

Lecture 5: Universal One-Way Function and Computational Number Theory

Instructor: Rafael Pass

Scribe: Edward Lui

1 Universal One-Way Function

Recall the definition of a (strong) one-way function and a weak one-way function:

Definition 1 A function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ is a (strong) one-way function if the following holds:

1. f is easy to compute: There exists a PPT algorithm that computes f .
2. f is hard to invert: For every NU PPT algorithm A , there exists a negligible function ϵ such that for all $n \in \mathbb{N}$,

$$\Pr(x \xleftarrow{r} \{0, 1\}^n : A(1^n, f(x)) \in f^{-1}(f(x))) \leq \epsilon(n).$$

Definition 2 A function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ is a weak one-way function if the following holds:

1. f is easy to compute: There exists a PPT algorithm that computes f .
2. f is slightly hard to invert: There exists a polynomial p such that for every NU PPT algorithm A and sufficiently large $n \in \mathbb{N}$,

$$\Pr(x \xleftarrow{r} \{0, 1\}^n : A(1^n, f(x)) \in f^{-1}(f(x))) \leq 1 - \frac{1}{p(n)}.$$

Last lecture, we showed that strong one-way functions exist iff weak one-way functions exist. How does one find a one-way function? The existence of one-way functions implies that $P \neq NP$. Thus, if $P = NP$, then there are no one-way functions. By assuming that one-way functions exist, we are also assuming that $P \neq NP$. (In fact, we are also assuming that $NP \not\subseteq BPP$.) We currently do not know whether one-way functions exist, although there are candidate functions that are widely believed to be one-way. We now present a fixed “universal” function that is one-way iff one-way functions exist.

Theorem 1 There exists an explicit function f_{univ} such that if there exists a one-way function, then f_{univ} is one-way.

Proof. We first prove the following claim:

Claim. If there exists a one-way function, then there exists a one-way function that can be computed in time n^2 .

Proof of Claim. Given a one-way function f that can be computed in time n^c , define the function f' by $f'(a||b) = a||f(b)$, where $|a| = m^c$ and $|b| = m$ for some $m \in \mathbb{N}$. (We use $||$ to denote concatenation of strings.) One can easily verify that f' can be computed in time n^2 , where n is the length of the input. Now, suppose that f' is not a one-way function. Then, there exists an NU PPT algorithm A' and a polynomial p' such that for infinitely many m ,

$$\Pr(x \xleftarrow{r} \{0, 1\}^{m^c+m} : A'(1^{m^c+m}, f'(x)) \in f'^{-1}(f'(x))) \geq \frac{1}{p'(m^c + m)}.$$

Now, let A be the following algorithm:

On input $(1^n, y)$,

1. $a \xleftarrow{r} \{0, 1\}^{(n^c)}$.
2. $a' || b' \leftarrow A'(1^{n^c+n}, a || y)$, where $|a'| = \min\{n^c, A'(1^{n^c+n}, a || y)\}$.
3. Output b' .

Let $p(n) = p'(n^c + n)$; we note that p is a polynomial. Since A' is a NU PPT algorithm, we see that A is also a NU PPT algorithm. By our construction of A , we also have

$$\begin{aligned} & \Pr(x \xleftarrow{r} \{0, 1\}^n : A(1^n, f(x)) \in f^{-1}(f(x))) \\ & \geq \Pr(x \xleftarrow{r} \{0, 1\}^{n^c+n} : A'(1^{n^c+n}, f'(x)) \in f'^{-1}(f'(x))) \\ & \geq \frac{1}{p'(n^c + n)} \\ & = \frac{1}{p(n)} \end{aligned}$$

for infinitely many $n \in \mathbb{N}$. This contradicts the fact that f is a one-way function.

■ (Claim)

Recall that there are only countably many Turing machines (for any fixed alphabet). Let M_1, M_2, M_3, \dots be a list of all the Turing machines such that the length of the description of M_n is bounded by some polynomial in n . Now, let

$$f_{univ}(x) = M_1^{(|x|^2)}(x) || M_2^{(|x|^2)}(x) || \dots || M_{|x|}^{(|x|^2)}(x)$$

where $M_i^{(|x|^2)}(x)$ denotes the output of M_i on input x right after $|x|^2$ steps of computation. One can verify that f_{univ} can be computed in polynomial time. Now, suppose there exists a one-way function f . Then, there exists a Turing machine in our list, say M_K , that computes f . We note that for $x \in \{0,1\}^*$ such that $|x| \geq K$, $f_{univ}(x)$ contains $M_K(x) = f(x)$. One can finish the proof using standard reduction techniques. ■

See the lecture notes on the course website for an alternate f_{univ} .

