CSE 599b: Cryptography **(Winter 2006)**

Lecture 5: Pseudorandom Generators; One-way Functions

18 January 2006

Lecturer: Paul Beame *Scribe: Paul Beame*

1 More queries don't hurt much: The hybrid argument

We now see that for two computationally indistinguishable ensembles that are polynomial-time samplable, if we allow PPT algorithms access to a polynomial number of queries rather than just one query then they still have negligible advantage. (We will really think of using this in the context of allowing extra queries in the the definition of psuedorandom generators but the version we will do has all the ideas and is formally a bit simpler.) This will introduce the basic idea of the reduction method we will use throughout the course and will introduce the 'Hybrid Argument' that appears frequently in analyzing security.

Theorem 1.1. *If $\mathcal{E}$ and $\mathcal{D}$ are computationally indistinguishable and poynomial-time samplable then for any polynomially bounded $q : \mathbb{N} \to \mathbb{N}$, the function $\epsilon(n)$ is negligible where*

$$\begin{aligned}\epsilon(n) &= \Pr[A(x_1, \ldots, x_{q(n)}, 1^n) = 1 \mid x_1, \ldots, x_q(n) \leftarrow \mathcal{E}_n] \\ &\quad - \Pr[A(x_1, \ldots, x_{q(n)}, 1^n) = 1 \mid x_1, \ldots x_q(n) \leftarrow \mathcal{D}_n].\end{aligned}$$

Note that unlike the single query case the algorithm A is also given 1^n since it may not be able to determine n solely from the length of its input Using the notation $\mathcal{D}^k$ to denote a vector of k independent trials from a distribution $\mathcal{D}$, we can conveniently re-write the expression for $\epsilon(n)$ in the above theorem as

$$\epsilon(n) = \Pr[A(\mathcal{E}_n^{q(n)}, 1^n) = 1] - \Pr[A(\mathcal{D}_n^{q(n)}, 1^n) = 1].$$

Proof. We will show that given a PPT A that takes $q(n)$ queries and has advantage $\epsilon(n)$ then there is a PPT A' that takes only 1 query and has advantage $\epsilon'(n) = \epsilon(n)/q(n)$ (all we really care is that $\epsilon'(n)$ is large enough relative to $\epsilon(n)$). Thus $\epsilon(n) \leq q(n)\epsilon'(n)$ which is still negligible since $q(n)$ is a polynomial. This is the standard way of arguing, namely we base the security under one definition on that of another by showing if one can break the first one then one can break the second.

The method we use is called a *hybrid argument*. It is based on creating hybrids of the two distributions $\mathcal{E}_n^{q(n)}$ and $\mathcal{D}_n^{q(n)}$.

For $0 \leq k \leq q(n)$ define distributions $\mathcal{H}_{k,n}$ on $(\{0,1\}^n)^{q(n)}$ by independently choosing $x_i \leftarrow \mathcal{D}_n$ for $1 \leq i \leq k$ and $x_i \leftarrow \mathcal{E}_n$ for $k < i \leq q(n)$.

Thus $\mathcal{H}_{k,n} = \mathcal{D}_n^k \times \mathcal{E}_n^{q(n)-k}$. It follows that $\mathcal{H}_{0,n} = \mathcal{E}_n^{q(n)}$ and $\mathcal{H}_{q(n),n} = \mathcal{E}_n^{q(n)}$.

Let $p_{k,n} = \Pr[A(\mathcal{H}_{k,n}, 1^n) = 1]$. Then $\epsilon(n) = p_{0,n} - p_{q(n),n}$.

Define statistical test PPT A' as follows:

On input x of length n:
1. Choose $k \in \{1, \ldots, q(n)\}$ uniformly at random
2. Sample each of $x_1, \ldots, x_{k-1}$ from $\mathcal{D}_n$ independently
3. Sample each of $x_{k+1}, \ldots, x_{q(n)}$ from $\mathcal{E}_n$ independently
4. Submit $(x_1, \ldots, x_{k-1}, x, x_{k+1}, \ldots, x_{q(n)}, 1^n)$ to A and output what A does.

