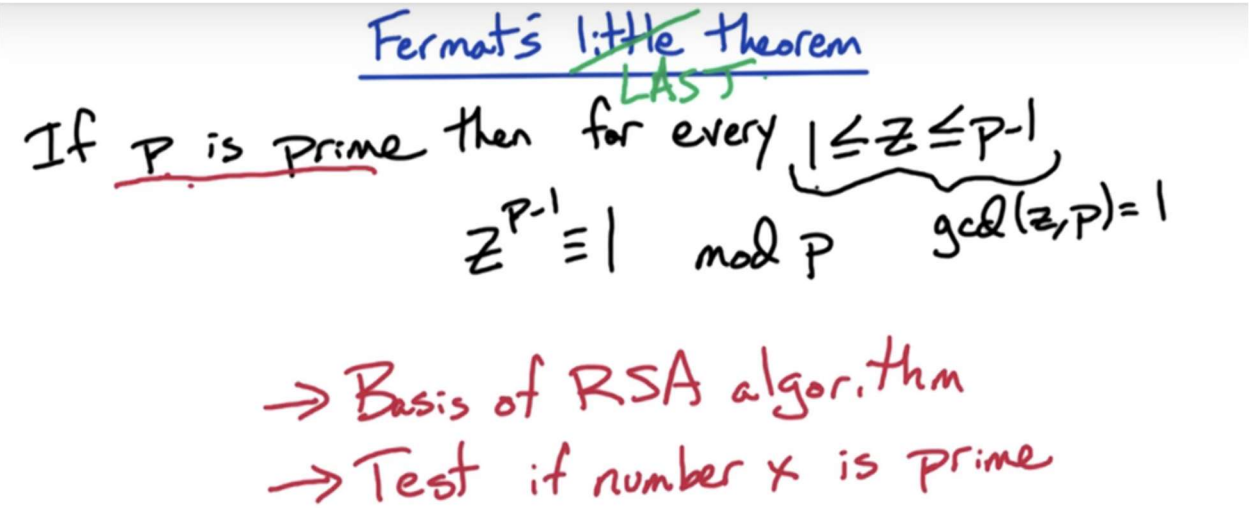
LESSON: RA2: RSA

(Slide 1) Fermat's Little Theorem

***



Before we dive into the RSA cryptosystem, we're going to dive into the mathematical concept which is the basis for the whole cryptosystem.

The key concept is Fermat's little theorem. Take a prime number p, then for every number z, which is $1 \le z \le p-1$. if we look at $z^{p-1} \bmod p$, what is that going to equal? That's going to be congruent to 1: $z^{p-1} \bmod p \equiv 1 \bmod p$. So for any z, which is relatively prime to p, this statement here – "which is $1 \le z \le p-1$", can be replaced by "for any z, which is relatively prime to p" -- So, $\gcd(z,p) = 1$. For any such z, if p is prime, $z^{p-1} \equiv 1 \bmod p$.

This easy-to-state fact is going to be the basis of the RSA cryptosystem.

It's also going to be the basis of a primality testing algorithm. So, if we're given a number x, and we want to test whether x is prime or not, we're going to devise an algorithm based on Fermat's little theorem.

Now, we're going to look at the proof of Fermat's little theorem. It's not too hard; it's not too easy; it's a reasonable proof; but, it's a very beautiful proof, very elegant. And there are several important ideas in the proof. For example, how does it come up that p is prime? How does that come into the proof?

Just a little bit of foreshadowing: if p is prime, what does that tell you about z? Well, these z's are all relatively prime to p. So, what does that imply? -- What that implies is that these z's all have an inverse mod p. That's the key thing about a prime number p and that's going to come

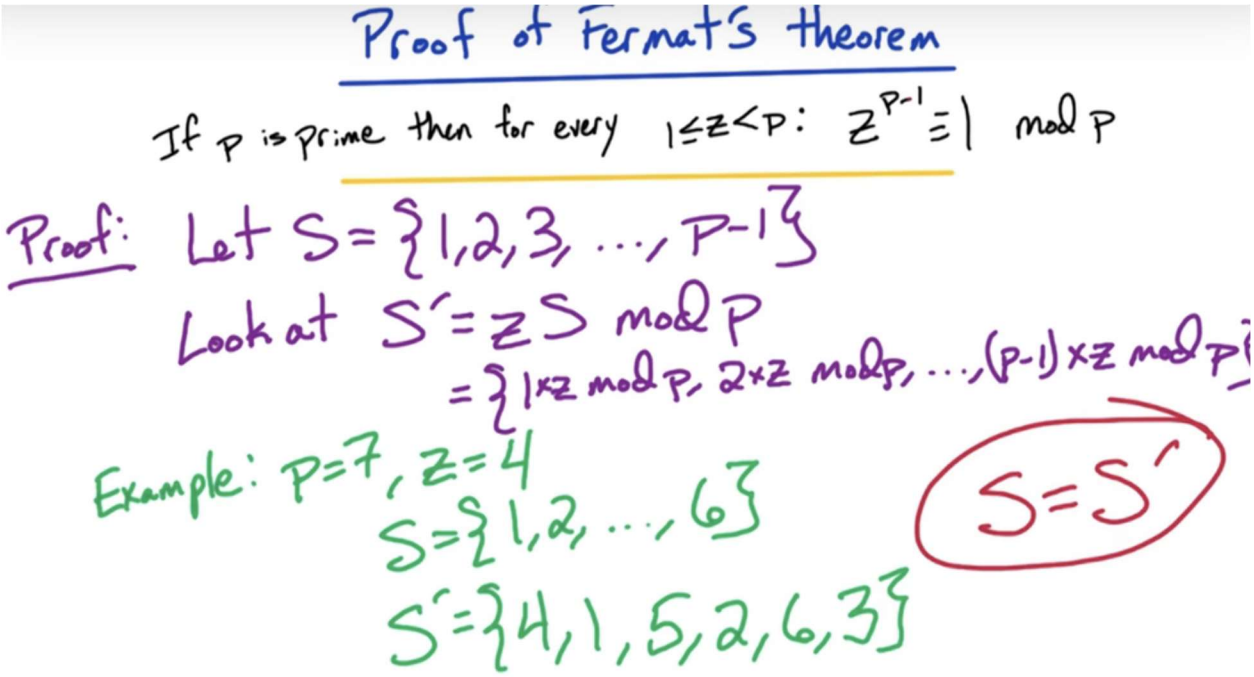
out in the proof.

One last historical note. This is Fermat's little theorem, it's not Fermat's last theorem. Fermat's last theorem is the one which he claimed to have a proof and he couldn't fit it into the margin and which was proofed roughly 400 years later by Andrew Wiles, after 10 years of hard work.

So, this Fermat's little theorem is quite easy to prove and we're going to prove it now. We'll go through the proof of Fermat's last theorem in the homework. Don't worry, we'll give you a few hints.

***

(Slide 2) Fermat's Thm.: Proof



So, here's Fermat's little theorem again. For prime number p, for every z between one and p-1, if we look at z^{p-1}, z^{p-1} mod p = 1.

So, well, let's look at the proof of this theorem. Let me look at the set of possible z's. What are these? These are all the numbers between one and p-1: 1, 2, 3,…,p-1. Let's define the set S as the possible z's. So, these are the numbers S = {1, 2, 3,…,p-1}.

Now let's look at another set S'. S' is going to be defined by taking the elements of S, each of these p-1 elements, multiplying each one by z and then for each of those elements we take it mod p: S' = {z * 1 mod p, 2 * z mod p, 3 * z mod p,…,(p-1) * z mod p}.

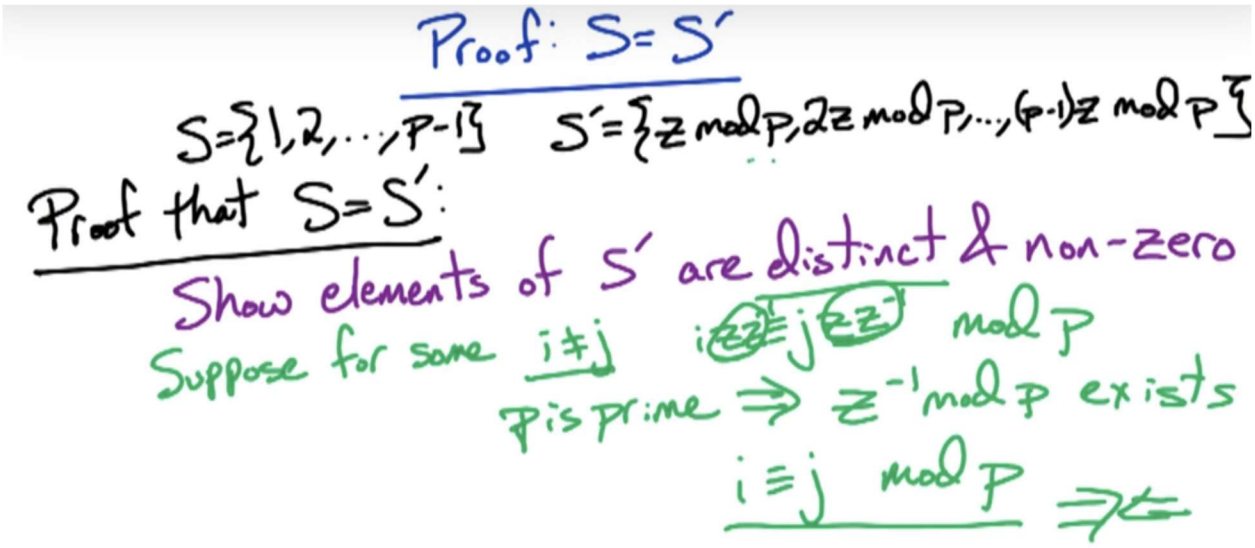
> Let's look at a simple example. Let's take p=7 and let's take z=4. First off, what's the set S? S there are the numbers {1,2,3,4,5,6}. What's S'?
>
> - Well, the first element is 1 x 4 = 4 mod 7, which is going to be 4.
> - The second element is going to be 2 x 4 = 8 mod 7 = 1.
> - The third element is going to be 3 x 4 = 12 mod 7 = 5
> - and 4 x 4 = 16 mod 7 = 2.
> - 5 x 4 = 20 mod 7 = 6
> - Finally, 6 x 4 = 24 mod 7 = 3.

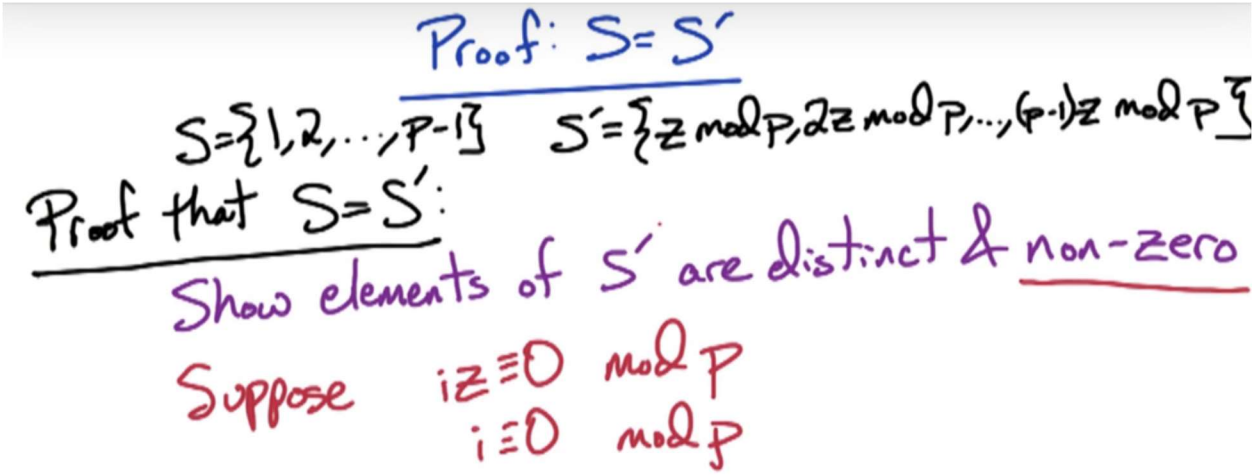
Look at this set S′ = {4,1,5,2,6,3}. It's the same as S except in different order. So the set of elements are the same – just that it's a permutation of S. So the key thing is that S and S′ are the same sets, just in different order.

Let me prove to you this fact that, in general, S and S′ are the same sets. And, then, we're going to use this fact that S and S′ are the same sets to prove the Fermat's little theorem.

***

(Slide 3) Proof: Key Lemma





Let me prove to you now that this fact that S and S′ are the same sets, just in different order.

Recall S = {1, 2, 3, …, p-1}, where p is a prime number from the hypothesis of the theorem statement. And the set S′ = {z mod p, 2z mod p, …, (p-1) * z mod p}. Now, let's go ahead and prove that S and S′ are the same.

We're going to show that the elements of S′ are distinct and non-zero.

- What are the possible values for the elements of S′? We take them all mod p, right? So they're all between 0 and p-1 -- okay? So, 0,1,2,3,…,p-1 are possible values -- zero up to p-1. Now if we show that they're all non-zero, then the possible values are 1, 2, …, p-1. So, this is p-1 possible values.

- How many elements are there in S'? Well there's p-1. So, since |S| = p-1, and the possible values are p-1 possible values, then if they're all distinct, they must contain exactly each of these elements one time and, therefore, they're the same -- S' is the same as S, just in possibly different order.

  So, once we show that the elements of S' are distinct and non-zero, we're done.

- Let's start off by showing that they're distinct. We'll do it by contradiction.

  (proof by contradiction) So let's suppose they're not distinct. So let's suppose for some $i \neq j$ that the ith element and the jth element of S' are the same. The ith element of S' is iz mod p and the jth element of S' is jz mod p. So, we're supposing that the ith element and the jth element are the same, so $iz \equiv jz \bmod p$.

  Now let's use the fact that p is a prime. If p is a prime, what do we know? We know then that every element in S has an inverse mod p. So, z is an element in S, right? z is a number between one and p-1.

  So, to recap, p is a prime so that implies that z has an inverse mod P. So, let's use that inverse.

  What can we do? Look at this equation:

  $$iz \equiv jz \bmod p.$$

  Let's multiply both sides by z^{-1} inverse -- it exists -- I don't know what it is, but no matter what it is multiply both sides by z^{-1}, okay? So what do we get? So, I multiplied the left hand side by z^{-1} and the right hand side by z^{-1}:

  $$i\, z * z^{-1} \equiv j\, z * z^{-1} \bmod p$$

  What do I get? Now, I have z times the inverse. What is that? That's 1. z times the inverse, that's 1. What am I left with?

  $$i \equiv j \bmod p.$$

  What does that mean? That means they're the same when I look between one and p-1.

So, therefore, they're the same index. We supposed that they were different indexes which gave me the same value; but, in order to come up with the same value, they have to be the same index. So I get my contradiction. Therefore, there are no repeat elements in S′ and they are all distinct.

- Now, we've shown that the elements of S′ are distinct. Now let's show that the elements of S′ are non-zero. Again, let's do a proof by contradiction.

