00:00 The goal of encryption is to garble data is such a way so that no one who has the data can read it unless they're the intended recipient.

And the encryption of pretty much all private information sent over the internet relies immensely on one numerical phenomenon - as far as we can tell, it's really really hard to take a really big number and find its factors using a normal, non-quantum computer.

Unlike multiplication, which is very fast (just multiply the digits together and add them up ), finding the prime numbers that multiply together to give you an arbitrary, big, non-prime number appears to be slow - at least, the best approach we currently have that runs on a normal computer - even a very powerful one - is very slow.

Like, to find the factors of this number , it took 2000 years of computer processor time!

Now, it's not yet proven that we won't eventually find a fast way to break encryption just with normal computers, but it's certain that anybody with a large working quantum computer today would pose an immediate privacy and security threat to the whole internet.

And that's due to something called "Shor's Algorithm."

Well actually it's due to quantum superposition and interference; they're just taken advantage of by an algorithm developed by Peter Shor, which I'm now going to attempt to explain.

The kind of encryption we're talking about garbles or "locks" messages using a large number in such a way that decrypting or "unlocking" the data requires knowing the factors of that number . If somebody doesn't have the factors, either they can't decrypt the data, or they have to spend a really really long time or a huge amount of investment in computing resources finding the factors.

Our current best methods essentially just guess a number that might be a factor, and

check if it is . And if it isn't, you try again.

And again.

And again.

It's slow.

There are so many numbers to check that even the fast clever ways to make really good guesses are slow.

For example, my computer took almost 9 minutes to find the prime factors of this number.

So if you used this number to encrypt your data, it would only be safe from me for 9 minutes.

If, on the other hand, you used a number like the one that took 2000 years of computer processor time to factor , your data would definitely be safe from me and my laptop, but not from somebody with access to a server farm . This is similar to how putting a lock on your door and bars on your windows doesn't guarantee you won't have stuff stolen from your house, but does make it take more time and more work.

Encrypting data isn't a guarantee of protection - it's a way of making it harder to access; hopefully enough harder that no one thinks it's worth trying.

But quantum computation has the potential to make it super super easy to access encrypted data - like having a lightsaber you can use to cut through any lock or barrier, no matter how strong.

Shor's algorithm is that lightsaber.

Roughly speaking, to factor a given number Shor's algorithm starts with a random crappy guess that might share a factor with your target number, (but which probably doesn't), and then the algorithm transforms it into a much better guess that probably DOES share a factor!

There's nothing intrinsically quantum mechanical about this - you can, in fact, run a version of Shor's algorithm on a regular computer to factor big numbers, but perhaps unsurprisingly the "turning your bad guess into a better guess" part of the process takes a very very long time on a normal computer.

On the other hand, this key step happens to be ridiculously fast on quantum computers.

So, our task is to explain how Shor's algorithm turns a crappy guess into a better guess (which is purely mathematics), and why quantum computers make that fast (which is where the physics comes in).

It all starts with a big number, N, that you'll need to find the factors of to break into some encrypted data.

If you don't know what the factors are (which you don't), you can make a guess; just pick some number g that's less than N . We actually don't need the guess to be a pure factor of N - it could also be a number that shares some factors with N, like how 4 isn't a factor of 6, but shares a factor with it.

Numbers that share factors are ok because there's a two-thousand-year-old method to check for and find common factors - it's called Euclid's algorithm and it's pretty darn efficient.

All this is to say that to find a factor of N, we don't have to guess a factor of N - guessing numbers that share factors with N works, too, thanks to Euclid.

And if Euclid's algorithm found any shared factors with N, then we'd be done!

You could just divide N by that factor to get the other factor and break the encryption.

But for the big numbers used in encryption, it's astronomically unlikely that any single guess will actually share a factor with N.

Instead, we'll use a trick to help transform your crappy guess into a pair of guesses that are way more likely to share factors with N. The trick is based on a simple mathematical fact: for any pair of whole numbers that don't share a factor, if you multiply one of them by itself enough times, you'll eventually arrive at some whole number multiple of the other number, plus 1 . That is, if a and b are integers that don't share factors, then eventually $a^p$ will be equal to m times b + 1, for some power p and some multiple m.

We don't have the time to get into why this is true, but hopefully a few illustrations can at least give you a feeling for it.

For example, 7 and 15.

While seven squared isn't one more than a multiple of 15, and neither is seven cubed, seven to the fourth is.

05:01 Or take 42 and 13 - 42 squared isn't one more than a multiple of 13 , but 42 cubed is.

This same kind of thing works for any pair of numbers that don't share factors, though the power p might be ridiculously large.

So, for the big number, N, and your crappy guess, g, we're guaranteed that some power of g is equal to some multiple of N, plus 1 . And here's the clever part - if we rearrange this equation by subtracting the 1 from both sides, we can rewrite $g^p - 1$ as $(g^p/2 + 1)(g^p/2 - 1)$. You can multiply that back together to convince yourself that it works.

And now we have an equation that almost looks like "something" times "something" is equal to N, which is exactly what we're trying to find - factors of N!

