QuIC Seminar 12

Shor's Algorithm Part 1

Contents

12.1 Cryptography **76**

12.1.1 Private key cryptography 76

12.1.2 Public key cryptography 77

12.2 How to break public key cryptography **80**

12.2.1 RSA 80

12.2.2 Diffie-Hellman-Merkle 81

12.3 Reduction of factoring to period finding **81**

12.3.1 A function that's periodic 81

12.3.2 What does the period tell us? 82

12.3.3 Euclid's algorithm 83

12.4 Summary **84**

I know what you're thinking. "Alright, alright, enough of these artificial black box "query model" algorithms. What's the big idea here? Why would I care if the function was constant or balanced, Deustch? Which evil villain is conjuring up all these secret strings for us to figure out quickly, Berstein, Vazirani, and Simon? And plus, we never even looked at how to implement the black boxes!"

These are fair points, as we've discussed. In no time at all, however, Peter Shor put all of them to rest in triumphant fashion with the publication Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. No more can we complain about artificial problems, unless you think keeping your credit card information secure over internet transactions is artificial. The result by Shor was so big that people started switching fields, computer scientists' jaws hit the floor, physicists went dancing, mathematicians did whatever mathematicians do to celebrate[1]. The paper has nearly ten thousand citations and is one of the monuments of modern science. It's perhaps the most academically rich, beautiful result in history, tying together number theory and cryptography and computational complexity and quantum physics. Let's understand it.

But why did we still need to care about those darn artificial problems? The answer is that Shor's algorithm, at least the quantum part, is extremely similar to Simon's algorithm. Simon's algorithm built off Bernstein and Vazirani's algorithm, and their algorithm built of Deustch and Josza's algorithm. Our understanding of these is a huge step towards understanding Shor's algorithm.

That being said, there are a lot of intricacies to the algorithm, and it will take us a few seminars to cover in satisfactory detail. This seminar will be mostly setting the stage. To rid all complaints of artificial

[1]Okay, perhaps I'm slightly exaggerating. We could also say that engineers started weeping at the idea of building such a device.

problems, we'll first spend some time discussing public key cryptography and how factoring can break the RSA cryptosystem. Then we'll discuss some group theory and show how factoring can be reduced to period finding. This will set the stage for the quantum part of Shor's algorithm, which computes the period of a certain function, in the next seminar(s).

12.1 Cryptography

Cryptography is the art (science) of private communication. There's many examples where one party, Alice, wants to communicate with another, Bob, but she's worried some eavesdropper, Eve, could intercept the message. For example, Alice and Bob could be generals in a war, and Eve could be on the opposing forces. Or, Alice could be an online shopper trying to communicate her credit card information to the store, Bob, in the presence of an online hacker, Eve.

12.1.1 Private key cryptography

One way to achieve this is to encode the message with some private *key* that Alice and Bob know but Eve doesn't. This is formally known as *private key cryptography.* Assuming Eve has not gained knowledge of the private key, these systems can range from impossible to break to extremely simple to break, all depending on how the message is encrypted and what the private key is.

We'll proceed with a series of examples ranging from easy to impossible to break. In these examples, we'll take our message to be a short phrase, for simplicity "QuIC." Each letter will be represented by it's numeric position in the alphabet, starting with 1 for A, 2 for B, and so on. Capitalization won't matter. Since I still hardly know the alphabet, I've taken the time to enumerate this in the table below.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Table 12.1: Reference table for encoding letters as numbers and vice-versa.

The message "QuIC" thus becomes $(17, 21, 9, 3)$.

Example 1: The Caeser cipher.
The Caesar cipher is an example of a private key cryptosystem that's really *easy* to break. It works as follows.

Alice and Bob pick a key that's a single number—1 through 26—as the private key. Each number in the message is then shifted by this amount (modulo 26). As an example, let's say the key is $k = 7$. The message "QuIC," which is $(17, 21, 9, 3)$, then becomes $(24, 2, 16, 10)$, which is "XBPJ." Since Bob knows the key, he simply subtracts this off when he gets the encrypted message to read the original message.