  (proof by contradiction) Suppose one of the elements is 0. So, take the ith element of S′ and suppose it's zero. So that means:

  $$i * z \equiv 0 \bmod p.$$

  What do we know? We know the inverse of z exists mod p, right? So let's multiply both sides by z^{-1} again, and what are we left with? We're left with:

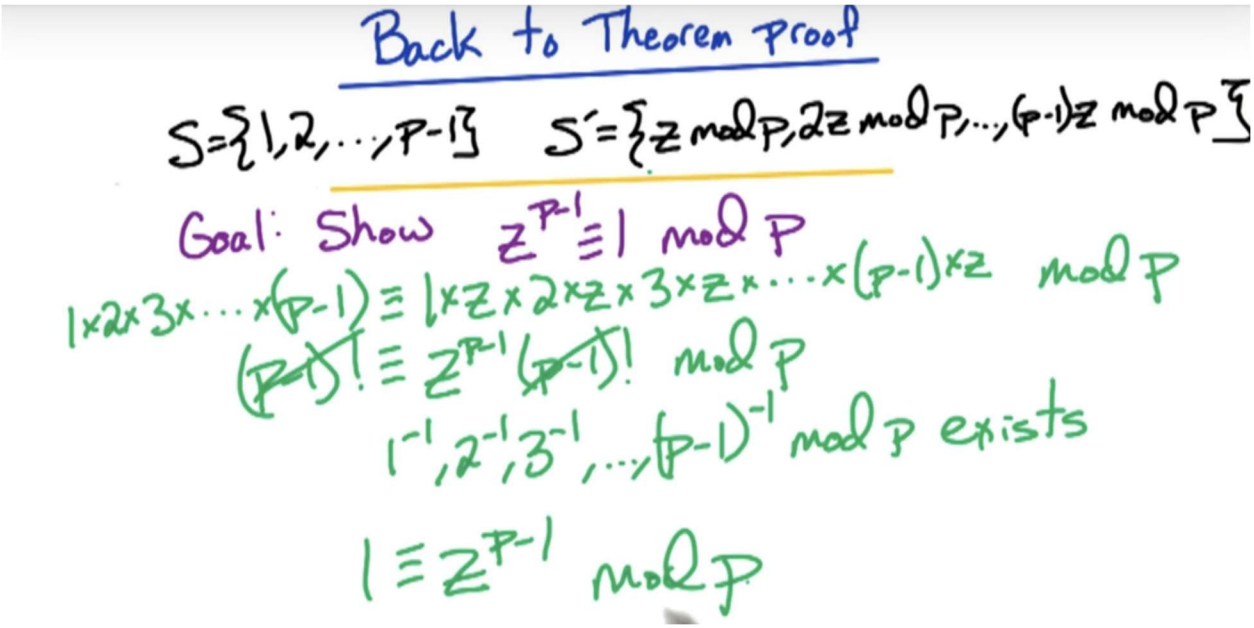
  $$i \equiv 0 \bmod p$$

  which means that i is not one of the indices here, the indices here are between 1 and p-1. So, this is not an element of S′. So we get a contradiction.

Therefore, the elements of S′ are non-zero and they're distinct and, therefore, they contain the elements 1 thru p-1 just in possibly different order and therefore, S and S′ are the same.

***

(Slide 4) Proof: Finishing Up



So let's recap where we are. Our goal is to show that $z^{p-1} \equiv 1 \bmod p$. What have we done so far?

We've looked at these two sets S, which are the possible values for z – 1,2,…,p-1 -- and we have this set S', which we obtained by taking the elements of S, multiplying each one by z, and then taking at mod P: S' = {z mod p, 2z mod p,…, (p-1) z mod p}. So, we have this set of elements, and we've shown, that S and S' are the same. Just in possibly different order.

Now, we're going to use the fact that S and S' are the same in order to prove the theorem statement. So, what are we going to do? We're going to multiply the elements of S together, and we're going to multiply the elements of S' together:

> (Proof) The elements are the same, just in possibly different order. So, when we multiply the elements of S together, we should get the same product as we get when we multiply the elements of S' together.
>
> So, when we multiply the elements of S together, what do we get? - We get 1 x 2 x 3 x ⋯ (p-1).
>
> When we multiply the elements of S' together, what do we get? We get 1 x z x 2 x z x 3 x z x ⋯ x (p-1) x z

And these are the same, modulo p:

$$1 \times 2 \times 3 \times \cdots (p-1) \equiv 1 \times z \times 2 \times z \times 3 \times z \times \cdots \times (p-1) \times z \mod p$$

So, on the left hand side, we've got the elements of S multiplied together. On the right hand side, we've got the elements of S' multiplied together. So, we've have the same numbers, just in different order, and we're taking it mod p, because these sets are the same with respect to mod p.

What do we want on the left hand side? Let's just simplify it a little bit - the left hand side is p-1! (p-1 factorial), right?

$$(p-1)! \equiv 1 \times z \times 2 \times z \times 3 \times z \times \cdots \times (p-1) \times z \mod p$$

What do we have on the right hand side? Well, we have z, z, z -- how many z's do we have? We have p-1 z's: z^{p-1}.

$$(p-1)! \equiv z^{p-1} \times 1 \times 2 \times 3 \times \cdots \times (p-1) \mod p$$

Now that looks good. Now we're starting to look like the theorem statement. And what else do we have? We have $1 \times 2 \times 3 \times \cdots \times (p-1)$.. So, we have another (p-1)! And these are the same mod P:

$$(p-1)! \equiv z^{p-1} (p-1)! \mod p$$

Okay? Can we get rid of this p-1 factorial? In normal real arithmetic, we can just cancel it out. We cancel out here. Well, does the inverse exist? Notice p is prime. So, that means that there is an inverse for 1, 2, up to p-1 -- Each of these numbers has an inverse mod p. Each of these p-1 numbers has an inverse mod p, because p is a prime. So, they're all relatively prime to p. So, we can multiply both sides by these inverses. And what happens? We get rid of this $1 \times 2 \times 3 \times \cdots \times (p-1)$. So that cancels out the p-1 factorial. And then, what are we left with?

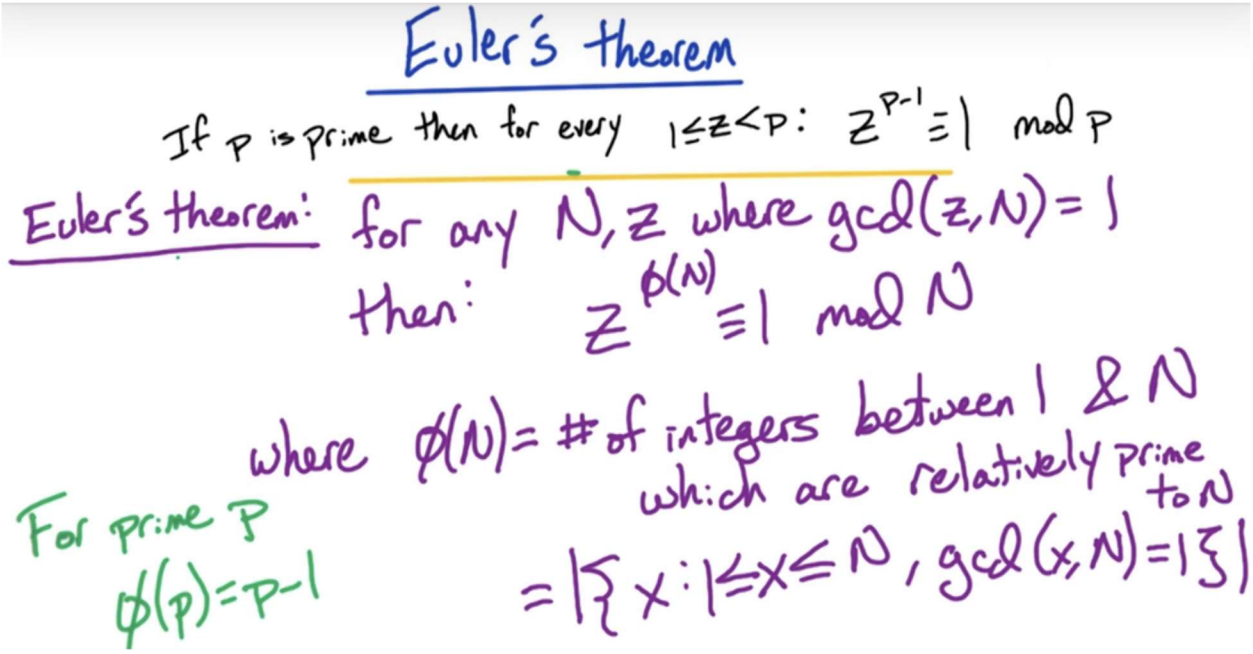
$$1 \equiv z^{p-1} \mod p$$

which is the theorem statement we're trying to prove.

So, we're done!

***

(Slide 5) Euler's Theorem



Now I misled you a little bit … we're going to use a generalization of Fermat's little theorem called Euler's theorem. So, from this "little theory", as Gann says, if p is a prime, then for every z between 1 and p-1, $z^{p-1} \equiv 1 \mod p$. Euler's theorem is going to generalize it to any number n instead of just a prime number p.

So Euler's theorem replaces p by any N, not necessarily a prime number, and any z but we need that z and N are relatively prime to each other. By relatively prime, we mean that gcd(z,N) =1. Now if z and N are relatively prime to each other then we're going to perform an operation similar to Fermat's a little theorem.

So, we're going to raise z to the power phi of N - $z^{\phi(N)}$ (I'll tell you what $\phi(N)$ is in a second), and that's is going to be congruent to one mod N: $z^{\phi(N)} \equiv 1 \mod N$

Now, what is this crazy $\phi(N)$? When N is a prime then $\phi(N)$ is p-1 or N-1. In general, what does it ϕ (N)? It's the number of integers between 1 and N which are relatively prime to N. So, in other words, it's the size of the set of those x's where x is between 1 and N and x is relatively prime to N, which means that gcd(x,N) = 1.

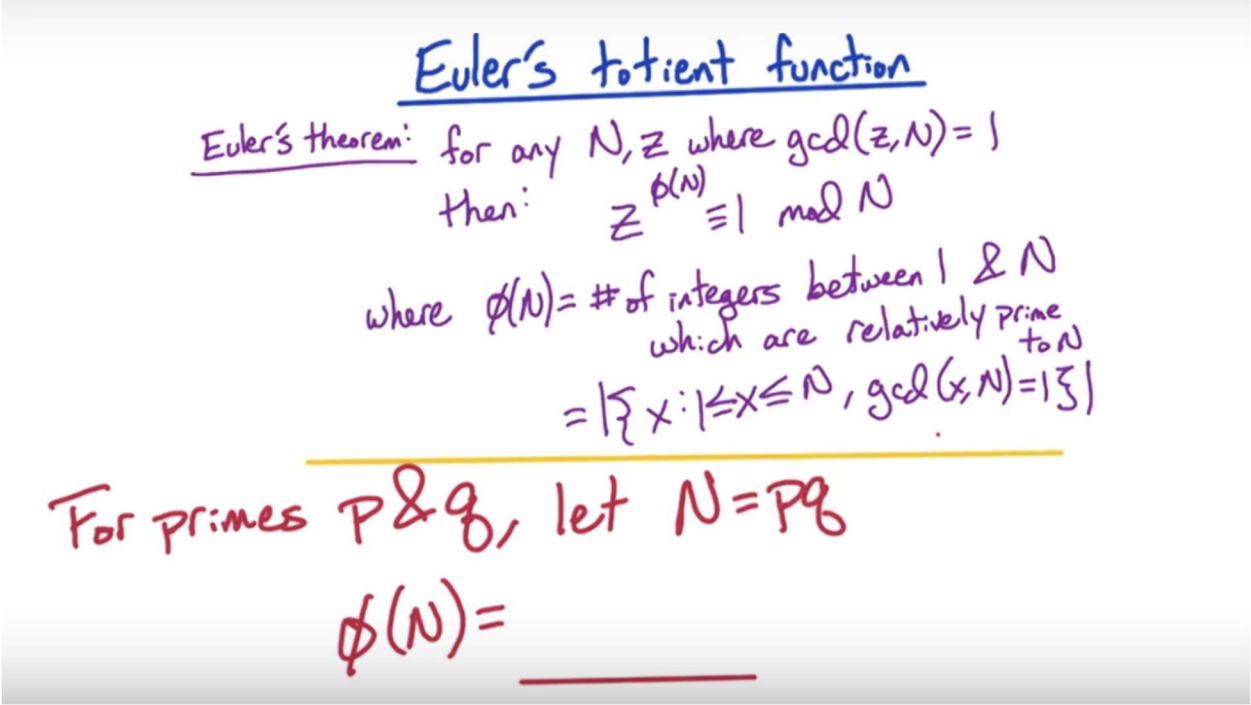
And, we want to look at the cardinality of this set. How many x's between 1 and N, or N-1, have gcd of 1 with N? This function $\phi(N)$ is called Euler's totient function (try to say that ten times quickly).

So let's take a look at what Euler's totient function is for the case when N = p, where p is a prime number. For prime number p, what is $\phi(N)$? How many numbers between one and p are relatively prime to that prime number p? -- well, every number between one and p-1: 1, 2, 3 up to p-1 are all prime -- relatively prime to p. So the number of relatively prime to p is p-1.

So, if you look at Euler's theorem for the case when N is a prime number p, then we are raising z to the power of p-1, and that's congruent to one mod P -- so, it's exactly the same as Fermat's little theorem. So, Euler's theorem is a generalization of Fermat's little theorem to arbitrary N.

***

(1st Slide 6) Euler's Totient Quiz



Now here's Euler's theorem again. Let's look at this Euler's totient function for the case that we're going to use.

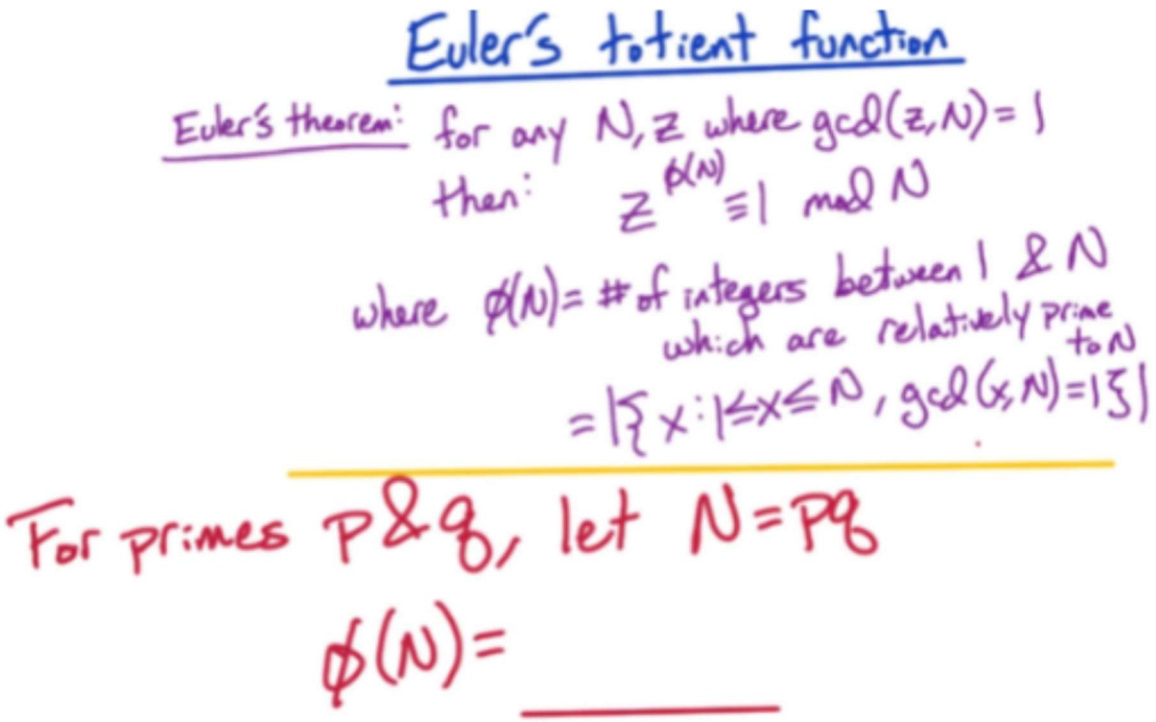
We're going to have a pair of prime numbers p and q. So, we saw when N equals a prime number p, then Euler's theorem is the same as Fermat's little theorem. We're going to apply Euler's theorem for the case when N = pq (the product of p and q), where p and q are both prime numbers.

What is $\phi(N)$ in this case? Why not go ahead and try to figure out what is $\phi(N)$ for the case when N = pq for prime numbers p and q?

So, we want to figure out how many numbers between 1 and pq are relatively prime to pq.

