[SQUEAKING]

[RUSTLING]

[CLICKING]

BRYNMOR CHAPMAN: OK, can people hear in the back? Are we good? Yeah, thumbs up? OK, nice one.

So you've already seen a couple of lectures on number theory now. And so some of you have been asking like, why are we doing this? Why is this in 6.1200? And today, we are going to see how GCD's and modular arithmetic can be very important for something that you use every day-- computer security.

So cryptography has been a very important subject for daily life for the last 20-odd years. Any time you do an online credit-card transaction or anything like that, that's using modern cryptography. We'll see some examples from history where it was quite important.

It's very much a computer science topic, not just math. But it does use all sorts of exciting mathematics, including number theory. So can anybody tell me, just in a nutshell, very informally, what is cryptography? Anyone? Yeah?

AUDIENCE: Insecure information over-- or sorry, secure information over an insecure channel.

BRYNMOR CHAPMAN: OK, secure information over an insecure channel. Yeah, that's a pretty good description. So I would say that it's the art or science of dealing with such, dealing with protecting information.

So the basic idea, well, at least for what we're going to be looking at today, is suppose you have an insecure channel and you want to send some message to a recipient, but you don't want other people who might be eavesdropping on your channel to know what you're doing.

So basically, you're going to want to garble a message that you send privately in such a way that only certain parties can read it. So let's take a look at what we mean. So the art or science of protecting information.

So in our setup, the convention is that we're going to have two parties, Alicekazam and then Bobsaur. And they're going to be trying to communicate over this communication channel, which may or may not be secure.

So maybe Alicekazam wants to send a message. It's Pi Day, so she's going to send a pie over to Bobsaur. So maybe she puts it in an envelope, pops it in the post, and Bobsaur receives it. He can open up the envelope and get the message.

So does that basic setup make sense to people? OK, so nothing too fancy yet. But what could go wrong? Can anybody tell me a problem with this? Yeah?

AUDIENCE: Somebody could come and steal it.

BRYNMOR CHAPMAN: Yeah, somebody could come and steal the message, right? So maybe we've got some eavesdropper, Eevee. And Eevee wants to eavesdrop on their conversation, wants to figure out what they're talking about.

So it's quite possible that maybe Alicekazam pops this in the mail, Eevee then opens the mail, puts it back in the mail, sends it along to Bobsaur, but now Eevee knows what the message is. So this is a problem, right? What else could go wrong? Any other ideas? Yeah?

AUDIENCE: Maybe they could send a message trying to be Alice.

BRYNMOR CHAPMAN: Yeah, what if Eevee is not just an eavesdropper but some meddling adversary? So maybe Bobsaur now wants to say thank you for the pie. Bobsaur puts this in an envelope, sends it over to Alicekazam. But Eevee's intercepted it along the way.

And so now, when Alicekazam gets it, it's not quite the same message as Bobsaur sent. So that's also a problem. We'd also like to prevent this.

So what can we do? Well, if we think about the actual postal system, we could basically put a seal on it, like an old-fashioned wax seal. But basically, Alicekazam and Bobsaur are going to share some key.

So they have a message that they'd like to share. It's supposed to be secret. So instead of just putting the envelope in the post, they're going to put it in a locked box. And Alicekazam will lock it with her key.

And now when she sends it over the channel, Eevee can't do anything with it. It's locked. Eevee doesn't have the key. But Bobsaur has the same key. So he can now remove the lock and open the message. And everything is awesome.

So key concepts in cryptography-- oops, that won't work. OK, so the basic idea-- we're trying to send private messages. So we're going to want to, what we call, encrypt a message. So this means just garbling it so that it looks like random noise so only parties who share your secret key will be able to read it.

Decryption is the reverse. You're going to take one of these garbled messages if you're the intended recipient. And hopefully you can ungarble it to recover the original message. But also, hopefully nobody else can do the same thing.

And in order to do both of these things, the two parties are going to share a secret key, which is some data that only they have. Nobody else should have it. And it should look random. And it should be hard for somebody else to derive or guess. Yeah?

AUDIENCE: In that analogy you used before, how do they get the keys without someone taking the key midway?

BRYNMOR CHAPMAN: That's a great question. So, yeah, they have to meet up and actually share a key. But then that allows them to share messages over this insecure channel later. But they do have to establish a secure way of sharing the key first.

