

Lecture 11: Prime Numbers

Lecture Notes on “Computer and Network Security”

by Avi Kak (kak@purdue.edu)

March 3, 2009

©2009 Avinash Kak, Purdue University

Goals:

- Fermat’s Little Theorem
- The Totient of a Number
- The Miller-Rabin Probabilistic Algorithm for Testing for Primality
- The Agrawal-Kayal-Saxena Deterministic Algorithm for Testing for Primality
- Chinese Remainder Theorem for Representing Large Numbers in Modular Arithmetic
- Discrete Logarithms

11.1: Prime Numbers

- Many public-key cryptographic algorithms require selecting **one or more large prime integers**.
- So an important concern in public-key cryptography is to test a randomly selected integer for its **primality**.
- An integer is prime if it has exactly two **distinct** divisors, the integer 1 and itself. That makes the integer 2 the **first prime**.
- We will also be very interested in two integers being **relatively prime** to each other. Such integers are also called **coprimes**. Two integers m and n are coprimes if and only if $\gcd(m, n) = 1$. Therefore, whereas 4 and 9 are coprimes, 9 and 9 are not.
- Much of the discussion in this lecture uses the notion of **coprimes**, as defined above. The same concept used in earlier lectures was referred to as **relatively prime**. **But as mentioned above, the two mean the same thing.**
- Obviously, the number 1 is **coprime** to every integer.

11.2: Fermat's Little Theorem

- Fermat's Little Theorem plays an important role in public-key cryptography.
- This theorem states that if p is a **prime number**, then for **every integer** a the following must be true

$$a^p \equiv a \pmod{p} \tag{1}$$

Another way of saying the same thing is that for any prime p and any integer a , $a^p - a$ will always be divisible by p .

- A simplified form of Fermat's Little Theorem states that when p is a **prime**, then for **any** integer a that is **coprime** to p , the following relationship must hold:

$$a^{p-1} \equiv 1 \pmod{p} \tag{2}$$

That obviously does NOT include a 's such that $a \equiv p \pmod{p}$. **That is, $a = 0$ and a 's that are multiples of p are**

excluded specifically. Another way of stating the theorem in Equation (2) is that for every prime p and every a that is coprime to p , $a^{p-1} - 1$ will always be divisible by p .

- To prove the theorem as stated in Equation (2), let's write down the following sequence assuming that p is prime and a is a non-zero integer that is coprime to p :

$$a, 2a, 3a, 4a, \dots, (p-1)a \tag{3}$$

It turns out that if we reduce these numbers modulo p , we will simply obtain a **rearrangement** of the sequence

$$1, 2, 3, 4, \dots, (p-1)$$

In what follows, we will first show two examples of this and then present a simple proof.

- For example, consider $p = 7$ and $a = 3$. Now the sequence shown in the expression labeled (3) above will be 3, 6, 9, 12, 15, 18 that when expressed modulo 7 becomes 3, 6, 2, 5, 1, 4.

- For another example, consider $p = 7$ and $a = 8$. Now the sequence shown in the expression labeled (3) above will be 8, 16, 24, 32, 40, 48 that when expressed modulo 7 becomes 1, 2, 3, 4, 5, 6.

- Therefore, we can say

$$\{a, 2a, 3a, \dots, (p-1)a\} =$$

$$\text{some permutation of } \{1, 2, 3, \dots, (p-1)\} \pmod{p} \quad (4)$$

for every prime p and every a that is coprime to p .

- The above conclusion can be established more formally by noting first that, since a cannot be a multiple of p , it is impossible for $k \cdot a \equiv 0 \pmod{p}$ for $k, 1 \leq k \leq p-1$. The product $k \cdot a$ cannot be a multiple of p because of the constraints we have placed on the values of k and a . Additionally note that $k \cdot a$ is also not allowed to become zero because a must be a non-zero integer and because the smallest value for k is 1. Next we can show that for any j and k with $1 \leq j, k \leq (p-1), j \neq k$, it is impossible that $j \cdot a \equiv k \cdot a \pmod{p}$ since otherwise we would have $(j-k) \cdot a \equiv 0 \pmod{p}$, which would require that either $a \equiv 0 \pmod{p}$ or that $j \equiv k \pmod{p}$.

- Hence, the product $k \cdot a \pmod{p}$ as k ranges from 1 through $p - 1$, both ends inclusive, must yield some permutation of the integer sequence $\{1, 2, 3, \dots, p - 1\}$.
- Therefore, multiplying all of the terms on the left hand side of Equation (4) would yield

$$a^{p-1} \cdot 1 \cdot 2 \cdots p - 1 \equiv 1 \cdot 2 \cdot 3 \cdots p - 1 \pmod{p}$$

Canceling out the common factors on both sides then gives the Fermat's Little Theorem as in Equation (2). (The common factors can be canceled out because they are all coprimes to p .)

