The RSA crypto systems is one of the lovely and really important applications of number theory in computer science. So let's start talking about it.

The RSA crypto system is what is known as a public key cryptosystem, which has the following really amazing properties-- namely, anyone can send a secret encrypted message to a designated receiver. This is without there being any prior contact using only publicly available information.

Now, if you think about that, it's really terrific because it means that you can send a secret message to Amazon that nobody but Amazon can read even though the entire world knows what you know and can see what you sent to Amazon. And Amazon knows that it's the only one can decrypt the message you sent.

This in fact is hard to believe if you think about it. It sounds paradoxical. How can secrecy be possible using only public info?

And in fact, the existence of this public key cryptosystem has some genuinely paradoxical consequence, which kind of are a mind bender. So let me tell you about one of them.

I don't know if you've heard of mental chess, but it's a standard thing in the chess world. Chess masters are so talented and have such deep insight into the game that they don't need a chessboard, and they don't need chess pieces. They can just go for walk on a country lane talking to each other and saying pond to king 4 and knight to bishop 3 and just talking chess code and play an entire chess game that way.

That's known as mental chess. It's quite impressive. In fact, the grand masters can play multiple games of mental chess against opponents who are staring at the chessboard and win the great majority of the games. Of course, these are not against other grand masters, but still.

OK. So now, this is what I propose. How about playing mental poker? If you know how to play poker, we deal our cards and we bet and so on. And my only condition is that I'll deal.

Now, that sounds like a joke and an absurd thing for you to agree to do, but it's amazing. It's actually possible.

One of the famous papers of Rivest and Shamir was how to play mental poker using public key crypto. So I once tried to persuade an eminent MIT dean who's a physicist researcher about this, and he just wouldn't believe it. He argued that it was just impossible logically.

And what he was thinking about was that if you know how to compute a function, then of course you can figure out

how to invert it. That is to say if I know how to compute some function f of a number and let's say that the function is one arrow in-- that is an injection-- then if I know what f of n, there's a unique n that it came from. So how can I not be able to find n?

And it's an insight of computer science and complexity theory that says it's quite possible. It's not that you can't find the n that produced f of n. It's that the search for it will be prohibitive. There are, in short, one-way. That is, functions that are easy to compute in one direction but hard to invert. They're easy to compute but hard to invert.

In particular, we're thinking about multiplying and factoring.

It's an observation that it's easy to compute the product of two large prime numbers. We all know how to multiply. And in fact, there are faster ways to multiply than you know.

But the current state of our knowledge of number theory and complexity theory is that given a number n that happens to be the product of two primes, it seems to be hopelessly hard in general to factor n into the components p and q.

Now, this is an open problem. It's similar to the p equals np question-- that famous open problem. It's actually a weaker-- it's quite possible that you could factor, and np would not equal to np. But nevertheless, it's the same kind of problem. And more generally, the existence of one way functions is closely related to that p equals np question.

Nevertheless, even though it's an open problem and theoretically has not been settled either way, it's widely believed-- the banks, the governments, and the commercial world have really bet the family jewels on the difficulty of factoring when they use the RSA protocol.

So I like to make the joke that my most important contribution to MIT was being involved in the hiring of our S and A. So this is A, Adi Shamir, R, Ron Rivest, and A, Len Adleman back in the late '70s when they first came up with these ideas.

So let's look at the way this RSA protocol actually works.

So here's what happens. To begin with, you have to make some information public so that people can communicate with you. We're looking at two players here. There's a receiver who's going to get encrypted messages, and there's a sender who is trying to send an encrypted message to the receiver.

So what the receiver does before hand is generates two primes, p and q. Now, in practice, you want these to be pretty big primes-- hundreds of digits. And we'll examine it in a moment, the question of how you find them.

But the receiver's job is to find two quite substantial large primes, p and q, chosen more or less randomly because if you have any kind of predictable procedure for how you got them, that would be a vulnerability. But if you just choose them at random, then there's enough primes in the hundreds of digits that it's hopeless that people would guess which one you wound up with.

OK. What do you do to begin with is multiply p and q together, which is easy to do. Let's call that number n.

And now the other thing the receiver is going to do is find a number e that's relatively prime to this peculiar number p minus 1, q minus 1. Now as a hint, you might notice that p minus 1, q minus 1 is in fact Euler's function of n-- phi of n. But for now, we don't need to understand that this is Euler's function. It's just the recipe of what the receiver has to do.

Find a number e that's relatively prime to p minus 1, q minus 1. Again, you don't want e to be too small, and we'll discuss in a moment how do you find such an e. But the receiver's job is to find such an e.