So we won't be talking too much about how you can actually do that nowadays. But it is in the notes if you'd like to read it. OK, so a cryptographic scheme is basically just these things. It's an encryption algorithm paired with a decryption algorithm. And they should use the same secret key to invert each other.

So anybody give me examples, historical examples, of cryptographic schemes? Does anybody know what the first one was? Yeah?

AUDIENCE: Caesar cipher.

BRYNMOR CHAPMAN: Yeah, the Caesar cipher. So what was the Caesar cipher? How do you encrypt a message? Yeah?

AUDIENCE: You offset every letter in the message by, like, n letters after it.

BRYNMOR CHAPMAN: So the answer was you offset every message in the letter by n letters after it. Not quite. You are thinking of something very slightly different. The Caesar cipher was specifically using 3, not some key n.

So you take every letter, shift it over by three letters. So C becomes D-E-- F. And you shift every letter individually and then send the shifted message. And to decrypt it, you do the reverse. You subtract 3 modulo 26.

Does that make sense to everybody? Now, what you're thinking of is the Caesar shift, which is slightly better. So one problem with the Caesar cipher is you don't really have a key. It's kind of what we call cryptography by obscurity.

So the only reason it might be secure is if nobody else knows you're doing it. Unfortunately, if you're a Roman emperor, legion, or whatever, using the Caesar cipher, it doesn't protect against, say, maybe one of your generals defects and now goes to the enemy and is like, hey, they're using this cryptographic scheme. Here's how you can break all their messages.

And so then, from that point forward, nothing you send is secure. So one way we can fix that is by, as you said, introducing a key. So the Caesar shift is very similar. But instead of just shifting by 3, you shift by some secret key, k, modulo 26. Does anybody see a problem with this? Mhm?

AUDIENCE: It's pretty easy to just check all these things.

BRYNMOR CHAPMAN: Yeah, that it is. Oops, let's not capslock that. So here is a-- oh, that did not work. What is going on?

OK, so here's an encrypted message. It's been encrypted using the Caesar cipher-- sorry, Caesar shift. As you said, it's fairly easy to guess what the key is. So does anybody have any guesses before I start just putting in values? Yeah?

AUDIENCE: 13.

BRYNMOR CHAPMAN: 13. Why 13?

AUDIENCE: I just guessed the first word is "the." I just--

BRYNMOR CHAPMAN: [CHUCKLES] Yeah, that's a very good guess. So, yeah, knowing some information about the message. This is probably in English. You have a standalone N in that first line. That's probably an A or an I.

You've already got lots of information that you can use to narrow down the key. And even if you don't, like if we just start putting in random values here, most of them are going to look like garbage. But one of them is actually going to make sense.

So when we try all of the possible values, we're going to figure out something that is almost certainly going to be the correct key. So how can we avoid this problem? Any ideas? Yeah?

AUDIENCE: Change the encryption method based on the index of the letter.

BRYNMOR CHAPMAN: Change the encryption method based on the index of the letter. OK, so what you're describing is what we call the one-time pad. That is a good idea. We will not be covering it just yet. Yeah?

AUDIENCE: [INAUDIBLE] 26, [INAUDIBLE] number.

BRYNMOR CHAPMAN: OK, similar idea. Let's start by making a smaller, more incremental change. Yeah?

AUDIENCE: Maybe include numbers.

BRYNMOR CHAPMAN: OK, include numbers. Well, the idea that I had in mind was to just use a larger key space. Use a permutation instead of just a shift. So this is something that was done in medieval times. Instead of shifting every letter over by the same amount, you'd take a random permutation of the letters, and so you're kind of shifting each letter differently.

Now, can anybody think of a way that you could attack this? No, why is this not working? Sorry, technical difficulties. Oh, well, I guess it doesn't matter too much. I can describe what's going on.

So, yeah, what your classmate suggested was that, again, we can use the known plaintext attack that we described a moment ago with the Caesar shift. If we know something about our message, like, for instance, if we know that it's probably written in English, then we can count how many times each letter appears.

And whichever letter appears most frequently probably decrypts to E because E is the most common English letter. Or even if not, we've got a decent guess. It's probably not going to be like X or Z. It might be S or T or something like that. But the point is, it's very easy to start narrowing down the key space.

And once we have a guess, we can then start building on it. So similarly to what your colleague did earlier, if you have E and you see three-letter words ending with E, that might be "the." So then you've got two more letters.

