YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 461b: Foundations of Cryptography *Notes 8 (rev. 1)*
Professor M. J. Fischer *February 5, 2009*

Lecture Notes 8

20 Collections of Functions

Although we have defined one-way functions on the infinite domain of all binary strings, for cryptographic applications we are often interested in strings of of a fixed length n, where n is determined by a *security parameter*. We have already seen in section 15.1 of lecture 5 that a one-way function defined only on lengths in a set I can be used to build a one-way function defined on all lengths, provided that I is polynomial-time enumerable. Intuitively, this just means that I is not too sparse and there are feasible algorithms for enumerating I and deciding membership in I—all the things you would expect from a “nice” set, for example, the set of all even numbers.

Later, when we want to talk about trapdoor functions, we’ll again need to consider lengths separately, since a trapdoor string (think RSA private key for now) only works for a finite set of inputs (the numbers in $\mathbf{Z}_n^*$ for RSA). Because of this, we shift gears slightly and consider infinite collections of finite functions instead of a single infinite functions.

Definition: A *collection of functions* consists of an infinite set of *indices* $\bar{I}$ and a family of functions $\{f_i\}_{i\in\bar{I}}$. For each $i \in \bar{I}$, the domain of f_i is a finite set D_i.

For cryptographic applications, we need some additional properties. Namely, given an index (description) $i \in \bar{I}$, we need to be able to compute $f_i(x)$ given $i \in \bar{I}$ and $x \in D_n$. In addition, we need to be able to be able to randomly sample (not necessarily uniformly) from both $\bar{I} \cap \{0,1\}^n$ and $fromD_i$.

Definition: A collection of functions $\mathcal{F} = \{f_i\}_{i\in\bar{I}}$ indexed by $\bar{I}$ is said to be *useful* if there are probabilistic polynomial-time algorithms I and D and deterministic algorithm F that satisfy the following:

1. The output distribution of $I(1^n)$ is a random variable ranging over $\bar{I} \cap \{0,1\}^n$.
2. The output distribution of $D(i)$ for $i \in \bar{I}$ is a random variable ranging over D_i.
3. $F(i,x) = f_i(x)$ for all $i \in \bar{I}$ and $x \in D_i$.

We say that $\mathcal{F}$ is *described* by $\bar{I}$, I, D, and F.

20.1 Collections of One-Way Functions

Now we can define the strongly one-way property for useful collections of functions.

Definition: Let $\mathcal{F}$ be a useful collection of functions described by $\bar{I}$, I, D, and F. We say $\mathcal{F}$ is *strongly one-way* if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n,

$$\Pr[A'(I_n, f_{I_n}(X_n)) \in f_{I_n}^{-1}(f_{I_n}(X_n))] < \frac{1}{p(n)},$$

where I_n is the random variable $I(1^n)$, and X_n is the random variable $D(I_n)$. Following the usual convention, all occurrences of I_n refer to the same randomly chosen value $i = I(1^n)$, so in particular, I_n and X_n are not assumed to be independent; rather, $X_n = D(i)$ for the chosen i.

In words, an experiment consists of choosing random values $i = I(1^n)$ and $x = D(i)$ using the probabilistic algorithms I and D, computing $y = F(i, x)$, and choosing random $x' = A'(i, y)$. The experiment is said to *succeed* if and only if $F(i, x') = y$. The strongly one-way property requires that the overall probability of success of an experiment be at most $1/p(n)$. Note that this probability is averaged over several sources of randomness: the choice of i, the choice of x, and the randomness in the algorithm A'. Because the requirement is only on the average success probability, it does not preclude higher success probabilities for particular values of $i \in \bar{I}$ or $x \in D_i$.

20.2 Examples of One-Way Collections

RSA collection of functions The index set $\bar{I}$ consists of pairs (N, e), where $N = PQ$ for distinct primes P, Q of equal length and $e \in \mathbf{Z}_N^*$. For $i = (N, e) \in \bar{I}$, we have $D_i = \mathbf{Z}_N^*$ and $f_i(x) = x^e \pmod{N}$.

