Lecture 12: Generating Large Primes

Generating n-bit Primes

- Let us begin by constructing an algorithm that outputs n-bit prime numbers **IF** we can efficiently test whether a number is a prime or not
- We shall assume that Is_Prime(X) returns true if and only if X is prime number; otherwise, it returns false.
- **Idea of the Algorithm.** We will pick a random number X in the range $\left[2^{n-1}, 2^n - 1\right]$ and test whether X is a prime number or not. If X is a prime number, then we are done. Otherwise, we repeat.
- Consider the code for this algorithm

Generate_Prime(n) :

1. While (true):
    1. $X \xleftarrow{\$} \left[2^{n-1}, 2^n - 1\right]$
    2. If Is_Prime(X) then return X

- To understand the running time of this algorithm, we will have to understand the distribution of the primes. In particular, how densely are prime numbers distributed over natural numbers? Towards this, we shall use the following theorem without proof.

Theorem (Prime Number Theorem)

The total number of primes $< N$ is (roughly) $P_N = N/\log N$.

- So, the number of primes in the range $\left[2^{n-1}, 2^n - 1\right]$ is $P_{2^n} - P_{2^{n-1}}$. Let us calculate this expression

$$\begin{aligned} P_{2^n} - P_{2^{n-1}} &= \frac{2^n}{n} - \frac{2^{n-1}}{n-1} \\ &= 2^{n-1}\left(\frac{2}{n} - \frac{1}{n-1}\right) \\ &\approx 2^{n-1} \cdot \frac{1}{n} \end{aligned}$$

- Note that the total number of integers in the range $\left[2^{n-1}, 2^n - 1\right]$ is exactly 2^{n-1}
- **First Conclusion.** As we have seen above, among these 2^{n-1}, we have (roughly) $2^{n-1} \cdot \frac{1}{n}$ prime numbers. If we pick a number at random from the range $\left[2^{n-1}, 2^n - 1\right]$, then the probability that the number is a prime is

$$\frac{2^{n-1} \cdot \frac{1}{n}}{2^{n-1}} = \frac{1}{n}$$

- Intuitively, if we want to generate an $n = 1000$ bit prime number, then the probability that a random number in the range $\left[2^{99}, 2^{100} - 1\right]$ is a prime number is (roughly) $\frac{1}{n} = \frac{1}{1000}$.

- **Failing to generate a prime in t attempts.** Note that the probability that we fail to generate a prime number in one iteration of the loop is given by

$$\left(1 - \frac{1}{n}\right)$$

So, the probability of failing to generate a prime number in two iterations of the loop is given by

$$\left(1 - \frac{1}{n}\right)^2$$

Similarly, the probability of failing to generate a prime number in t iterations of the loop is given by

$$\left(1 - \frac{1}{n}\right)^t$$

- **Recall from calculus classes.** Consider $f(x) = \left(1 - \frac{1}{x}\right)^x$. We know that
$$\lim_{x \to \infty} f(x) = \frac{1}{e}$$
- If $t = \alpha n$, then the probability that t iterations of the loop fail to generate a prime is given by
$$\left(1 - \frac{1}{n}\right)^t = \left(1 - \frac{1}{n}\right)^{\alpha n} \approx \exp(-\alpha)$$
- **Conclusion about the running time of the** Generate_Prime **Algorithm.** Now, we can conclude that the algorithm runs the loop t-times and does not find a prime number is at most $\exp(-t)$

Ideal Primality Testing

- Our objective is to construct the Is_Prime algorithm
- Ideally, we would like the following input/output behavior from the Is_Prime algorithm. In the following table, we write the probability that the algorithm says true/false conditioned on the fact that the input X is a prime or composite.

	Is_Prime says true	Is_prime says false
X is a prime	1	0
X is a composite	0	1

Read the table as follows. Conditioned on the fact that X is a prime number, we want Is_Prime(X) to return true with probability 1. And, conditioned on the fact that X is a composite number, we want Is_Prime(X) to return false with probability 1.

- This algorithm should run in time that is polynomial in $x = \log X$, the number of bits needed to represent the number X
- This ideal algorithm was discovered in 2002 by Agrawal–Kayal–Saxena. In practice, this algorithm is not used for primality testing because this turns out to be too slow.
- In practice, we use a randomized algorithm, namely, the Miller–Rabin Test, that successfully distinguishes primes from composites with very high probability. In this lecture, we will study a basic version of this Miller–Rabin primality test

Miller–Rabin Primality Testing Algorithm

- Suppose we implement the Is_Prime algorithm using the Miller–Rabin Primality Testing Algorithm. This is a randomized algorithm with a one-sided error. It takes as an additional parameter t and assures the following

	Is_Prime says true	Is_prime says false
X is a prime	1	0
X is a composite	$\leqslant 2^{-t}$	$\geqslant 1 - 2^{-t}$

- When X is a prime number, the algorithm does not err! However, when X is a composite number, the algorithm might mistakenly declare it as a prime with probability 2^{-t}. So, we say that it has a one-sided error.
- The running time of the algorithm is polynomial in x (the number of bits needed to represent the number X) and t
- For example, we can test the primality of a 1000-bit prime with confidence $1 - 2^{-500}$ in running time that is polynomial in $x = 1000$ and $t = 500$.