- We therefore have a formal proof for Fermat's Little Theorem as stated in Equation (2). But what about the theorem as stated in Equation (1)? Note that Equation (1) places no constraints on a . That is, Equation (1) does **not** require a to be a coprime to p .
- Proof of the theorem in the form of Equation (1) follows directly from the theorem as stated in Equation (2) by multiplying both sides of the latter by a . Since p is prime, when a is not a coprime to p , a must either be 0 or a multiple of p . When a is 0, Equation (1) is true trivially. When a is, say, $n \cdot p$, Equation (1) reduces

trivially to Equation (2) because the *mod p* operation cancels out the *p* factors on both sides of Equation (1).

- If you are tempted to use Fermat's Little Theorem directly for primality testing, note the following problem you would run into: Let's say you have a number n you want to test for primality. So you come up with a small integer a that you are sure is coprime to n . (You could, for example, use a small prime number for a that is guaranteed to be coprime to all numbers including your n .) Now let's say you have a magical procedure that can efficiently compute $a^n - 1 \pmod{n}$. If the answer returned by this procedure is NOT 1, you can be sure that n is NOT a prime. But, should the answer equal 1, then you *could* be — as they say — up the creek without a paddle. You see, if the answer is 1, then n may either be a composite or a prime. [A non-prime number is also referred to as a composite number.] That is because the relationship of Fermat's Little Theorem is also satisfied by numbers that are composite. For example, consider the case $n = 25$ and $a = 7$:

$$7^{25-1} \equiv 1 \pmod{25}$$

So what is one to do if Fermat's Little Theorem is satisfied for a given number n for a given choice for a ? One could try another choice for a . [Remember, Fermat's Little Theorem must be satisfied by **every** a that is coprime to n . It is trivially satisfied by $a = 1$. As we will see later, it is also trivially satisfied by $a = n - 1$ for prime n .] For the case of $n = 25$, we

could next try $a = 11$. If we do so, we get

$$11^{25-1} \equiv 16 \pmod{25}$$

which tells us with certainty that 25 is not a prime.

- Although Fermat's Little Theorem is not used directly for primality testing, the same sort of primality testing logic as described above is used in the Miller-Rabin primality testing algorithm that we will take up in Section 11.5.

11.3: Euler's Totient Function

- An important quantity related to positive integers is the Euler's Totient Function, denoted $\phi(n)$.
- As you will see in Lecture 12, the notion of a totient plays a critical role in the famous RSA algorithm for public key cryptography.
- For a given positive integer n , $\phi(n)$ is the number of positive integers less than or equal to n that are coprimes to n . (Integers a and b are coprimes to each other if $\gcd(a, b) = 1$; that is, if their greatest common divisor is 1.)
- $\phi(n)$ is known as the **totient** of n .
- It follows from the definition that $\phi(1) = 1$. Here are some positive integers and their totients:

ints:	1	2	3	4	5	6	7	8	9	10	11	12
totients:	1	1	2	2	4	2	6	4	6	4	10	4

To see why $\phi(3) = 2$: Obviously 1 is coprime to 3. The number 2 is also coprime to 3 since their gcd is 1. However, 3 is **not** coprime to 3 because $\gcd(3, 3) = 3$.

- Obviously, if p is prime, its totient is given by $\phi(p) = p - 1$.
- Suppose a number n is a product of two primes p and q , that is $n = p \times q$, then

$$\phi(n) = \phi(p) \cdot \phi(q) = (p - 1)(q - 1)$$

This follows from the observation that in the set of numbers $\{1, 2, 3, \dots, p, p + 1, \dots, pq - 1\}$, the number p is obviously **not** a coprime to n since $\gcd(p, n) = p$. By the same token $2p, 3p, \dots, (q - 1)p$ are **not** coprimes to n . By similar reasoning, $q, 2q, \dots, (p - 1)q$ are **not** coprimes to n . That then leaves the following as the number of coprimes to n :

$$\begin{aligned} \phi(n) &= (pq - 1) - [(q - 1) + (p - 1)] \\ &= pq - (p + q) + 1 \\ &= (p - 1) \times (q - 1) \\ &= \phi(p) \times \phi(q) \end{aligned}$$

[**An aside:** Euler's Totient Function and the Euler's Theorem to be presented next are named after Leonhard Euler who lived from 1707 to 1783. He was the first to use the word "function" and gave us the notation $f(x)$ to describe a function that takes an argument. He was an extremely high-energy and rambunctious sort of a guy who was born and raised in Switzerland and who at the age of 22 was invited by Catherine the Great to a professorship in St. Petersburg. He is considered to be one of the greatest mathematicians and probably the most prolific. His work fills 70 volumes, half of which were written with the help of assistants during the last 17 years of his life when he was completely blind.]

11.4: Euler's Theorem

- This theorem states that for **every** positive integer n and **every** a that is **coprime** to n , the following must be true

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

- Note that **when n is a prime**, $\phi(n) = n - 1$. In this case, Euler's Theorem reduces the Fermat's Little Theorem.
- **However**, Euler's Theorem holds for **all** positive integers n . To demonstrate this, let's say that

$$R = \{x_1, x_2, \dots, x_{\phi(n)}\}$$

is the set of all integer less than n that are relatively prime (the same thing as co-prime) to n .