To show that the RSA collection satisfies the definition for a useful collection of functions, we must give algorithms I_{RSA}, D_{RSA}, and F_{RSA}. Briefly, $I_{\mathrm{RSA}}(1^n)$ selects uniformly distinct primes P and Q of length n, sets $N = PQ$, selects uniformly an integer $e \in \mathbf{Z}_N^*$, and returns the pair (N, e). Algorithm $D_{\mathrm{RSA}}((N, e))$ simply selects uniformly an element from $\mathbf{Z}_N^*$. $F_{\mathrm{RSA}}((N, e), x) = x^e \pmod{N}$. All three of these algorithms were presented in CPSC 467 in the context of RSA. I_{RSA} is the key generation algorithm, which also uses the algorithm D_{RSA} for sampling from $\mathbf{Z}_N^*$ in order to select e. F_{RSA} is the algorithm for computing the RSA encryption function.

It is an open problem if RSA is strongly one-way, or even if it is as hard to invert as factoring, but no feasible algorithm for inverting it has yet been discovered.

Other collections See the textbook for other examples of one-way collections, namely, the Rabin Function, the Factoring Permutations, and Discrete Logarithms.

20.3 Trapdoor One-Way Permutations

A collection of strongly one-way trapdoor permutations is similar to the collection of strongly one-way functions defined in section 20.1, except that the functions are required to be permutations (i.e., one-to-one and onto), and the index set is expanded to include the index of an algorithm for computing the inverse $f^{-1}(y) = x$. Namely, $\bar{I} = \bar{I}_1 \times \bar{I}_2$, where $\bar{I}_1$ is the set of indices of permutations in the collection, and $\bar{I}_2$ is the set of indices of the inverses of functions in the collection.

The algorithm $I(1^n)$ returns a pair $(i, t) \in \bar{I}_1 \times \bar{I}_2$, where i has length n, and t has length polynomial in n. The index i describes a permutation f_i in the collection and is interpreted by the algorithm F as before, namely, $F(i, x) = f_i(x)$ for all $x \in D_i$. The index t describes the inverse f_i^{-1} of f_i and is interpreted by a fourth required algorithm, F^{-1}, with the property that $F^{-1}(t, y) = f_i^{-1}(y)$.

See the textbook for more details as well as variations on this definition that allow the required algorithms to have small probability of failure (as for example the RSA key generation algorithm does in the unlikely case that the probabilistic primality test fails and uses a non-prime for P or Q).

Example The RSA collection of functions is easily extended to the RSA collection of trapdoor permutations. Namely, $\bar{I}_2$ consists of pairs (N, d), and $I_{\mathrm{RSA}}(1^n) = ((N, e), (N, d))$, where (N, e)

is the public RSA key, (N, d) is the private key, and $F^{-1}((N, d), y) = y^d \pmod{N}$.

21 Hard-Core Predicates

Let f be a strongly one-way function. By definition, no algorithm can invert f with a non-negligible success probability, but this doesn't preclude algorithms that can "partially" invert f in the sense of recovering a lot of information about the inverse. For example, we observed earlier that if f is strongly one-way, then so is

$$g(u, v) = (f(u), v),$$

where $|u| = |v|$, even though given $y = (f(u), v)$, we can easily recover half of the bits of an $x' = (u', v')$ for which $f(x') = y$, namely, we know that $y' = y$.

We'd like to know the answers to questions such as the following:

1. Suppose $f(x) = y$. Is there some bit x_i of x that cannot be predicted with non-negligible success probability, or can every individual bit be so predicted?

2. Suppose neither bits x_i nor x_j can be predicted for $i \neq j$ given only $y = f(x)$. Can we predict some function of them, e.g., $x_i \oplus x_j$?

3. How many "hard" bits does a one-way function give us?

To begin thinking about these questions, we define the notion of a "hard-core" predicate b of a strongly one-way function f. The idea is that b is a predicate that depends on x, but b cannot be predicted with better than $\frac{1}{2} + \varepsilon$ accuracy given only $y = f(x)$.