Does that make sense to people? So knowing a little bit of information about your key, you can incrementally figure out the rest of the key and decrypt the message. Yeah?

AUDIENCE: So if you were to have a message like that, you could have the same letter-- or two different letters to the same letter?

BRYNMOR CHAPMAN: That's right. So it is a permutation. So it's a one-to-one map. It's a bijection. Other questions? Maybe I can just post this demo on the website so that you can play around with it. Sorry it didn't work.

So the next example comes from World War II. Has anybody heard of the Enigma cipher? Yeah, a few hands. Can anybody tell me what it did? Mhm?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: OK, so the answer was it changed the encryption method every day. That's not quite true. It changed the key every day. It didn't change the algorithm that they were using. Yes?

AUDIENCE: [INAUDIBLE] letter.

BRYNMOR CHAPMAN: Yeah, so it basically-- oh, so the answer was it changed the permutation every letter. So, yes, that's the basic idea behind Enigma.

It started with some permutation of the letters. And every time you encrypted a letter, it would then apply some change to this permutation so that the next letter would be encrypted differently. So that made it resistant to a frequency analysis.

But unfortunately for the Germans, it was still viable to do a known plaintext attack. So basically, the way that the Allies ended up breaking this encryption scheme was they knew something about the messages that were being sent every day. Many of them were weather reports or something. Many of them had the phrase "Heil Hitler" or something like that.

But if you know that you're getting a weather report from a certain location, you know the weather in that location, you've got a bunch of information about how this message might start. And so similarly to a frequency analysis attack, you can use this information to narrow down your key space and figure out the secret key.

And once you've got that, you can decrypt everything that's being sent that day. And so this is what the Allies were doing in Bletchley Park to win the war. So Enigma was kind of the last major example of what we-- or what I call cryptography by hubris.

Basically, the security relied on some German engineer being smarter than everybody else on the planet. He came up with this encryption scheme and was like, OK, this looks weird and random. I can't figure out how to break it. I bet nobody else can either.

So the Germans worked with this assumption for the duration of the war, and it was wrong. So nowadays we like to try and prove security based on more reasonable assumptions than that. So one example, which is in the notes that we won't be covering too much today, is called Diffie-Hellman. The one that we'll be focusing on, though, is RSA.

So RSA was-- it's a modern cryptographic scheme named after its inventor, Clifford Cocks. That's totally a lie. So Clifford Cocks did, in some sense, invent RSA. But he made the rather suspicious life choice of working for the government. So his work was classified for 20, 25 years, something like that.

And in the meantime, R, S, and A came along and invented his same scheme. They did it here at MIT. So this was Ron Rivest, Adi Shamir, and Leonard Adleman. They got all the credit for it. They won a Turing Award. And poor Mr. Cocks was kind of screwed.

So moral of the story, don't work for the government. Just kidding. But, yeah, kind of sucked for him. But, yeah, why don't we get into what RSA actually does? So RSA is an example of what we call a public key cryptosystem.

So before, I described what we actually call a private key cryptosystem, where both encryption and decryption use the same secret key. In a public key cryptosystem, that's no longer the case. So we actually have two keys. So let's call them KP and KS.

So KP is going to be a public key, which everybody can see. And then it's going to be paired with the secret key, KS, which you keep to yourself. And now the encryption and decryption algorithms, each of them uses a different key.

So we've got encryption. So our encryption algorithm is going to be E of some message and our public key. And this will output some ciphertext C.

And then we're going to have a different decryption algorithm, D, which will hopefully take this ciphertext C and, this time, the secret key. And it should output the original plaintext message, M.

So slightly different from the cryptosystem that I defined earlier. In that one, these two keys were the same. And so you can't publish KP. So only people who share the same secret key can actually communicate.

Here, this allows anybody to send a message. So you can think of this as, suppose Alicekazam wants to send a message to Bobsaur. So she's going to use Bobsaur's public key to send him a message.

You can think of this as Bobsaur's Gmail address. This is something that anybody can find. But then Bobsaur has his Gmail password, which he can then use to read the message.

OK, so now we are ready to actually define how RSA works. OK, so the basic idea behind RSA is that the encryption and decryption algorithm together should basically make use of Fermat's little theorem to make sure that we get back to the same message.