Suppose Eve intercepted the encrypted message without the key. She would read only "XBPJ." This is nonsense! What is that, a radio station? How could Eve possibly make sense of this? We can be sure this message is unbreakable, right?

The issue here is that there's only 26 possible keys. Eve could simply, and very quickly, try every single one of them until the letters started forming sensible words[a]. How would she know it was a Caeser cipher? She could have guessed it or had gained some knowledge of it somehow.

[a]Not that QuIC is a sensible word or anything, but we're all in the seminar, so we know what it means.

The Caeser cipher is thus not a very secure private key cryptosystem. It's a good illustration of how private key cryptography works, in particular the encryption and decryption schemes, and the importance of the key used.

Exercise 80: Using a Caeser cipher with key $k = -1$, encode the message "IBM." Have you ever read the Arthur C. Clarke *Space Odyssey* series or seen the Stanley Kubrick film *2001: A Space Odyssey*? If so, you should recognize the encrypted message as one of the main characters. Both Clarke and Kubrick denied any meaning behind this coincident Caesar cipher, but some think there is given the plot of the story/film.

Example 2: The Enigma machine.
In World War II, Germany used a device called *The Enigma machine* to encrypt all of their messages so the Allied forces could not intelligibly intercept them.

An Enigma operator would type each letter of a message into the machine. For each letter, Enigma would return a pseudo-random substitution for this letter, which would be the encryption. This process would continue for the entire message, then the encrypted message would be sent, typically over radio in Morse code.

The message would be decrypted by having the receiver type the encrypted message into another Engima machine with identical settings. Thus, unless the Allied Forces knew the exact inner workings of Enigma—which had settings that changed daily—they could not crack the messages.

But they did end up cracking the messages. The way this was accomplished was by noting patterns in messages resulting from poor operation. I don't know if this is historically accurate, but according to the movie *The Imitation Game* it's because each message contained the same text, something like "heil Hitler." By identifying this message, the key became much easier to break. This realization, and cryptography in general, played a huge role in the outcome of WWII.

The last example of a private key cryptosystem—I think we get the idea by now—that I'll mention is the *one-time pad*, which is truly unbreakable. It's related to the previous examples in that each letter is shifted, but each letter is shifted by a different amount, and the amount is *truly* random. Given these assumptions, one can show that Eve cannot decipher the message without knowing the key, as long as the key is only used once. (Hence the *one-time* pad. Why *pad*? It's a name for "what the key is written on," like a notepad.) It's not hard to see why this is true—each letter is shifted by a random amount, so the message is thoroughly jumbled with no correlation or way of determining any letter.

12.1.2 Public key cryptography

Private key cryptography is all well and good and has interesting historical examples, but there's a serious impediment in practice: Alice and Bob have to somehow exchange a key without anyone intercepting it. Well, the problem of private communication was the original problem we started with! If Alice is shopping online, it's extraordinarily impractical for her to try and exchange a private key with the store owner, Bob.

The fix to this is to publish the key to the world! Well, at least part of the key. This is how *public key cryptography* works. It sounds like it couldn't be possible when you first hear it or imagine it. The key idea[2] is only exchanging half of the key and relying on a process that's easy to do one way but extremely hard to do in the reverse way. The best way I know how to illustrate these currently vague points is by example. We'll use the RSA public key cryptosystem for this example, in which Alice generates a key as follows:

RSA key generation

1. Select two *large*[3] prime numbers p and q.
2. Compute the product $n := pq$.

   Note that this is efficient—multiplying two numbers together is easy (computationally speaking).
3. Select at random a small odd integer e that is relatively prime to
$$\phi(pq) \equiv \phi(n) := (p-1)(q-1). \tag{12.1}$$
   The function ϕ is known as *Euler's totient function* or *Euler's phi function.*
4. Compute d, the multiplicative inverse of e module $\phi(n)$.
5. The *public key* is then the tuple $P := (e, n)$. The *secret key* is the tuple $S := (d, n)$.

[2]No pun intended.
[3]How large is large? This will be clarified shortly.

