Primality and One-Way Functions

Slides by Dan Fu

Based on notes by Hoffoss (UCSD) and Kleinberg (Cornell)

Importance of primes

- Much of modern day digital encryption depends on large, prime numbers
- Multiplication of any two numbers, prime or not, is polynomial time
- Factoring prime numbers is a difficult problem
- How can we efficiently find prime numbers to use?
  - Large numbers can be generated quickly using PRNGs
  - How do we efficiently verify that they are primes?

Fermat’s little theorem

- Primality testing is an application of randomized algorithms
- For a number p, we can check whether it is *probably* prime or composite
- If a base a does not divide p, and p is prime, then for $1 < a < n - 1$.
$$a^{p-1} \equiv 1 \bmod p$$
- It’s not efficient to check all a, pick some random a, then if the congruency holds, it could be prime, and we repeat this process.
- But if for any a, the congruency does not hold, then the number is definitely composite.

Given the above definition of Fermat’s little theorem for deciding primes, what complexity class best describes it?

coRP

Definition. *The complexity class **co-RP** is the class of all languages L for which there exists a p.p.t. M such that*

$$x \in L \Rightarrow P[M(x) = 1] = 1$$
$$x \notin L \Rightarrow P[M(x) = 0] \geq 1/2$$

- If x is prime, then Fermat's little theorem outputs prime
- If x is not prime, then Fermat's little theorem sometimes outputs prime, but with suppressible error
- Fortunately, for p that is composite and not a Carmichael number, at least half of the values $1, 2, \ldots, p-1$ are Fermat witnesses

Fermat’s little theorem

- Example: Is $p = 221$ prime?
  - Let $a = 38$
  - $38^{220} \equiv 1 \bmod 221$
  - Possibly prime, 221 is pseudoprime to 38

  - Let $a = 24$
  - $24^{220} \equiv 81 \bmod 221 \ \not\equiv 1 \bmod 221$
  - $\therefore p$ is composite, 24 is a *Fermat witness*, and 38 is a *Fermat liar*

Carmichael numbers

- Makes a problem for Fermat's little theorem
- Carmichael numbers are a set of composite numbers that satisfy the modular congruency method
- Example, 561 is the smallest Carmichael number:
  - $p = 561, a = 5$
  - $5^{560} \equiv 1 \bmod 561$
  - $561 = 3 \times 11 \times 17$
- There are few Carmichael numbers, but still a problem, although it is not expensive to solve

Primality Testing

- Rely on one more condition that primes satisfy, which is a “fake square root”
- Show that a number x satisfying $x^2 \equiv 1 \bmod p$ and $x \not\equiv 1 \bmod p$
- That is, if p is prime, then its modular square root should be only $\pm 1 \bmod p$
- Then if there are any other different square roots, then p is composite
- This is known as the Euler method of primality testing, but still has undetectable pseudoprimes (try 1729 and 2465)

Miller-Rabin test

- More comprehensively applies the Euler test
- Let $p - 1 = 2^s \cdot m$, where m is odd and $s \geq 1$
- Then begin with a^m and run the Euler test
- If $a^m \equiv \pm 1 \; mod \; p$, return possibly prime and stop, otherwise square a^m and continue
- If $a^{2m} \equiv 1 \; mod \; p$, return composite and stop, because a^m is also a square root of a^{2m}
- If $a^{2m} \equiv -1 \; mod \; p$, return possibly prime and stop, otherwise square the base a again and continue
- Repeat this process until returning to a^{p-1}, and return composite
- Can be shown that the false positive rate (Miller-Rabin declares that a composite number is prime) is at most 1/4 of integers a, $1 \leq a < n$ and $\gcd(a, p) = 1$.

Examples

- Fermat's little theorem:
  - $2^{340} \equiv 1 \; mod \; 341$ (incorrectly returns prime)
  - $5^{560} \equiv 1 \; mod \; 561$ (incorrectly returns prime)
- Euler test
  - $2^{170} \equiv 1 \text{ mod } 341$ (incorrectly returns prime)
  - $5^{280} \equiv 67 \; mod \; 561$ (correctly returns composite for Carmichael number)
- Miller-Rabin
  - For $m = 85, s = 2$
    - $2^{85} \; mod \; 341 \equiv 32 \; mod \; 341$ (correctly returns composite)
  - For $m = 35, s = 5$
    - $5^{35} \; mod \; 561 \equiv 23 \; mod \; 561$ (correctly returns composite)

One-way functions

- A function f that is computable in polynomial time, but unlikely to be inverted, that is, any adversary $\mathcal{A}$ operating on output $y = f(x)$ finds some string x^* that exactly produces $y = f(x) = f(x^*)$ with negligible probability
- A one-way permutation f is computable in polynomial time, and for an input w, given $y = f(w)$, any adversary $\mathcal{A}$ finds w with negligible probability
- It is important that $FACTORIZATION$ is a one-way function, although, the formal existence of one-way functions has not been proven
- If one-way functions are proven to exist, it also implies that $P \neq NP$.

Let f be a length-preserving one-way function (i.e. $|f(x)| = |x|$). Prove or disprove that that $g(x) = f(f(x))$ is a one-way function.

- If $g(x)$ is a one-way function, there will be some application of f such that any adversary $\mathcal{A}$ is unlikely to find some $g(x^*) = f\big(f(x)\big)$
- We must construct such an f and also show that it is one-way and length-preserving

Let f be a length-preserving one-way function (i.e. $|f(x)| = |x|$). Prove or disprove that that $g(x) = f(f(x))$ is a one-way function.