***

RA2: RSA: Euler's Totient Quiz



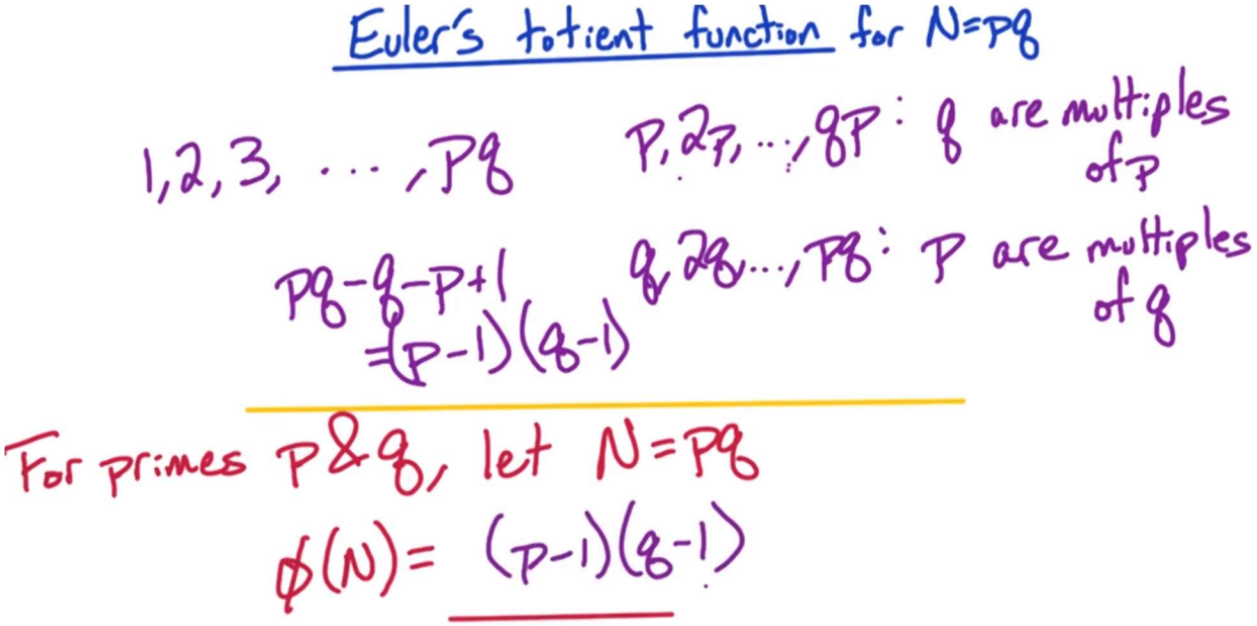
Question:

What is the value of the totient function when N=pq?

- p (q -1)
- (p - 1) q
- (p - 1)(q - 1)
- pq

***

(2nd Slide 6) Euler's Totient Quiz (Answer)



Now, let's go ahead and look at what $\phi(N)$ is for this case when N = p x q.

- Let's write out all the numbers that we're looking at. We're looking at the numbers between 1 and pq. So, this pq numbers here: 1, 2, 3, … ,pq.
- How many of these numbers are multiples of p? Well, p, 2p, 3p, up to qp, are all multiples of p. How many of those are there? There's q of them. So, q are multiples of p.
- How many are multiples of q? There's q, 2q, up to pq. So, p of these numbers are multiples of q.
- So, each of these numbers p, 2p,…,qp has a common divisor with pq – namely, p. And each of these numbers q, 2q,…,pq has a common divisor with pq, namely q.

  So, we have to exclude all these numbers. So we had pq numbers, originally. q of them are multiples of p, and p of them are multiples of q. And look, we double counted: pq is counted twice, so let's add one while I can:

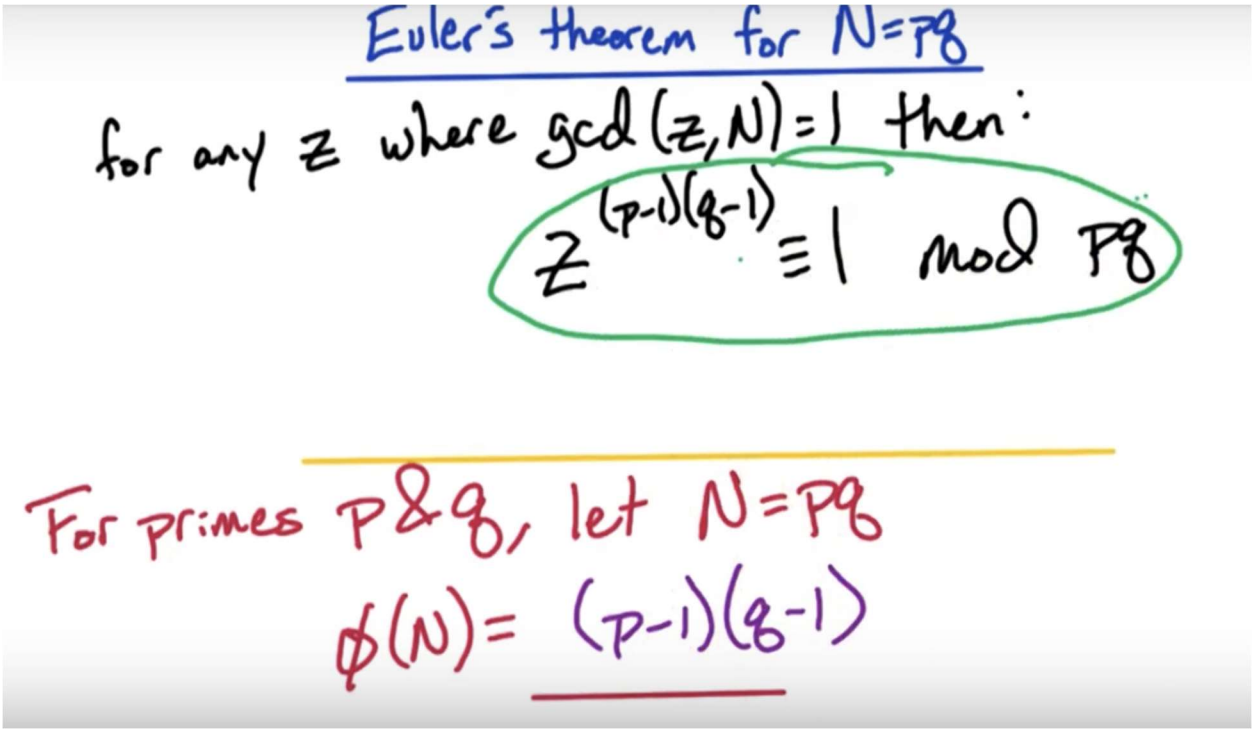
  $$\phi(N) = pq - q - p + 1.$$

  This can be rewritten as $\phi(N) = (p - 1)(q - 1)$.

  So, what is $\phi(N)$ for the case when N = p x q for prime numbers p and q?
  It's $(p - 1)(q - 1)$.

***

(Slide 7) Euler's Thm. For N=pq



What does Euler's Theorem tell us for the case when N = pq, where p and q are prime numbers? -- For any z which is relatively prime to N (which means that gcd(z,N) = 1), and since we have that $\phi(N) = (p-1)(q-1)$), and N = pq:

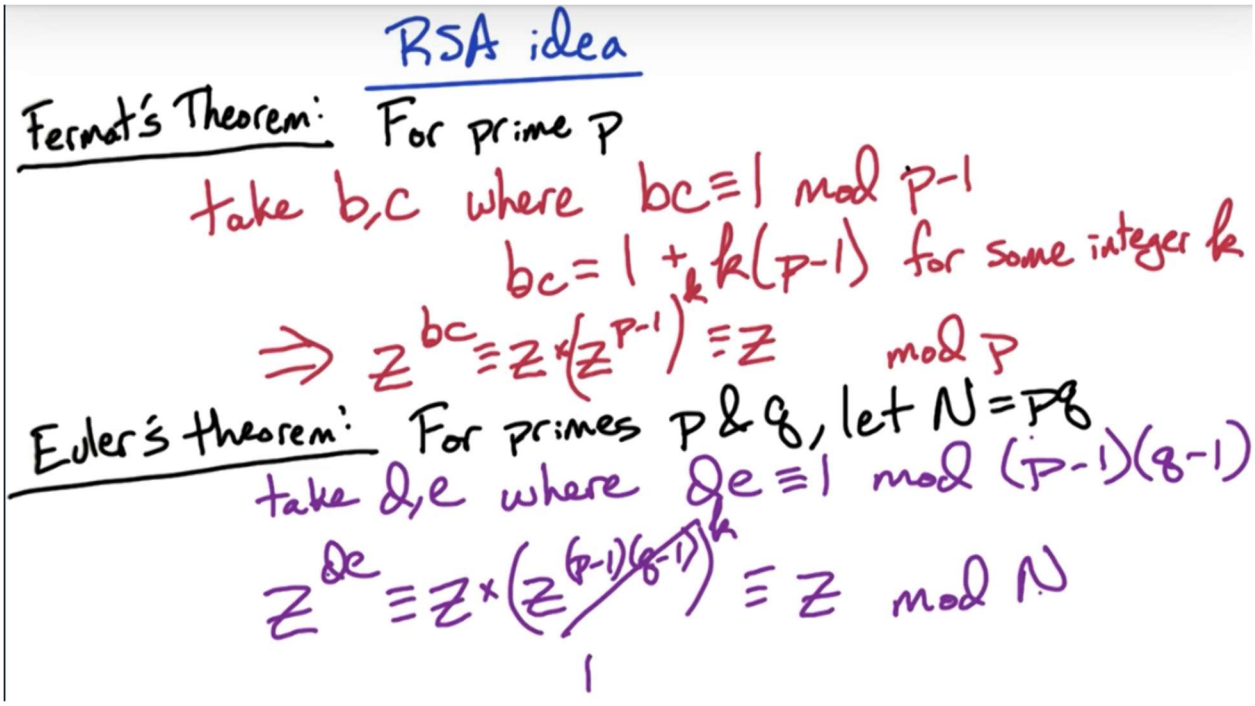
$$z^{\phi(N)} = z^{(p-1)(q-1)} \equiv 1 \bmod pq$$

This fact is going to be the basis for the RSA algorithm: We're going to generate a pair of prime numbers, p and q. And we're going to look at p times q. And, we're going to have a message, and we're going to use (p-1) times (q-1) to generate a encryption key and the decryption key. And this is going to give us a basic fact, which is going to allow the encryption and decryption.

Let's investigate this a little bit more and then we'll see the basic idea of the RSA algorithm.

***

(Slide 8) RSA Alg. Idea



Now we can get to the basic idea behind the RSA algorithm.

Let's first look at Fermat's Theorem. It talks about a prime number p. Let's look at a pair of numbers, b and c, which are relatively prime to each other mod p-1. Why p-1? Because, remember, in Euler's totient function, or in Fermat's Little Theorem, the exponent there is p-1. That's where that p-1 comes from.

So we're taking a pair of numbers b and c which are inverses to each other modulo p-1. So what does that mean? That means $bc \equiv 1 \bmod p-1$, or in other words $bc = 1 + k\,(p-1)$, where k is some integer. So it means that when we take b times c and we divide it by p-1, then we get a remainder of 1.

Let's take a number z between 1 and p-1, and let's raise it to the power bc, and let's take the whole thing mod p: $z^{bc} \bmod p$. Now we can replace bc using this formula $bc = 1 + k\,(p-1)$:

$$z^{bc} \equiv z^{1 + k\,(p-1)} \equiv z \times (z^{p-1})^k \bmod p$$

Now what do we get? Well, the z sticks around. What about the z to the p-1? What do we know? Fermat's Little theorem tells us that $z^{p-1} \equiv 1 \bmod p$. So we can replace z^{p-1} by 1 and then one to the K is what? It's 1. So this whole thing is one. So it drops out and we're left with z:

$$z^{bc} \equiv z^{1 + k(p-1)} \equiv z \times (z^{p-1})^k \equiv z \times (1)^k \bmod p \equiv z \bmod p$$

So z raised to the power B times C is z mod P. So we get back z itself. Okay?

We're starting to get the idea for the encryption and decryption algorithm: I'm going to take a message z, and now, if I raise it to the power b to encrypt it and then later I raise it to the power c to decrypt it, that's the same as taking z to the power b times c and that's going to end up with z itself. So if I encrypt by raising to power b and then decrypt by raising it to the power c, I end up with the original message itself.

Now, the problem is that I have to tell everybody p, so that they can do these operations. If I tell them P then they know p-1, and therefore, given b then they can figure out c, which is the inverse of B with respect to mod p-1.

So I want to conceal this fact. I want to conceal this p-1 term. How am I going to do that? -- well, I'm going to use Euler's theorem instead of Fermat's little theorem. For Euler's theorem, I want to have a pair of primes, p and q. I'm going to look at their product N. Now, what am I going to do? I'm going to take a pair of numbers, d and e, which are inverses of each other with respect to p-1 times q-1. So, $d\ e \equiv 1 \bmod (p-1)(q-1)$.

> Why did they use p-1 and q-1 instead of p-1? Well, in Fermat's little theorem, when I raise z to the p-1, I get 1 mod p. What happens in Euler's theorem? Well, the totient function of N we saw was (p-1)(q-1) for this case when it's p times q. So, if I take a number z and I raise it to (p-1)(q-1), I'm going to get 1 when I look at it mod N: $z^{(p-1)(q-q)} \equiv 1 \bmod N$

So, that's what I'm going to do. I'm going to take a number z; I'm going to raise it to the power d x e. What do I get? $d\ e \equiv 1 \bmod (p-1)(q-1)$.

That means $d\ e = 1 + k(p-1)(q-1)$, for some integer k, so:

$$z^{de} \equiv z \times (z^{(p-1)(q-1)})^k \equiv 1 \bmod N$$

Now, what is this, $z^{(p-1)(q-1)}$? I'm looking at this whole thing mod N. Euler's theorem tells me that if z is relatively prime to N, then this thing is one. So then, one raised to the power of k, just like before, is one. So this term goes away, I'm left with Z, and there it is:

$$z^{de} \equiv z \times (z^{(p-1)(q-1)})^k \equiv z \times (1)^k \equiv z \bmod N$$

Notice the terms “d” and “e” -- decryption, encryption. So, you give me a message z. I'm going to tell you e and I'm going to tell you N, what are you going to do? If you want to send me the message z, you're going to take z and you're going to raise it to power e and then get mod N, and then you're going to send it to me. That's the encrypted message.

What do I do? I compute this d, which is the e^{-1} mod (p-1)(q-1). That's my decryption key. So, I take that encrypted message with the z^e mod N; I raise it to the power d. And, what am I left with? I'm left with z itself, the original message.

Now, the point is you know N, which is p times a. I know p and q, but you don't know them. You just know the product of them, you know N. So from N, can you figure out p-1 times q-1? I don't know how to do that unless you tell me p and q. If you just tell me their product, p times q, I don't know how to get p-1 times q-1 out of p times q.

But, I know p and q, so I can take p and q and compute p-1 times q-1, and then I can run the Extended-Euclid algorithm to compute e^{-1} mod (p-1)(q-1). So I can compute d but nobody else knows how to compute d because they don't know p-1 q-1. They simply know N, P times Q.

