# Math Circles – Cryptography

Nickolas Rollick – nrollick@uwaterloo.ca

February 21, 2018

## Introduction

In our next two meetings, we will explore the math behind *cryptography*, the science of sending secret messages. This week, we'll look at what was known before World War II, along with the big post-war idea that revolutionized cryptography. In our second meeting (two weeks from now), we'll look at the RSA cryptosystem, the amazing idea that made it possible to send credit card numbers and other confidential data over the internet. Of course, you'll have plenty of opportunity to play with these ideas yourself in the question sheets...

## Basics

Essentially, cryptography is about scrambling a message you don't want people to read, in such a way that only the intended recipient can unscramble and read it. The scrambling process is called *encryption*, and the descrambling process is called *decryption*. Before encryption, the message is called the *plaintext*, and after encryption, it is called the *ciphertext*. Sounds like a spy movie!

Why would anybody want to encrypt a message? Well, you can imagine a general who wants to send her battle strategy to her soldiers, but knows it would be very bad if the enemy intercepted it. If the message is encrypted first, it doesn't matter if it falls into enemy hands, since it looks like gibberish. Only the lieutenant on the battlefield with the decryption key has the ability to recover the original message from the gibberish (at least, that's the idea). But there are other, less poetic applications: say you want to send your credit card number to a store over the internet, but don't want anyone but the store to know what it is.

Before the 1970s, encryption always worked the same way. Two people wishing to communicate (who for some reason are usually called Alice and Bob) would have to exchange a secret *key* beforehand. Alice uses this key to convert the plaintext into ciphertext, and when Bob receives the message, he uses the same key to change the ciphertext back into plaintext. Anyone reading the ciphertext without the key is faced with a seemingly meaningless string of characters.

At the moment, there is a big question remaining: how do we *actually* encrypt and decrypt a message? In essence, there are two approaches: *transposition* and *substitution*. Encrypting by transposition means rearranging the characters of the plaintext to get the ciphertext. Let's take a look at one type of example. Suppose that Alice wants to send Bob the message:

Retreat immediately.

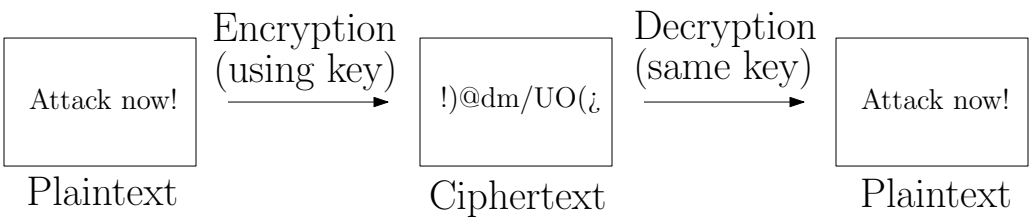


Figure 1: Encryption before the 1970s

Alice and Bob would agree on a certain number of letters, say five. First, Alice removes all the spaces and punctuation and makes the letters uppercase to remove any distinguishing features, leaving her with

RETREATIMMEDIATELY

Next, she arranges the letters in rows of five, the number agreed with Bob:

| | | | | |
|---|---|---|---|---|
| R | E | T | R | E |
| A | T | I | M | M |
| E | D | I | A | T |
| E | L | Y | | |

Finally, Alice arranges the columns in some order pre-arranged with Bob, say last to first. She goes down each column and writes out the letters, resulting in the gibberish ciphertext:

EMTRMATIIYETDLRAEE

When Bob receives this message, he just has to reverse the procedure. He knows the letters were arranged in rows of five. Since the message has 18 letters, the last two columns will be one letter shorter than the rest. Because Bob knows the columns were taken in reverse order, from last to first, he puts the first three letters of the message in the last column, the next three in the fourth column, and so on, resulting in the arrangement of letters Alice started with. By reading the letters across the rows, Bob can get the original message back.

