CMSC/Math 456: Cryptography (Fall 2022)

Lecture 12

Daniel Gottesman

Administrative

Problem set #4 due, Solution set #3 out soon, Problem set #5 available.

Midterm: Thursday, Oct. 20 (2 weeks from today)

- In class
- Open book (including textbook), no electronic devices
- Will cover classical cryptographic, private key encryption, and public key encryption and key exchange, including all topics discussed under those general subjects (such as number theory).
- Those with accommodations remember to book with ADS.

About Modular Arithmetic

When you are thinking about modular arithmetic, you should be thinking of this as a *separate* arithmetic system than integer arithmetic that simply happens to agree with integer arithmetic sometimes.

> In computer science terms, modular arithmetic is working with a different *type* than integer arithmetic. When an equation has a "mod N" label, that is a sign that you are working with modular type, not integer type.

You can convert back and forth between modular type and integer type by taking advantage of the fact that they agree for numbers less than N, but you should bear in mind which you are working with.

Chinese Remainder Theorem

Chinese Remainder Theorem: Let N = ab, with a and b relatively prime. Given any pair of non-negative integers (x_a, x_b), with $x_a < a$ and $x_b < b$, there exists a unique non-negative integer $x < N$ such that $x = x_a \bmod a$ and $x = x_b \bmod b$. There is an efficient algorithm to compute x.

Algorithm:

1. Using Euclid's algorithm, compute X and Y such that $aX + bY = 1$.
2. Let $x = x_b aX + x_a bY$.

Why does this algorithm work?

$bY = 1 - aX$, so $x = (x_b X - x_a X)a + x_a$, so $x = x_a \bmod a$.

This class is being recorded

Chinese Remainder Thm Example

Example:

Suppose we want to find an x such that

$$x = 3 \bmod 5 = x_a \bmod a$$
$$x = 5 \bmod 14 = x_b \bmod b$$

We apply Euclid's algorithm to see that

$$3 * 5 - 1 * 14 = 1$$

We thus have aX = 15 and bY = -14.

We then have

$$x = 5 * 15 - 3 * 14 = 33$$

Solving Discrete Log

Suppose you are given p, g, with p-1 a product of small primes.
You are given y and wish to find x such that $g^x = y \bmod p$.

This class is being recorded

Solving Discrete Log

Suppose you are given p, g, with p-1 a product of small primes.
You are given y and wish to find x such that $g^x = y \bmod p$.

If $p - 1 = \prod_i p_i$, calculate $y_i = y^{(p-1)/p_i} \bmod p$ for all i.

This class is being recorded

Solving Discrete Log

Suppose you are given p, g, with p-1 a product of small primes. You are given y and wish to find x such that $g^x = y \bmod p$.

If $p - 1 = \prod_i p_i$, calculate $y_i = y^{(p-1)/p_i} \bmod p$ for all i.

Now, $(g^{(p-1)/p_i})^x = (g^x)^{(p-1)/p_i} = y_i \bmod p$. But $g_i = g^{(p-1)/p_i}$ has only order p_i in $\mathbb{Z}_p^*$, so we can easily check all powers to find an x_i such that $g_i^{x_i} = y_i \bmod p$.

Solving Discrete Log

Suppose you are given p, g, with p-1 a product of small primes. You are given y and wish to find x such that $g^x = y \bmod p$.

If $p - 1 = \prod_i p_i$, calculate $y_i = y^{(p-1)/p_i} \bmod p$ for all i.

Now, $(g^{(p-1)/p_i})^x = (g^x)^{(p-1)/p_i} = y_i \bmod p$. But $g_i = g^{(p-1)/p_i}$ has only order p_i in $\mathbb{Z}_p^*$, so we can easily check all powers to find an x_i such that $g_i^{x_i} = y_i \bmod p$.

Since g_i has order p_i, $g_i^x = g_i^{x_i} \bmod p$ whenever

$$x = x_i \bmod p_i$$

we use the Chinese remainder theorem in order to find such an x. Then $y = g^x$. (This follows because the x given by the Chinese remainder theorem is unique.)

Discrete Log Example

Example: g = 65, p = 71, so $p - 1 = 2 \cdot 5 \cdot 7$.

Given y = 54, what is x such that $65^x = 54 \bmod 71$?

$65^{10} = 20 \bmod 71$ $\quad 65^{14} = 5 \bmod 71$ $\quad 65^{35} = 70 \bmod 71$

Calculate $54^{10} = 1 \bmod 71$ and compare to $20^{x_0} \bmod 71$.

$x_0 = 0 \bmod 7$

Calculate $54^{14} = 25 \bmod 71$ and compare to $5^{x_1} \bmod 71$.

$x_1 = 2 \bmod 5$

Calculate $54^{35} = 1 \bmod 71$ and compare to $70^{x_1} \bmod 71$.

$x_2 = 0 \bmod 2$

$x = 0 \bmod 14, x = 2 \bmod 5$

Formula for Chinese remainder theorem

$\Rightarrow$ $x = 15 \cdot 0 - 14 \cdot 2 = -28 = 42 \bmod 70$

$\boxed{65^{42} = 54 \bmod 71}$

This class is being recorded

Safe Primes

So for Diffie-Hellman to be secure, we need to find a prime base p such that p-1 has at least one large factor, and we also need to be able to find g with large order mod p.

> We will look for a prime of the form $p = rq + 1$, where r is small (e.g., r=2) and q is also prime. This guarantees a large prime factor for p-1.