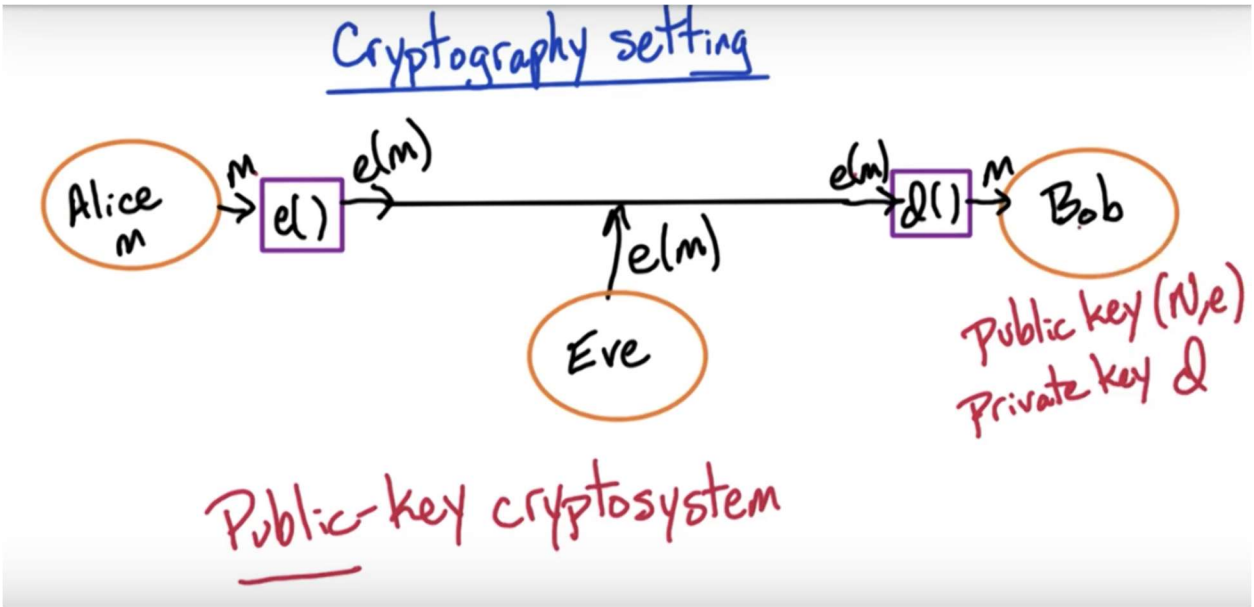
So, everybody's going to know N, and everybody is going to know e; but, I, alone, know d and that's the decryption key. So I tell everybody e and N, they take the message z, raise it to the encrypted power e, take it mod N -- so, z^e mod N -- that's the encrypted message.

Send that. The whole world can see that, but only I can decrypt it.

So, now, let's go ahead and detail the RSA algorithm.

***

(Slide 9) Crypto Setting



Here's the general Crypto setting. I have Alice over here and we have Bob over here. Alice has a message m, that she wants to send to Bob. Now, they have a communication line that they can talk over. The problem is that there's someone else eavesdropping on this conversation. So, whatever is transmitted over this communication line can be seen in the clear by Eve, who's the eavesdropper on this communication. So, Eve sees everything in the clear that's communicated on this line. So, Alice doesn't want to send the message by itself. She wants to encrypt the message.

So, Alice is going to take the message m; she is going to feed that into the encryption scheme and out is going to pop e(m), the encrypted form of m, the message. She's going to send that along the transmission line. Bob, meanwhile is going to have his decryption box. So, he's going to take this encrypted message, feed that as input into his decryption box, an arrow is going to pop the decrypted message.

Meanwhile, Eve what did she see? She sees this encrypted message, because that's what sent over this communication line. And, from the encrypted message, she doesn't know how to decrypt. Only, Bob knows how to decrypt.

Now, this is a public key cryptosystem. By public key, we mean that no communication needs to be happen between Alice and Bob in private. So, they don't need to have any coordination beforehand. What's going to happen is that Bob is going to do a computation on his own and he's going to compute a public key and a private key. His public key is going to be two

numbers, N and e. N is going to correspond to p times q, the product of two primes and e is going to be some number which is relatively prime to p-1 times q-1. He's going to announce to the world, N and E.

Now, anybody that wants to send a message to Bob is going to take his public key. They're going to take their message m, they're going to encrypt their message using N and e, the public key. But, the thing is that only Bob knows how to decrypt messages that are encrypted using his public key. So, he's going to have a private key d, what is d? d is going to be e^{-1} mod (p-1)(q-1).

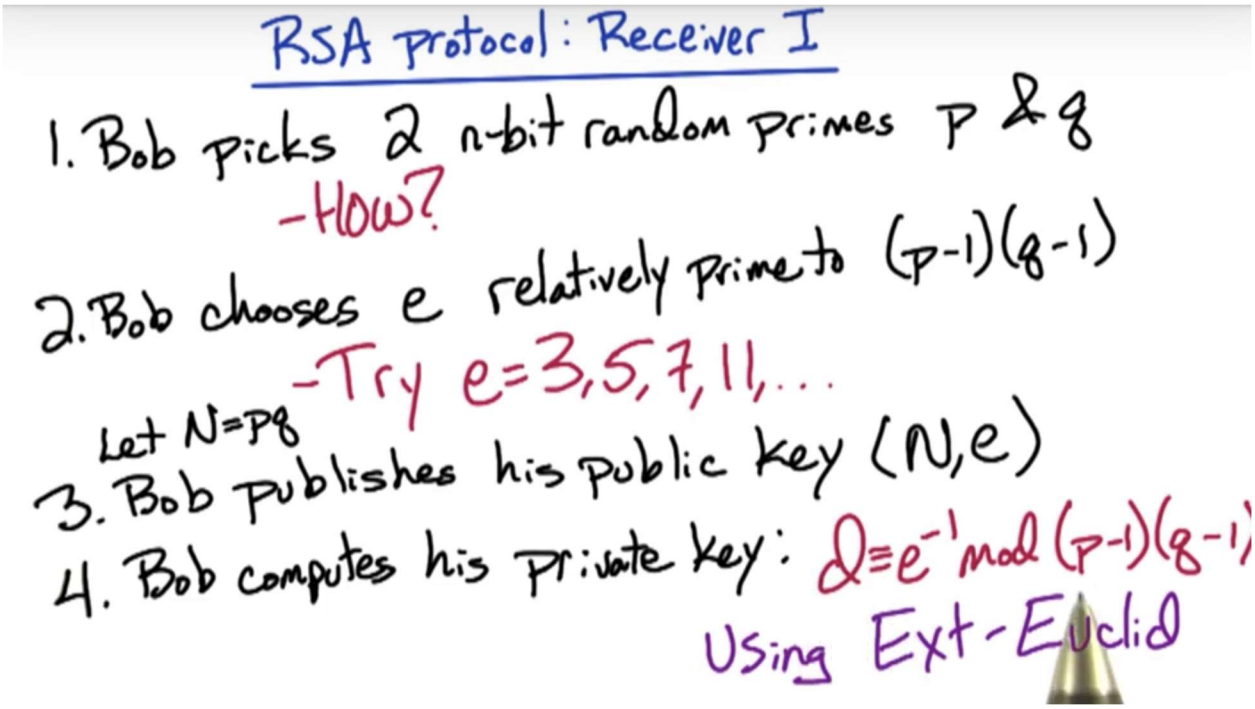
Only he knows p and q, so only he can compute d. He can compute d and then he can take this encrypted message and he can raises it to the power d and then take that mod N and that will give him the original message. But, only Bob knows how to compute this private key, so only he can do this decryption.

There's no coordination that needs to happen between Alice and Bob. Bob does this computation on his own. He computes N and e and announces it to the world and he computes this private d and keeps it to himself. Meanwhile, anybody that wants to send a message to Bob looks up his public key, sees his N and e, he uses that to decrypt the sender's encrypted message in the clear. So, anybody can see the encrypted message and then, only Bob can decrypt it.

So, let's detail this scheme -- this is a RSA cryptosystem.

***

(Slide 10) RSA Protocol: Keys



Let's detail the RSA protocol. Let's start with the receiver, Bob. So what is Bob going to do?

(Receiver)

(Step 1) Bob has to compute his public key and his private key and he has to announce to the world his public key. How does he do it? The first step that Bob is going to do is that he's going to choose two n-bit numbers p and q which are primes. And, he's going to do that at random.

So he's going to choose two random prime numbers p and q. How does he do that? -- We haven't discussed that at all. And, we're going to skip it until after we see the whole protocol -- the idea is that, first, he's going to generate two random n-bit numbers p and q. And then he's going to check whether those random n-bit numbers are prime or not.

> So, we're gonna have a primarily testing algorithm, so that we can efficiently test whether a number is prime or not. So, he's going to generate random numbers; test whether they are prime; if they are prime, then they are random prime; if they're not prime, he's going to repeat and generate a new random number and check whether it's prime again, and keep going until he gets a prime number.

How do you generate a random number? You just try and generate a random string of 0s and 1s of length n that gives you an n-bit random number.

How do you check whether it's prime or not? Well, that's a little bit more complicated. And it turns out that a random number has a reasonable chance of being prime. Primes are dense in some sense -- We'll see details of all that next, after we go through the whole RSA protocol in detail.

(Step 2) Bob chooses an e which is relatively prime to p-1 times q-1 -- How does he do that? -- He's going to try e = 3 and he's going to check whether the gcd(3,(p-1)(q-1) = 1. If it is, then they are relatively prime to each other.

How does he check it? He runs Euclid's GCD algorithm. And if 3 is not relatively prime, then what does he do? He tries 5 and then 7 and then 11 and so on.

> You know, if you get up to 13 or 17 or q-1, what do you do? Usually, you go back to the first step, choose two new primes p and q and try again. It's nice to keep this encryption key small. It makes it easy for somebody to encrypt their message, to send to you.

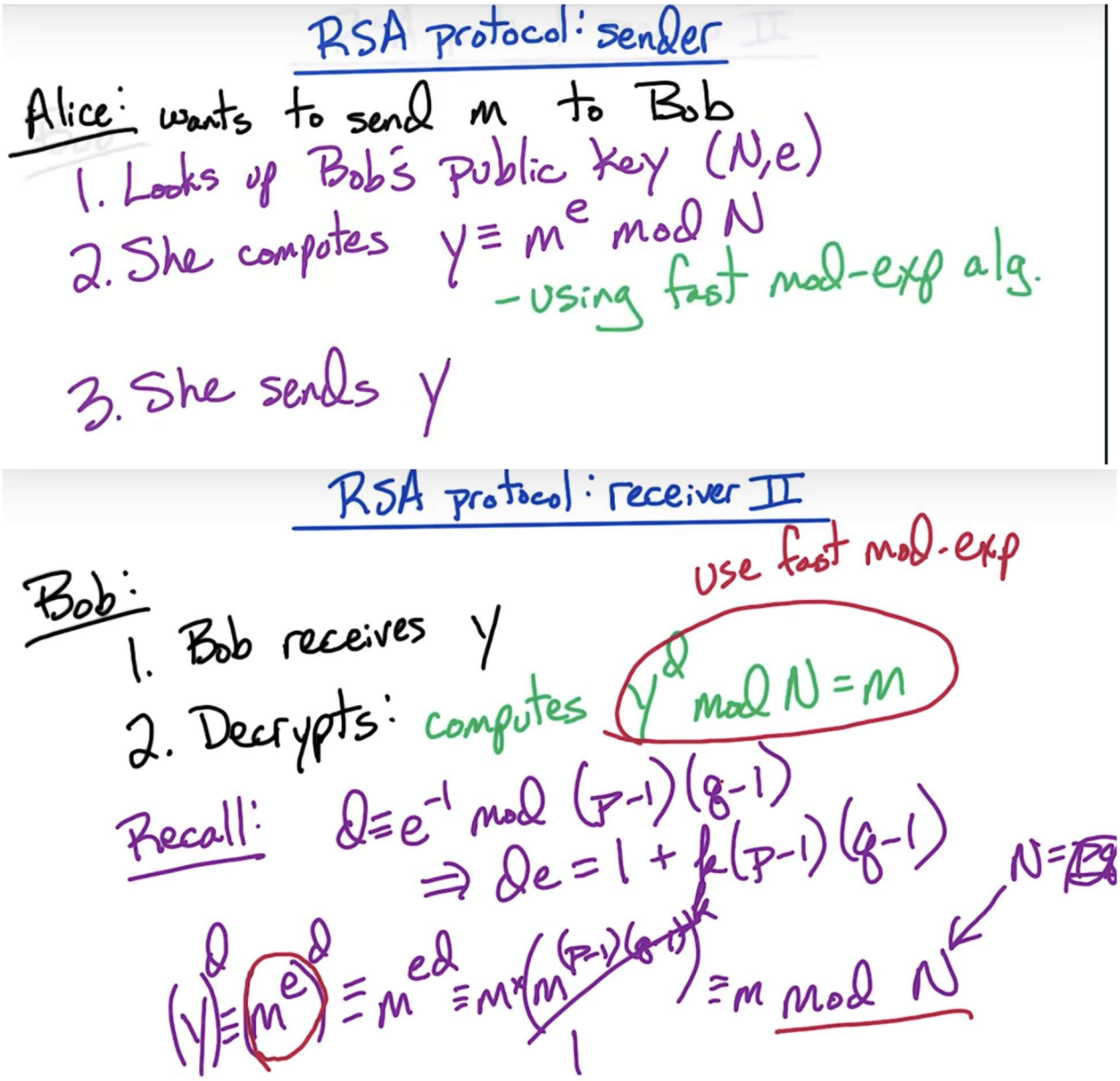
Now what do you do? Let's let N equal P times Q.

(Step 3) Now, Bob can publish his public key, N and e. N is the product of p times q. So he's publishing that product. He's not publishing p or q, he's just telling what the product is and he's telling them what e is, where e is relatively prime to (p-1)(q-1). So the whole world can know N and e.

Meanwhile, Bob computes his private key. What's his private key? It's the e^{-1} relative to (p-1)(q-1). So $d = e^{-1} \bmod (p-1)(q-1)$. How do we know that d exists? Because e is relatively prime to (p-1)(q-1). We chose it so that it was relatively prime; therefore, it has an inverse mod (p-1)(q-1). And we can find that inverse, how? By using the Extended Euclid algorithm, we can find the inverse. This is going to be Bob's decryption key. He keeps it private, he doesn't tell anybody. He tells the whole world N and e, but he keeps private his decryption key, d.

***

(Slide 11) RSA Protocol: Encrypting



Now, let's look at things from Alice's perspective.

(Sender)

(Step 1) Alice has a message m that she wants to send to Bob. What's the first step that Alice does? She looks up Bob's public key which is this pair N, e.

(Step 2) She needs to encrypt her message using Bob's public key. What does she do? She encrypts using their public key. She encrypts the message using m^e mod N. And that's her encrypted message, y, that she sends to the world.

Now, one key thing is that e might be a little bit large. So, how does she raise m to the power e mod N? She uses our Fast modular exponentiation algorithm that we just saw earlier in this lecture.

(Step 3) Finally, Alice can send the message y.

Now, let's look at the final step of the procedure. What happens for Bob?

(Step 1) Bob receives this encrypted message y that Alice sent.

(Step 2) Now, Bob decrypts this message. How does he decrypt it? He computes y, this encrypted message, raises it to the power d which is his private key and he takes that mod N.

What is that going to equal? That's going to give him back m. So, he's going to end up with the original message m. Let me recall why that's the case.

Remember, how did we choose d? $d = e^{-1} \bmod (p-1)(q-1)$. That means $d \times e \equiv 1 \bmod (p-1)(q-1)$. That means $d \times e = 1 + k(p-1)(q-1)$, for some integer k.

Now, what are we doing? We're starting with the message m. Alice is encrypting it by taking that message m by raising it to the power E, then taking it mod N. Recall, N is p times q where p and q are prime numbers. This m^e is y: $y \equiv m^e \bmod N$.

Now, what does Bob do? Bob takes this as message y and he decrypts it by raising it to the power d: $y^d \equiv (m^e)^d$ What do we get then? $y^d = m^{e\,d}$. What do we know about e times d? That's equal to one plus some multiple of (p-1)(q-1): $y^d = m^{e\,d} = m^{1+k(p-1)(q-1)} = m \times (m^{(p-1)(q-1)})^k$ What is $m^{(p-1)(q-1)}$? -- well, when m is relatively prime to N, then Euler's theorem tells us that this is 1. So, this whole term drops out and what are we left with? We're left with m, the original message. So, we take this message m raise it to power e and then raise it to the power d, what do we end up with? $(m^e)^d = m$, the original message. This is the case when m is relatively prime to N.

And also it holds when m and N have a common factor namely, p or q. In which case you can prove this statement still holds. But it takes a little bit more work. You want to use Chinese Remainder Theorem. But that's the basic idea. We use Euler's theorem. So, that gives us this (p-1)(q-1) term. That's the whole RSA algorithm. It's fairly simple.

The only thing that's missing now for us is how do we generate a random prime number? How do we get this p and this q? We said, generate a random number and then check whether it's prime. So, we've want to look at how to check whether a number p is prime or not. We'll do that next.