- Now let S be the set obtained when we multiply modulo n each element of R by some integer a co-prime to n . That is

$$S = \{a \times x_1 \bmod n, a \times x_2 \bmod n, \dots, a \times x_{\phi(n)} \bmod n\}$$

- We now claim that S is simply a permutation of R . To prove this, we first note that $(a \times x_i \bmod n)$ cannot be zero because, as a and x_i are coprimes to n , the product $a \times x_i$ cannot contain n as a factor. Next we can show that for $1 \leq i, j \leq \phi(n), i \neq j$, it is not possible for $(a \times x_i \bmod n)$ to be equal to $(a \times x_j \bmod n)$. If it were possible for $(a \times x_i \bmod n)$ to be equal to $(a \times x_j \bmod n)$, then $(a \times x_i - a \times x_j \equiv 0 \pmod{n})$ since both $a \times x_i$ and $a \times x_j$ are coprimes to n . That would imply that either a is $0 \bmod n$, or that $x_i \equiv x_j \pmod{n}$, both clearly violating the assumptions.

- Therefore, we can say that

$$S = \text{merely a permutation of } R$$

implying that multiplying **all** of the elements of S should equal the product of **all** of the elements of R . That is

$$\prod_i s_i \in S \bmod n = \prod_i r_i \in R \bmod n$$

- Looking at the individual elements of S , multiplying all of the elements of S will give us a result that is $a^{\phi(n)}$ times the product of all of the elements of R . So the above equation can be expressed as

$$a^{\phi(n)} \times \prod_i r_i \in R \equiv \prod_i r_i \in R \pmod{n}$$

which then directly leads to the statement of the theorem.

11.5: Miller-Rabin Algorithm for Testing for Primality

- One of the most commonly used algorithms for testing a randomly selected number for primality is the Miller-Rabin algorithm.
- But note that this algorithm makes only a **probabilistic assessment of primality**: If the algorithm says that the number is **composite** (the same thing as **not a prime**), then the number is definitely not a prime. On the other hand, if the algorithm says that the number **is** a prime, then with a very small probability the number may **not** actually be a prime. (With proper algorithmic design, this probability can be made so small that, as someone has said, there would be a greater probability that, as you are sitting at a workstation, you'd win a lottery and get hit by a bolt of lightening at the same time.)

11.6: Miller-Rabin Algorithm is Based on the Following Decomposition of Odd Numbers

- Given any odd positive integer n , we can express $n - 1$ as a product of a power of 2 and a smaller odd number:

$$n - 1 = 2^k \cdot q \quad \text{for some } k > 0, \text{ and } \mathbf{odd} \ q$$

This follows from the fact that if n is odd, then $n - 1$ is even. It follows that after we have factored out the largest power of 2 from $n - 1$, what remains, meaning q , must be odd.

11.7: Miller-Rabin Algorithm Uses the Fact that $x^2 = 1$ Has No Non-Trivial Roots in Z_p

- When we say that $x^2 = 1$ has only trivial roots in Z_p for any prime p , we mean that only $x = 1$ and $x = -1$ can satisfy the equation $x^2 = 1$.
- Let's first try to see what -1 stands for in the finite field Z_p for any prime p .
- Let's consider the finite field Z_7 for a moment:

Natural

nums: ... -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 ...

Z_7 : ... 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 ...

We notice that -1 is congruent to 6 modulo 7. In general, we can say that for any prime p , we have in the finite field Z_p :

$$-1 \equiv (p - 1) \pmod{p}$$

- What is interesting is that there exist only two numbers -1 and 1 that when squared in Z_p give us 1. That is

$$\begin{aligned} 1 \cdot 1 &= 1 \\ -1 \cdot -1 &= 1 \end{aligned}$$

- Actually, there is more to the above two equations than meets the eye. A more meaningful way to express these two thoughts would be that for any number $a \equiv 1 \pmod{p}$, we have

$$(a \bmod p) \cdot (a \bmod p) = 1 \bmod p$$

and for any number $b \equiv -1 \pmod{p}$, we have

$$(b \bmod p) \cdot (b \bmod p) = 1 \bmod p$$

- There does **not** exist any other number $x \in \mathbb{Z}_p$ that when squared will return $1 \bmod p$.
- We will prove the above assertion **by contradiction**. So let's assume that there does exist an $x \in \mathbb{Z}_p$, $x \neq 1$ and $x \neq -1$, such that

$$x \cdot x = 1 \bmod p$$

which is the same thing as saying that

$$x^2 \equiv 1 \pmod{p}$$

And that is equivalent to the following

$$\begin{aligned}x^2 - 1 &\equiv 0 \pmod{p} \\x^2 - x + x - 1 &\equiv 0 \pmod{p} \\(x - 1) \cdot (x + 1) &\equiv 0 \pmod{p}\end{aligned}$$

- Now remember that in our **proof by contradiction** we are not allowing x to be either -1 or 1 . Therefore, for the last of the above equivalences to hold true, it must be the case that either $x - 1$ or $x + 1$ is congruent to 0 modulo the prime p . But we know already that, **when p is prime, no number in Z_p can satisfy this condition if x is not allowed to be either 1 or -1** . Therefore, the above equivalences must be false unless x is either -1 or 1 . (As mentioned earlier, -1 is a standin for $p - 1$ in the finite field Z_p .)
- We summarize the above proof by saying that in Z_p the equation $x^2 = 1$ has only two **trivial** roots -1 and 1 . There do **not** exist any **non-trivial** roots for $x^2 = 1$ in Z_p for any prime p .