As you will see on the question sheet, there are many reasons why using straight transposition to encrypt a message is not a good idea. Historically, *substitution* was the preferred technique. When encrypting using substitution, each letter in the message is replaced with a different symbol (usually also a letter), but the order of the letters remains the same. The simplest example is known as the *shift cipher* and goes back thousands of years. The idea is to encrypt by shifting every letter of the alphabet by a certain number of places. For example, suppose we wanted to shift by 2. We would replace every "A" in the message by "C", which is two places further in the alphabet. Similarly, "B" would be replaced by "D", "C" replaced by "E", and so on, finishing by replacing "Z" with "B" (looping around to the beginning of the alphabet). We can write this out neatly in the form of a table:

| Plaintext | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext | C | D | E | F | G | H | I | J | K | L | M | N | O |
| Plaintext | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Ciphertext | P | Q | R | S | T | U | V | W | X | Y | Z | A | B |

The "plaintext" row of the table represents a letter in the plaintext, and the "ciphertext" row represents the corresponding letter to use in the ciphertext. For example, suppose Alice wants to use this shift cipher to send RETREATIMMEDIATELY to Bob. She takes the first letter, R, and looks it up in the plaintext row of the table. She replaces R with T, the letter below it. Next, she takes the second letter of the message, E, and replaces it with G, the letter below the E in the plaintext row. Doing this for the entire message, Alice sends the following ciphertext to Bob:

TGVTGCVKOOGFKCVGNA

To decrypt it, Bob just has to look up each letter of the ciphertext in the bottom row of the table and replace it with the corresponding letter in the top row of the table (shifting each letter two places back in the alphabet).

In the shift cipher, the number of places to shift by forms the key. For instance, Alice and Bob could decide to shift by 10, so that "A" is replaced by "K", ten letters later in the alphabet, and similarly for the other letters. It is easy to draw up a corresponding table, and encrypt away:

| Plaintext | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| Plaintext | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Ciphertext | X | Y | Z | A | B | C | D | E | F | G | H | I | J |

While this looks like a promising form of encryption, there are in fact several major problems with the security of the shift cipher. The biggest one is that there are only 25 possible keys, one for every shift size. Of course, we don't count the shift by 0 places, which does nothing to the message. If someone intercepts the message and suspects a shift cipher, they only have to try all 25 possible keys. Usually, only one does not lead to a string of gibberish, and that's the plaintext.

Thankfully, this problem is easily fixed. Rather than just shifting each letter a certain number of places forward in the alphabet, we can instead replace each letter in a less predictable fashion. In other words, we can allow any arrangement of the letters of the alphabet in the ciphertext row of the encryption table. This more general technique has a fancy name: the *monoalphabetic substitution cipher*. For example, we could use this table:

| Plaintext | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext | G | Z | I | C | N | P | F | H | X | D | L | Q | T |
| Plaintext | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Ciphertext | M | Y | W | A | J | B | S | R | V | U | K | E | O |

By allowing for any arrangement of the ciphertext letters, the possible number of keys increases dramatically. Along with that, the time required to try them all is hopelessly huge. You'll later work out just how long it would take, but suffice it to say it is no longer a good option for someone who intercepts the ciphertext. That's good news, right? The monoalphabetic substitution cipher looks unbreakable... Of course, you already know that can't be true, or else we could wrap up now and go home!

Before we try breaking this cipher, though, it's time for you to get your hands on some problems of your own. At this point, you have the tools to attempt Questions 1–4.

## Cryptanalysis of Monoalphabetic Substitution

Up until now, we've mostly put ourselves in the shoes of someone wanting to send and receive encrypted messages. It's now time to think like someone who wants to defeat the system of encryption. Mathematicians call this process *cryptanalysis*. So, suppose that you've gotten your hands on an encrypted message, and you're curious to know what it says. Maybe it contains secret pieces of wisdom?