That's all there is to it! Before we see an example, note the following: the *public key* is the one that anyone can have access to. Bob sees it and uses it to encrypt his message, which we'll discuss shortly. But Eve can also see it, anyone can see it—and they still can't break the cryptosystem *within any reasonable amount of time.* We'll see why this is shortly, but first let's practice generating RSA keys with an example.

Example 3: Generating an RSA key.
We'll proceed with the above steps giving a small numeric value to p, q and e to see how the process works.
Let's say $p = 5$ and $q = 11$.
This implies that $n := pq = 55$.
This implies that $\phi(n) = \phi(pq) = (p - 1)(q - 1) = (4)(10) = 40$. Let's pick $e = 3$, for example.
The multiplicative inverse of e module $\phi(n)$ is $e = 27$, since $de = 81 = 1 \bmod 40$.
Thus we have the public key $P := (e, n) = (3, 55)$ and the secret key $S := (d, n) = (27, 55)$.

Exercise 81: Show that the public key for RSA with $p = 3$, $q = 11$, and $e = 3$ is $P = (3, 33)$, and the secret key is $S = (7, 33)$. List out all steps as above.

These numbers are small—consisting of only a few bits in binary representation—and not secure! The current standard is using a number n which has 2048 bits (RSA-2048). More bits corresponds to increased difficulty in factoring the integer n, and we'll see shortly that factoring can break RSA. First, let's see how Bob uses the public key to encrypt his message.

Let m be a numerical value representing a character in Bob's message. Given $P = (e, n)$, this value is encoded via

$$E(m) := m^e \bmod n. \tag{12.2}$$

This quantity can be efficiently computed by a trick known as *repeated squaring.* This trick will actually be crucial in Shor's algorithm for a different but similar reason—we'll leave the explanation of how repeated squaring works for when we come to it later. It's not critical to understand now. What is important is that $E(m)$ can be computed efficiently, and everything in RSA key generation can be done efficiently.

Example 4: Encrypting with an RSA public key.
Given the public RSA key $P = (3, 33)$, let's encrypt each character in the message "QuIC" which we represent as $(17, 21, 9, 3)$.
The first character becomes $E(17) = 17^3 \bmod 33 = 29$.
The second character becomes $E(21) = 21^3 \bmod 33 = 22$.
The third character becomes $E(9) = 9^3 \bmod 33 = 3$.
The fourth character becomes $E(3) = 3^3 \bmod 33 = 27$.
Thus, the encrypted message is $(18, 22, 3, 27)$ or "RVCA" (taking each number modulo 26).

Exercise 82: Encode the message "RSA" using the RSA public key $P = (3, 55)$.

Now that Bob has encrypted his message, he can send it to Alice. It's ok if this message get's intercepted by Eve—she still won't be able to decipher it because she doesn't have the secret key—only Alice does. Once the message reaches Alice, she can decipher it using her secret key $S := (d, n)$ which only she knows. Let's see how this works.

We'll first proceed by example to see the magic of RSA decryption. (Then we'll prove some theorems to understand the magic.) We know Bob's encrypted message was $(29, 22, 3, 27)$ and his original message was $(17, 21, 9, 3)$. Consider what happens when Alice does the decoding

$$D(E(m)) = E(m)^d \bmod n \tag{12.3}$$

with her secret key $S = (d, n) = (7, 33)$.

- The first encrypted number gets mapped to $D(29) := 29^7 \bmod 33 = 17$.

  This is the first number in Bob's original message! So far so good for the proposed decryption scheme (12.3).

- The second encrypted number gets mapped to $D(22) = 22^7 \bmod 33 = 22$.

  Two for two!

- The third encrypted number gets mapped to $D(3) = 3^7 \bmod 33 = 9$.

- The fourth encrypted number gets mapped to $D(27) = 27^7 \bmod 33 = 3$.

Thus the original message $(17, 22, 9, 3)$, or "QuIC," is recovered! This is great news, but why does it work? Will it always work? The following theorem answers these questions. First we'll formally define the encryption and decryption schemes.