So we're going to try and come up with some e and d and some n such that if we exponentiate, first raised to the power of e, then raised again to the power of d, we should get back to where we started. So we're going to want for any m, m to the e, raised to the power of d, should be equivalent mod n to m.

So let's just start with correctness for now. Let's not worry too much about security just yet. But how could we achieve this? How could we choose n, e, and d such that this will at least work, even if it's not secure? Any ideas?

So what did Fairmont tell us? So Fairmont tells us that m to the p minus 1 is congruent mod p to 1. So how can we massage this into this? Well, first, how might we choose n? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Sorry?

AUDIENCE: Let e and d be factors of n.

BRYNMOR CHAPMAN: OK. So the idea was to let e and d be factors of n. So e times d equals n. Does anybody see a problem with this? Yeah?

AUDIENCE: Be like ed plus 1?

BRYNMOR CHAPMAN: So that's not the problem. So remember, the exponent here is p minus 1. So if you multiply both sides by m, you get m to the p is congruent mod p to m. So that is actually what we want. What assumption did we have on p? Yeah?

AUDIENCE: p is prime.

BRYNMOR CHAPMAN: Yeah, p is prime. So we can't do this. It is close, though. So maybe I should rewrite this as-- the suggestion was e times d should be equal to p minus 1 plus 1.

Does that make it any more clear how we can iterate on this? What other things would work here instead of p minus 1 plus 1? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Yes, that's right. Sorry. [CHUCKLES] So the answer was we can multiply this by whatever we want. any k here, it'll still work. As long as e times d is equivalent to 1, mod p minus 1-- so ed equivalent mod p minus 1 to 1-- as long as we satisfy this, we get this. Or, sorry, we get this. Yeah?

AUDIENCE: [INAUDIBLE] plus 1 or k times [INAUDIBLE]?

BRYNMOR CHAPMAN: So the question was like, what are we multiplying by k? We're multiplying this p minus 1. Yeah, so the reason-- FLT says that m to the p minus 1 is congruent to 1. So m to the k times p minus 1 is going to be 1 to the k. So that's just 1.

So no matter what k we put here, this will still work. Now, when you multiply it by m, that's just adding 1 onto the exponents. So then you'll have m to the k times p minus 1 plus 1 should be congruent to the m. Does this setup make sense to people? Yeah? Question?

AUDIENCE: Why do you add 1 at the end?

BRYNMOR CHAPMAN: Ah, so the question was, why are we adding 1 onto the end? So the answer was, FLT tells us that we're going to get 1. What we really wanted was m. So what we want is m to the ed is congruent to m, not congruent to 1. So that's why.

So as long as we choose ed congruent mod p minus 1 to 1, we'll get this. So more practically speaking, how could we generate e and d? How can I find a solution to this congruence? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Sorry?

AUDIENCE: Try different values of k.

BRYNMOR CHAPMAN: OK, so the answer was "try different values of k." So we're going to want to be using very large numbers here. So that's going to be intractable. We want to be able to do it more efficiently.

So recall from Tuesday, this is saying that e and d are modular inverses, modulo p minus 1. How can we compute a modular inverse? Well, one way is to use the pulverizer. So if we choose e randomly and then pulverize, this will give us d.

So we've got a way to generate keys. We've got a way to encrypt-- we raise m to the e. We've got a way to decrypt-- we raise c to the d. Can anybody tell me a security flaw? Why is this insecure?

So remember, for public key cryptosystem, everybody knows the public key. And we don't want anybody to be able to figure out the private key from that. So in this case, our public key is going to be kp, is going to be the pair p comma d-- oh, sorry, p comma e. And our secret key is going to be the pair p comma d.

So this is what we need in order to encrypt. This is what we need in order to decrypt. I claim that this is insecure. Yeah?

AUDIENCE: Can't anyone else also just pulverize the final thing?

BRYNMOR CHAPMAN: Yes, so the answer was, can't anybody else do exactly what we did? We started with the public key and then used it to derive our secret key. If we then share that public key with everybody else, everybody else can do exactly the same thing.

So our secret key isn't exactly secret. So we would like to try and do something similar to this but in a way where other people can't repeat our computation. Can anybody think of how we might do this? How can I make sure that nobody else can do the same pulverization? Yeah?

AUDIENCE: Choose a random d and then pulverize to find e.

BRYNMOR CHAPMAN: OK, so the answer was choose a random d and then pulverize to find e. I claim that this is exactly the same. It's the same process.

