An Introduction to Cryptography

James Grime

Introduction

Secret writing has been used for thousands of years; probably for as long as we have had secrets to keep. These may be messages in war, messages between corporations, or just personal secret messages. Even languages may be thought of as a kind of code.

When we send secret messages there are three types of participants. The *sender*, the intended recipient or *receiver*, and possibly the *interceptor* or 'bad guy'. The act of disguising your message is known as *encryption*, and the original message is called the *plaintext*. The encrypted plaintext is known as the *ciphertext* (or *cryptogram*), and the act of turning the ciphertext back into the original plaintext is *decryption*.

Steganography is the act of physically hiding your message; in ancient times you might write your message on the shell of a boiled egg using special ink that would soak through the porous outer shell. When the egg was cracked and peeled, your secret message would be found written on the egg white. In the 20th century, agents involved in espionage would use the microdot, shrinking their entire message smaller than could be seen by the naked eye.

Codes on the other hand involve turning words or phrases into other words. For example, a command such as 'attack at midnight' might simply become 'eagle'. The problem with codes is that they involve carrying a code book, essentially a dictionary, to translate what you want to say into code. If this book falls into enemy hands then your code is revealed and is now useless, and replacing each code book is not something that can be done frequently. Also, a code may not have the flexibility to deal with all messages you might conceivably wish to send.

In comparison, *ciphers* work on the level of the individual letters of your message. Ciphers may replace letters in a message with other letters, or numbers, or symbols. For example, if 'a' becomes 0, 'b' becomes 1, 'c' becomes 2 and so on, then a word like 'secret' becomes '18 4 2 17 4 19'. (Note, counting from 'a' as zero is a necessary convention that we will continue to use later on). This provides a much greater flexibility in our messages, and it will be ciphers that we will be dealing with during this course.

Ciphers comes in two parts: The first part is the *algorithm*; this is simply the method of encryption. The second part is the *key*. The idea is these work together like a lock-and-key and, for the code breaker, having one without the other is just half the problem. The key may change frequently meaning the greater the number of keys the more difficult the cipher becomes break by brute force (an exhaustive check of all possible keys).

Secret writing has always been a constant struggle between the code maker and the code breaker. *Cryptography* is the study of making secret messages, whereas *cryptanalysis* is the study of breaking those secret messages. *Cryptology* is the collective name for both cryptography and cryptanalysis; the word cryptology coming from the Greek words *kryptos* meaning hidden and *logos* meaning word.

1 Monoalphabetic Ciphers

1.1 The Caesar Shift

We begin with a simple cipher as described by Julius Caesar in the Gallic Wars. In it he describes taking two alphabets, from a to z, one above the other. However, the second alphabet is shifted three places to the left. The top alphabet represents letters in the plaintext, while the the alphabet underneath shows what these letters become in the ciphertext. Note, throughout this course I will represent the plaintext in lowercase letters, and the ciphertext in uppercase letters.

plaintext	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
ciphertext																										

So, a plaintext message like

veni, vidi, vici

becomes

in the ciphertext.

Clearly, the shift does not have to be 3, (it can be 4, 5, etc). The idea of shifting the alphabet is the algorithm, and the size of the shift is the key. This is not a very secure cipher since there are only keys (including plain English when you shift by). This cipher can be quickly broken by a simple exhaustive check of all possible keys. (The original strength of this cipher may have come from Caesar's enemies not realising he was using a cipher at all).

Definition. A *set* is a collection of objects (numbers, letters, symbols, or other objects).

Let denote the set of letters alphabet a, b, c, ..., z. Let denote the alphabet of the plaintext, usually . Let denote the ciphertext alphabet, this set may or may not be the same as .

Definition. A *monoalphabetic cipher* is a rule which associates each letter of with a unique letter in . Furthermore, different letters in are sent to different objects in .

Below are three diagrams using the first five letters of the alphabet. The first is not a valid example of a monoalphabetic cipher because one letter in the plaintext alphabet is sent to two different letters in the ciphertext alphabet. The second is not valid because two letters in the plaintext alphabet are sent to the same letter in the ciphertext alphabet. Finally, the third is a valid example of a monoalphabetic cipher.

