COM S 687 Introduction to Cryptography September 7, 2006

Lecture 5: Existence of OWF and Computational Number Theory

Lecturer: Rafael Pass *Scribe: Michael George*

1 Introduction

In the last few lectures we have come to suitable definitions for strong and weak one-way functinos. Although the existence of one-way functions is still a matter of conjecture, in this lecture we will find some likely candidates. We will construct two useful contenders:

Complexity Theoretic We will construct a function that is guaranteed to be one way if any one-way functions exist

Number Theoretic We will show that multiplication is one-way under a simple and widely accepted conjecture

Finally we will introduce the slightly more general notion of a one-way *Collection* of functions and produce a collection that is both practical and secure.

2 Levin's One Way Function

It is not known whether one-way functions exist. However, it would be nice to assume only that some one-way function exists, rather than assuming that a *particular* function is one-way. The following theorem gives a single constructible function that is one-way if there are any one way functions:

Theorem 1 *There exists a particular polynomial-time computable function f such that f is one-way if and only if there are any one-way functions.*

Proof. We will construct a function f_{Levin} that is weakly one-way. We can then apply the hardness amplification construction from lecture 4 to get the strong one-way function.

The idea behind the contruction is that f_{Levin} should somehow combine the computations in all efficient functions in such a way that inverting f_{Levin} allows us to invert all other functions. The naïve approach is to interpret the input y to f_{Levin} as a machine-input pair $\langle M, x\rangle$, and then let the output of f_{Levin} be $M(x)$. The problem with this approach is that f will not be computable in polynomial time, since M may or may not even terminate on input x.

Instead, we will define f_{Levin} as follows. On input y, f_{Levin} will

1. Interpret y as $\langle M, x\rangle$ where $|M| = \log|x|$
2. Run M on input x for $|y|^3$ steps
3. If M terminates, then $f_{\text{Levin}}(y) = M(x)$, otherwise $f_{\text{Levin}}(y) = \bot$.

We claim that this function is weakly one-way.

Clearly f_{Levin} is computable in $O(n^3)$ time, and thus it satisfies the "easy" criterion for being one-way. To show that it satisfies the "hard" criterion, we must assume that there exists some function g that is strongly one-way.

Without loss of generality, we can assume that g is computable in $O(n^2)$ time. This is because if g runs in $O(n^c)$ for some $c > 2$, then we can let $g'(\langle a, b\rangle) = \langle a, g(b)\rangle$, where $|a| \approx n^c$ and $|b| = n$. Then g' is computable in time

$$\underbrace{|a|}_{\text{copying a}} + \underbrace{|b|^c}_{\text{computing } g} + \underbrace{O(m^2)}_{\text{parsing}} < 2m + O(m^2) = O(m^2)$$

where $m = |\langle a, b\rangle|$ is the input length to g'. Moreover, g' is still one-way, since we can easily reduce inverting g to inverting g'.

Now, if f_{Levin} is not weakly one-way, then there exists a machine $\mathcal{A}$ such that

$$\Pr\left[y \leftarrow \{0,1\}^k; \mathcal{A}(f(y)) \in f^{-1}(f(y))\right] > 1 - 1/q(k)$$

for some polynomial q. But with probability $1/2^{log|n|} = 1/n$, we choose $y = \langle M_g, x\rangle$, so

$$\begin{aligned}
P &= \Pr\left[x \leftarrow \{0,1\}^k; \mathcal{A}(g(x)) \in g^{-1}(g(x))\right] \\
&= \Pr\left[x \leftarrow \{0,1\}^k; \mathcal{A}(f(\langle M_g, x\rangle) \in f^{-1}(f(\langle M_g, x\rangle))\right] \\
&\geq \frac{1}{n}\Pr\left[x \leftarrow \{0,1\}^k; M \leftarrow \{0,1\}^{\log k}; \mathcal{A}(f(\langle M, x\rangle)) \in g^{-1}(g(\langle M, x\rangle))\right] \\
&\geq \frac{1}{n} - \frac{1}{nq(n + \log n)}
\end{aligned}$$

which shows that g is not strongly one-way. This is a contradiction, so f_{Levin} must be weakly one-way. ■

This theorem gives us a function that we can safely assume is one-way (becuase that assumption is equivalent to the assumption that one-way functions exist). However, it is extremely impractical to compute. A very open problem is to find a "nicer" universal one way function (e.g. it would be very nice if f_{mult} is universal).

3 Some Number Theory

Unfortunately the complexity theoretic function presented above is very impractical, for two reasons. First, it is difficult to compute (it involves repeatedly interpreting random turing machines). Second, it will require very large key lengths before the hardness kicks in. To find more practical one-way functions to base our cryptography on, we will turn to number theory.