11.8: Miller-Rabin Algorithm: The Special Conditions That Must Be Satisfied by a Prime

- First note that for any prime p , it being an odd number, the following relationship must hold (as stated previously)

$$p - 1 = 2^k \cdot q \quad \text{for some } k > 0, \text{ and odd } q$$

- Now for any integer a in the range $1 < a < p - 1$ (note that a is **not** allowed to take on **either the first two or the last value of the range** of the integers in Z_p , all of the allowed values for a being coprime to p if p is truly a prime), **one of the following** conditions is true:

CONDITION 1: Either it must be the case that

$$a^q \equiv 1 \pmod{p}$$

CONDITION 2: Or, it must be the case that one of the numbers $a^q, a^{2q}, a^{4q}, \dots, a^{2^{k-1}q}$ is congruent to -1 modulo p . That

is, there exists some number j in the range $1 \leq j \leq k$, such that

$$a^{2^{j-1}q} \equiv -1 \pmod{p}$$

11.9: Proof for the Conditions 1 and 2 That Must be Satisfied by a Prime

- We will now present a proof for the Conditions 1 and 2 stated at the end of the previous section. As mentioned there, if p is a prime, then either **Condition 1** or **Condition 2** must be satisfied.
- Since $p - 1 = 2^k \cdot q$ for some k and for some odd integer q , the following statement of Fermat's Little Theorem

$$a^{p-1} \equiv 1 \pmod{p}$$

can be reexpressed as

$$a^{2^k \cdot q} \equiv 1 \pmod{p}$$

for any positive integer a that is coprime to p . For prime p , that obviously includes all values of a such that $1 < a < (p - 1)$.

- Now let's examine the following sequence of numbers

$$a^q \bmod p, \quad a^{2q} \bmod p, \quad a^{2^2q} \bmod p, \quad a^{2^3q} \bmod p, \quad \dots, \quad a^{2^kq} \bmod p$$

Note that every number in this sequence is a square of the previous number. Therefore, either it **must** be the case that the first number satisfies $a^q \bmod p = 1$, in which case every number in the sequence is 1; **or** it **must** be the case that one of the numbers in the sequence is -1 (**the other** square-root of 1), which would then make all the subsequent numbers equal to 1. This is the proof for **Condition 1** and **Condition 2** of the previous section.

- In the logic stated above, note the role played by the fact that when $x^2 = 1$ in Z_p , then it must be the case that either $x = 1$ or $x = -1$. (This fact was established in Section 11.7.) Also recall that in the finite field Z_p , the number -1 is the same thing as $p - 1$.

11.10: Miller-Rabin Algorithm is Based on Conditions 1 and 2

- The upshot of the points made so far is that if for a given number n there exists a number a that is greater than 1 and less than $n - 1$ and for which neither of the **Conditions 1 and 2** is satisfied, then the number n is definitely **not** a prime.
- Since we have **not** established a “if and only if” sort of a connection between the primality of a number and the two **Conditions**, it is certainly possible that a composite number may also satisfy the two **Conditions**.
- Therefore, we conclude that if neither **Condition** is true for a randomly selected $a < n$, then n is definitely **not** a prime. However, if the **Conditions** are true for a given $a < n$, then n may be either a composite or a prime.
- From experiments it is known that if the **Conditions** are true for a randomly selected $a < n$, then n is likely to be prime with a very high probability. To increase the probability of n being a prime, one can repeat testing for the two **Conditions** with

different randomly selected choices for a .

11.11: Miller-Rabin Algorithm: The Computational Steps

- Miller-Rabin is a probabilistic algorithm for testing for primality.
- To test a number n for primality:

choose a parameter t that determines the accuracy of the test

write $n-1$ as $2^k \cdot q$

repeat t times:

 choose a number 'a' randomly such that $1 < a < n-1$

 if $a^q \% n == 1$:

 return "prime with high probability"

 repeat for index i , $0 < i < k-1$:

 if $(a^{2^i \cdot q} \% n == -1)$:

 return "prime with high probability"

 return "definitely a composite"

The operator $\%$ used above carries out modulo n division.

11.12: Miller-Rabin Algorithm: Liars and Witnesses

- When n is **known** to be composite, then the dual test

$$a^q \not\equiv 1$$

and

$$a^{2^i \cdot q} \not\equiv -1 \pmod{n} \quad \text{for all } 0 < i < k - 1$$

will be satisfied by only a certain number of a 's, $a < n$. All such a 's are called **witnesses for the compositeness** of n .