QHFKHGKHGIKSHECFKSLZSSAFKAQZJPKCZSHPSLZSQSZEDQTQZEEHTPEAUKQNTSAPZ
SLKPNKEAKUKSLZSSAFKAQZIHFJZGAHGVLHOHKQVASLTQHGSLKBHTPGKXPKFAGCQTQS
HILKPAQLKUKPXFHFKGSNKIZTQKSLKXEEGKUKPIHFKZOZAGVLZSVKEKZUKNKLAGCAQG
HSZQAFJHPSZGSZQLHVVKEAUKC

You suspect that this ciphertext was the result of a monoalphabetic substitution cipher. How should we start trying to decrypt it without the key? Well, the biggest downfall of this cipher is that the ciphertext still has the irregularities of language. Two "e"s or "t"s in a row in the plaintext will still be encrypted as two of the same letter in a row. Even more helpfully, the frequencies of the letters are preserved, even if the letters themselves might be relabelled. For example, "e", "t", and "a" are the most common letters in the English language, in that order. Therefore, if the ciphertext is large enough, it is reasonable to assume that the three most common letters in the ciphertext correspond to "e", "t", and "a".

With this in mind, we count how many times each letter appears in the ciphertext, and compare those frequencies to letter frequencies for the English language. Here are the frequency counts for this ciphertext:

| Letter | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Occurrences | 16 | 1 | 5 | 1 | 10 | 10 | 13 | 18 | 5 | 3 | 32 | 13 | 0 |
| Frequency (%) | 7.2 | 0.5 | 2.3 | 0.5 | 4.5 | 4.5 | 5.9 | 8.1 | 2.3 | 1.3 | 14.4 | 5.9 | 0.0 |
| Letter | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Occurrences | 4 | 2 | 11 | 16 | 0 | 22 | 7 | 6 | 6 | 0 | 3 | 0 | 18 |
| Frequency (%) | 1.8 | 0.9 | 5.0 | 7.2 | 0.0 | 10.0 | 3.2 | 2.7 | 2.7 | 0.0 | 1.3 | 0.0 | 8.1 |

Now, we compare this information to a table of letter frequencies for the English language, rounded to the nearest tenth of a percent. It is possible to make your own, but I obtained this one at `https://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html`.

| Letter | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frequency (%) | 8.1 | 1.5 | 2.7 | 4.3 | 12.0 | 2.3 | 2.0 | 5.9 | 7.3 | 0.1 | 0.7 | 4.0 | 2.6 |
| Letter | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Frequency (%) | 7.0 | 7.7 | 1.8 | 0.1 | 6.0 | 6.3 | 9.1 | 2.9 | 1.1 | 2.1 | 0.2 | 2.1 | 0.1 |

From this, we see that "e" is the most common English letter by a comfortable margin. Since K is the most common letter of the ciphertext, we will guess that it corresponds to "e" in the plaintext. The next most common ciphertext letter is S, while the next most common English letter is "t". So, we guess that "t" is encrypted as S. Using these guesses, we transform the corresponding letters in the ciphertext and see where we end up. To make things clear, we'll use lowercase letters to indicate our guesses at the plaintext.

QHFeHGeHGIetHECFetLZttAFeAQZJPeCZtHPtLZtQtZEDQTQZEEHTPEAUeQNTtAPZtLePNeEAe
UetLZttAFeAQZIHFJZGAHGVLHOHeQVAtLTQHGtLeBHTPGeXPeFAGCQTQtHILePAQLeUePXFHF
eGtNeIZTQetLeXEEGeUePIHFeZOZAGVLZtVeEeZUeNeLAGCAQGHtZQAFJHPtZGtZQLHVVeEAUeC

So far, so good. But now we have a bit of a problem: *two* letters in the ciphertext are third most common - both Z and H have the same frequency. Which one should correspond to "a", the third most common English letter? Clearly, following the frequencies alone will not give us a complete decryption. We expect things like this to happen with smaller messages: the bigger the message, the more it will behave like the average frequencies.