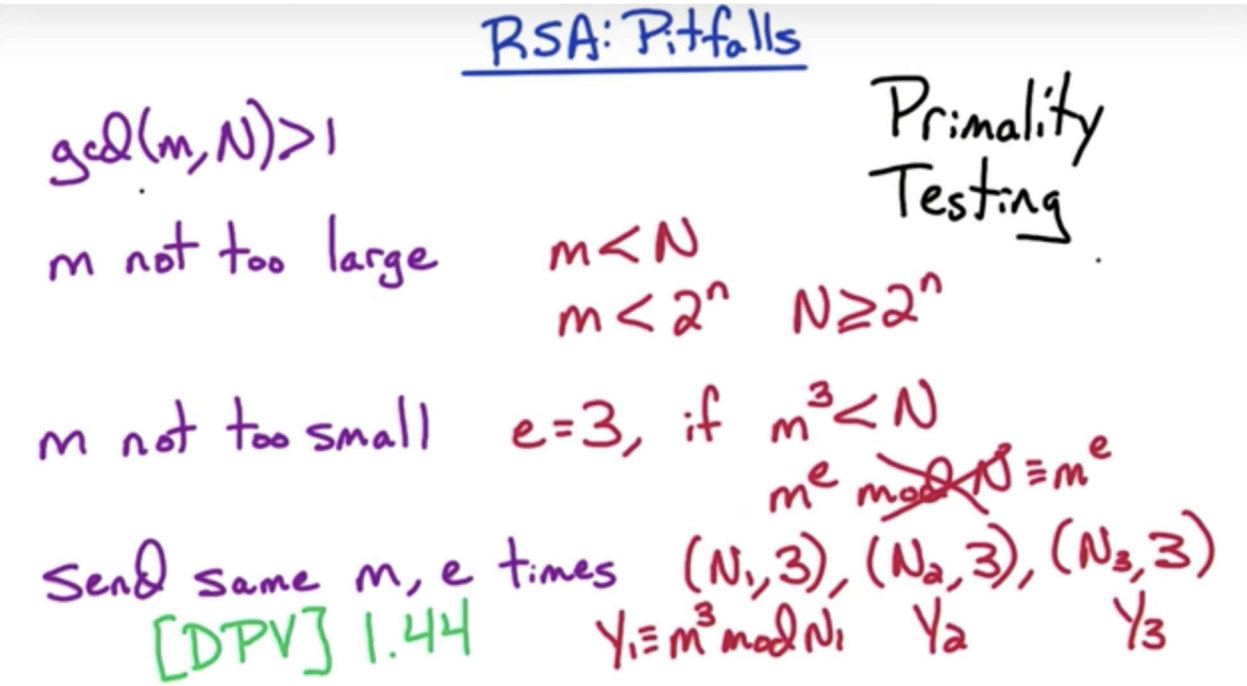
But, let's first look at some simple issues that might arise in the RSA algorithm that you have to be careful about.

One important note before we move on. How do we compute this y^d mod N? Notice that we, typically, tried to make e small. Why did we make e small? -- So that it's easy -- it's fast to compute the message raised to the power e mod N -- So that it's easy and it is fast to encrypt the message. So, we want to make it easy for somebody to send us an encrypted message. But, then, we're going to put in extra work in order to decrypt it.

Why? Because this y^d, if e is smaller than d, it's is probably going to be a huge number. So, how do we take this encrypted message y and raise it to the power d? Here is where we really need to use our Fast modular exponentiation algorithm. This was the algorithm based on repeated squaring. And then using this, we can compute y^d efficiently, mod N. But if we don't use the Fast algorithm -- if we use a naive approach -- then this is going to be exponential time and there's no way we're going to be able compute it efficiently.

***

(Slide 12) RSA: Potential Pitfalls



Let's look at some of the issues that can arise when we're implementing RSA.

First off, suppose in our message m is not relatively prime to N. So, gcd(m,N) > 1. Now, N is a product of two primes p and q -- it's only divisors are p and q. So, what can be the common divisor of m and N? It's has to be either p or q.

So, let's suppose gcd(m,N) = p. What happens in the RSA protocol? -- we take this message m raise it to the power e, and look at that mod N and then the receiver takes this encrypted message raises it to the power d and then takes it mod N and what do we get back? Well, $(m^e)^d$ equals to the original message m.

Now, we didn't prove this case when gcd(m,N) > 1. We proved the case when they're relatively prime to each other and that followed by Euler's theorem and we claimed that this case when they're not relatively prime followed by the Chinese remainder theorem.

This is, in fact true, but there's a potential problem in this case. What's the problem? – well, if p divides m and N and, if we look at the encrypted message y, which is m^e mod N, then since P divides m and N, then it's also going to divide y. So, gcd(y,N) = p. What does this mean? Well, anybody that's eavesdropping is going to see this encrypted message -- and from this encrypted message and the public key N, by simply running Euclid's algorithm, they can take

the gcd of these two numbers and find that the gcd is p - the prime p - and once they know prime p, then they can factorize n, and, therefore, they can break the RSA cryptosystem -- they can find the decryption key d. So, before using this message m and sending the encrypted version of it, we have the first check that m and N are relatively prime to each other. Otherwise, if they share a common factor (ex: gcd(m,N) = p), then anybody will be able to use this encrypted message to break this crypto system.

What are some other issues that can arise? We need that m is not too large. In particular, we need that $m < N$. Now, typically our message is text. So, we first have to convert it into a number. How might we do that? Well, we can take a binary version of the text. Now, the binary version of text is gonna be a huge number. So what we do is we break this huge number to n-bit segments. These are segments of length, little n and therefore little m will be at most strictly less than 2^n. So, we break the message into little n bit strings then we get this property.

And, if we ensure that p and q are sufficiently large (ex: $N > 2^n$) to ensure that their leading bit is 1, then, if we look at capital N, we know that capital $N \geq 2^n$. Therefore $m < N$. So this property that little m is not too large is easy to ensure – but, we also need that m is not too small. Why is that the case?

Well, suppose that e = 3 which is a common practice and suppose that m is very small number – so, $m^3 < N$. So, when we look at m^3 mod N, what do we get?

Well, the mod N isn't doing anything because $m^3 < N$. So $m^3 \bmod N \equiv m$. So this mod N is not doing anything when this number m is too small and that means that our encrypted message - the message we're sending in cleartext - is simply m^3. The mod N isn't doing anything.

Now, if we see this message m^3, how do we decrypt it? We just take the cube root – anybody that sees this encrypted message can simply take the cube root and get the original message back. So, it's easy to decrypt; it's easy to break the script… a system, in this case.

> Note that cube roots are difficult to do when we're doing it with respect to modular arithmetic. But in real arithmetic, cube roots are easy to do.

So, how do we avoid this issue when I have a small message? We can choose a random number r and we can look at $m \oplus r$ - m xor r - and we can send this new message, this padded message. and we can also send the second message which is just r itself. So we send two messages - the padded message, and r itself. And, as long as r is not too small, then this will be okay. And, if r

is too small, just choose a new random string until we get an r which is sufficiently large.

Now, there is one last issue that I want to point out: If we use the same message multiple times, then we have a potential problem. Suppose, we have the same message that we just want to send to three different people each of these three people have a different public key, but suppose they're all using e=3:

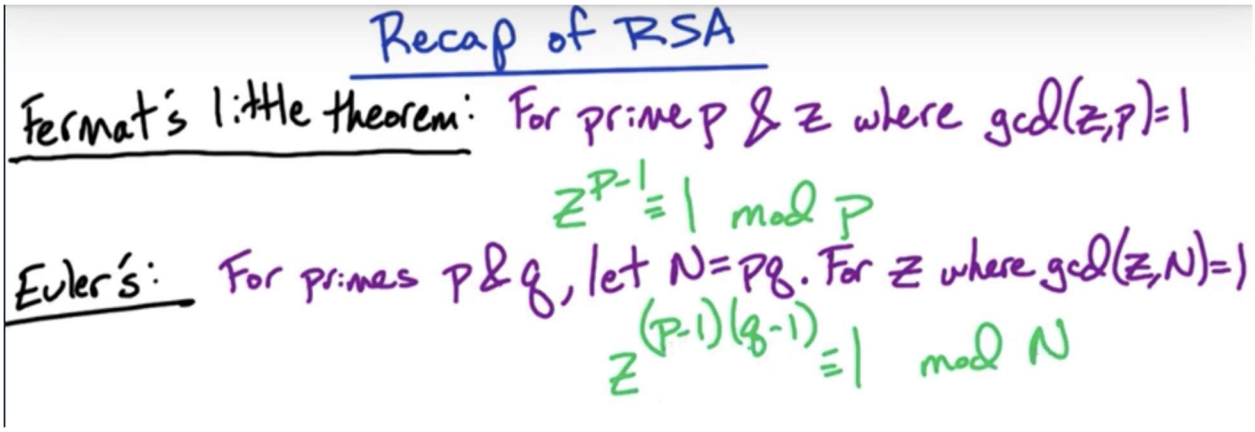
- The first recipient has e=3, N_1.
  Second recipient has e=3,N2.
  Third recipient has e=3, N3
- And, suppose we the same message m to send to these three people.
- Now, the encrypted messages are going to be different --
  the first encrypted message Y1 is going to be m^3 mod N1.
  Second encrypted message Y2 is going to be m^3 mod N2.
  The third one is going to be $Y3 \equiv m^3 \bmod N3$.
- So we got three different encrypted messages. Y1, Y2, and Y3. But it turns out that if somebody sees three encrypted messages which all come from the same message the same number m -- Then, they can decrypt -- they can figure out m from Y1, Y2, and Y3. How do they do that? They use the Chinese remainder theorem. This is a homework problem in the textbook and this is the [DPV] textbook - in Chapter 1, this is problem 44.

Now all that remains for specifying the RSA protocol is to describe how we do primality testing.

So let's dive into that.

***

(Slide 13) Recap of RSA



Let's take a moment now to recap the RSA algorithm and also the mathematics behind the algorithm.

We start with **Fermat's little theorem**, that was the basis of the whole algorithm. Let's recap that: so the setting is: we have a prime number p and we have another number z, where z and p are relatively prime to each other, which means that the gcd of z and p is 1. This means that they're relatively prime if their gcd is one.

Now, the theorem says, if we take z and raise it to the power p-1 and we look at it mod p, then what are get? We get 1: $z^{p-1} \equiv 1 \mod p$. And that's true for any z, so take any z which is not a multiple of p and we raise it to power p-1 we get 1 mod p.

Actually, we didn't use Fermat's little theorem, we used the generalization known as **Euler's theorem**. Now Euler's theorem is a general theorem that holds for any capital N, but we used it for the particular case where capital N was the product of two primes, p and q. So let's recap it for this particular case that we're interested in:

So, for primes p and q, look at their product, capital N (N = pq), and take a z, which is relatively prime to capital N. Now in this case, if we raise z to the power (p-1)(q-1), then we get the analog of Fermat's little theorem: $z^{(p-1)(q-1)} \equiv 1 \mod N$..

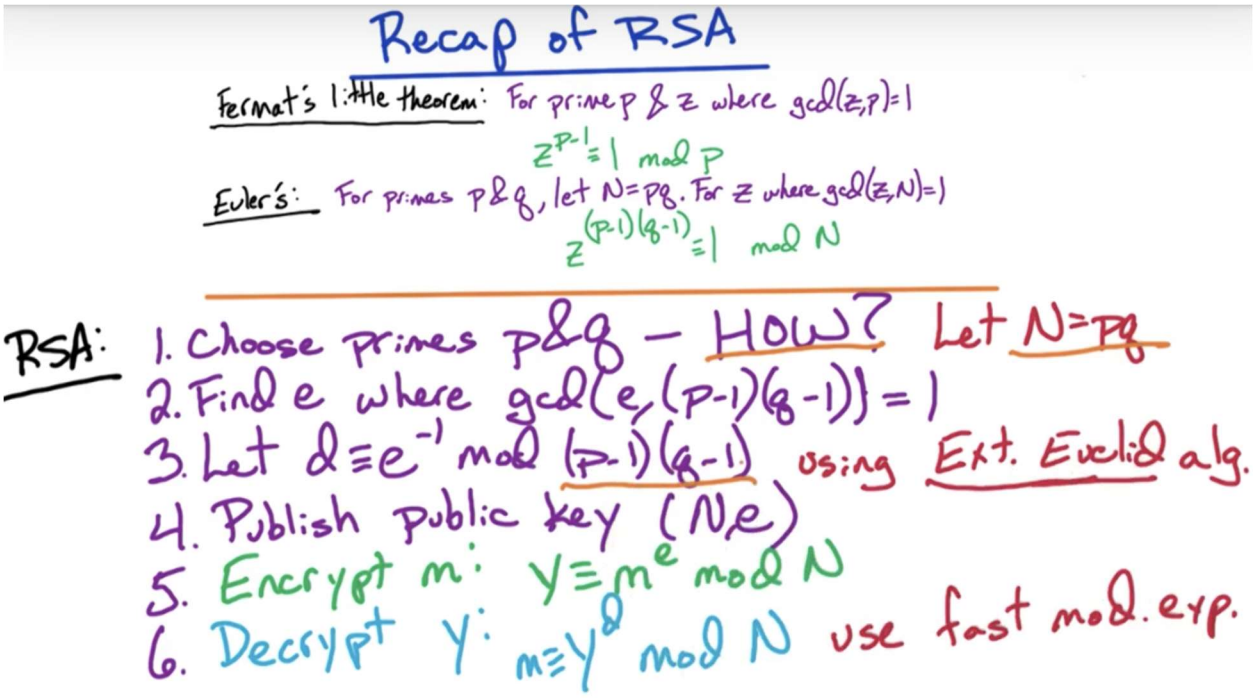
Now, where did this exponent (p-1)(q-1) come from? Well, that came from when we looked at how many numbers were between one and N are relatively prime to N.

- For the case where N = p, then all numbers from 1 to p-1 are relatively prime to this N. So that's why the exponent is p-1.

- For the case where N = pq, then the number of numbers between 1 and pq which are relatively prime to N are (p-1)(q-1). That's what we solved before.

***

(Slide 14) Recap of RSA #2



Now, Euler's theorem was the basis for RSA algorithm. Now, let's go ahead in detail again the RSA algorithm.

RSA:

(Step 1) The first step is to choose a pair of primes p and q. These are the ones that we haven't seen actually how to do. We're going to explain how to choose prime numbers after we review the RSA algorithm. Now, after we choose the pair of primes p and q, we look at their product. Hence let N = p x q.

(Step 2) The next step is to find an e which is relatively prime to (p-1)(q-1). By relatively prime, again we mean that gcd(e, (p-1)(q-1)) = 1. So they have no common factors. Now, why did this (p-1)(q-1) come up? Because -- recall Euler's theorem -- that's the exponent here.

(Step 3) So, what is the key implication of the fact that e is relatively prime to (p-1)(q-1)? That means that e has an inverse. So, the third step is to find the inverse of e relative to (p-1)(q-1).

So, let d be the inverse of e mod (p-1)(q-1). We know it exists, because e is relatively prime to (p-1)(q-1). How do we find this inverse? We use the Extended Euclid

algorithm.

(Step 4) Now, we can publish our public key N and e. We tell the whole world this public key N and e and anybody that wants to send us a message will encrypt that message using this public key, but nobody is going to know our private key d. We're going to keep this private key d secret, because anybody that finds this private key d can decrypt messages.

(Step 5) Now, anybody that wants to send us a message, let's say, m, they're going to encrypt the message using our public key. They take that message m, they raise it to power of e and look at a mod N. And then they send that y, which is congruent to m^e mod N -- they send that message y and the whole world can see that message y, but only we can decrypt it, because only we know d.

(Step 6) Finally, we receive this message y, this encrypted message. How do we decrypt it? We use d in the following way. We take this encrypted message y, we raise it to the power of d and we look at that mod N: y^d mod N. And it turns out that this equals m -- y^d mod N = m.

What are the algorithms that we need to use?

- Well, first off, to find this e, which is relatively prime to (p-1)(q-1), what do we do? We try e=3, 5, 7, 11, 13, at some point we give up. For each of those e's, though, how do we check whether it's relatively prime to (p-1)(q-1)? We check its gcd which we do using **Euclid's algorithm**.