- When a randomly chosen a for a known composite n does not satisfy the dual test above, it is called a **liar** for the compositeness of n .
- It has been shown theoretically that, in general, for a **composite** n , at least 3/4th of the numbers $a < n$ will be witnesses for its compositeness.
- It has also been established theoretically that if n is indeed composite, then the Miller-Rabin algorithm will declare it to be a prime with a probability of 4^{-t} . (See previous slide for what's t .)

- In reality, the probability of a composite number being declared prime by the Miller-Rabin algorithm is significantly less than 4^{-t} .
- If you are careful in how you choose a candidate for a prime number, you can safely depend on the Miller-Rabin algorithm to verify its primality.
- Because of the probabilistic nature of primality verification, you'd never want to accept a number that is claimed to be a prime number from a questionable source.

11.13: Computational Complexity of the Miller-Rabin Algorithm

- The running time of this algorithm is $O(t \times \log^3 n)$ where t is the different values of a for which we execute the main loop on the previous slide.
- A more efficient FFT based implementation can reduce the time complexity measure to $O(t \times \log^2 n)$.
- In the theory of algorithms, the Miller-Rabin algorithm would be called a **randomized algorithm**.
- A **randomized algorithm** is an algorithm that can make random choices during its execution.
- As a randomized algorithm, the Miller-Rabin algorithm belongs to the class **co-RP**.

- The class **RP** stands for **randomized polynomial time**. This is the class of problems that can be solved in polynomial time with randomized algorithms provided errors are made on only the “yes” inputs. What that means is that when the answer is known to be “yes”, the algorithm occasionally says “no”.
- The class **co-RP** is similar to the class **RP** except that the algorithm occasionally makes errors on only the “no” inputs. What that means is that when the answer is known to be “no”, the algorithm occasionally says “yes”.
- The Miller-Rabin algorithm belongs to **co-RP** because occasionally when an input number is known to **not** be a prime, the algorithm declares it to be prime.
- The class **co-RP** is a subset of the class BPP. BPP stands for **bounded probabilistic polynomial-time**. These are randomized polynomial-time algorithms that yield the correct answer with an exponentially small probability of error.
- The fastest algorithms that behave deterministically belong to the class **P** in the theory of computational complexity. **P** stands for **polynomial-time**. All problems that can be solved in ex-

ponential time in a deterministic machine belong to the class **NP** in the theory of computational complexity.

- The class **P** is a subset of class **BPP** and there is no known direct relationship between the classes **BPP** and **NP**. In general we have

$$P \subset RP \subset NP$$
$$P \subset co-RP \subset BPP$$

11.14: The Agrawal-Kayal-Saxena (AKS) Algorithm for Primality Testing

- Despite the **millennia old obsession** with prime numbers, until 2002 there did not exist a computationally efficient test with an unconditional guarantee of primality.
 - A deterministic test of primality (as opposed to a randomized test) is considered to be **computationally efficient** if it belongs to class **P**. That is, the running time of the algorithm must be a polynomial function of the size of a number whose primality is being tested. (**The size of n is proportional to $\log n$** . Think of the binary representation of n .)
 - Obviously, if you are not concerned about computational efficiency, you can always test for primality by dividing n by all integers up to \sqrt{n} . The running time of this algorithm is directly proportional to n , which is **exponential** in the size of n .
 - Only very small integers can be tested for primality by such a brute-force approach even though it is unconditionally guar-

anted to yield the correct answer.

- Hence the great interest by **all** (the governments, the scientists, the commercial enterprise, etc.) in discovering a computationally efficient algorithm for testing for primality that guarantees its result unconditionally.
- So when on **August 8, 2002** The New York Times broke the story that the trio of Agrawal, Kayal, and Saxena (all from the Indian Institute of Technology at Kanpur) had found a computationally efficient algorithm that returned an unconditionally guaranteed answer to the primality test, it caused a big sensation.

11.15: Generalization of Fermat's Little Theorem to Polynomial Rings Over Finite Fields

- The Agrawal-Kayal-Saxena (AKS) algorithm is based on the following generalization of Fermat's Little Theorem to polynomial rings over finite fields. This generalization states that if a number a is coprime to another number p , $p > 1$, then p is prime **if and only if** the **polynomial** $(x + a)^p$ defined over the finite field Z_p obeys the following equality:

$$(x + a)^p \equiv x^p + a \pmod{p} \quad (4)$$

Pay particular attention to the '**if and only if**' clause in the statement above the equation. That implies that the equality in Equation 4 is both a **necessary** and a **sufficient** condition for p to be a prime. It is this fact that allows the AKS test for primality to be deterministic. By contrast, Fermat's Little Theorem is only a necessary condition for the p to be prime. Therefore, a test based directly on Fermat's Little Theorem can only be probabilistic in the sense explained earlier.