Observe that

$$\begin{aligned}
\epsilon(n) &= \Pr[A'(\mathcal{E}_n) = 1] - \Pr[A'(\mathcal{D}_n) = 1] \\
&= \frac{1}{q(n)} \sum_{k=1}^{q(n)} (\Pr[A(\mathcal{H}_{k-1,n}, 1^n) = 1] - \Pr[A(\mathcal{H}_{k,n}, 1^n) = 1]) \\
&= \frac{1}{q(n)} \sum_{k=1}^{q(n)} (p_{k-1,n} - p_{k,n}) \\
&= \frac{1}{q(n)} [(p_{0,n} - p_{1,n}) + (p_{1,n} - p_{2,n}) + \cdots + (p_{q(n)-1,n} - p_{q(n),n})] \\
&= \frac{1}{q(n)} (p_{0,n} - p_{q(n),n}) \\
&= \epsilon(n)/q(n)
\end{aligned}$$

□

How can we get PRNG's?

2 One-way functions

Definition 2.1. *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is a* (strong) one-way function *if and only if*

(1) Easy to Compute: f is computable by a deterministic polynomial-time algorithm.

(2) Hard to Invert: For every PPT A, the function $\epsilon(n)$ is negligible where

$$\epsilon(n) = \Pr[A(f(x), 1^n) \in f^{-1}(f(x)) \mid x \leftarrow \mathcal{U}_n].$$

(I.e. $\epsilon(n) = \Pr[f(A(y, 1^n)) = y \mid x \leftarrow \mathcal{U}_n; y \leftarrow f(x)]$.)

Note that for the notion of hardness of inverting f there is no requirement that the invertin algorithm produce x which may be impossible if f is not one-to-one, but merely that f produce some value that maps the same place that x does.

Also note that we give the adversary A, 1^n in addition to $f(x)$ to allow A polynomial time in the length of x; otherwise any function that maps n bits to $\log n$ bits, say, would be hard to invert since just writing down x would be exponential in the size of the output of f.

A related but weaker notion is also useful to consider since it is a much less stringent requirement that seems easier to satisfy.

Definition 2.2. *A* weak one-way function *satisfies the same conditions as a one-way function except that the function is only slightly hard to invert in that (2) above is replaced by the requirement that $\epsilon(n) \leq 1 - 1/n^c$ for some constant c. In particular this means that the probability that A does not invert f is not negligible (although it is a little strong in that it requires the bound hold for all input lengths).*

Theorem 2.3. *The existence of weak one-way functions implies the existence of strong one-way functions.*

Proof. Let f be weak one-way and let c be the constant from the weak one-way definition for f. Define

$$g(x_1, \ldots, x_{n^{c+1}}) = (f(x_1), \ldots, f(x_{n^{c+1}})).$$

The claim is that g is a one-way function. This proof is tricky but some intuition may help: Suppose that an inverting algorithm A for g worked independently on each coordinate. The chance that each coordinate is correct is at most $(1 - 1/n^c)$ and sicne there are n^{c+1} independent copies, the success would be at most

$$(1 - 1/n^c)^{n^{c+1}} = ((1 - 1/n^c)^{n^c})^n \leq e^{-n}$$

since $1 + x \leq e^x$ and substituting $x = -1/n^c$. The actual proof is quit a bit subtler than this but we will skip it. □

2.1 Candidate weak one-way functions

Integer Multiplication $f(xy) = x \cdot y$ where $|x| = |y|$.

Inverting amounts to factoring if the output is a product of two primes. The Prime Number Theorem implies that a $\Theta(1/n)$ fraction of n-bit numbers are prime. This means that at least a $Theta(1/n^2)$ fraction of possibilities seem hard.

Subset Sum $f(x_1, \ldots, x_n, I) = (x_1, \ldots, x_n, \sum_{i \in I} x_i)$ where I is n bits and $x_1, ..., x_n$ are m-bit integers (typically $m = n + 1$ or so).

Inverting seems to imply solving an NP-hard problem but we require much more than worst-case hardness; we require that it be hard to solve on average.

2.2 More is needed

The above theorem about converting weak one-way functions to strong ones is actually not very useful with these functions since it requires such huge blow up in input size. We will find that by generalizing our notion of a single one-way function to a collection of one-way functions we will find natural candidates where there is no loss in going between the weak and strong definitions. It will turn out that this notion of a collection of one-way functions is also critical for the definition of trapdoor functions. We will consider this next time.