This means the size of is equal to or greater than the size of , and each letter in is paired with an image in . This will allow us to define an inverse rule from to that will decrypt our messages from the ciphertext back to the original plaintext.

1.2 Modular Arithmetic

In the Caesar Shift we saw how a shift of 3 caused the cipher letters A, B, C to wrap around at the end. Now, instead of the letters let's use the numbers 0 to 25, where a becomes 0, b becomes 1, and so on until z becomes 25.

Then a shift of 3 is equivalent to adding 3 to each number, where , and .

plaintext	0	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
ciphertext																										

This is called *modular arithmetic*. It works like a clock, where (for a 12 hour clock) if you start at o'clock and add three hours it's o'clock. However there is nothing special about the number twelve, in fact we could imagine a clock of any size and for ciphers we will mostly be imagining a clock of size . For this reason modular arithmetic is sometimes called *clock arithmetic*.

Definition. Given integers and and positive integer , we write if:

- strictly divides - written as ; or equivalently
- there exists an integer such that .

Here is called the *modulus* and we say that and are *congruent modulo* .

Example. If we set and , then or .

The set of numbers congruent to modulo are .
For example, the set of numbers congruent to mod are

If , with , then is the remainder of on division by and . Any integer is congruent modulo to exactly one of , as these are all the possible remainders on division by . For example, if then all integers are congruent to one of , , , , modulo .

Our next result shows that addition and multiplication work modulo m.

Proposition 1.2.1 *If* *are integers and* *a positive integer with* *and* *, then* *and* *.*

Proof. By definition, we have and , for some integers . Thus;

so that , and

so that . ■

(Note, the term *theorem* is reserved for important results, the term *proposition* is used for smaller results. Proofs are traditionally set out this way, beginning with 'Proof', and terminating with a small box to show the proof is finished).

The above proposition is extremely useful because it allows us to substitute a value for its remainder, modulo m, when adding and multiplying.

Example. If , , then their remainders modulo are , and:

and

1.3 Additive Ciphers

The Caesar Shift is also known as an *additive cipher*. As we know, there are 26 possible keys. If is the value of your plaintext letter, is the value of your key, then the value of your ciphertext letter is given by;

Example. If we encipher the message 'I like the park' using a key we get;

plaintext:	i		l	i	k	e		t	h	e		p	a	r	k
in numbers:	8		11	8	10	4		19	7	4		15	0	17	10
:															
ciphertext:															

To decrypt a cipher letter we need to find the decryption algorithm, or decryption key, to obtain the original plaintext letter . If our message was sent using an additive cipher we simply subtract the key-value ; or equivalently we need to find the *additive inverse* such that .

Proposition 1.3.1 *If is the encryption key for an additive cipher, then the decryption key .*

Proof. If is our cipher letter, then

Hence, the additive inverse of modulo is . ■

1.4 Common Divisors

Given two non-zero integers and , we call a *common divisor* (or *common factor*) if is an integer that exactly divides and with no remainder, i.e. and . The largest such divisor is called the *greatest common divisor* (*highest common factor*).

Example. The divisors of are . The divisors of are . The common divisors of and are and the greatest common divisor is .

Next we will show a procedure to find the greatest common divisor of two integers.

Proposition 1.4.1 *If and are integers, there exists integers and such that . The set of common divisors of and are the same as the set of common divisors of and .*

Proof. If divides and then also divides , hence divides and .

On the other hand, if divides and then also divides , hence divides and .

This means the common divisors of and are the same as the common divisors of and . ■

Given integers and we may use division with remainder to find , where . We can then set up a chain as follows:

By above, the common divisors of and are the same common divisors of and . Similarly, the common divisors of and are the same common divisors of and . Continue this way down the chain and we conclude that the common divisors of and are the same as the common divisors of and .