There are two reasons why this might be the case. One is that if f is not one-to-one, then f loses information, so we might have $y = f(x') = f(x'')$ for $x' \neq x''$, yet $b(x') \neq b(x'')$. In this case, our chance of correctly guessing b given y is exactly $\frac{1}{2}$, since it is equiprobable given y whether $x = x'$ or $x = x''$.

On the other hand, if f is one-to-one, then the only thing that makes b hard to compute is if f is hard to invert. Hence, we are primarily interested in the case of functions f that are both strongly one-way and also one-to-one. Nevertheless, we state the definition of hard-core in the more general form without the requirement that f be 1-1.

Definition: Let f be a function. A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is a *hard core* of f if, for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n,

$$\Pr[A'(f(U_n)) = b(U_n)] < \frac{1}{2} + \frac{1}{p(n)}.$$

Given any such algorithm A', we define the *advantage* of A' (at predicting $b(x)$ given $f(x)$) to be the quantity

$$\varepsilon_{A'}(n) = \Pr[A'(f(U_n)) = b(U_n)] - \frac{1}{2}$$

Thus, b is a hard core of f if $\varepsilon_{A'}(n)$ is a negligible function for every p.p.t. algorithm A'.

Our goal in the next two lectures is to prove the theorem

Theorem 1 *If strongly one-way functions exist, then there exists a strongly one-way function that has a hard-core predicate.*

We proceed by considering strongly one-way functions g of a special kind. Let f be strongly one-way and length-preserving. For $|x| = |r|$, define

$$g(x, r) \overset{\text{df}}{=} (f(x), r).$$

Lemma 2 *g is strongly one-way (on even lengths).*

Proof: The fact that g is strongly one-way on even lengths is obvious, since the first half x of any inverse (x, r) of $g(y, r)$ is also an inverse of $f(y)$. Hence, an algorithm F to invert f, given an algorithm G for inverting g, merely guesses a string r of the same length as y, computes $(x', r') = G(y, r)$, and outputs x'. Algorithm F has the same success probability at inverting f given y as algorithm G has at inverting g given y and r. Since F has negligible success probability, then so does G; hence, g is strongly one-way (on even lengths) since f is strongly one-way.[1] ■

For $|x| = |r|$, define the predicate

$$b(x, r) \overset{\text{df}}{=} x \cdot r \bmod 2.$$

Here, $x \cdot r$ is the ordinary vector dot product when the strings x and r are regarded as bit vectors. More formally, if x_i and r_i denote the i^{th} bits of x and r, respectively, then $x \cdot r = \sum_{i=1}^{n} x_i r_i$, and $x \cdot r \bmod 2 = (\sum_{i=1}^{n} x_i r_i) \bmod 2$. This can be expressed using the Boolean operators $\wedge$ ("and") and $\oplus$ ("exclusive-or"):

$$x \cdot r \bmod 2 = \bigoplus_{i=1}^{n} (x_i \wedge r_i).$$

Lemma 3 *b is a hard core of g.*

The general outline of the proof is similar to the proof that a strongly one-way function can be constructed from a weakly one-way function. Namely, we assume that b is not a hard core of g, so there is an algorithm G for predicting b with a non-negligible advantage $\varepsilon_G(n)$. Using G, we construct an algorithm A for inverting f. We then analyze the success probability of F and show that it is non-negligible. This contradicts the assumption that f is strongly one-way, from which we conclude that b really is a hard core of g.

[1] A careful proof will have to take account of the fact that the lengths of the strings on which F and G have the same success probability (y for F and (y, r) for G) differ by a factor of 2, so the bound on F's success probability on strings of length n is the same as the bound on G's success probability on strings of length $m = 2n$. One must show for all polynomials $p(\cdot)$ that the latter is bounded by $\frac{1}{p(m)}$ for all sufficiently large even lengths m if the former is bounded by $\frac{1}{p(n)}$ for all sufficiently large n.