- Then once we find an e which is relatively prime, we compute its inverse. How do we do that? We used the **Extended Euclid algorithm**, and we publish our key, usually e is small, so that taking m to the e mod N is relatively easy.

- If it's not, then we can use repeated squaring -- our **Fast modular exponentiation** algorithm, and definitely here, when we're raising this encrypted message Y to the power d, that's probably going to be to a huge power. So, how do we do y^d mod N? Here we need to use our fast modular exponentiation algorithm - the algorithm based on repeated squaring - and that's going to take time which is polynomial in little n, the number of bits in these numbers y and d and N.

Finally, the key thing about why this works is look at what's happening to the message. We're raising the message m to the power e and then to the power d. And recall that e d ≡ 1 mod (p-1)(q-1). So, when we apply Euler's theorem what we're going to get out is -- we're going to get the message m back out. So, m raised to the power e times d modulo N is congruent to m, because of Euler's theorem.

What remains?

- We need to look at how we choose these prime numbers p and q.

- The other thing is let's just make an aside about why this algorithm is hard to break.

  Notice that the whole world knows N, which is p times q, but only we know (p-1)(q-1), and therefore only we can compute e^{-1} mod (p-1)(q-1). The point is, can you get (p-1)(q-1) from N without knowing p and q, individually?

  The assumption is that no, we can not do that, that the only way to get (p-1)(q-1) is to factor N into p and q. If you don't know how to factor N into the pair of primes which compose it, then you cannot get (p-1)(q-1). That's our assumption.

  So, this algorithm is as hard as factoring N. We don't know any other way to get (p-1)(q-1). And our assumption is that factoring is difficult, computationally difficult to solve.

Now, to finish off the RSA algorithm, let's look at how we find prime numbers p and q.

***

(Slide 15) RSA Exercise

See RA2: RSA: RSA Exercise

***

RA2: RSA: RSA Exercise

For all of the following, we are working with primes **p = 13, q = 31**

Question 1
Encryption Key
What is the smallest encryption key **e?**

Question 2
Public Key
What is the public key **(N, e)**?

Answer format: (N,e)

Question 3
Private Decryption Key
What is the private decryption key **d**?

***

(No Slide)

RA2: RSA: RSA Exercise (Answer)

Problem

Given primes $p = 13, q = 31$.

The smallest encryption key e = 7.

Why? $(p-1)(q-1) = 360$ and $e = 3$ and $e = 5$ are not relatively prime to 360 (i.e., $gcd(360, 3) = 3 > 1$ and $gcd(360, 5) = 5 > 1$). Therefore, the next smallest prime is 7.

Answer: Public key $(N, e) = (403, 7)$.

Quiz: Private Decryption Key

What is the private decryption key **d**?

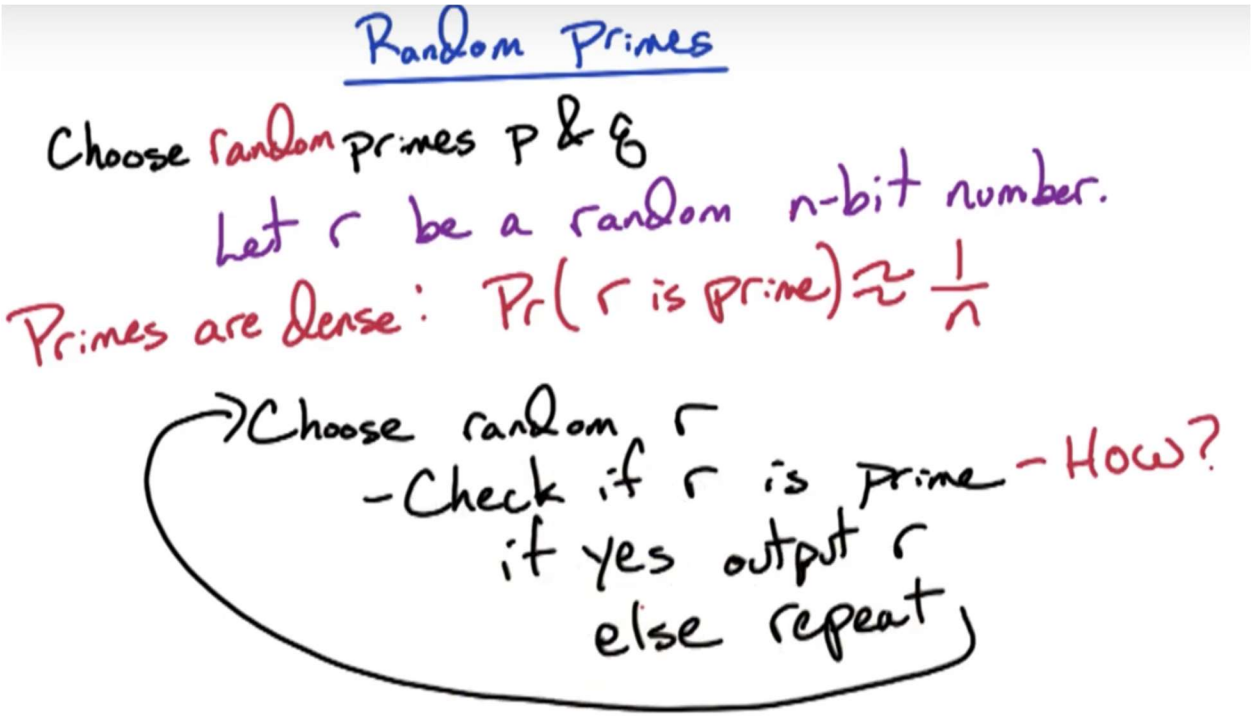
The matching decryption key $d = 103$.

Why? $103 \times 7 \equiv 1 \bmod 360$ so 103 is the multiplicative inverse of $7 \bmod 360$, i.e., $7^{-1} \equiv 103 \bmod 360$.

This can be found by running the Extended Euclid algorithm; this was the example given in the lecture about the Extended Euclid algorithm.

***

(Slide 16) Random Primes



Now, we want to choose a pair of primes p and q.

One way to do that is to have a table of prime numbers, and then just go through that table of prime numbers. What's the problem with that? Well, if somebody has access to our table, then it'll be easy to crack our cryptographic scheme. So, we want something more secure.

So what's the better approach? A better approach is to choose random primes p and q, and we want these primes p and q to be chosen so that every time we run the algorithm they are being chosen independently from the previous runs.

So, how do we choose these primes at random? Well, first off, how do we choose random numbers? That's what we're going to do first. Let r be a random n-bit number. How do we choose r? Let's say, little n is six, actually in practice though, little N is going to be a huge number like a 1000 or 2000. How do we generate this random 6-bit number? We have a one dimensional array of size 6 and we generate each of the bits. For each bit, we choose a random number 0 or 1. And, we make sure that every bit is generated independently of every other bit. And, then every time we run the algorithm, the bits are generated independent of previous runs of the algorithm. So, this is a quite easy to generate random N-bit number, but we want a random N-bit prime number. So what do we do?

We choose a random n-bit number regardless of whether it's prime or not. And then we check whether this random number is prime or not. How do we do that? We'll see how to do that in a moment.

But, suppose we have a test for primality – so, given a number, we can check whether it's prime or not. Then, if this random number happens to be prime then, what do we know? We know it's a random prime number. So in that case, if r is prime, we can output it because it is a random prime number. What do we do if it's not a prime number? Then we repeat this procedure. We generate a new random number, check whether it's prime or not.

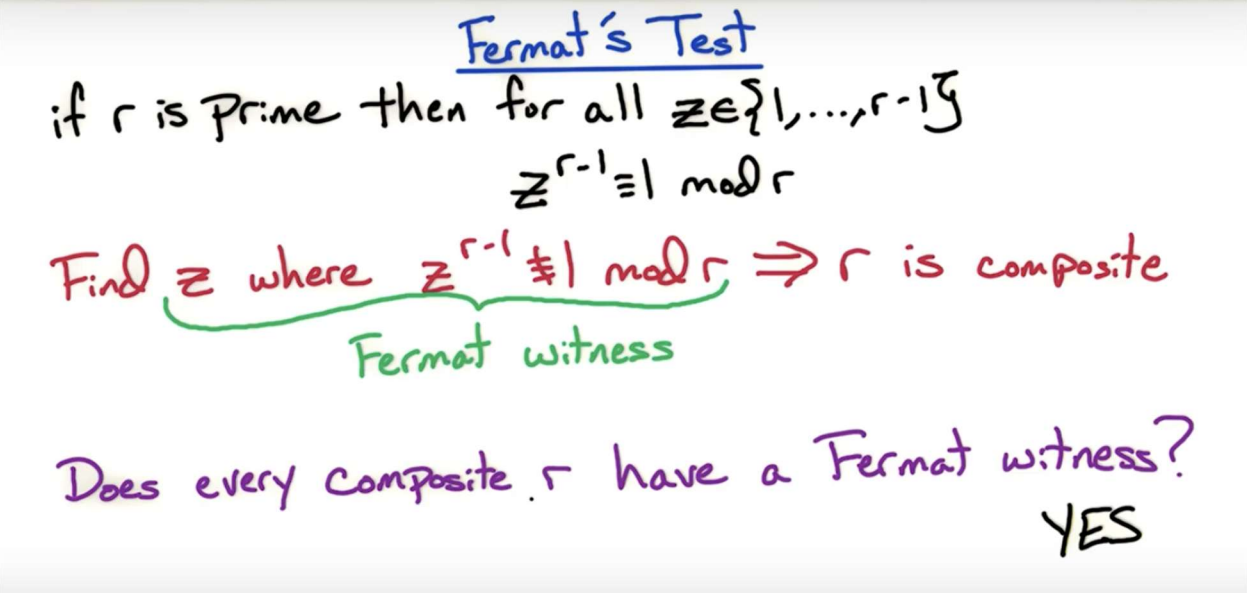
How long is this algorithm going to take before it finds a prime number? Well, the key thing is that primes are dense. What do we mean more precisely? -- the probability that this random number r happens to be a prime number is roughly 1/n, where n is the number of bits in it.

So, for generating a 1000 bit number or a 2000 bit number, the probability that it's going to be prime is about 1/1000 or 1/2000 -- which means how many times are we going to have to repeat this procedure before we find a prime number? In expectation, it is going to be about little n -- it's going to be about a 1000 or 2000 times -- which is not a big deal … to repeat this a 1000 or 2000 tries before we find a prime number.

But, the question remains: how do we check whether a given number r is prime or not?

***

(Slide 17) Primality: Fermat's Test



Now, we're going to derive a Primality Testing Algorithm using Fermat's Little Theorem.

Let's first recall Fermat's Little Theorem. If r is a prime number, then for all z in this set, all z between 1 and r-1, if we look at z^{r-1} mod r, then it's congruent to 1: z^{r-1} mod r ≡ 1. Now, we're going to use this as the basis for figuring out whether a given number r is prime or not. So, if it's prime, it satisfies this.

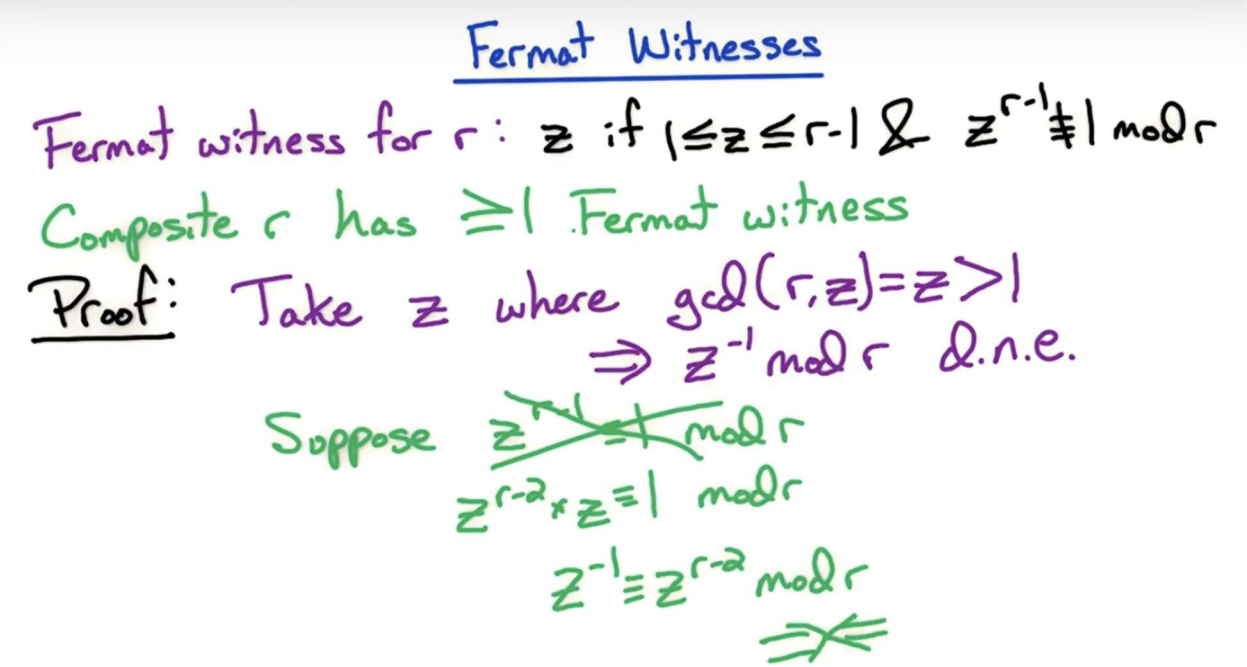
What if it's composite? Well, what if we find a z between 1 and r-1, where z^{r-1} mod r !≡ 1 mod r? Well, by Fermat's Little Theorem, if this is the case, then what do we know about r? -- r must be composite if there's such a z. So, this z where z^{r-1} !≡ 1 mod r is a witness. It proves that r is composite. And hence, we call this z a **Fermat witness** because it's a witness with respect to Fermat's Little Theorem.

The first question we want to ask is whether every composite number has a Fermat witness. The answer to this first question is yes, it does -- Every composite number has at least one Fermat witness and we'll demonstrate this in a moment. After we demonstrate this, then we have to look at how we find a Fermat witness. After we show that every composite number has at least one Fermat witness, then we'll look at how to find Fermat witnesses. What we will show is that every composite number has many Fermat witness. Well, there are some exceptions called pseudo-primes; but, if we ignore pseudo-primes, then every other composite number has many Fermat witness. So, in fact it will be easy to find Fermat witnesses for composite numbers.

And that will give us our Primality Testing Algorithm. And then we'll see how to deal with these pseudo-primes. Let's first prove this fact that every composite number has at least one Fermat witness.

***

(Slide 18) Trivial Witness



Once again, we have a number r and we're trying to determine if r is prime or composite.

Now we say number z is a Fermat witness for r. First off, z lies between 1 and r-1. And, crucially, $z^{r-1} \not\equiv 1 \bmod r$. Then by Fermat's little theorem, we know that r is composite because if R is prime, for every such Z, $z^{r-1} \equiv 1 \bmod r$. So, if we get a z where this is not true then this r must be composite. So z is a witness to the fact via Fermat's little theorem that r is composite.

We want to first prove that every composite number has at least one Fermat witness. So let's prove this fact.

> (proof) Now, r is composite so we know it has at least two divisors. So, take a z which is a divisor of R. So, clearly this z, which is a divisor of r, lies between 1 and r-1. What else do we know about this z? Well, look at its gcd.
>
> What do we know about gcd(r,z)? Well, if we take z to be a divisor of r, then we know the greatest common divisor of these two numbers is z itself. And since R is composite, we know this divisor is greater than one -- Well, actually, this proof works for any z where this statement (gcd(r,z) = z) is true. So, for any z which is not relatively prime to r (note that if r is composite, we always know there is at least two such z's which are not relatively prime to r) -- so if gcd(r,z) > 1, what do we know? -- well, if you recall from our

previous lecture about modular arithmetic, what about z^{-1} mod r? We know it does not exist. There is no inverse of z mod R. Why? Because the inverse of z mod R exists if and only if gcd(r,z) = 1 -- they're relatively prime to each other.