- To establish Equation (4), we can expand the binomial $(x + a)^p$ as follows:

$$(x + a)^p = \binom{p}{0}x^p + \binom{p}{1}x^{p-1} \cdot a + \binom{p}{2}x^{p-2} \cdot a^2 + \cdots + \binom{p}{p}a^p \quad (5)$$

where the binomial coefficients are given by

$$\binom{p}{i} = \frac{p!}{i!(p-i)!}$$

- To prove Equation (4) in the forward direction, suppose p is prime. Then all of the binomial coefficients, since they contain p as a factor, will obey

$$\binom{p}{i} \equiv 0 \pmod{p}$$

Also, in this case, by Fermat's Little Theorem, we have $a^{p-1} = 1$. As a result, the expansion in Equation (5) reduces to the form shown in Equation (4).

- To prove Equation (4) in the opposite direction, suppose p is composite. It then has a prime factor $q > 1$. Let q^k be the greatest power of q that divides p . Then q^k does NOT divide the binomial coefficient $\binom{p}{q}$. That is because this binomial coefficient has factored out of it some power of q and therefore the binomial coefficient cannot have q^k as one of its factors. [To make the same

assertion contrapositively, let's assume for a moment that q^k is a factor of $\binom{p}{q}$. Then it must be the case that a larger power of q can divide p which is false by the assumption about k .] We also note that q^k must be coprime to a^{p-q} since we started out with the assumption that a and p were coprimes, implying that a and p cannot share any factors (except for the number 1). Now the coefficient of the term x^q in the binomial expansion is

$$\binom{p}{q} \cdot a^{p-q}$$

We have identified a factor of p , the factor being q^k , that does **not** divide $\binom{p}{q}$ and that is coprime to a^{p-q} . For the coefficient of x^q to be $0 \pmod{p}$, it must be divisible by p . But for that to be the case, the coefficient must be divisible by all factors of p . But we have just identified a factor, q^k , that divides neither $\binom{p}{q}$ nor a^{p-q} . Therefore, the coefficient of x^q **cannot** be $0 \pmod{p}$. This establishes the proof of Equation (4) in the opposite direction, since we have shown that when p is **not** a prime, the equality in Equation (4) does not hold.

- The generalization of Fermat's Little Theorem can be used directly for primality testing, but it would **not** be computationally efficient since it would require we check each of the p coefficients in the expansion of $(x + a)^p$ for some a that is coprime to p .

- There is a way to make this sort of primality testing more efficient by making use of the fact that if

$$f(x) \bmod p = g(x) \bmod p \quad (6)$$

then

$$f(x) \bmod h(x) = g(x) \bmod h(x) \quad (7)$$

where $f(x)$, $g(x)$, and $h(x)$ are polynomials whose coefficients are in the finite field Z_p . (But bear in mind the fact that whereas Eq. (6) implies Eq. (7), the reverse is **not** true.)

- As a result, the primality test of Equation (4) can be expressed in the following form for some value of the integer r :

$$(x + a)^p \bmod (x^r - 1) = (x^p + a) \bmod (x^r - 1) \quad (8)$$

with the caveat that there will exist some **composite** p for which this equality will also hold true. So, when p is known to be a prime, the above equation will be satisfied by all a coprime to p and by all r . However, when p is a composite, this equation will be satisfied by some values for a and r .

- The main AKS contribution lies in showing that, when r is chosen appropriately, if Equation (8) is satisfied for appropriately chosen values for a , then p is guaranteed to be a prime. **The amount of work required to find the value to use for r and the number of values of a for which the equality in Equation (8) must be tested is bounded by a polynomial in $\log p$.**

11.16: The Agrawal-Kayal-Saxena Primality Test: The Computational Steps

```
p = integer to be tested for primality

if ( p == a^b for some integer a and for some integer b > 1 ) :
    then return 'p is COMPOSITE'

r = 2

### This loop is to find the appropriate value for the number r:
while r < p:
    if ( gcd(p,r) is not 1 ) :
        return return 'p is COMPOSITE'

    if ( r is a prime greater than 2 ):
        let q be the largest factor of r-1
        if ( q > (4 . sqrt(r) . log p) ) and
            ( p^{(r-1)/q} is not 1 mod r ) :
            break
    r = r+1

### Now that r is known, apply the following test:
for a = 1 to (2 . sqrt(r) . log p) :
    if ( (x-a)^p is not (x^p - a) (mod x^r - 1), p :
        return 'p is COMPOSITE'

return 'p is PRIME'
```

11.17: Computational Complexity of the Agrawal-Kayal-Saxena Primality Testing Algorithm

- The computational complexity of the AKS algorithm is

$$O((\log p)^{12} \cdot f(\log \log p))$$

where p is the integer whose primality is being tested and f is a polynomial. So the running time of the algorithm is **proportional** to the twelfth power of the number of bits required to represent the candidate integer times a polynomial function of the logarithm of the number of bits.

- There exist proposals for alternative implementations of the AKS algorithm for which the running time approaches the fourth power of the number of bits required to represent the number.