So these two are basically symmetric. So if I share one of them, then anybody else can then find the other. So the idea behind RSA is that this modulus here, the thing that we're trying to pulverize with, and the modulus here, the part of the key that we're sharing, here, they're off by 1. But in some sense, they don't have to be.

So we'd like to find a way to share this but not have this be so easily computable. How could we do that? Well, instead of using a single prime, suppose we have two primes. So let's call them p and q.

And now our n is going to be their product, p times q. So that's going to be what I want to share here. But now this no longer quite works. So instead of pulverizing with n minus 1, we're going to pulverize with-- sorry-- p minus 1 times q minus 1. Why?

Well, suppose we have done this already. Suppose that ed is congruent mod-- let's call this, say, m. No, m is for message. Let's call it n prime. Congruent mod n prime to 1. What happens if we take m and raise it to the ed power? What is this equivalent to, mod n? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: OK, so the answer was m. Yes. Why?

AUDIENCE: I don't know.

BRYNMOR CHAPMAN: Don't know. [CHUCKLES] Fair enough. I mean, this is kind of what we were aiming for. So very good educated guess. How could we prove this? Yeah?

AUDIENCE: Is that supposed to be n prime or just--

BRYNMOR CHAPMAN: So this is supposed to be an n. So remember that when we encrypt and decrypt, those computations are being done mod n. It's only the pulverizing that should be done using n prime.

So we're assuming that ed is congruent mod n prime to 1. And so we find that by pulverizing with n prime. But after that, we can forget about n prime. We don't need n prime anymore. We're just going to do all of our computations mod n using e and d.

So does anybody have any ideas how we could prove this statement here? What exactly is this statement saying if we go back to the definitions? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Yeah, exactly. So you've skipped a couple of steps. But, yes, I will get to that in a moment. So this is saying that n divides m to the ed minus m.

Now, at this point, we can break apart this n. So all we really need to do is prove that p and q separately divide m to the ed minus m, or, equivalently, that m to the ed is congruent mod p and q separately to m. So if we prove these two things, then we're done.

So p and q are symmetric. So let's only prove it for p. The same exact proof will work for q. So how do we prove now that m to the ed is congruent mod p to m? How can we do this? Any ideas? Yeah?

AUDIENCE: So use the fact that [INAUDIBLE] is [INAUDIBLE]

BRYNMOR CHAPMAN: That's right. So the answer was basically that we've already done it. So remember that ed is congruent mod n prime to 1. So this means that n prime divides ed minus 1.

Well, what is n prime? n prime equals p minus 1 q minus 1. So in particular, q minus 1-- no, p minus 1 divides ed minus 1.

So said otherwise, ed is congruent mod p minus 1 to 1. So that's exactly the condition we had over here. Yeah, question?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: So the question was, how do we know p minus 1 divides ed minus 1? p minus 1 divides n prime. We defined n prime to be the product of q minus 1 and p minus 1. So we know that n prime divides this. And one of our rules of divisibility was that then any factors of n prime also divide it.

So we know that p minus 1 divides ed minus 1. So that's just saying that ed is congruent to 1 mod p minus 1. And that's the condition we had here.

So now, by FLT, m to the ed is congruent mod p to m. So did people follow that? So we've got congruence mod p. By symmetry, we also have congruence mod q. p and q are-- they're basically symmetric.

And now, because they're prime, we can say that if p and q both independently divide this difference, their product must divide it as well. So that gives us congruence mod pq. OK, questions? Yeah?

AUDIENCE: Can you explain how we went from ed to [INAUDIBLE] 1 [INAUDIBLE]?

BRYNMOR CHAPMAN: OK, so the question was, can we go into this step here in more detail? So remember, that's what we were doing over here. So Fermat's little theorem says that m to the p minus 1 is congruent to 1 mod p.

So if we raise these to any power k, it's still going to be congruent to 1. So m to the k times p minus 1 will also be congruent to 1. So m to the k times p minus 1 plus 1 will be congruent to 1.

This here is saying that ed has that form. It's some multiple of p minus 1 plus 1. So basically, we're applying FLT repeatedly just as we did over here. Are people reasonably happy with this? Yeah?

AUDIENCE: How did you get p minus 1 and q minus 1 [INAUDIBLE]?