Now, we want a proof for this z that $z^{r-1} \not\equiv 1 \bmod r$. What are we going to do? Let's suppose that $z^{r-1} \equiv 1 \bmod r$ -- That's the opposite of what we're trying to prove -- So, what will show that if this holds, then we get a contradiction. How will we get a contradiction? We will show that z has an inverse mod r. And we know that's not the case; therefore, we will have a contradiction -- therefore this assumption cannot hold, and therefore $z^{r-1} \not\equiv 1 \bmod r$. Okay?

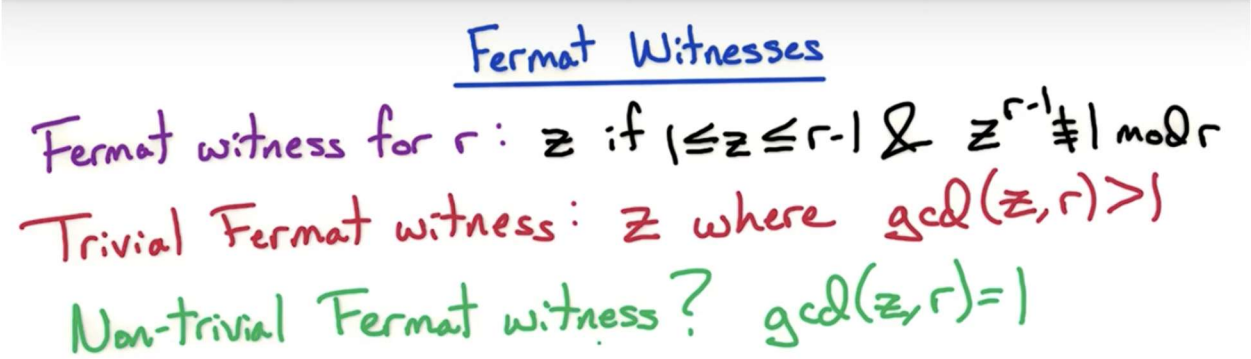
Now, we have the fact that $z^{r-1} \equiv 1 \bmod r$. Now, z^{r-1} is the same as the following: z^{r-1} is the same as $z^{r-2} \times z$. So, if $z^{r-1} \equiv 1 \bmod r$, then $z^{r-2} \times z \equiv 1 \bmod r$. Now, look at this statement. $z \times z^{r-2} \equiv 1 \bmod R$. That means that z has an inverse. What's its inverse? It's z^{r-2} – because, when we look at the product of these two, it's 1 mod R. That's the definition of the inverse.

So z^{r-2} is the inverse of z and z is the inverse of z^{r-2} – but, we know that z does not have an inverse mod r, therefore we have a contradiction.

And thus, the assumption cannot hold and therefore we know that $z^{r-1} \not\equiv 1 \bmod r$. And that proves that every composite R has at least one Fermat witness.

***

(Slide 19) Non-Trivial Witness



We call this Fermat witness that we just derived a trivial Fermat witness. Why is it a trivial Fermat witness? -- this is a Fermat witness, z, where it also has the property that gcd(z,r) > 1. So, z and r are not relatively prime.

Notice, having such a z where gcd(z,r) > 1, already proves that r is composite. If z and r have a non-trivial divisor, then we know that r has a non-trivial divisor, and therefore, it's not prime. So, any z, where this is the case, proves that r is composite -- there's no reason to run Fermat's test. So that's why we called such a z a trivial Fermat witness.

Now there always exists a trivial Fermat witness for composite numbers. Why? Because every composite number has at least two non-trivial divisors. The question is whether a composite number r has a non-trivial Fermat witness. This is a number z which is relatively prime to r.

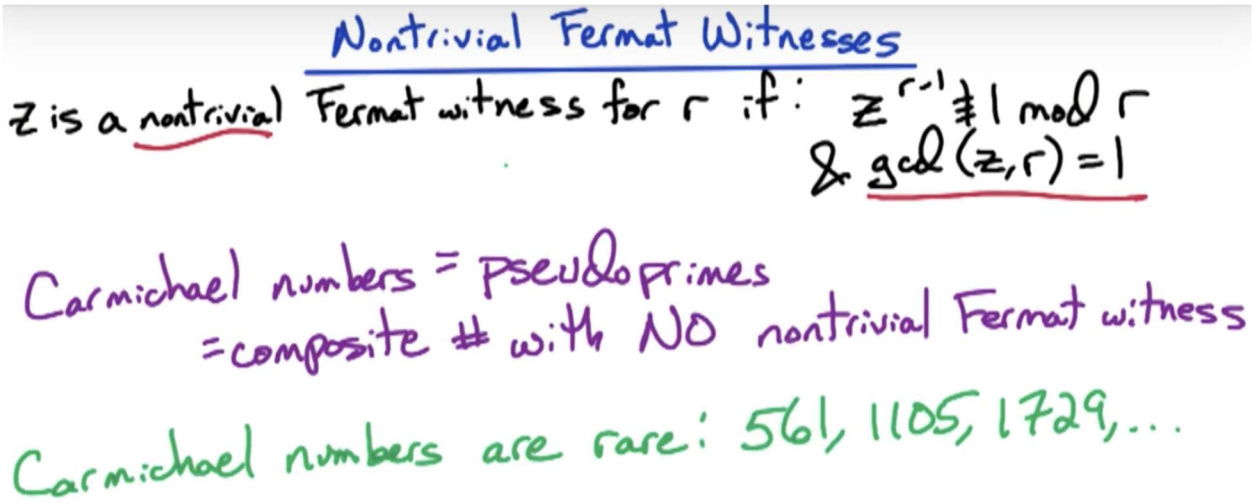
Now, it turns out that some composite numbers have no non-trivial Fermat witnesses -- these are called pseudo-primes, but those are relatively rare. For all the other composite numbers, they have at least one non-trivial Fermat witness, and, if they have at least one, then in fact, they have many Fermat witnesses.

And, therefore it will be easy to find a Fermat witness and that's the key point. Trivial Fermat witnesses always exist. Every composite number has at least two trivial Fermat witnesses. But if a composite number has a non-trivial Fermat witness, then there are many Fermat witnesses; they're dense, and, therefore, they're easy to find.

And that's what we'll utilize for our primality testing algorithm.

***

(Slide 20) No Non-Trivial Witnesses?



Recall, for a number r, if we're checking whether r is prime or not, we say that z is a Fermat witness if it proves that r is composite, according to Fermat's little theorem. What does that mean exactly? That means that if $z^{r-1} \not\equiv 1 \bmod r$, then that z proves that r is composite.

> Now, we say it's **non-trivial** if, in addition, gcd(z,r) = 1 -- So, z and r are relatively prime. If gcd(z,r) > 1 -- they're not relatively prime to each other -- then that z gives us a non-trivial divisor of r, and therefore we know already, trivially, that r is composite.

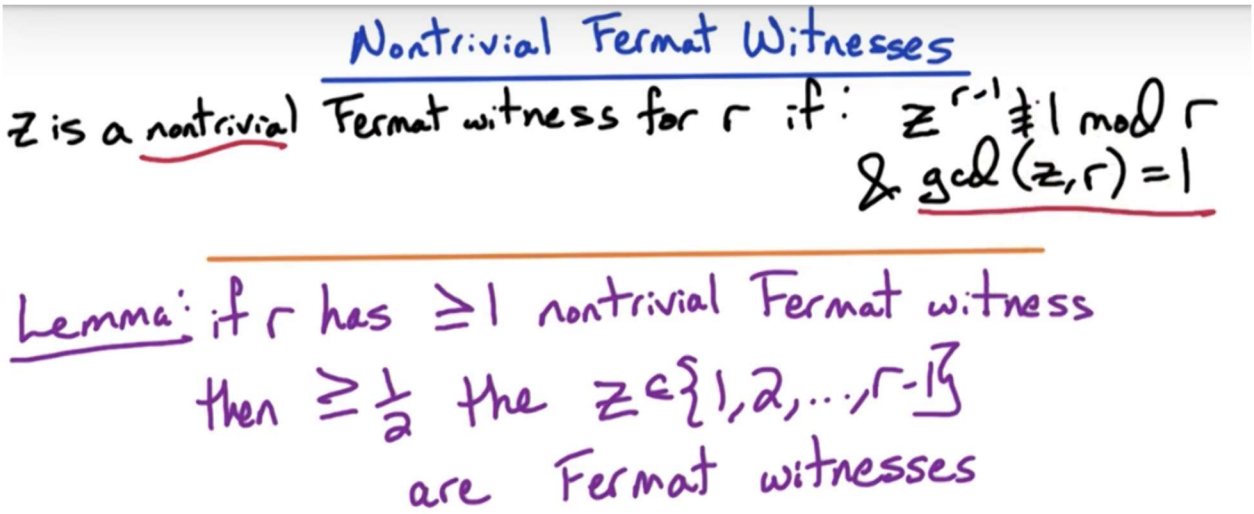
Now, the question is, does every composite number have a non-trivial Fermat witness? We know it has a trivial Fermat witness -- namely, if one of its divisors is a trivial Fermat witness. Does, it have a non-trivial Fermat witness?

Well, in fact, there are composite numbers which do not have non-trivial Fermat witnesses. These are called **Carmichael Numbers**. These are sort of pseudo-primes -- They behave like primes with respect to Fermat's little theorem. A Carmichael number is a composite number r, which has no non-trivial Fermat witnesses. Therefore, such a number is going to be inefficient to use Fermat's test, to prove that r is a composite number. We're going to have to find a different way to deal with Carmichael numbers. But, the key thing is that Carmichael numbers are rare. The smallest one is 561 and 1105 and so on.

Since they're relatively rare, let's ignore them for now, and when we ignore Carmichael numbers, we're going to have a simple algorithm to check whether number r is prime or not, using Fermat's test and trying to find a Fermat witness when it's composite.

***

(Slide 21) Primality: Many Witnesses



So, let's ignore Carmichael numbers, for now, since they're relatively rare. Let's assume that every composite number r, has at least one non-trivial Fermat witness.

If we assume that it has at least one non-trivial witness, how many witnesses does it have, actually? How prevalent are these non-trivial witnesses? The key fact, or Lemma, is that if r has at least one non-trivial Fermat witness, then there are many non-trivial Fermat witnesses. In fact, if we look at the set of possible witnesses, 1 thru r-1, then at least half the numbers in this set are Fermat witnesses.

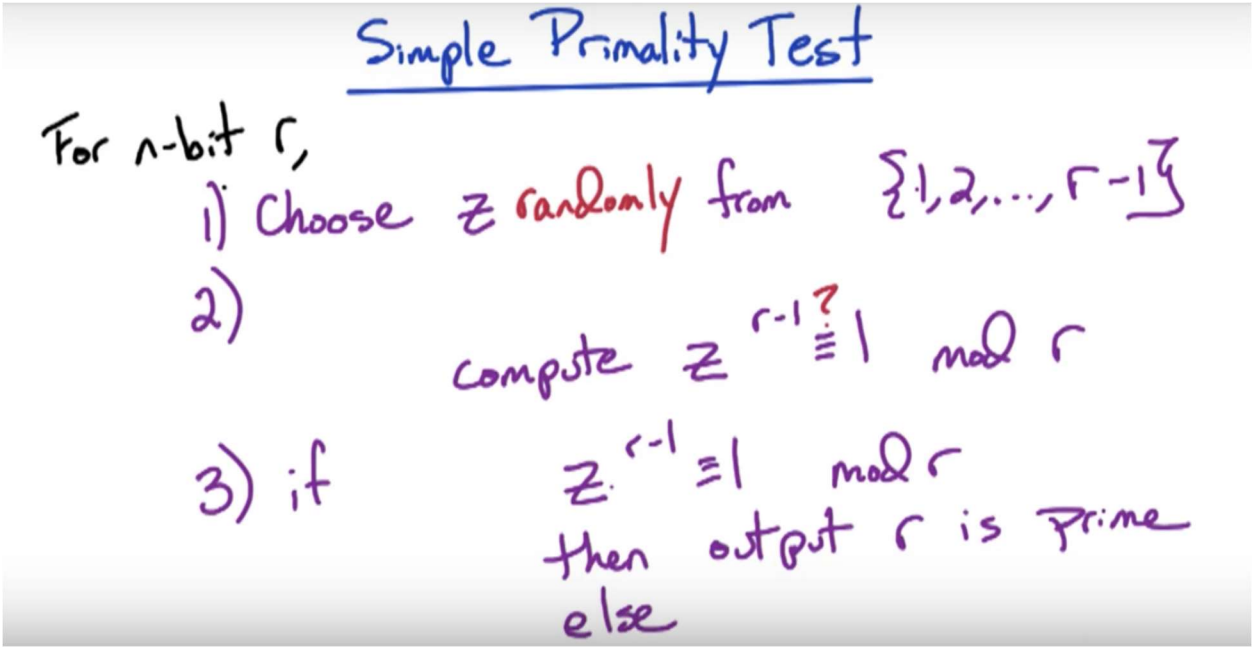
Now, this is a relatively simple fact to prove and it's a very nice proof. The proof is in the book. But let's get the proof for now and let's try to use the Lemma to have an algorithm for checking whether a number is prime or not … when we ignore Carmichael numbers.

> (proof) So, we assume that a composite number has at least one non-trivial witness. Then the Lemma tells us that at least half the numbers in this set {1,2,…,r-1} are Fermat witnesses -- are witnesses to the fact that r is composite.
>
> Can we use this fact to get a test for whether r is composite or not? The point is that when r is prime, all of these numbers, when we raise them to the power r-1 are congruent to 1 mod r. And when r is composite, then at least half these numbers, when we raise them to the power r-1, are not congruent 1 mod r.
>
> So how are we going to check whether a number is prime or not? -- we're going to take

a z from this set {1,2,..,r-1}, raise it to the power r-1, look at mod r, and see whether it's equal to 1 or not.

***

(Slide 22) Simple Primality Alg.



We have an n-bit number r, and here's our simple algorithm for checking whether r is prime or not based on Fermat's Little Theorem. And this is ignoring Carmichael numbers. It will be an efficient algorithm for primality testing.

(Step 1) Recall, that if r is composite then, at least half these numbers in this set are witnesses to that fact using Fermat's Little Theorem. Whereas, if r is prime, then none of these numbers in this set, {1,2,…,r-1}, are witnesses to the fact because r is prime. So, what are we going to do? We're going to choose a z from this set. How do we choose a z from this set? Well, we choose a z randomly, uniformly at random from this set. So, we choose a random number between one and r-1.

(step 2) Now, we do Fermat's Test for this z. So, we compute z^{r-1} mod r. and

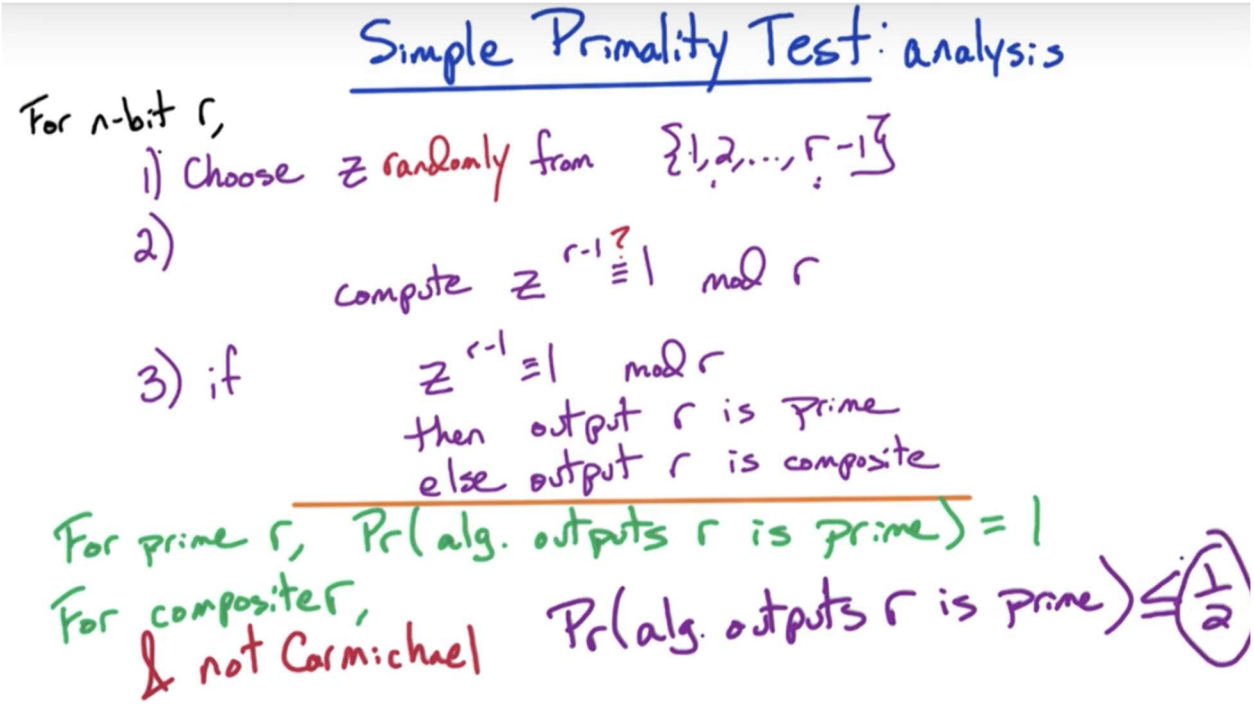
(step 3) we check whether that's congruent to one, or not:

- If $z^{r-1} \equiv 1 \mod r$, then what do we know? Actually, we don't know for sure either way, but we think that r is prime.