These two terms are precisely the new and improved guesses that Shor's algorithm prescribes: take the initial crappy guess, multiply it by itself p/2 times, and either add or subtract one!

Of course, since we're dealing with a multiple of N rather than N itself, the terms on the left hand side might be multiples of factors of N, rather than the factors themselves.

Like how $7^4/2 + 1 = 50$, and $7^4/2 - 1 = 48$, neither of which is a factor of 15.

But we can find shared factors by using Euclid's algorithm again, and once we do, we'll have broken the encryption!

So is this all Shor's algorithm is?

Where's the quantum mechanics?

Why can't we use this to break encryption right now?

Well, indeed, there are three problems with these new and improved guesses.

First, one of the new guesses might itself be a multiple of N, in which case the other would be a factor of m and neither would be useful to us in any way.

And second, the power "p" might be an odd number , in which case p/2 isn't a whole number and so our guess taken to the power of p/2 probably isn't a whole number either, which is no good.

However, for a random starting guess, it turns out that at least 3/8ths of the time neither of these problems happens and p does generate guesses that share factors with N and break the encryption!

This is worth repeating - for ANY initial guess that we make, at least 37.5

Which means we're 99

However, problem number three is the big one.

Remember, to turn a crappy guess into a good guess we need to know how many times you have to multiply our guess by itself before we get a multiple of N, plus 1.

And for a normal computer, the act of finding that power p takes a ton of work and time.

It's not hard for small numbers like 42 and 13, but if our big number is a thousand digits long, and our crappy guess is 500 digits long, then trying to figure out how many times you have to multiply our guess by itself before you get some multiple of the big number, plus one, takes a ridiculous amount of trial and error on a normal computer - more effort than it would have taken to just factor N by brute force in the first place!

So finally, this is where quantum mechanics comes in and speeds things up an INSANE amount.

Unlike a normal computation which gives only one answer for a given input, a quantum computation can simultaneously calculate a bunch of possible answers for a single input by using a quantum superposition - but you only get one of the answers out at the end, randomly, with different probabilities for each one.

The key behind fast quantum computations is to set up a quantum superposition that calculates all possible answers at once while being cleverly arranged so that all of the wrong answers destructively interfere with each other.

That way when you actually measure the output of the calculation, the result of your measurement is most likely the right answer.

In general it can be really hard to figure out how to put any particular problem into a quantum form where all the wrong answers destructively interfere, but that's what Shor's algorithm does for the problem of factoring large numbers - well, actually, it does it for the problem of finding the power "p".

Remember, at this point we've made a crappy guess g, and we're trying to find the power p so that g to the p is one more than a multiple of N. A p that does that also means that $g^p/2ś1$ is very likely to share factors with N.

So to begin the quantum computation, we need to set up a quantum mechanical computer that takes a number x as input, and raises our guess to the power of x.

For reasons we'll see later, we need to keep track of both the number x, and our guess to that power.

The computer then needs to take that result and calculate how much bigger than a multiple of N it is.

We'll call that the "remainder", and we'll write it as plus "something"for whatever something the remainder is (remember, we want a remainder of 1).

So far, no different from a normal computer.

But since it's a quantum computer, we can send in a superposition of numbers and the computation will be done simultaneously on all of them, first resulting in a superposition for each p of all possible powers our guess could be raised to , and then a superposition

for each p of how much bigger each of those powers are than a multiple of N.

We can't just measure this superposition to get the answer - if we did, we'd get a single random element of the superposition as output, like "our guess squared is 5 more than a multiple of N" . Which is no better than just randomly guessing powers, which we can do with a normal computer.

10:04 No, we need to do something clever to get all the non-p answers to destructively interfere and cancel out, leaving us with only one possible answer: p.

Which it turns out we can do, based on another mathematical observation.

This mathematical observation isn't particularly complicated, but it is a tad subtle and it may not be immediately clear why we care.

However, it's the key idea that allows us to turn the problem of finding p into one that works well on a quantum computer, and so in some ways it's the crux of Shor's algorithm - which is to say, it's worth the effort!

Ok, so remember that IF we knew what p was, we could raise our guess to that power and get one more than a multiple of N. On the other hand, if we take our guess to a random power , it's probably going to be some other number more than a multiple of N - say, 3 more . But check this out - if we raise our guess to that random power plus p, it's again 3 more than a multiple of N . If we raise our guess to that random power plus 2 p, it's again 3 more than a multiple of N. And so on.

It's pretty straightforward to show why this works by multiplying out "something times N plus 1" with "something else times N plus 3"; you get "a different something times N, again plus 3" . And this works for any power x - if $g^x$ is r more than a multiple of N, then $g^(x+p)$ will also be r more than a multiple of N (though a different multiple).

So the power p that we're looking for - the one that allows us to improve our crappy guess and find factors of N and break encryption - it has a repeating property where if we take another power and add (or subtract) p to it, the amount more than a multiple of N stays the same.

This repeating property isn't something you could figure out from taking our guess to just one power - it's a structural relationship between different powers, and we can take advantage of it since quantum computations can be performed on superpositions of different possible powers.