Definition 12.1 (RSA Encryption and Decryption). Let $P := (e, n)$ and $S := (d, n)$ be public and secret RSA keys, respectively. A message m is **encrypted** using the public key P as

$$E(m) := m^e \bmod n, \tag{12.4}$$

and an encrypted message $E(m)$ is **decrypted** using the secret key S as

$$D(E(m)) = E(m)^d \bmod n. \tag{12.5}$$

Now we prove that the decryption always recovers the original message. The proof uses some standard theorems from number theory which we will state and use without proof, in particular *Euler's theorem*, which states that, if a and n are relatively prime positive integers, then

$$a^{\phi(n)} = 1 \bmod n \qquad \text{(Euler's theorem)} \tag{12.6}$$

where ϕ is the same Euler totient function in (12.1). (Integers a and n are relatively prime if their greatest common divisor is one. For example, 6 and 25 are relatively prime, but 6 and 24 are not.)

Theorem 12.1 (RSA decryption recovers the original message). For any RSA encoded message $E(m)$ as in Definition 12.1, we have

$$D(E(m)) = m. \tag{12.7}$$

Proof. Consider the case when m is relatively prime to n. Then,

$$\begin{aligned} D(E(m)) &:= E(m)^d \bmod n \\ &= (m^e)^d \bmod n \\ &= m^{ed} \bmod n. \end{aligned}$$

Now, by construction of e and d, we know that $d = e^{-1}$ modulo $\phi(n)$. By definition, this means that $ed = 1 + k\phi(n)$ for some integer k. Thus, we have that

$$\begin{aligned} D(E(m)) &= m^{1+k\phi(n)} \bmod n \\ &= m \cdot (m^{\phi(n)})^k \bmod n \\ &= m \cdot (1) \bmod n \\ &= m. \end{aligned}$$

In the second to last step we used Euler's theorem (12.6) (we assumed m and n were relatively prime), and in the last step we used that $m < n$ to get rid of the modulo n.

This proves the theorem when m and n are relatively prime. The case where m and n are not relatively prime is left as a (not that important) exercise. □

Exercise 83: Complete the proof of Theorem 12.1 by considering the case when m and n are not relatively prime.

At this point, we've proposed a public key cryptosystem known as RSA, detailed how to generate public and secret keys, how to encrypt messages, how to decrypt messages, and proved decryption recovers the original message. What's left is to consider how RSA could be broken and determine how difficult such a task would be. This is the subject of the next section.

12.2 How to break public key cryptography

Public key cryptography, in principle, is able to be cracked. The reason we say "in principle" is that, for a good public key cryptosystem, it takes a *very* long time to crack, even if the method is known for how to do this. For most of the following discussion, we'll focus on cracking RSA, but at the end we'll mention another protocol, the Diffie-Hellman-Merkle key exchange.

12.2.1 RSA

Recall that the RSA public key contains the integer n. Suppose we were able to factor this integer into its two primes p and q. This would then give us the ability to construct the secret key S and decrypt the message in the same way that Alice does. Thus, if we could factor the integer n (quickly), we could break RSA.

In an above example, we had $n = 33$. One can factor this instantaneously to get $p = 3$ and $q = 11$, then generate the secret key. The key property that makes RSA safe is that factoring n gets hard when n contains many digits. For example, imagine if someone asked you to factor the RSA-2048 number[4]

$$\begin{aligned} n = {} & 2519590847565789349402718324004839857142928212620403202777713783604366202070 \\ & 7595556264018525880784406918290641249515082189298559149176184502808489120072 \\ & 8449926873928072877767359714183472702618963750149718246911650776133798590957 \\ & 0009733045974880842840179742910064245869181719511874612151517265463228221686 \\ & 9987549182422433637259085141865462043576798423387184774447920739934236584823 \\ & 8242811981638150106748104516603773060562016196762561338441436038339044149526 \\ & 3443219011465754445417842402092461651572335077870774981712577246796292638635 \\ & 6373289912154831438167899885040445364023527381951378636564391212010397122822 \\ & 120720357. \end{aligned}$$

This is much more difficult. The reason it becomes very difficult very quickly is that, for all known classical factoring algorithms, the runtime scales exponentially in the number of bits. It could be that there exists a fast classical algorithm for factoring, but no one has been able to discover one in over three decades of trying to break RSA.