BRYNMOR CHAPMAN: OK, so why are we using p minus 1 and q minus 1 in the first place? In short, I pulled them out of my ass that it would work. So this is-- it's basically a generalization of FLT called Euler's theorem.

It turns out that this p minus 1 times q minus 1 is what we call phi of n, where phi is Euler's totient function. This counts the number of-- numbers up to n that are coprime with n.

So basically, we can use the same proof as we did Tuesday to show that if we want to get rid of this assumption that p is prime over here, we can replace this p minus 1 with phi of n. And then we get congruence mod n.

We just proved the special case where n is the product of two primes. But, yeah, basically it's kind of a guess and check. All we really need to know is that this works. We don't really need to know how we got there.

So we've proven that if we encrypt something by raising it to the e-th power mod n and then we decrypt it by raising it to the e power of mod n, we're going to get back to the same thing we started with. So this cryptosystem is at least correct. We can encrypt a message and then decrypt it again. Is it secure?

Oh, sorry. Actually, why don't I just do that? So is RSA secure? We said before that if we're just using a prime instead of this product of two primes, then anybody else can compute our secret key.

What are we doing now? Well, one, we're choosing random primes, large primes. So think of these as like having hundreds of digits.

Next, we're going to set n equals their product. And, let's just call it, phi of n now equals p minus 1 times q minus 1. And next, we're going to choose a random e coprime with this.

And then we want to pulverize it. We want to compute an inverse of e. So why don't I write it as this? Compute a multiplicative inverse modulo phi of n.

Now, what is the public key? Public key is going to be-- k sub p is going to be n comma e. k sub s is going to be n comma d. So before, when we had-- oh, question?

AUDIENCE: Yeah, how hard is it to choose e that's coprime? Or do you just choose a relatively small prime?

BRYNMOR CHAPMAN: OK. How hard is it to choose e that's coprime? It's actually not that hard. So in practice, usually you would not choose it totally randomly. You'd have a fairly small e so that encryption is efficient. But if you wanted to do it totally randomly, like how many things are not coprime to phi of n?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: So, yeah, multiples of the prime factors, yes. So let's see, have you-- did we do the Carmichael numbers yet? No? OK.

Well, it turns out that if you choose a random number, it'll actually be coprime with reasonably good probability. But in practice, we're going to just choose a fairly small e.

So we said that if we just forget about q, use n equals p, this doesn't work because anybody else can then compute our secret key in the same way as we computed it ourselves. Why does this not happen here? Why can nobody else compute the secret key given the public key? Yeah?

AUDIENCE: It's hard to find factors and--

BRYNMOR CHAPMAN: Yeah, so the basic idea is that we're relying on factoring being hard. So the basic idea is that if you have n, it should be hard to come up with p and q. And I claim that this is actually a much safer assumption than was used with Enigma because this is a very well-studied problem.

People have been trying to factor things more efficiently for hundreds of years. And mathematicians have not found an efficient way of doing this. So in particular, the adversary will also presumably not be able to do it. It's not just saying like, I personally couldn't come up with a good factoring algorithm. This is relying on many years of work by many mathematicians.

It still is possible that factoring isn't hard. And in fact, maybe somebody already knows how to do it. They've got their algorithm. They just haven't published it. But the assumption is basically like, if one could do it, then one of these mathematicians would have figured it out, like one of these world experts. So more formally, though, if we could compute the--

AUDIENCE: Isn't that cryptography by hubris again?

BRYNMOR CHAPMAN: Ah, OK, question was, isn't this just cryptography by hubris? OK, I claim that I'm not relying solely on my own intelligence. I'm relying on everybody else, like the entire field of mathematics.

Many mathematicians have spent their entire careers trying to factor things quickly, and nobody's figured out how to do it yet. It's not just like, I was spending half an hour in my basement, and I couldn't figure out a good algorithm in that time.

The Enigma engineer, he just came up with something and then couldn't figure out a way to break it and so just assumed security from that. So it's kind of me versus the entire field of mathematics.

OK, so, yeah, I guess the question is, given-- so if we had an algorithm that would compute the secret key from the public key, how would this factor in? Notice that the secret key doesn't actually contain p and q.

So I suppose, as described, we aren't quite relying on the hardness of factoring. We're relying on something slightly different. Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Oh, sorry.

AUDIENCE: Are you asking how did you come up with that p or q--

BRYNMOR CHAPMAN: So if we could find the private key, the secret key given the public key, how would this then factor in? We would like to rely on the hardness of factorization.