We don't have much information to go on, so we'll just have to take a guess. If we later get a piece of plaintext that looks like nonsense, we'll have to backtrack and try again. Let's say that Z is the encryption of "a" and H is the encryption of the fourth most common English letter, "o". This leads to the following:

QoFeoGeoGIetoECFetLattAFeAQaJPeCatoPtLatQtaEDQTQaEEoTPEAUeQNTtAPatLePNeEAeUetL
attAFeAQaIoFJaGAoGVLoOoeQVAtLTQoGtLeBoTPGeXPeFAGCQTQtoILePAQLeUePXFoFeGtNeIaT
QetLeXEEGeUePIoFeaOaAGVLatVeEeaUeNeLAGCAQGotaQAFJoPtaGtaQLoVVeEAUeC

None of the plaintext looks like complete gibberish yet, so let's keep moving on. Now, we run into a dilemma again, since the next two most common ciphertext letters, A and Q, both occur with 7.2% frequency. The next two most common English letters are "i" and "n", but which of A and Q corresponds to "i" and which one to "n"? At this stage, we'll just take another guess. Let's say the letter "i" in the plaintext was replaced with A in the ciphertext, and "n" was replaced with Q. If we test out this guess, we end up with:

noFeoGeoGIetoECFetLattiFeinaJPeCatoPtLatntaEDnTnaEEoTPEiUenNTtiPatLePNeEieUetLattiFeina
IoFJaGioGVLoOoenVitLTnoGtLeBoTPGeXPeFiGCnTntoILePinLeUePXFoFeGtNeIaTnetLeXEEGeUePIo
FeaOaiGVLatVeEeaUeNeLiGCinGotaniFJoPtaGtanLoVVeEiUeC

Before accepting this guess as correct, we have to be careful: any section of plaintext that looks like gibberish could tell us we've made a wrong move. Look at the first line: right now, we have a chunk of plaintext reading "atnta". Even inserting word breaks, there seems to be no way to make sense of this chunk of text, unless it were talking about the explosive "tnt"! Notice this is true even if we made the wrong guess about Z and H earlier. Making the other guess would just swap "a" and "o" in the message, and we would still have the gibberish text "otnto".

Let's see what happens to this part of the message if we make the other guess. In other words, assume "i" was replaced by Q, and "n" was replaced by A. This has the effect of swapping all the "i"s and "n"s in the text above. Now this string of letters becomes "atita", which looks a little better. But elsewhere in the first line, we will get "attnFenia". No matter what letter F represents, it seems unlikely that sensible English words can come from that string of text.

Uh-oh! Following the frequencies seems to go wrong both ways! Therefore, we do what a mathematician does: go back and question our assumptions. Both of our guesses led to really strange strings of text, so maybe the frequencies in the ciphertext do not match up exactly with the frequencies of normal English. This is common for small ciphertexts, which is why comparing letter frequencies alone is not good enough. So, let's take back our guesses about these two letters:

QoFeoGeoGIetoECFetLattAFeAQaJPeCatoPtLatQtaEDQTQaEEoTPEAUeQNTtAPatLePNeEAeUetL
attAFeAQaIoFJaGAoGVLoOoeQVAtLTQoGtLeBoTPGeXPeFAGCQTQtoILePAQLeUePXFoFeGtNeIaT
QetLeXEEGeUePIoFeaOaAGVLatVeEeaUeNeLAGCAQGotaQAFJoPtaGtaQLoVVeEAUeC

Blindly following the frequencies didn't work, so let's try something a little more clever. Since "the" and "that" are common words in English, we can try to figure out what letter is used to encrypt "h" by looking for blocks of text of the form t?e and t?at. There are a number of times where tLat and tLe shows up above (enough times to be convincing), so let's guess that "h" has been replaced with L. Turning all the L's into lowercase h's gives us

QoFeoGeoGIetoECFethattAFeAQaJPeCatoPthatQtaEDQTQaEEoTPEAUeQNTtAPathePNeEAeUethat
tAFeAQaIoFJaGAoGVhoOoeQVAthTQoGtheBoTPGeXPeFAGCQTQtoIhePAQheUePXFoFeGtNeIaTQe
theXEEGeUePIoFeaOaAGVhatVeEeaUeNehAGCAQGotaQAFJoPtaGtaQhoVVeEAUeC

Seeing "that" showing up so many times in the above tells us we probably made the right guess when we said "a" was encrypted as Z earlier. Making the other guess would mean that "thot" shows up a bunch of times, which would be rather strange.