- What happens if $z^{r-1} \not\equiv 1 \mod r$? Then, in this case, we're sure that r is composite because this z is a witness to the fact that r is composite.

***

(Slide 23) Primality Alg. Analysis



Let's take a look at how this simple Primality testing algorithm performs.

If r happens to be prime, what does the algorithm do? It always outputs that r is prime, because for every z in this set, $z^{r-1} \equiv 1 \mod r$, by Fermat's little theorem. So, the probability that the algorithm outputs that r is prime, is 1, this is always going to do so. So, it's always correct when r is prime.

Now, what happens when r is composite? And let's assume that r is not a Carmichael number. Now, sometimes the algorithm is going to be correct, it's going to output that r is composite. When is that the case? -- when it finds a z which is a Fermat witness. So $z^{r-1} \not\equiv 1 \mod r$.

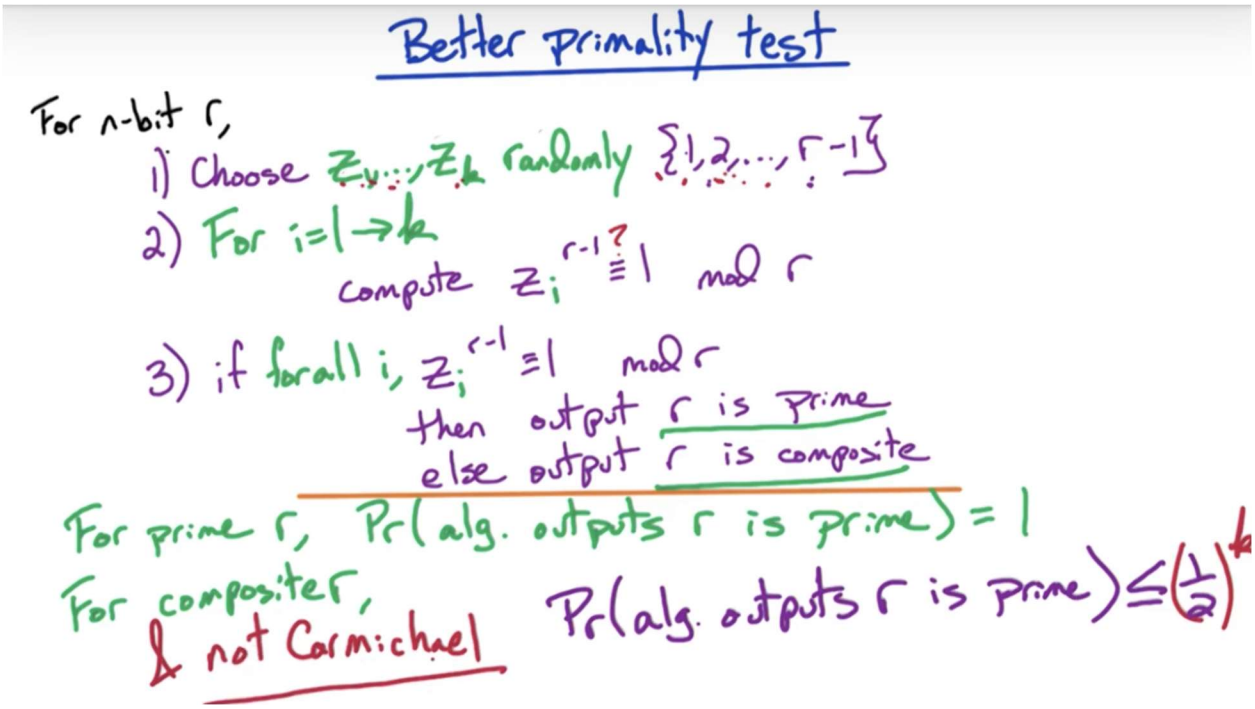
But sometimes it's going to get confused -- it's going to make a mistake -- and it's going to find a z, where $z^{r-1} \equiv 1 \mod r$. So it's going to think that r is prime.

> What is the probability that the algorithm outputs that r is prime? -- so, it makes a false positive statement? Well, we know that at least half the z's in this set are Fermat witnesses. So, what's the chance that it finds a non-witness? Well, at most half of them are not witnesses, so therefore, the probability of finding a non-witnesses is at most a half. So, the probability of a false positive is, at most, 1/2.

Okay, so, we have a reasonable algorithm with probability $\leq 1/2$ that we get a false positive, and when it is prime, it's always correct. But can we do this better? Can we improve this error probability of a false positive?

***

(Slide 24) Boosting Success



Here again is our original primality testing algorithm. The problem was that it had a false positive rate of at most 1/2. We want to improve that false positive rate. We want to get it down smaller. So, what are we going to do?

Recall the algorithm starts by choosing a random number between 1 and r-1, and then we run Fermat's test for that z that we chose at random. Now, what we're going to do is that instead of choosing one number at random from this set, {1,2,…,r-1}, we're going to choose k numbers at random from this set, where k is a parameter that we're going to choose.

Now, for all of these k numbers, we're going to run Fermat's test. So we're going to take z_i and we're going to raise it to the power r-1 and we're going to check mod r (i.e., check whether it's congruent to 1 or not).

Now what we need is just one of these z_i zi's to be a Fermat witness. If any of these zi's is a Fermat witness - so, if $z_i^{r-1} \not\equiv 1 \bmod r$ - then we know for sure that r is composite. Whereas, if all of these z_i's are not witnesses, then we have a very good chance that r is prime.

So our final check is whether any of these zi's is a witness or not. So, if, for all i's, it passes the test -- so $z_i^{r-1} \equiv 1 \bmod r$ -- then what do we know? We know there's a good chance that r is prime. And, if any of these z_i 's is a witness then what do we know? Then we know for sure

that R is composite.

More precisely, let's look at our analysis from before.

Suppose that r is a prime number, what does the algorithm do? It always outputs that r is prime because for every zi, $z_i^{r-1} \equiv 1 \bmod r$, by Fermat's little theorem. So, the probability that the algorithm outputs r is prime is 1, when r is, in fact, prime.

What happens when R is composite but not Carmichael? The previous algorithm had a false positive with probability at most 1/2, because at most half of the z's are non-witnesses. Now, we're choosing k z's -- We just need that one of them is a witness. What's the probability that none of them are witnesses?

> So, think of the following analogy: say it's a witness, if I flip a coin and it's heads, and if it's tails, it's a non-witness. So, the probability of a tails for each of these k flips is at most 1/2.
>
> So, suppose it is fair coin, what's the chance that I have k tails? That means that all K of these are non-witnesses. If I have just one head, or at least one head (then I have a witness). What's the chance that none of them are witnesses? So the chance that all k of them are tails? -- that's at most $(1/2)^k$. So the probability of a false positive in this scenario is at most $(1/2)^k$.
>
> So if I choose k to be, let's say,100. So, I choose 100 numbers at random from this set -- this is a huge set -- this is about on the order of 2^{1000}, or 2^{2000}. So, choosing 100 numbers from there, running this procedure 100 times is not much of a cost. Then the probability of a false positive is at most $(1/2)^{100}$, which is a tiny, tiny probability. So there is a very minuscule chance of that.

So I'm willing to take that chance of a false positive -- in which case my scheme, my cryptographic scheme might be easy to break -- the chance of that is very minuscule.

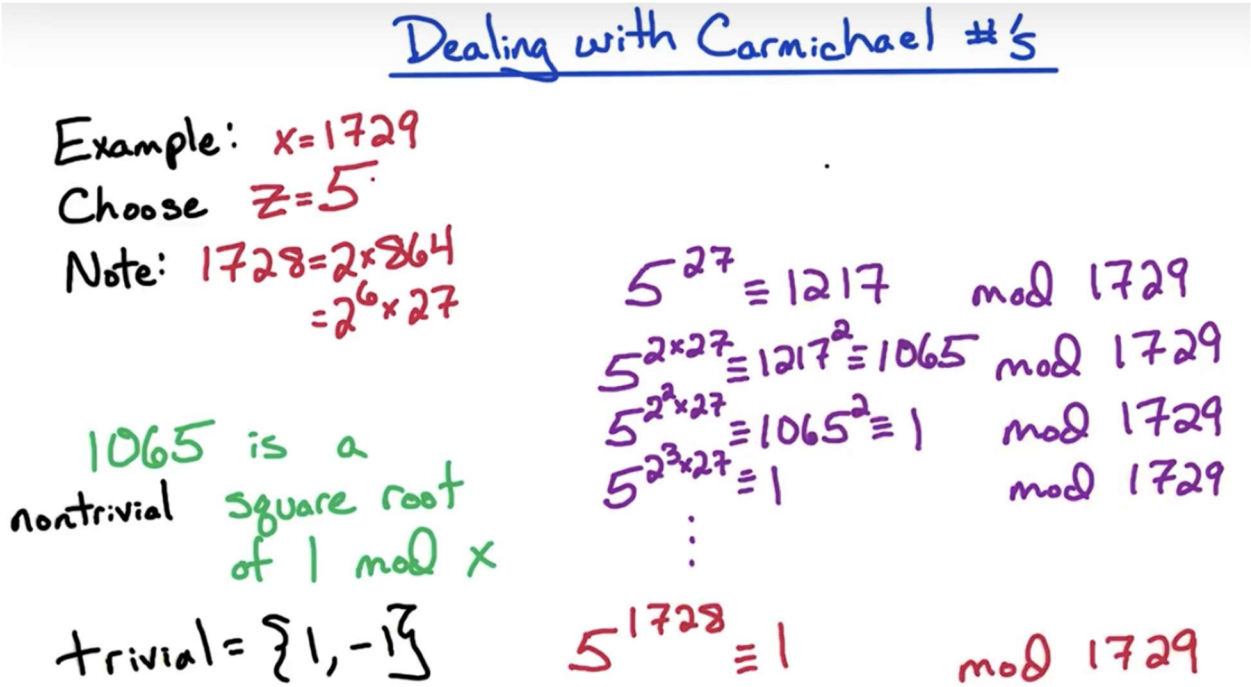
So that completes our primality testing algorithm in the case when we assume that the composite numbers are not Carmichael numbers. So we're ignoring these pseudo-prime numbers.

It turns out that, actually, to deal with these pseudo-prime numbers, Carmichael numbers, it's

not that much more complicated of an algorithm. That's detailed in the textbook and I'll leave that to there.

***

(Slide 25) Addendum: Pseudoprimes



Here is the general algorithm for primality testing which handles Carmichael numbers.

The algorithm is essentially the same as before with one small observation. I'll explain the algorithm using an example.

Let's consider the example of x = 1729, this is a composite number, in fact, it's a Carmichael number. So our previous algorithm is unlikely to detect that is composite. Now, let's first recall our previous approach:

- We first choose a random number z between one and 1728. For concreteness and simplicity, let's suppose that z =5 (I chose a small number to make it simpler).
- Now, our previous approach takes these numbers z, which is 5, and we raise it to the power of 1728, which is x-1, and we take that mod 1729, which is x.
- So, we look at z^{x-1} mod x. Now, if this is not one, then we have a proof that this number is composite. Well, since this is a Carmichael number, this is unlikely to work and in fact it doesn't work: $5^{1728} \equiv 1 \bmod 1729$. So, Fermat's test fails in this case.

Well, let's go back and look at how we compute five raised to this power: 5^{1728}. Well of course, we do use repeated squaring. Now, in the spirit of repeated squaring, let's take this number 1728 and let's take out all the factors of two that we can.

Now, notice that this number is even. Why? -- this number x, we're checking whether it's prime or not -- So, it's odd and therefore x-1 is even:

$1728 = 2 * 864$

we take out another factor of two and repeat:

$1728 = 2 * 864 = 2^2 * 432$

We can do it six times, so we get

$1728 = 2 * 864 = 2^2 * 432 = 2^6 * 27$

We stop when we reach an odd number.

Let's start a new approach by computing 5^{27} mod 1729. Now of course, this exponent might be very large, so of course, we're going to use the fast modular exponentiation algorithm to compute it.

Suppose we ran it, and we computed this exponent -- it turns out it's congruent to 1,217:

$5^{27} \equiv 1217 \bmod 1729$

Now, let's apply repeated squaring six times to get to this result. So let's take this result and square it. So, we're computing five to the power 54, and we're doing this mod x. So we take the previous results squared, and that's congruent to 1,065 mod X:

$5^{2 \times 27} \equiv 1217^2 \equiv 1065 \bmod 1729$

then we take this previous result and we square it, and it turns out that 1,065 squared is congruent to 1 mod x:

$5^{4 \times 27} \equiv 1065^2 \equiv 1 \bmod 1729$

Now, we continue, of course, once we get 1, it's going to continue 1, so the next result will be 1^2 which is 1, of course, and we square it again and we're going to get 1 again. We do it a few more times, and eventually we get to 5 raised to the power, 2^6 x 27, and that will be 1, which we know from before, mod 1729

Now, let's look backwards. So, we end with 1 here, we get this string of 1s. Let's look at the first number which is different from 1 -- $5^{2 \times 27}$ mod 1729 == what do we know about it? In this case, it's 1,065 but 1,065 squared is 1 mod x. Since $1065^2 \equiv 1$ mod x, therefore, in fact, 1065 is what we call a non-trivial square root of 1 mod x.

Why is it non-trivial? What are trivial squared roots of one mod x? Well, the trivial square roots of 1 mod x are 1 and -1. Why? Well, we always know that $1^2 = 1$ and $(-1)^2 = 1$. That's true in real arithmetic and it's also true in modular arithmetic. So, every number x has these two trivial square roots of 1 mod x.

It turns out that prime numbers x only have these two square roots of 1 mod x. So, 1 and -1 are the only two square roots of 1 mod x when x is prime.

But, if we can show a non-trivial square root of 1 mod x, then therefore, that implies that x is composite because prime numbers only have the trivial ones. So, if we show a non-trivial one, that proves that x is composite.

In this case, we've shown that this number x has a non-trivial square root of 1, namely 1,065. It turns out that for a composite number x, even if it's Carmichael, for at least three quarters of the choices of z, this algorithm works. Namely, if we compute z^{x-1} mod x and if we get 1, and we go backwards in this approach -- so we use this repeated squaring, and then we work backwards from the 1 to the first non-1, this leads to a non-trivial square root of 1 mod x, and therefore, proves that x is composite and this works for at least three quarters of the choices of z.

Now, the mathematics for proving that at least three quarters of the choices of z work is quite complicated but the algorithm itself that we're using here is basically the same as before with the repeated squaring added in.

So, to deal with Carmichael numbers, we use basically the same algorithm as before, except when Fermat's test fails, we go back and we check whether we get a non-trivial square root of 1 mod x.