11.18: The Chinese Remainder Theorem

- Discovered by the Chinese mathematician Sun Tsu Suan-Ching around 4th century A.D. Particularly useful for modulo arithmetic operations on very large numbers.
- CRT says that, for any positive integer value for M , any integer in the set $Z_M = \{0, 1, 2, \dots, M - 1\}$ can be reconstructed from residues with respect to a set of different moduli provided the moduli are coprime to each other on a pairwise basis.
- For example, the prime factors of 10 are 2 and 5. Now let's consider an integer 9 in Z_{10} . Its residue modulo 2 is 1 and the residue modulo 5 is 4. So 9 can be represented by the tuple (1, 4).
- Let us express a decomposition of M into factors that are **pair-wise coprime** by

$$M = \prod_{i=1}^k m_i$$

Therefore, the following must be true for the factors: $\gcd(m_i, m_j) = 1$ for $1 \leq i, j \leq k$ and $i \neq j$. As an example of such a de-

composition, we can express the integer 130 as a product of 5 and 26, which results in $m_1 = 5$ and $m_2 = 26$. Another way to decompose the integer 130 would be express it as a product of 2, 5, and 13. For this decomposition, we have $m_1 = 2$, $m_2 = 5$ and $m_3 = 13$.

- CRT allows us to represent any integer A in Z_M by the k-tuple:

$$A \equiv (a_1, a_2, \dots, a_k)$$

where each $a_i \in Z_{m_i}$, its exact value being given by

$$a_i = A \bmod m_i \quad \text{for } 1 \leq i \leq k$$

Note that each a_i can be any value in the range $0 \leq a_i \leq m_i$.

- CRT makes the following two assertions about the k-tuple representations for integers:
 - The mapping between the integers $A \in Z_M$ and the k-tuples is a **bijection**, meaning that the mapping is one-to-one and onto. That is, there corresponds a **unique** k-tuple for every integer in Z_M and vice versa. (More formally, the

bijjective mapping is between Z_M and the Cartesian product $Z_{m_1} \times Z_{m_2} \times \dots \times Z_{m_k}$.)

- Arithmetic operations on the numbers in Z_M can be carried out equivalently on the k-tuples representing the numbers. When operating on the k-tuples, **the operations can be carried out independently on each of coordinates of the tuples**, as represented by

$$\begin{aligned} (A + B) \bmod M &\Leftrightarrow ((a_1 + b_1) \bmod m_1, \dots, (a_k + b_k) \bmod m_k) \\ (A - B) \bmod M &\Leftrightarrow ((a_1 - b_1) \bmod m_1, \dots, (a_k - b_k) \bmod m_k) \\ (A \times B) \bmod M &\Leftrightarrow ((a_1 \times b_1) \bmod m_1, \dots, (a_k \times b_k) \bmod m_k) \end{aligned}$$

where $A \Leftrightarrow (a_1, a_2, \dots, a_k)$ and $B \Leftrightarrow (b_1, b_2, \dots, b_k)$ are two arbitrary numbers in Z_M .

- To compute the number A for a given tuple (a_1, a_2, \dots, a_k) , we first calculate $M_i = M/m_i$ for $1 \leq i \leq k$. Since each M_i has for its factors all the other prime moduli m_j , $j \neq i$, it must be the case that

$$M_i \equiv 0 \pmod{m_j} \quad \text{for all } j \neq i$$

Let's now construct a sequence of numbers c_i , $1 \leq i \leq k$, in the following manner

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for all } 1 \leq i \leq k$$

Since M_i is coprime to m_i , there must exist a multiplicative inverse for $M_i \bmod m_i$. Now we can write the following formula for obtaining A from the tuple (a_1, a_2, \dots, a_k) :

$$A = \left(\sum_{i=1}^k a_i \times c_i \right) \bmod M$$

To see the correctness of this formula, we must show that ' $A \bmod m_i$ ' produces a_i for $1 \leq i \leq k$. This follows from the fact that $M_j \bmod m_i = 0$, $j \neq i$, implying that $c_j \bmod m_i = 0$, $j \neq i$, and the fact that $c_i \bmod m_i = 1$.

11.19: A Demonstration of the Usefulness of CRT

- CRT is extremely useful for manipulating very large integers in modulo arithmetic. We are talking about integers with over 150 decimal digits (that is, numbers potentially larger than 10^{150}).
- To illustrate the idea as to why CRT is useful for manipulating very large numbers in modulo arithmetic, let's consider an example that can be shown on a slide.
- Let's say that we want to do arithmetic on integers modulo 8633. That is, $M = 8633$. This modulus has the following decomposition into two pairwise coprimes:

$$8633 = 89 \times 97$$

So we have $m_1 = 89$ and $m_2 = 97$. The corresponding M_i integers are $M_1 = M/m_1 = 97$ and $M_2 = M/m_2 = 89$.

- By using the Extended Euclid's Algorithm (see Lecture 5), we can next figure out the multiplicative inverse for M_1 modulo m_1 and the multiplicative inverse for M_2 modulo m_2 . (These multi-

plicative inverses are guaranteed to exist since M_1 is coprime to m_1 , and M_2 is coprime to m_2 .) We have

$$\begin{aligned}M_1^{-1} \bmod m_1 &= 78 \\M_2^{-1} \bmod m_2 &= 12\end{aligned}$$

You can verify the correctness of the two multiplicative inverses by showing that $97 \times 78 \equiv 1 \pmod{89}$ and that $89 \times 12 \equiv 1 \pmod{97}$.