With a p of this form, $p - 1 = rq$, so we can easily find an element g of order q with the following procedure:

1. Choose random $x \in \mathbb{Z}_p^*$.
2. Let $g = x^r \bmod p$.
3. Repeat until $g \neq 1$.

Steps 1 and 2 generate a random element of the order q cyclic subgroup of $\mathbb{Z}_p^*$. Since q is prime, all elements of that subgroup have order q except for 1.

Finding Primes

How can we find a prime, let alone a prime of a specific form?

1. Choose a random number p of the desired length.
2. Check that p is prime.
3. Check that $(p-1)/r$ is prime.
4. Repeat until both p and $(p-1)/r$ are prime.

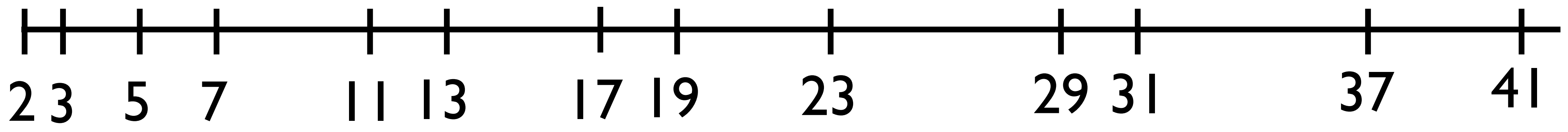
Remarkably, this works. But there are two pieces needed to make it work:

- We need to be sure that primes are sufficiently common that we can find a prime in a reasonable time.
- We need an efficient algorithm to check that a number is prime.

For secure Diffie-Hellman, we will need p that is at least thousands of bits long, so efficiency is important.

How Common Are Primes?

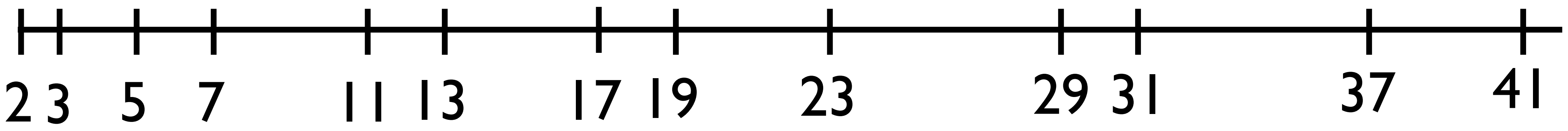
What do we expect?



Primes get rarer as the numbers get larger, but only slowly.

How Common Are Primes?

What do we expect?



Primes get rarer as the numbers get larger, but only slowly.

Prime Number Theorem: Let $\pi(n)$ be the number of primes less than or equal to n. Then

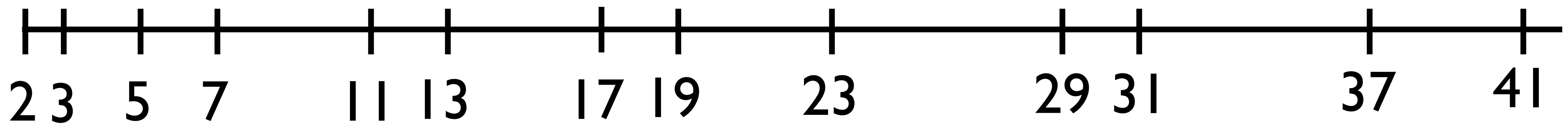
$$\pi(n) \approx n/\ln n$$

The probability that a random s-bit number is prime is about 1/s.

This class is being recorded

How Common Are Primes?

What do we expect?



Primes get rarer as the numbers get larger, but only slowly.

Prime Number Theorem: Let $\pi(n)$ be the number of primes less than or equal to n. Then

$$\pi(n) \approx n / \ln n$$

The probability that a random s-bit number is prime is about 1/s.

Therefore, if we choose s-bit numbers at random, we find a prime after O(s) tries, which is efficient.

This class is being recorded

Testing Primes

We also need a method to test for primes: Given N, is N prime?

Fermat's Little Theorem states that, if N is prime, then

$$x^N = x \bmod N$$

for all x.

This suggests the following algorithm:

1. Choose random x
2. Calculate $y = x^N \bmod N$
3. If $y \neq x$, end the loop and return: Composite
4. Repeat steps 1-3 a number of times
5. If we are still going, return: Prime

Vote: Does this algorithm work? (Yes/No) Answer: No

This class is being recorded

Pseudoprimes

Unfortunately, there are some composite numbers N such that

$$x^N = x \bmod N$$

for all x. These are called pseudoprimes or Carmichael numbers.

The smallest one is 561. They seem to be rarer than prime numbers, but it is not clear if they are sufficiently rare that we can neglect the probability of choosing one if we choose N at random.

(Carmichael numbers fail the test $x^{N-1} = 1 \bmod N$ when x is not relatively prime to N. Unfortunately, it is possible that only a small fraction of possible x's share a common factor with N.)

Miller-Rabin Primality Test

Some modifications are needed to make the previous algorithm work.

The Miller-Rabin primality test is a probabilistic test but one that works (except with negligible probability) for all N, including pseudoprimes.

It takes advantage of the fact that if N is composite, then exists some a such that $a \neq \pm 1 \bmod N$ but $a^2 = 1 \bmod N$.

E.g., for N = 561, $188^2 = 1 \bmod 561$.

This follows from the Chinese remainder theorem:

If $N = uv$, there is a solution to

$$a = -1 \bmod u$$
$$a = 1 \bmod v$$

which must satisfy the desired two conditions.