Suppose there is a length-preserving, one-way permutation $p(x)$
Then let us define $f(x)$ as the following:

$$f(x) = \begin{cases} 0^{|x|} & \text{if } x_{\frac{n}{2}+1}, \dots, x_n = y0^{|x|/2} \\ p(x_1, \dots, x_{n/2})0^{|x|/2} & \text{otherwise} \end{cases}$$

In other words, if x has trailing zeros, f replace the entire string by a length of 0s. Otherwise, f is a one-way permutation of the first $n/2$ bits of x, with trailing 0s added.

Clearly in this construction, $f(f(x))$ is always a string of 0s, and it is trivial for some g to find some x^* that generates $f(f(x))$.

Let f be a length-preserving one-way function (i.e. $|f(x)| = |x|$). Prove or disprove that that $g(x) = f(f(x))$ is a one-way function.

We must now show that our assumption f is indeed a length-preserving one-way function.

Clearly, f is **length-preserving** as it always pads the string up to $|x|$.

To show that f is **one-way**, we use a proof by contradiction.

Suppose that f is not a one-way function. Then it is possible there is some adversary $\mathcal{A}$ where f can be inverted with non-suppressible error.

It follows that we can construct an adversary $\mathcal{A}'$ that queries $\mathcal{A}$ when receiving some input y. Instead of inverting some $y \in \{0,1\}^{|x|/2}$, $\mathcal{A}'$ queries $\mathcal{A}$ on a padded input string $y0^{|x|/2}$. Since we know $\mathcal{A}$ can be inverted, that is, it is a p.p.t. that finds x with high likelihood, then $\mathcal{A}'$ can also be inverted in non-negligible time. We see that $x = p(x)$.

This contradicts our assumption that $p(x)$ is a one-way permutation, and thus not invertible.

$\therefore$ $g(x) = f(f(x))$ is not necessarily a one-way function

What if f is a length-preserving one-way permutation (in the homework!)

Show that if **P**=**NP** then one-way functions do not exist.

A one-way function relies on some inverse being hard to compute, so our goal should be to show that some adversary $\mathcal{A}$ can easily compute the inverse, if $P = NP$.

Show that if **P**=**NP** then one-way functions do not exist.

Let x be a unary string 1^n, where $n = |x|$ and $f(x) = y$. Let x_0 also be some prefix of x. n helps to bounds the size of x.

Define a language L that includes the set of all x that satisfy $f(x) = y$

Claim: $L \in NP$

x is a trivial witness, since one only needs to verify that $f(x) = y$, and since f is polynomial-time computable, then that satisfies the existence of a polynomial-time verifier.

Show that if **P**=**NP** then one-way functions do not exist.

If $P = NP$, this implies that the language $L \in P$, meaning it is possible to *solve* for some x satisfying $f(x) = y$ in polynomial time.

Suppose we have some adversarial machine $M_{\mathcal{A}}$ that inverts y. It can do so by beginning with an empty string ε, then adding 1 bits to the string, and verifying whether that (x_0, y) is a satisfying assignment of $f(x_0) = y$. Since $M_{\mathcal{A}}$ always operates in polynomial-time, then the process of finding some x_0 that will generate the output of f guarantees that f can be inverted in polynomial time.

Thus, f cannot be a one-way function.

Show that if **P**=**NP** then one-way functions do not exist.

By extension, it is also worth nothing that pseudorandom generators exist **if and only if** one-way functions exist.

The forward direction is trivial. If g is a PRNG then it must also be a one-way function. We can see this by what we mean to be pseudorandom, that is, that no efficient computation exists which can distinguish a pseudorandom sequence from a truly random, uniform distribution of values with non-negligible probability.

The “only if” direction shows that if one-way functions exist, we can build a PRNG from it. This is much more involved and relies on disproving an assumption about being able to predict the hardcore bit in the Goldreich-Levin theorem such that stretching a pseudorandom sequence by 1 bit can be inverted with some probability > 1/2. Then, (as you’ll show with one-way permutations in your homework) this concept is straightforwardly extended to any length n^c, $c > 0$.

Suppose $g: \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ is any pseudorandom generator. Then use g to describe a pseudorandom generator that stretches n bits to n^k for any constant $k > 1$.

We presume that the generator function g holds for any value of n. There always exists an application of g on a pseudorandom sequence n that "stretches" it to a pseudorandom sequence $n + 1$.

Briefly, if the procedure g is a one-way function, then it suffices to describe the PRG w.r.t. a circuit C that cannot distinguish any application of g from a random sequence. In this case, the probability that a random sequence and a pseudorandom sequence differ when generating $n + 1$ from n should be negligible within some error bound ε.

Suppose $g:\{0,1\}^n \to \{0,1\}^{n+1}$ is any pseudorandom generator. Then use g to describe a pseudorandom generator that stretches n bits to n^k for any constant $k > 1$.

Let $T: \{0,1\}^n \to \{0,1\}$ be some test for determining the next bit when given a sequence of n bits.

Then,

$$\left|Pr_{R \leftarrow U_n}[T(R) = 1] - Pr_{S \leftarrow U_n}[T\big(g(S)\big) = 1]\right| \le \varepsilon$$

, where $R \leftarrow U_n$ is a uniform random distribution, and $S \leftarrow U_n$ is a uniform pseudorandom string.

It's clear that this applies recursively for any n, so it is also a valid description of any extension of n bits to n^k for $k > 1$.

This is what is commonly known as the seed-extending pseudorandom generator approach.

Note that if we wanted to prove the pseudorandomness of the PRG, then we must also find some derandomization machine D applying the test T that cannot succeed except at some negligible error $\rho + \varepsilon$, where ρ is some measure of the pseudorandomness.