There is another means of breaking RSA which involves finding the period of a special function known as the modular exponential function. This is what the quantum part of Shor's algorithm does. The purpose of the next section is to relate the problem of factoring to this problem of period finding.

[4]I got this number from https://en.wikipedia.org/wiki/RSA_numbers, which is an interesting read about a contest for factoring such numbers and the outcomes of the contest.

12.2.2 Diffie-Hellman-Merkle

Before this, we'll mention one other important fact *extremely* briefly—another popular public key cryptosystem known as Diffie-Hellman[5] can be broken by computing a discrete logarithm. As suggest in the title of his paper, Peter Shor also devised a quantum algorithm for this. Not one, but two! of the biggest public key cryptosystems hacked with a (hypothetical) quantum computer.

This is all that I'll say about Diffie-Hellman and discrete logs—when I refer to Shor's algorithm, I mean the one for factoring. (Well, period-finding, really, as we'll soon see.)

12.3 Reduction of factoring to period finding

This discussion involves a bit of group theory and number theory—physicists be warned. At face value, the factoring problem, formally defined below, appears to have nothing at all to do with period finding.

Definition 12.2 (Factoring problem). Let p and q be prime numbers. The **factoring problem** is: Given $n = pq$, determine p and q.

12.3.1 A function that's periodic

Let's see how we can relate the factoring problem to period finding. Let $(\mathbb{Z}_n, \times)$ denote the multiplicative group of integers modulo n. For simplicity of notation, we'll just denote this group as $\mathbb{Z}_n$. This group consists of integers that are relatively prime to n.

Example 5: The group $\mathbb{Z}_{15}$.
If you're not familiar with $\mathbb{Z}_n$, this example should help. Let's enumerate the elements of $\mathbb{Z}_{15}$. The candidates are $1, 2, 3, ..., 15$. Certainly $1 \in \mathbb{Z}_{15}$ since 1 is relatively prime to any integer. Also $2 \in \mathbb{Z}_{15}$ since 2 and 15 are relatively prime. (In fact, any prime is relatively prime to any other integer.) However, $3 \notin \mathbb{Z}_{15}$ since $\gcd(3, 15) = 3$, thus they are not relatively prime. One can continue in this manner to show that

$$\mathbb{Z}_{15} = (\{1, 2, 4, 7, 8, 11, 13, 14\}, \times). \tag{12.8}$$

Exercise 84: Prove that (12.8) is a group.

Exercise 85: Let p be prime. Show that the elements of $\mathbb{Z}_p$ are all integers modulo p.

How many elements are going to be in $\mathbb{Z}_n$? The answer is known from number theory for the case $n = pq$.

Theorem 12.2 (Cardinality of $\mathbb{Z}_{pq}$). Let $n = pq$ with p and q prime. Then, $|\mathbb{Z}_n| = \phi(n) = (p - 1)(q - 1)$.

Exercise 86: Count the number of elements in $\mathbb{Z}_{15}$ from (12.8). Show this number is equal to $\phi(15)$.

Exercise 87: List the elements of the group $\mathbb{Z}_{21}$ and show that there are exactly $\phi(21) = 12$ of them.

We'll now define the function whose period will allow us to factor an integer n.

Definition 12.3 (Modular exponential function). Let $n = pq$ with p and q prime, and let $x \in \mathbb{Z}_n$. The **modular exponential function** $f_x : \mathbb{Z}_+ \to \mathbb{Z}_n$ is defined by

$$f_x(r) := x^r \bmod n. \tag{12.9}$$

We implicitly claimed that this function is periodic above. Why is this the case? There's (at least) two ways to see this. The first relies on the fact that $\mathbb{Z}_n$ is finite and the following theorem from group theory:

[5] More properly Diffie-Hellman-Merkle, but you'll often see just Diffie-Hellman.