This pair of numbers e and n will be the public key which the receiver publishes widely where it can easily be found by anyone who cares to look for it. Basically there's a phone directory where if you want to know how to send somebody a secret message, you look them up, and you find the receivers name in there. And then you see his public e and n, and that's what you use to send him a message.

Now, how do you use it to send him a message? Well, I'll explain that in a minute, but let's look at one more thing that the receiver needs to do to set himself up.

The receiver is going to find an inverse of this number e that he's published-- the part of his public -- modulo p minus 1, q minus 1. That is, this e since it's relatively prime to p minus 1, q minus 1, it will have an inverse in Z star p minus 1, q minus 1.

Let's let that inverse be d. And of course, we know how to find d because you can do that with a Pulverizer. D is the private key. That's this crucial piece of information that the receiver has and that the receiver is not going to tell anybody.

Only the receiver knows that because the receiver chose the p and the q and the e more or less randomly-- maybe even as randomly as they can manage-- and then they find the d. And that's their secret. OK. That's what the receiver does.

How does the sender send a message? Well, to send a message, what the sender wants to do is choose a message that is in fact a number in the range from 1 to n where-- we're thinking again, of n, if it's a product of two

primes of a couple of hundred digits each, then the product is around 400 digits. And so you can pick any message m that can be represented by a 400 digit number.

Now, there's a lot of messages that will fit within 400 digits. And of course, if it's bigger, you just break it up into 400 digit pieces. So that's the kind of message you're going to send.

So the message is going to be a number in this range from 1 to n. And what the sender is going to do is look up the public key e and the other part of the public key n and raise the secret message to the power e in Z n.

So we're going to compute m to the e in Zn and send that encoded message m hat. So m hat is what we think of as the encrypted version of the message m.

So then we have the problem if that's what the sender sends to the receiver, how does the receiver decode the m hat, and the answer is the receiver just computes m hat to the power d-- the secret key-- also in the ring Zn. And the claim is that in fact, that's equal to m.

Now, you can check in class problem, and it's easy to see that the reason why that method of decrypting works is precisely an application of Euler's theorem-- at least when m happens to be relatively prime to n.

Now, the odds of finding an m that's not relatively prime to n are basically negligible because if you'd find such an m, it would enable you to factor them. And we believe factoring is very hard. But in fact, it actually works for all m, which is a nice theoretical results. And you'll work this out in class problem.

OK. That's how it works.

The receiver publishers e and n, keeps a secret key d. The sender exponentiates the message to the power e. The receiver simply decodes by raising the received message to the power d and reads off what the original was.

OK. So we need to think about the feasibility of all of this because we believe that it's impossible to decrypt, but there's a lot of other stuff going on there that the players have to be able perform. And let's examine what their responsibilities and abilities have to be.

So the receiver to begin with has to be able to find large primes. And how on earth do they do that? Well, without going into too much detail, we can make the remark that there are lots of primes. That is to say by appealing to the prime number theorem, we know that among the n digit numbers, about log n of them are going to be primes so that you don't have to go too long before you stumble upon a random prime. That is, if you're dealing with a 200 digit n and you're searching for a prime of around that size, you're not going to have to search more than a few hundred numbers before you're likely to stumble on a prime.

And of course, how do you know that you stumbled on a prime? Well, you need to be able to check whether a number is prime or not-- and efficientlY-- in order for this whole thing to be feasible. So we'll have to discuss that brieflY-- how do you test whether or not a number is prime in an efficient way?

The other thing the receiver has to do is find an e that's relatively prime to p minus 1, q minus 1. But that's easy. Well, it's easy because first of all, if you just kind of randomly guess a medium sized e and then search consecutively from some random number you've chosen somewhere in the middle of the interval up to p minus 1, q minus 1. Again, you're very likely to find in a few steps a number e that is relatively prime to p minus 1, q minus 1.

How do you recognize that it's relatively prime? Well, you just compute the GCD, which we know how to do using Euclid's algorithm. So that's really quite efficient. Recognizing that it's relatively prime is easy, you just don't have to search very many numbers until you stumble on an e. OK.

The other thing you have to do is find the d that an e inverse modulo p minus 1, q minus 1. And again, that is the extended Euclidean algorithm, the extended GCD, namely the Pulverizer.

So those are the pieces that the receiver has to do.

Now, let's look at this a little bit more and think about the information about the prime. So the famous theorem about the primes is their density, which is if you let a pi of n be the number of primes less than or equal to n, then it's a deep theorem of number theory that pi event actually approaches a limit in an asymptotic sense-- which we'll discuss in more detail-- that pi of n as n grows gets to be very close to n over log n. That's the natural log of n.