Now, we want a better approach to finding the encryptions of the next most common letters. After "e", "t", "a", and "o", there are four other very common letters, with frequencies between 6% and 8%. These letters are "i", "n", "s", and "r".

The corresponding ciphertext letters should still be fairly common, and the next four common letters in the ciphertext are A, Q, G and L. We've already matched up L in the ciphertext with "h" in the plaintext, so we're left with A, Q, and G. We want to match these up with three out of the four common plaintext letters. It may help to try to find the encryptions of these letters in pairs. For instance, "in" happens far more often than "ni", and "rs" is more common than "sr". Common pairs of A, Q, and G happening in the ciphertext include AQ and AG, and AQG shows up at one point. Given that A keeps showing up first, it seems more likely that it corresponds to a vowel, and "i" is the only candidate. Let's try replacing the A's with plaintext "i"s.

QoFeoGeoGIetoECFethattiFeiQaJPeCatoPthatQtaEDQTQaEEoTPEiUeQNTtiPathePNeEieUethattiFei
QaIoFJaGioGVhoOoeQVithTQoGtheBoTPGeXPeFiGCQTQtoIhePiQheUePXFoFeGtNeIaTQetheXEEGeU
ePIoFeaOaiGVhatVeEeaUeNehiGCiQGotaQiFJoPtaGtaQhoVVeEiUeC

Looking good! Now, we still want to find the plaintext letters corresponding to Q and G. We can get some substantial hints from the first line above. For instance, looking at "thatQta" shows that substituting Q for "r" or "n" makes no sense, while substituting it for "s" might work. Similarly, looking at "eoGeoG" right at the beginning of the first line, replacing the G's with "r"s leads to "eoreor", while replacing it with "n" produces "eoneon". Neither one is necessarily gibberish with the right word breaks, but the second option seems better, since the first forces the plaintext to include the unusual word "ore". So, let's take a guess that "s" was replaced by Q, and that "n" was replaced by G. Now we start to see meaningful chunks of text emerge:

soFeoneonIetoECFethattiFeisaJPeCatoPthatstaEDsTsaEEoTPEiUesNTtiPathePNeEieUethattiFeisaIoF
JanionVhoOoesVithTsontheBoTPneXPeFinCsTstoIhePisheUePXFoFentNeIaTsetheXEEneUePIoFeaOai
nVhatVeEeaUeNehinCisnotasiFJoPtantashoVVeEiUeC

Everything still seems to make sense, and we have now decrypted a large enough chunk of the ciphertext that the remaining deductions come quickly. Looking at "Vho" in the second line and "VhatVe" in the last line suggests V should be replaced by "w". There are many further examples, but I'll let you finish the encryption yourselves on the question sheet, because I'm sure you're tired of me blathering on!

This technique of using letter frequencies to do cryptanalysis is called *frequency analysis*. The earliest known author to describe it is a ninth-century Arab scientist with an incredibly impressive name: Abū Yūsūf Ya'qūb ibn Is-hāq ibn as-Sabbāh ibn 'omrān ibn Ismaīl al-Kindī. Before this time, everyone believed that a monoalphabetic substitution cipher was uncrackable (such beliefs are common in the history of cryptography, and usually wrong). All of a sudden, this cipher was no longer secure, if someone with the right know-how intercepted a message.

## The Onetime Pad

Nonetheless, it would take more than 1000 years before anyone substantially improved on the monoalphabetic substitution cipher. The idea behind the breakthrough is this: frequency analysis works because each letter in the plaintext is substituted according to the same rule. What if each plaintext letter was encrypted using a *different* rule? For example, what if each letter is encrypted by a shift cipher, but the shift is different each time? Ciphers like this are called *polyalphabetic substitution ciphers*.