Theorem 12.3. Let G be a finite group. Then, for any $g \in G$, we have $g^{|G|} = 1_G$.

Proof. Consider the subgroup $\langle g \rangle$ generated by g. By Lagrange's theorem, $|\langle g \rangle|$ divides $|G|$, hence $|g| \leq |G|$. By definition of order, $g^{|g|} = 1_G$, so it follows that $g^{|G|} = 1_G$. □

Thus, the modular exponential function satisfies

$$f_x(r + |G|) = x^r x^{|G|} \text{ mod } n = x^r \text{ mod } n = f_x(r). \tag{12.10}$$

Here, $G = \mathbb{Z}_n$, so we know by Theorem 12.2 that $|G| = |\mathbb{Z}_n| = \phi(n)$.

The second way to see that f is periodic actually uses the above fact directly by means of Euler's Theorem (12.6). Since $x \in \mathbb{Z}_n$, we know that x and n are relatively prime, so this theorem applies. In either case, we have shown the following important result:

Theorem 12.4 (The modular exponential function is periodic). Let f_x be as in Definition 12.3. Then,

$$f_x(r + \phi(n)) = f_x(r). \tag{12.11}$$

Exercise 88: The group $\mathbb{Z}_6$ consists of the elements $\{1, 5\}$ so that $|\mathbb{Z}_6| = 2$. Show that $1^{|\mathbb{Z}_2|} = 1$ and $5^{|\mathbb{Z}_2|} = 1$ modulo 6. This is an example of Theorem 12.3.

Exercise 89: Do the same exercise as above but start with $\mathbb{Z}_{10}$.

12.3.2 What does the period tell us?

At this point, you may think I've left the reservation like a proper theoretical mathematician. The whole point was to somehow relate factoring to period finding. We now have this strange modular exponential function, which we've shown to be periodic, but what the heck does this have to do with factoring?

The relationship is as follows. Suppose we determine $\phi(n)$ by *determining the period of* f_x. How does this help us factor n? We know that $n = pq$ and

$$\phi(n) = (p-1)(q-1) = pq - p - q + 1 = n - p - q + 1. \tag{12.12}$$

Recall that we know n—Alice published this to the world with her public key $P = (e, n)$! This gives us one equation

$$p + q = n - \phi(n) + 1 \tag{12.13}$$

relating p and q to known quantities. This isn't enough to determine p and q individually, however—we need one more piece of information. What could this be? Well, we also know

$$pq = n. \tag{12.14}$$

This will work! Letting $m := n - \phi(n) - 1$, we can thus write

$$p^2 - mp + n = 0, \tag{12.15}$$

which can be solved for with your favorite method for solving quadratic equations—completing the square, quadratic formula, etc. In any event, the answer is

$$p = \frac{m}{2} - \frac{1}{2}\sqrt{m^2 - 4n} \tag{12.16}$$

$$q = m - p. \tag{12.17}$$

So there it is—we've just determined p and q (i.e., factored n) by determining the period of the modular exponential function. How can we determine the period, you ask? Enter the quantum part of Shor's algorithm[6]!

There's one important subtlety left, however. Although (12.11) is a true statement, the period of the modular exponential function can be smaller than $\phi(n)$. Consider the following example.

[6]You may have noticed this seminar has been quite classical up to this point.

Example 6: The period of f_x can be smaller than $\phi(n)$.

In an earlier example, we considered the group $\mathbb{Z}_{15}$. It's easy to compute that $\phi(15) = 8$, so we know that $f_x(r + 8) = f_x(r)$. However, the *fundamental period* of f_x in this case is smaller than 8. It's 4, as we'll now see.

The elements of the group are $\mathbb{Z}_{15} = \{1, 2, 4, 7, 8, 11, 13, 14\}$ as previously shown. You're asked to verify in the exercise below that $g^4 \bmod 15 = 1$ for each $g \in \mathbb{Z}_{15}$. This shows that the fundamental period of f_x is 4, as claimed. (Technically, one must check that the period isn't 2 or 3 to show 4 is the *fundamental period*, but these are quite simple checks.)