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: OK. So the answer was basically that you have to come up with the prime factors p and q before you can compute d from e. We would like to say that no matter how you do it, you're still basically going to end up with the factors p and q. You will have solved the problem even if you didn't intend to. Does that make sense?

OK, so suppose I have n, e, d. So suppose I've used the public key to come up with a secret key. So now I have these three pieces of information. Does this now allow me to factor n?

Does anybody remember the quadratic formula from grade school? It's OK, I don't either. So if I have e and d, ed minus 1 should be a multiple of p minus 1 times q minus 1. What happens if I reduce that mod n? Hm?

AUDIENCE: You get 1.

BRYNMOR CHAPMAN: I get 1? If I reduce it mod p minus 1 times q minus 1, I get 1. Yeah?

AUDIENCE: You get p and q.

BRYNMOR CHAPMAN: Get p and q?

AUDIENCE: Yeah.

BRYNMOR CHAPMAN: OK, let's make it a little bit simpler. Suppose I actually have p minus 1, q minus 1-- or p minus 1 times q minus 1. If I had that, could I factor n?

So if I had that, I could-- oh, yeah.

AUDIENCE: If you expand p minus 1 times q minus 1, you'll see that the expression for that is n pq minus p times q plus 1. And so from there, we can see what n is. We're given n. And then we just have, what, negative p times q plus 1. And that's easy.

BRYNMOR CHAPMAN: Yeah. So if we actually had p minus 1 times q minus 1, we could use that to figure out the sum of p and q. We know their product. We can use the quadratic formula. What happens if we just have ed-- sorry, e and d? Can we do something similar?

Oh, actually I'm running quite short on time. OK, why don't we hold off on that? Let's just take it on faith that we're relying on factoring being hard. Let's move on to the last thing that we wanted to cover today, which is a slightly different application of number theory, which you may find useful throughout computer science.

Notice-- where is it? OK, here. So here, we're trying to work mod n. n is the product of two primes. And we said that it's easier-- instead of working mod n, we're going to work mod the two primes individually.

So it turns out that this is something that you can do in general. This is what we call the Chinese remainder theorem. If you have-- in fact, if you have two coprime numbers p and q, working modulo their product is equivalent to working modulo each of them individually.

And often working modulo the smaller numbers individually is going to be much easier than working modulo the large number. So as a very simple example, suppose we have the following. Actually, strictly less than 55. And we know that x is congruent mod 5 to 4, x congruent mod 11 to 7.

OK, I claim that we have enough information now to uniquely identify x. Can anybody do it? How might we do it? Yeah?

AUDIENCE: This would be a slow way to do it. But you could go through every number between 0 and 55 and figure out what all the possibilities were for each of these [INAUDIBLE].

BRYNMOR CHAPMAN: OK, so the answer was try everything between 0 and 55, see what they're congruent to mod 5, see what they're congruent to mod 11, and see whether it satisfies both of those. I claim that we can do it slightly faster than that. Why don't we enumerate everything that satisfies this congruence first?

So what is congruent to 7 modulo 11 in this range? So that tells us that-- maybe I should write it down here. Whoops. x is in the following set-- 7, 18, 29, 40, 51. And if we add 11 again, we get past 55.

Now we only have to check five numbers to see which is congruent to 4 modulo 5. And if we look at them, 29 is the only one that works. So it turns out that there's a unique answer modulo 55.

And we can actually do this in general and, in fact, even more efficiently. So Chinese remainder theorem, often abbreviated CRT. So suppose p and q are coprime and a and b are integers. And we have the following system of congruences.

So if we have this, then there is a unique solution, there's a unique x, that satisfies this modulo pq. So how might we prove this? What are the elements that we need to prove? Yeah?

AUDIENCE: Sorry, I had a question about [INAUDIBLE]. Which Greek letter are you using for the quotient?

BRYNMOR CHAPMAN: Sorry?

AUDIENCE: For the top right-ish board, there's a Greek letter--

BRYNMOR CHAPMAN: Oh, here? Phi of n.

AUDIENCE: Huh?

BRYNMOR CHAPMAN: Here? Phi of n.

AUDIENCE: Oh.

BRYNMOR CHAPMAN: Yeah. So you don't have to worry about what this is actually called. We could call this n prime. But yeah. Sorry, this was just following up on what one of your classmates asked earlier.