Now, that's a deep theorem. But in fact, if we want a self-contained treatment for our purposes, there's an exercise that will be in the text where we can derive Chebyshev's bound, which is weaker than they tight prime number theorem. But Chebyshev's bound, which can be proved by more elementary means that's within our own ability at this point with the number theory we have-- to be able to show that n over 4 log n is a lower bound on pi of n.

So basically that says that if you're dealing with numbers of size n, which means they're of length log n a few hundred digits, then you only have to search maybe 1,000 digits before your very likely to stumble on a prime. And if you search 2,000 digits, it becomes extremely likely that you'll stumble on a prime.

So the primes are dense enough that we can afford to look for them, providing we can have a reasonably fast way to recognize when a number is prime. Well, one simple way that it almost is perfect-- but works pragmatically pretty well-- is called the Fermat test.

But let me just reemphasize this -- I got ahead of myself-- that if I'm dealing with 200 digit numbers, then about one in 1,000 is prime using just the weaker Chebyshev's bound. And that says that I don't have to search too long-- only a few thousand numbers to be able to find a prime. And a few thousand numbers is well within the ability of a computer to carry out, providing that the test for recognizing that a number is prime isn't too time consuming.

So one naive way that the really almost works to be a reliable primality test is to check whether Fermat's theorem is obeyed.

Fermat's theorem-- the special case of Euler's theorem-- says that if n is prime, then if I compute a number a to the n minus 1, it's going to equal 1 in Z n. And that's going to be the case for all a that are not 0 if n is prime.

Now that means that if this equality fails in Z n, then I immediately know a is not prime. Go on. Search for another one.

OK. So suppose I'm unlucky-- or lucky-- and I choose an a to test and it turns out that a to the n minus 1 is 1, does that mean that n is prime? Unfortunately not. It might be that I just hit an n that happened to satisfy Fermat's equation even though n was not prime.

But it's not a very hard thing to prove that if n is not prime, then half of the numbers from 1 to n are not going to pass the Fermat test. So if half of the numbers are not going to pass the Fermat test, then what I can do is just choose a random nonzero number in the interval from 1 to n, raise it to the n minus first power, and see what happens.

And if n is not prime, the probability that this random numbers that I've chosen fails this test is at least a half. So I try it 50 times. And if in fact 50 randomly chosen a's in the interval 1 to n all satisfy Fermat's theorem, then there's one chance in 2 to the 50th that n is not prime. That's a great bet. Leap for it.

So that basically is the idea of a probabilistic primarily test.

Now, there's a small complication which is that there are certain numbers n where this property that half the numbers will fail to satisfy Fermat's theorem doesn't hold. They're known as the Carmichael numbers, and they're known to be pretty sparse. So that really if you're choosing an n at random, which is kind of what we're doing when we choose random primes p and q, the likelihood that you'll stumble on a Carmichael number is another thing that you just don't have to worry about.

So really, the Fermat primality test is a plausible pragmatic test that you could use to pretty reliably detect whether

or not a number was prime-- what was the last component of the powers that we needed the receiver to have.

OK. So now we come to the question of why do we believe that the RSA protocol is secure? And the first thing to notice is that if you could factor n, then it's easy to break. Because if you can factor n, then you have the p and the q. And that means you know what p minus 1 times q minus 1 is. And therefore you can use the Pulverizer in exactly the same way the receiver did to find the inverse of the public key e. You could find d easily.

So surely if you can factor, then RSA breaks. No question about that.

What about the converse? Well, what you can approve-- and there's an argument that's sketched in class problem, not fully, in the book-- is that if I could find the private key d, then in fact, I can also factor n.

So if I believe that factoring is hard, then in fact finding the secret key is also hard. And we could try to be confident that our secret key is not going to be found even given the public.

Now, unfortunately this is not the strongest kind of security guaranteed you'd like because there's a logical possibility that you might be able to decrypt messages without knowing the secret key. Maybe there's some other walk around whereby you can decrypt the secret message m hat by a method other than raising it to the dth power.

And what you'd really like is a theorem of security that said that breaking RSA-- reading RSA messages by any means whatsoever-- would be as hard as factoring. That's not known for RSA. It's an open problem. And so RSA doesn't have the theoretically most desirable security assurance, but we really believe in it.

And the reason we really believe in it is that for 100 or more years, mathematicians and number theorists have been trying to find efficient ways to factor. And more pragmatically, the most sophisticated cryptographers and decoders in the world using the most powerful networks of supercomputers have been attacking RSA for 35 years and have yet to crack it.

Now, the truth is that in the course of the 35 years, various kinds of glitches were found that required some added rules about how you found the p and the q and how you found the e, but they were easily identified and fixed. And RSA really is a robust public key encryption method that has withstood attack for all these years. That's why we believe in it.