Specifically, if we take the superposition of all possible powers and JUST measure the "amount more than a multiple of N" part, then we'll randomly get one of the possible "amounts more than a multiple of N" as the output - say, 3.

The specific number doesn't matter to us, but what does matter is that this means we must be left with a superposition of purely the powers that could have resulted in a remainder of 3.

This is one of the special properties of quantum computation - if you put in a superposition and get an answer that could have come from more than one element of the superposition, then you'll be left with a superposition of just those elements!

And in our case, because of the repeating property, those powers are all numbers that are "p" apart from each other.

To recap, we're trying to find p because it will allow us to turn our crappy guess into a good guess for a number that shares factors with N, which will allow us to break the encryption.

And we now have a quantum superposition of numbers that repeat periodically with a period of p, or equivalently, they repeat with a frequency of 1/p . If we can find the frequency, we can find p and break the encryption!

And the best tool to find the frequencies of things is called a Fourier transform.

Fourier transforms are what allow you to input an audio signal as a wave and convert it into a graph showing the different frequencies that the wave is made up of.

13:00 And there's a quantum version of the Fourier transform, which we can apply to our superposition that repeats with a frequency of 1/p to cause all the different possible wrong frequencies to destructively interfere, leaving us with a single quantum state: the number 1/p.

So how does the quantum Fourier transform perform this magic?

Well, if you input a single number into the quantum Fourier transform, it will give you a superposition of all other numbers - but not any old superposition.

A superposition where the other numbers are all weighted by different amounts, and those weights look roughly like a sine wave with the frequency of the single number we put in.

If you put in a higher number, you get a sine wave-style superposition of all other numbers, but with a higher frequency.

And the magic is that when you put IN a superposition of numbers, you get out a superposition of superpositions and the sine waves add together - or subtract and cancel out.

And it happens that if you put in a superposition of numbers that are all separated by an amount p, all those sine waves interfere so that what you get out (and I'm oversimplifying a touch), is the single quantum state representing 1/p.

Which we can finally measure to get the output of the computation: 1/p!

Which we invert to find p, and as long as p is even we can now finally raise our guess to the power p over two and add or subtract one, and as long as we don't get an exact multiple of N, we are guaranteed to have a number that shares factors with N. And therefore we can use Euclid's algorithm to quickly find those factors, and thus we can finally take the encrypted data and decrypt it.

And thus we will have broken the encryption.

And that is Shor's algorithm - the lightsaber that can be used to cut through locks on the internet.

As complicated as this clearly is in practice... (and we've glossed over a ton of details), it's surprising to me how simple the core structure of Shor's algorithm actually is:

for any crappy guess at a number that shares factors with N, that guess to the power p/2 plus or minus one is a much much better guess, if we can find p.

And we CAN find p almost immediately with a single (if complex) quantum computation.

A normal computer would have to go one by one through all possible powers, which would take an incredible amount of time for any really really big number like the ones used in encryption, since p could be almost any number up to N. The quantum version is ridiculously ridiculously faster, and if a big enough quantum computer is ever built, then Shor's algorithm would allow the user to very easily decrypt any data encrypted with a large-number factoring based system - which would pretty much ruin the entire internet.

At this point, however, the biggest actual quantum implementations of Shor's algorithm don't have enough memory to hold more than a few bits, which only allows factoring of numbers like 15, 21, and 35 . Now, there are other methods of factoring using quantum computations that are a bit more advanced, and have factored numbers as big as a few hundred thousand using just a few quantum bits of memory . But they would still need 2000 times more quantum memory to factor even some of the smaller of the really big numbers used in modern encryption.

So, no need to worry about quantum computers just yet.

If all this talk of breaking encryption makes you a bit nervous and worried about your online safety, well, there's something you can do to improve your internet security right now - I've been a long time user of the password manager Dashlane who are sponsoring this video, and if you've never used a password manager before, Dashlane is amazing.

It generates and remembers a long, unique password for each site or service that I use so that I don't have to worry about remembering passwords; and of course all of my data and passwords are stored encrypted with very very large numbers.

And Dashlane is more than just a password manager - it lets you know when your passwords are old or weak or when a site or app you use has been hacked so you can change your passwords, it encrypts and lets you securely share passwords with family and co-workers,

it can be used to securely store or share your address, credit card info, and banking info, with just the people and sites you want to, it can be used as a VPN, and more.

Oh, and Dashlane uses 2048 bit numbers for its encryption - numbers that big are estimated

to take a trillion times more effort to factor than any that have so far been factored by brute force.

And of course Dashlane is free for up to 50 passwords for as long as you like, so you have nothing to lose checking it out.

But, if you want the very useful features of unlimited passwords, encrypted syncing of passwords, VPN, remote account access, and more, the first 200 people get 10

Dashlane premium by going to dashlane.com/minutephysics and using promo code minutephysics.

Again, that's dashlane.com/minutephysics with promo code minutephysics to simplify and encrypt your online life.

Dashlane has legitimately improved my online security and changed my password habits for the better.

What could it do for you?