If the different shifts are selected completely randomly, the result is called the *onetime pad cipher*. A version of this cipher was first described in 1882 by an American named Frank Miller, but ironically, no one paid attention. The technique was not used until its rediscovery in a different form in 1917, by a research engineer named Gilbert Vernam. To use the onetime pad cipher, Alice and Bob must share a large string of randomly generated letters, at least as long as the message. Suppose Alice's message to Bob is "The prime minister is ill". As usual, she strips away spaces and makes all the letters uppercase:

THEPRIMEMINISTERISILL

There are 21 letters in the message, so Alice and Bob need to share a random string of 21 letters as the key. For the sake of example, suppose the key is MSWZTFAOHAOHAJIERZYAL. The first letter of the key is M, so Alice draws up an encryption table for a shift cipher, with M being the first letter in the ciphertext row:

| Plaintext | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext | M | N | O | P | Q | R | S | T | U | V | W | X | Y |
| Plaintext | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Ciphertext | Z | A | B | C | D | E | F | G | H | I | J | K | L |

She uses this table to encrypt the first letter of the plaintext (in this case, T), and finds that the first letter of the ciphertext should be F. The second letter of the key is S, which means we use the encryption table starting with S in the second row.

| Plaintext | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext | S | T | U | V | W | X | Y | Z | A | B | C | D | E |
| Plaintext | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Ciphertext | F | G | H | I | J | K | L | M | N | O | P | Q | R |

The second plaintext letter, H, is therefore encrypted as Z. Proceeding in this manner, Alice ends up sending the following message to Bob:

FZAOKNMSTIBPSCMVZRGLW

For Bob to decrypt using the shared key, he just needs to reverse each shift cipher on each letter. From the perspective of breaking the cipher, notice that frequency analysis is no good here. For instance, the three E's in the plaintext are all encrypted differently: the first one becomes an A, the second becomes an S, and the last one becomes an M. In fact, as long as the key was selected *completely* at random, decrypting without the key is literally hopeless. If every possible random key is equally likely, then *every* plaintext could become this ciphertext if the right key is used. Mathematicians have actually proved that the onetime pad cipher offers *perfect secrecy*. This has a specific technical meaning, but essentially tells us that cryptanalysis of this cipher is impossible.

So, problem solved, right? The onetime pad is the perfect cipher! Well, maybe theoretically, but there are a couple practical hurdles making the onetime pad less than ideal. First, perfect secrecy is only guaranteed if the key is truly selected randomly, and the key can be used *only once*. Breaking either of these rules compromises the security of the cipher (as we'll see in the question sheet). So, to use the onetime pad properly, a large amount of randomness is required in order to generate the keys. Such randomness is rare in nature and hard to harness, making the process costly in terms of time and equipment. This means adoption of onetime pad ciphers has been limited mainly to top-secret diplomatic applications (such as the hotline between Moscow and Washington D.C. during the Cold War).

Because the onetime pad was so impractical, the search continued for ciphers that were more secure than monoalphabetic substitution, but not as cumbersome as the onetime pad. By this point, people were relying