Exercise 90: For each element g in $\mathbb{Z}_{15} = \{1, 2, 4, 7, 8, 11, 13, 14\}$, show that $g^4 \bmod 15 = 1$. A (classical) computer may be useful.

Let s denote the fundamental period of f_x. As we have shown above, $s \le \phi(n)$, but certainly s evenly divides $\phi(n)$ in light of (12.11). That is,

$$s \mid \phi(n) \tag{12.18}$$

This is a key fact that will let us determine the factors p and q as above.

To enumerate this process, suppose we learn that the period of f_x is s. This is equivalent to say that

$$x^s = 1 \bmod n. \tag{12.19}$$

Now, suppose s is even. One may expect that this happens roughly half the time[7]. Since s is even, we may factor the above equation as

$$(x^{s/2} - 1)(x^{s/2} + 1) = 0 \bmod n. \tag{12.20}$$

This is equivalent to saying

$$(x^{s/2} - 1)(x^{s/2} + 1) = tn \tag{12.21}$$

for some integer t.

For clarity, suppose $t = 1$ for the moment. What does this tell us about the quantities $(x^{s/2} - 1)$ and $(x^{s/2} + 1)$? Well, they're precisely p and q! This is because this equation is of the form $n =$ something times something else, and we know $n = pq$. So, this is the only possibility.

Now suppose $t > 1$. The idea is similar, but now p could be a factor of $(x^{s/2} - 1)$ or of $(x^{s/2} + 1)$. That is, if $t = ab$, then this equation tells us something like

$$(ap)(bq) = tn. \tag{12.22}$$

Of course we don't know what t is going to be. Regardless, we can still say in any case that, up to relabeling p and q,

$$p = \gcd((x^{s/2} + 1), n) \tag{12.23}$$

$$q = \gcd((x^{s/2} - 1), n). \tag{12.24}$$

Thus, finding the period s of f_x allows us to determine p and q, even when $s < \phi(n)$. Thus, we have shown that factoring reduces to finding the period of the modular exponential function f_x.

12.3.3 Euclid's algorithm

There's *one* more detail to cover. Above we just said "we can compute the greatest common divisor to figure out p and q." While this is true, we're talking about algorithms here—how do we know we can do it efficiently? Do we need to invent some fast quantum algorithm to do this? Fortunately, the answer is no. Computing the greatest common divisor has an efficient algorithm that was known by Euclid in c. 300 B.C.

Euclid's algorithm (also known as the *Euclidean algorithm*) is best explained in pseudo-code. Actually, it's so simple I'll include actual Python code. Let a and b be positive integers. The g.c.d. can be computed by the following function:

[7]There's a precise theorem here which I won't get into. Ok, fine, I'll get into the statement, but not the proof. The statement of the theorem is as follows. Given any n, if x is randomly chosen from $\mathbb{Z}_n$, then with probability at least 3/8, (1) s is even and (2) neither $(s^{s/2} - 1)$ nor $(s^{s/2} + 1)$ is a multiple of n.

```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)
```

(Recall that the % operator in Python is the modulus operator.)

We won't go into why this algorithm works or derive it's runtime, but we will state that (1) it works and (2) the runtime scales as $O(\log(ab))$. That is, linear in the number of bits of a and b. This is certainly efficient, so we can be safe in saying that "we can just compute the greatest common divisor to figure out p and q."

12.4 Summary

As anticipated, this turned out to be a long seminar. There's a lot of intricate details in Shor's algorithm, as we have seen, and these take some time to explain. And we haven't even made it to the quantum part! This was all just setting the stage. Don't worry, though—the quantum part is, in my opinion, simpler than the above exposition.

Here's a recap of what we know so far:

- Factoring is a useful, interesting problem because of the RSA cryptosystem.
- The modular exponentiation function f_x is periodic.
- Determining the period of f_x allows us to factor n into p and q.

We don't know how we can determine the period of f_x yet, but we'll figure out how we can do this (efficiently) with a quantum computer in the next seminar(s).

Exercise 91: Write a program to generate RSA keys for an input number of bits, e.g. RSA-128, RSA-256, etc.