ)-secure PRF if for all PPT adversaries A running in time at most t and making at most q queries, then $\mathsf{Adv}\ A \leq \epsilon$.*

3.1 Application: Password Derivation

Consider the problem of having to remember passwords for many websites. One possible solution is to pick a truly random function R. Then the password for website w is $R(w)$. The problem with this solution is that the description of R requires exponential space. Instead, let F be a PRF and choose a master password P uniformly at random. Then the password for website w is $F(P, w)$. The advantage of this solution is that the description of F is exponentially more succinct than the description of R.

3.2 Block Ciphers

A block cipher is a function $B : \mathsf{Keys} \times \mathcal{M} \to \mathcal{C}$, where $\mathsf{Keys} = \{0,1\}^k$, $\mathcal{M} = \{0,1\}^n$ is the set of plaintexts, and $\mathcal{C} = \{0,1\}^n$ is the set of ciphertexts. Furthermore, k is called the key length and n is called the block length. Block ciphers are designed to have fast hardware implementations, and they work as follows:

1. Key Generation – Pick $K \xleftarrow{\$} \mathsf{Keys}$.
2. Encryption – Define $E_K(m) = B(K, m)$, where $m \in \mathcal{M}$.
3. Decryption – There is an inverse function E_K^{-1} such that $E_K^{-1}(E_K(m)) = m$ for all $m \in \mathcal{M}$.

Thus, we see that for all $K \in \mathsf{Keys}$, E_K defines a permutation on $\{0,1\}^n$.

An example of a block cipher is the Advanced Encryption Standard (AES), which has a 128-bit block length, and the key length can be 128, 192, or 256 bits. The two known attacks against AES involve an exhaustive key search, and the other attack uses the fact that E_K is bijective. Thus, we have arrived at the following conjecture:

Conjecture. AES *with 128-bit keys is a $\left(t, q, \frac{t}{2^{128}} + \frac{q^2}{2^{128}}\right)$-PRF.*