3.1 Selecting a Random Prime

Our algorithm for finding a random k-bit prime will be to repeatedly guess a k-bit number, and then check that it is prime. In order for this scheme to work, we need two facts:

1. It is easy to determine whether a number is prime
2. There are many primes

Miller and Rabin's algorithm probabilistically determines primality in polynomial time, satisfying the first fact. In practice, this algorithm is very fast. More recently primality has been shown to be computable in deterministic polynomial time.

To satisfy the second requirement, we will resort to Chebychev's theorem:

Theorem 2 (Chebychev's Theorem) *The number of primes between* 1 *and* N *is* $\Omega\left(\frac{N}{\log N}\right)$.

In fact, Chebychev's theorem is a simple version of a stronger theorem:

Theorem 3 (Prime Number Theorem) *The number of primes between* 1 *and* N *approaches* $\frac{N}{\log N}$ *as* $N \to \infty$.

The important consequence is that if I select a k-bit integer uniformly at random, the probability that it is prime is greater than $(N/\log N)/N = \frac{c}{k}$ for sufficiently large k. Thus, the expected number of guesses in the guess-and-check method is polynomial in k, which in turn implies that the expected running time to sample a random prime is polynomial.

3.2 The Factoring Assumption

We will assume the following:

Conjecture 1 (Factoring Assumption) *For every PPT* $\mathcal{A}$*, there exists a negligible function* ϵ *such that*

$$Pr\left[p \leftarrow \{0,1\}^k \textit{ prime}; q \leftarrow \{0,1\}^k \textit{ prime}; n = pq : \mathcal{A}(n) \in \{p,q\}\right] < \epsilon(k)$$

The factoring assumption is a very important, well-studied conjecture that is widely believed to be true. The best provable algorithm for factorization runs in time $2^{O((k \log k)^{1/2})}$, and the best heuristic algorithm runs in time $2^{O(k^{1/3} \log^{2/3} k)}$. Factoring is hard in a concrete way as well: at the time of this writing, researchers have been able to factor a 663 bit numbers using 80 machines and several months.

3.3 Collections of One-Way Functions

Using the factoring assumption, it is easy to show that f_{mult} is a weak one-way function, where $f_{mult}(x, y) = xy$ with $|x| = |y|$. We could then amplify the hardness of f_{mult} using the previous lecture to obtain a strong one-way function as we did for f_{Levin}. However, the hardness amplification would reduce the practicality of f_{mult} as a cryptographic primitive. Instead, we will use a slightly more flexible definition that combines the practicality of a weak OWF with the security of a strong OWF:

Definition 1 (Collection of OWF) *A* collection of one-way functions *is a family $\mathcal{F} = \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I}$ satisfying the following conditions:*

1. *It is easy to sample a function, i.e. there exists a PPT Gen such that Gen(1^k) outputs some $i \in I$.*
2. *It is easy to sample a given domain, i.e. there esists a PPT that on input i returns a uniformly random element of $\mathcal{D}_i$*
3. *It is easy to evaluate, i.e. there exists a PPT that on input $i, x \in \mathcal{D}_i$ computes $f_i(x)$.*
4. *It is hard to invert, i.e. for any PPT $\mathcal{A}$ there exists a negligible function ϵ such that*

$$Pr\left[i \leftarrow Gen; x \leftarrow \mathcal{D}_i; y \leftarrow f_i(x); z = \mathcal{A}(1^k, i, y) : f(z) = y\right] \leq \epsilon(k)$$

We can prove the following result:

Theorem 4 *Let*

$$\begin{aligned} I &= \mathbb{N} \\ \mathcal{D}_i &= \{(p, q) \mid p, q \textit{ prime, and } |p| = |q| = \frac{i}{2}\} \\ f_i(p, q) &= pq \\ \mathcal{F} &= \{f_i : \mathcal{D}_i \to \mathcal{R}_i\}_{i \in I} \end{aligned}$$

Then $\mathcal{F}$ is a collection of OWF.

Proof. We can clearly sample a random element of $\mathbb{N}$. We showed above that the guess-and-check algorithm uniformly samples a random prime, so we can clearly uniformly sample an element of $\mathcal{D}_i$. It is easy to evaluate f_i since that's just multiplication, and the factoring assumption says that inverting f_i is hard. Thus all four conditions are satisfied. ■

Finally, we will show that the existence of a collection of one-way functions is equivalent to the existence of a strong one-way function:

Theorem 5 *There exists a collection of one-way functions if and only if there exists a single strong one-way function*

Proof idea: If we have a single one-way function f, then we can choose our index set ot be the singleton set $I = \{0\}$, choose $\mathcal{D}_0 = \mathbb{N}$, and $f_0 = f$.

The difficult direction is to construct a single one-way function given a collection $\mathcal{F}$. The trick is to define $g(r_1, r_2)$ to be $i, f_i(x)$ where i is generated using r_1 as the random bits and x is sampled from $\mathcal{D}_i$ using r_2 as the random bits. The fact that g is a strong one-way function is left as an excercise. ■