- Now let's say that we want to add two integers 2345 and 6789 modulo 8633.
- We first express the operand 2345 by its CRT representation, which is (31, 17) since $2345 \bmod 89 = 31$ and $2345 \bmod 97 = 17$.
- We next express the operand 6789 by its CRT representation, which is (25, 96) since $6789 \bmod 89 = 25$ and $6789 \bmod 97 = 96$.
- To add the two “large” integers, we simply add the two corre-

sponding CRT tuples modulo the respective moduli. This gives us (56, 16). For the second of these two numbers, we initially get 113, which modulo 97 is 16.

- To recover the result as a single number, we use the formula

$$a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} \quad \text{mod } M$$

which for our example becomes

$$56 \times 97 \times 78 + 16 \times 89 \times 12 \quad \text{mod } 8633$$

that returns the result 501. You can verify this result by directly computing $2345 + 6789 \text{ mod } 8633$ and getting the same answer.

- For the example we worked out above, we decomposed the modulus M into its prime factors. In general, it is sufficient to decompose M into factors that are coprimes on a pairwise basis.
- In the next lecture, we will see how CRT is used in a computationally efficient approach to modular exponentiation (a key step in public key cryptography).

11.20: Discrete Logarithms

- First let's define what is meant by a **primitive root modulo a positive number N** .
- You already know that when p is a prime, the set of remainders, Z_p , is a finite **field**.
- We can show similarly that for **any positive** integer N , the set of all integers $i < N$ that are coprime to N form a **group** with modulo N multiplication as the **group operator**.
- For example, when $N = 8$, the set of coprimes is $\{1, 3, 5, 7\}$. This set forms a group with modulo N multiplication as the group operator. What that implies immediately is that the result of multiplying modulo N any two elements of the set is contained in the set. For example, $3 \times 7 \bmod 8 = 5$. The identity element for the group operator is, of course, 1. And every element has its inverse with respect to the identity element within the set. For example, the inverse of 3 is 3 itself since $3 \times 3 \bmod 8 = 1$. (By the way, each element of $\{1, 3, 5, 7\}$ is its own inverse in the group.)

- For any positive integer N , the set of all coprimes modulo N is denoted $(\mathbb{Z}/N\mathbb{Z})^\times$ or \mathbb{Z}_N^* . Obviously, when $N = p$ is a prime, we can also just use the notation \mathbb{Z}_p .
- For some values of N , the set $(\mathbb{Z}/N\mathbb{Z})^\times$ contains an element whose various powers, when computed modulo N , are all distinct and span the entire set $(\mathbb{Z}/N\mathbb{Z})^\times$. Such an element is called the **primitive element** of the set $(\mathbb{Z}/N\mathbb{Z})^\times$ or **primitive root modulo N** .
- Consider, for example, $N = 9$. We have

$$\begin{aligned} \mathbb{Z}_9 &= \{0, 1, 2, 3, 4, 5, 6, 7, 8\} \\ (\mathbb{Z}/9\mathbb{Z})^\times &= \{1, 2, 4, 5, 7, 8\} \end{aligned}$$

Now we will show that 2 is a **primitive element** of the group $(\mathbb{Z}/9\mathbb{Z})^\times$, which is the same as **primitive root mod 9**. Consider

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &\equiv 7 \pmod{9} \\ 2^5 &\equiv 5 \pmod{9} \end{aligned}$$

$$\begin{array}{rcl}
\dots & \dots & \dots \\
2^6 & \equiv & 1 \pmod{9} \\
2^7 & \equiv & 2 \pmod{9} \\
2^8 & \equiv & 4 \pmod{9} \\
& \vdots &
\end{array}$$

- It is clear that for the group $(\mathbb{Z}/9\mathbb{Z})^\times$, as we raise the element 2 to all possible powers of the elements of \mathbb{Z}_9 , we recover all the elements of $(\mathbb{Z}/9\mathbb{Z})^\times$. That makes 2 a primitive root mod 9.

- A primitive root can serve as the base of what is known as a **discrete logarithm**. Just as we can express $x^y = z$ as $\log_x z = y$, we can express

$$x^y \equiv z \pmod{N}$$

as

$$d\log_{x,N} z = y$$

- Therefore, the table shown at the bottom of the previous page and at the top of this page for the powers of 2 can be expressed as

$$d\log_{2,9} 1 = 0$$

$$\begin{aligned}dlog_{2,9} 2 &= 1 \\dlog_{2,9} 4 &= 2 \\dlog_{2,9} 8 &= 3 \\dlog_{2,9} 7 &= 4 \\dlog_{2,9} 5 &= 5\end{aligned}$$

- It should follow from the above discussion that unique discrete logarithm mod N to some base a exists only if a is a primitive root modulo N .