So what are the elements that we need to prove in order to prove this theorem here? Well, we need to prove existence. And then we need to prove that there don't exist two solutions. So let's start with existence.

Why does there exist a solution? Maybe we can start thinking about a special case. So suppose, for a start, that a equals 0. How could we find a solution to x congruent to b modulo q and x congruent to 0 modulo p? Yeah?

AUDIENCE: I just have a question [INAUDIBLE].

BRYNMOR CHAPMAN: Sure.

AUDIENCE: So for the example question, you gave us the [INAUDIBLE]. Is that [INAUDIBLE]?

BRYNMOR CHAPMAN: Yeah, so the relation here is that we've got our p and q are 5 and 11. Our a and b are 4 and 7. And so we're saying that there's a unique solution to this system of congruences modulo 55, which is their product. Or you could also phrase that as there is a unique x in this range, like a unique integer x in this range.

So we're just going to generalize this to a more general p and q, more general a and b. OK, question? Oh, sorry. OK. Well, if we look at the special case where a equals 0, why don't we start by defining p minus 1, p inverse to be a modular inverse of p modulo q?

And now let's multiply that by p to get the integer e sub q. Now what can we say about e sub q? What is it congruent to modulo p? Yeah?

AUDIENCE: Wouldn't it be 0 because it's a multiple of p?

BRYNMOR CHAPMAN: Yeah, so we defined it to be a multiple of p. So e sub q is going to be congruent mod p to 0. What about modulo q? What is it congruent to modulo q? Yeah?

AUDIENCE: 1.

BRYNMOR CHAPMAN: 1. Yeah, so by definition, p inverse is a multiplicative inverse of p modulo q. So when you multiply them, you get 1 mod q.

So how can we use e sub q to get x? Remember we're thinking of the special case where a equals 0. So we're trying to find a multiple of p that's congruent to b modulo q. Anybody? Yeah?

AUDIENCE: Couldn't we take eq as a good one, but [INAUDIBLE] and multiply both sides by d?

BRYNMOR CHAPMAN: Yeah, so the answer was multiply eq by b. b times eq now is congruent to 0 modulo p and 1 times b, which is b modulo q. Does that make sense to everybody? So for the more general case, we are going to define e sub p analogously.

Now, how do you think we could solve the more general case? Instead of just b times eq, how do we then fold ep into it? Yeah?

AUDIENCE: You could maybe do something more like beq but then add [INAUDIBLE].

BRYNMOR CHAPMAN: Yeah. So claim x equals aep plus beq is a solution. So we can verify each of these two congruences separately.

So if we look at the first congruence, what happens if we reduce this x modulo p? Well, modulo p, this ep here, is congruent to 1. So this term becomes a.

We said that eq is defined to be a multiple of p. So the second term just goes away. So modulo p, x is now congruent to a.

Modulo q, it's exactly the opposite. This term goes away because the ep was defined to be a multiple of q. And-- sorry-- yes. This is defined to be a multiple of q.

And I've done it the wrong way around, haven't I? eq mod q-- no. No, we're fine. So ep is a multiple of q. The a term goes away. b times eq is going to be 1. Sorry-- b times eq is going to be b.

So what about uniqueness? How might we prove uniqueness? What is the setup going to look like? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: OK, so the idea was to show that any candidate solution, so let's say x prime, is actually the same solution. So let x and x prime be two solutions. Now what are we trying to show? Yeah?

AUDIENCE: [INAUDIBLE]

BRYNMOR CHAPMAN: Not equals. So we're not trying to prove they are unique integers. We're trying to prove that it's unique modulo pq. So we want to show that they're congruent modulo pq. How could we do this?

Well, what is the definition of this equivalence? We could do this by showing that pq divides x minus x prime. Now, in order to do that, we can look at p and q separately.

So why does p divide the difference? Well, both of them satisfy that first equation, or that first congruence. So both of them are congruent to a modulo p. If you subtract them, you get 0. So p divides the difference.

Now, because they both satisfy the second congruence, you subtract them. You get b minus b. So q divides the difference, as well-- so p and q individually.

And by assumption, they're co-prime. So their product must then also divide the difference. So that means that x and x prime must be congruent modulo pq.

So that's all we have time for today. Yeah, you'll get more practice with RSA and CRT in recitation tomorrow. And, yeah, feel free to come up if you have questions.