- In this section, we aim to implement the Is_Prime test using the basic Miller–Rabin test
- The basic Rabin–Miller test that we shall learn in this class NEARLY works. Except that for some VERY WEIRD inputs X, it messes up badly. These very weird inputs are composite; however, the basic Miller–Rabin algorithm returns true (i.e., it mistakenly declares it as a prime number)
- These VERY WEIRD composite numbers are called the Carmichael numbers. There are infinitely many Carmichael numbers. Otherwise, we could have tested if X belongs to this finite set! However, the good news is that they are very sparse. Erdös proved that the total number of Carmichael numbers $< N$ is given by

$$\frac{N}{\exp\left(n\frac{\lambda \log\log n}{\log n}\right)},$$

where $\lambda > 0$ is a constant and n is the number of bits needed to represent N. So, it is very unlikely that a random number being tested for primality is, in fact, a Carmichael number. The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$

- **What will Basic Rabin–Miller achieve?** The guarantees are summarized below.

	Is_Prime says true	Is_prime says false
X is a prime	1	0
X is Carmichael	≈ 1	≈ 0
X is a composite but not Carmichael	$\leqslant 2^{-t}$	$\geqslant 1 - 2^{-t}$

- **Atomic Test.** Consider the following loop

1. Pick $\alpha \xleftarrow{\$} \{1, 2, \ldots, X-1\}$
2. Compute $\beta = \alpha^{X-1} \mod X$
3. If $(\beta == 1)$: Return pass
4. Else: Return fail

Recall that we have proven in the homework that $\alpha^{X-1} = 1 \mod X$ for a prime X, for any integer $X \in \{1, 2, \ldots, X-1\}$. So, we can conclude the following.

	Atomic Loop says pass	Atomic Loop says fail
X is a prime	1	0

Even though they are composite numbers, Carmichael numbers are such that for $\alpha \in \{1, \ldots, X-1\}$ relatively prime to X satisfies $\alpha^{X-1} = 1 \mod X$. Typically there are a lot of numbers in $\{1, \ldots, X-1\}$ that are relatively prime to X. So, we can conclude that

	Atomic Loop says pass	Atomic Loop says fail
X is a prime	1	0
X is Carmichael	≈ 1	≈ 0

When X is a composite number that is not a Carmichael number, we shall use the following result without proof.

Lemma

Suppose X is a composite number that is not a Carmichael number. Then, at least half of $\alpha \in \{1, \ldots, X-1\}$ satisfy $\alpha^{X-1} \neq 1 \mod X$.

This result helps us conclude that the Atomic Loop will output fail with probability $\geqslant 1/2$. So, we can fill the final row in the table as

	Atomic Loop says pass	Atomic Loop says fail
X is a prime	1	0
X is Carmichael	≈ 1	≈ 0
X is composite but not Carmichael	$\leqslant 1/2$	$\geqslant 1/2$

- **Using Atomic Loop to get the Basic Miller–Rabin Test.**
  Suppose X is not a Carmichael number but is composite. Note that when we run the atomic loop once, the probability that the algorithm mistakenly says pass is $\leqslant 1/2$. Similarly, when we run the atomic loop twice. Then the probability that both loops say pass both times is $\leqslant (1/2)^2$. If we run the atomic loop t-times, then the probability that all loops say pass is $\leqslant (1/2)^t = 2^{-t}$.
  Note that if we say any atomic loop report fail, we know that X is composite (and not Carmichael).
  So, we consider the following algorithm.

Is_Prime(X, t)
/*Implemented using Basic Miller–Rabin Algorithm*/

1. For $i = 1$ to t
   1. $\alpha \xleftarrow{\$} \{1, 2, \ldots, X-1\}$
   2. $\beta = \alpha^{X-1} \mod X$
   3. If ($\beta \neq 1$): Return false
2. Return true

The guarantee of this algorithm is

	Is_Prime says true	Is_prime says false
X is a prime	1	0
X is Carmichael	≈ 1	≈ 0
X is a composite but not Carmichael	$\leqslant 2^{-t}$	$\geqslant 1 - 2^{-t}$