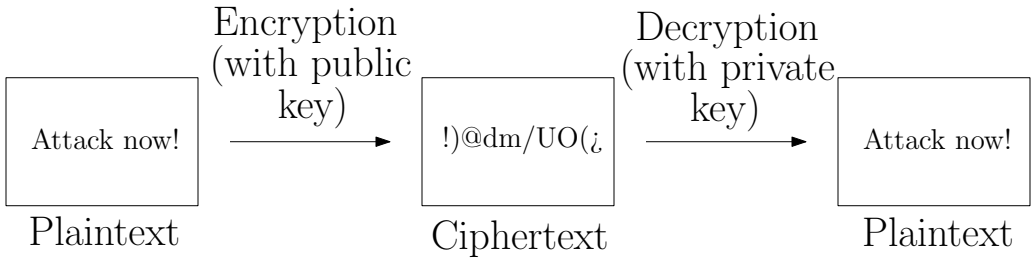


Figure 2: Public Key Cryptography

on machines (and eventually computers) more and more, so it made sense that the new breed of ciphers would be mechanized. I won't have time to tell you about these ciphers, but you will get to see an Enigma machine next week. This is a cipher machine used heavily by the German army during World War II, and breaking Enigma had a significant impact on the course of the war. If you're interested, two mechanized post-war ciphers that were (and are) widely used are DES (Data Encryption Standard) and AES (Advanced Encryption Standard). While DES is no longer considered secure, AES is very fast, appears to be very difficult to break, and is used in many modern applications.

Unfortunately, the onetime pad and all these later ciphers suffer from the same problem. In order to encrypt messages, both Alice and Bob need to get their hands on the key. This is not a problem if Alice and Bob can physically meet to exchange the key before they have to communicate, but becomes a huge problem if they live on opposite sides of the planet. If they phoned or e-mailed each other to exchange keys, anyone could tap the line and learn the key as well, because the first contact between Alice and Bob would necessarily be unencrypted.

Alice and Bob could use a trusted courier to carry a key from one to the other, but this is way too expensive and troublesome for the typical person. Without a solution to this problem, making purchases over the internet would be impossible. There is no way that a company can be expected to securely exchange keys with every potential customer before the customer sends along credit card details...

## Public Key Cryptography

Thankfully for all of us, a solution to this problem was discovered in the 1970s. The idea came independently from Whitfield Diffie, an American academic, and James Ellis, working for the British government. For the longest time, credit went to Diffie, because Ellis could not report on his government work to the general public. That's the nature of this top-secret business!

The big realization is that encryption and decryption do not have to use the same key. Suppose for a moment that Alice could generate a pair of keys, one *private key* and one *public key*. As the name suggests, Alice would keep the private key secret and never reveal it to anybody. The public key would be published for everyone to see. Anyone wanting to encrypt a message to Alice would use her public key to generate the ciphertext. Now here's the catch: Alice's public key is good for encryption only - it cannot be used for decryption! The private key, on the other hand, is good for *de*cryption only, and cannot be used to encrypt messages. So, Alice (and only Alice) can decrypt ciphertexts encrypted using her public key. This new method of encryption is usually called *public key cryptography*, and the old way of doing things is called *symmetric key cryptography*.

In Simon Singh's excellent book on cryptography, *The Code Book*, he gives a great metaphor explaining how public key cryptography works. Suppose Alice has an unlimited supply of identical open padlocks (public keys), but only one copy of the key opening these padlocks (the private key). Anyone wanting to send a secret message to Alice would pick up one of her (unlocked) padlocks from some public location. Then, they can put the message in a box and lock the padlock. Since Alice has the only key, she is the only one who can remove the padlock and read the message in the box.

Notice that public key cryptography removes the problem of getting keys from Alice to Bob. Alice can just publish her public key online, and Bob can download it at his convenience in order to send encrypted messages to Alice. Alice doesn't have to worry that her public key is so widely available, because it doesn't

help with decrypting messages. In the same way, if I want to encrypt my credit card number to send to a company online, I just need to download the company's public key and I'm good to go.

Both Diffie and Ellis started by phrasing public key cryptography in theoretical terms, but there was still work to do. For starters, is it possible to describe a cipher that uses public and private keys? No matter how the cipher works, there are two requirements. Firstly, it must be very hard to decrypt a message, *even knowing the public key*. In other words, the process of encryption must be easy to do, but extremely hard to undo. Secondly, there is a special piece of information (the private key) that suddenly makes undoing the encryption very easy. At first glance, it is unclear whether there are any encryption procedures that will meet both requirements.

Next time, we will take a look at the *RSA cryptosystem*, the first proposed cipher that meets the requirements of public key cryptography. In understanding this cryptosystem, we will need to look at a fascinating branch of math called *number theory*.

In the meantime, it's time to let loose on the question sheet! You are now able to complete the rest of the questions.