2 Computational Number Theory

It will be assumed that students have sufficient background knowledge in number theory. Students who do not have this background knowledge may want to consult the lecture notes on the course website. The lecture notes cover just enough number theory for a student to understand the material in this course.

Theorem 2 (Agrawal, Kayal, Saxena) *There exists a deterministic polynomial-time algorithm that checks whether a number is prime.*

See the paper “Primes is in P” for a proof of the above theorem. The following two theorems describe the asymptotic distribution of the prime numbers.

Theorem 3 (Chebyshev) *The number of primes between 1 and N is $\omega(\frac{N}{\log N})$.*

Theorem 4 (Prime Number Theorem) *The number of primes between 1 and N approaches $\frac{N}{\log N}$ as $N \rightarrow \infty$.*

To generate a random n -bit prime (with sufficiently high probability), one can simply choose a random $N \in \{0,1\}^n$ and then run a primality test (e.g. Miller-Rabin) on N to check whether N is prime (with sufficiently high probability). If the test says that N is prime, we are done. Otherwise, we choose another random n -bit number and start over. By the prime number theorem, the number of n -bit primes is approximately $\frac{2^n}{\log 2^n} = \frac{c2^n}{n}$ for some constant $c > 1$. Thus, a fraction $\approx \frac{c}{n}$ of the integers in $\{0,1\}^n$ are prime. Thus, the expected number of trials before we obtain a prime (in the above procedure) is linear in n .

Now, let f_{mult} be defined by $f_{mult}(x, y) = x \cdot y$, where $|x| = |y|$.

The Factoring Assumption: For every NU PPT algorithm A , there exists a negligible function ϵ such that for every $n \in \mathbb{N}$,

$$\Pr(p, q \stackrel{r}{\leftarrow} \{n\text{-bit primes}\}; N = pq : A(N) \in \{p, q\}) \leq \epsilon(n).$$

The factoring assumption is widely believed to be true. Much time and effort have been

spent on trying to find an efficient algorithm for factoring the product of two large prime numbers, but the best provable algorithm only runs in time $2^{O(n^{1/2} \log^{1/2} n)}$, and the best heuristic algorithm runs in time $2^{O(n^{1/3} \log^{2/3} n)}$. Factoring a 500-bit number (that is the product of two prime numbers) took 300 machines and 4 months.

Theorem 5 *Assuming that the factoring assumption is true, f_{mult} is a weak one-way function.*

Proof. Suppose f_{mult} is not a weak one-way function. Then, for every polynomial q , there exists a NU PPT algorithm A such that for infinitely many $n \in \mathbb{N}$,

$$\Pr(x, y \leftarrow \{0, 1\}^n : A(1^{2n}, f_{\text{mult}}(x, y)) \in f_{\text{mult}}^{-1}(f_{\text{mult}}(x, y))) \geq 1 - \frac{1}{q(2n)}.$$

If we choose q to be $q(n) = \frac{1}{2}n^2$, then there exists a NU PPT algorithm A such that for infinitely many $n \in \mathbb{N}$,

$$\Pr(x, y \leftarrow \{0, 1\}^n : A(1^{2n}, f_{\text{mult}}(x, y)) \in f_{\text{mult}}^{-1}(f_{\text{mult}}(x, y))) \geq 1 - \frac{1}{2n^2}.$$

Now, let A' be the following algorithm:

On input $z \in \{0, 1\}^{2n}$,

1. $x, y \xleftarrow{r} \{0, 1\}^n$.
2. If x and y are both prime, let $z' = z$. Otherwise, let $z' = x \cdot y$.
3. Output $A(1^n, z')$.

Since one can check whether an integer is prime in polynomial time, and since A is a NU PPT algorithm, A' is also a NU PPT algorithm. We will show that A' inverts the product of two random n -bit primes with nonnegligible probability, for infinitely many n .

For $x, y \in \{0, 1\}^n$, we say that (x, y) is *bad* if one of them is not prime; otherwise, we say (x, y) is *good*. For infinitely many n , we have

$$\Pr(x, y \xleftarrow{r} \{0, 1\}^n : (x, y) \text{ is bad}) \leq 1 - \frac{1}{n^2},$$

and

$$\Pr(A' \text{ fails to invert input} \mid (x, y) \text{ is good}) \leq \frac{1}{2n^2}.$$

Thus, we have

$$\Pr(A' \text{ fails to invert input}) \leq 1 - \frac{1}{n^2} + \frac{1}{2n^2} = 1 - \frac{1}{2n^2},$$

so

$$\Pr(A' \text{ succeeds at inverting input}) \geq \frac{1}{2n^2}.$$

Thus, A' inverts the product of two random n -bit primes with nonnegligible probability (for infinitely many n). This contradicts the factoring assumption. ■