In other words, the set of common divisors of and are simply the divisors of , and the greatest divisor of is itself. This procedure is known as *Euclid's Algorithm.*

Example. Let's use Euclid's Algorithm to find the greatest common divisor of and .

Hence, the greatest common divisor of and is .

If is the greatest common divisor of and , we may reverse the steps of Euclid's Algorithm and write in the form for some integers and . This is sometimes known as *Bézout's Identity.*

Example. We will reverse the steps of Euclid's Algorithm in the last example.

We call two non-zero integers and *coprime* if they share no common divisors other than , i.e. their greatest common divisor is , or equivalently there exists integers and such that .

Example. The integers and are coprime because they share no common divisors. Equivalently we may use Euclid's Algorithm to find:

This shows that their greatest common divisor is . We may now reverse the procedure as follows:

Note, if and can be written in the form then and are coprime. This is an equally valid definition of coprime and we will be using that definition repeatedly throughout the course.

1.5 Multiplicative Ciphers

Instead of adding a key-value to our plaintext, we could multiply by a key-value , such that

For example; if then

plaintext:	i		l	i	k	e		t	h	e		p	a	r	k
in numbers:	8		11	8	10	4		19	7	4		15	0	17	10
:															
ciphertext:															

However, we cannot pick any value for . In the example above, by choosing , the letters and in the plaintext are both encrypted as the letter . This violates the definition of a monoalphabetic cipher and would make decryption very hard, or impossible, to do. Which values for work?

In fact we must choose a value that is *coprime* to , i.e. shares no common factors with . In general, if the alphabet has size , then must be coprime to .

Here, the value fails because it shares a common divisor with , namely the common divisor . This will cause more than one letter to be sent to the same value. For example, .

Proposition 1.5.1 *Let integers and be coprime, such that for some integers . Then there exists such that . If is the encryption key for a multiplicative cipher, then the decryption key is .*

Proof. If , then , and

Hence the multiplicative inverse of is .

If is our plaintext letter, and is our cipher letter, then;

as required. ■

The value is the *multiplicative inverse* of modulo . This will always exist if the key-value and the alphabet size are coprime.

There are only valid key-values coprime to , namely .

Example. If then the message 'cab' becomes 'QAV'. To decrypt we need to find the multiplicative inverse of . Earlier, we saw that and are coprime with . This means the multiplicative inverse is . So, to decrypt, we multiply by (modulo 26).

1.6 Affine Cipher

An *affine cipher* is a multiplicative cipher followed by an additive cipher. If is coprime to , and then;

For each of the valid values for we have choices for . So altogether there are possible affine ciphers, for an alphabet of size .

1.7 General Substitution Ciphers

Remember, a monoalphabetic cipher is a one-to-one rule that pairs each letter in the plaintext alphabet with a letter in the ciphertext alphabet. However, not all monoalphabetic ciphers need to be defined by a formula, like those formulas we have seen for the additive, multiplicative and affine ciphers. Instead, we may define a cipher letter-by-letter, by defining where each letter in the plaintext alphabet is sent in the ciphertext alphabet.

Example. Let's use the first five letters a, b, c, d, e. Then we may define a cipher as;

We may represent this with a diagram;

So a plaintext message such as 'bead' becomes in the ciphertext.

To decrypt this message we simply reverse the direction of the arrows;

We may encrypt a message twice, performing one encryption after another, making a *composite* of ciphers. We can represent this by juxtaposing the two diagrams. We may represent the the result on one diagram by following the total path from left to right for each letter. The result is a new monoalphabetic cipher, however the new cipher is no more secure than the original.

How many monoalphabetic ciphers exist? To calculate this answer consider your choices for each step. Building our cipher from scratch, we have choices for the first letter of the cipher. We then have choices left for the second letter, and choices for the third letter, and so on. Continue this way until we reach the final letter of the cipher, for which we will have one choice left that we have to take.

plaintext	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
choices																										

To find out the total number of keys, i.e. the total number of *combinations* of A to Z, we multiply choices together. Multiplying the integers to inclusive is called *factorial* and is written so

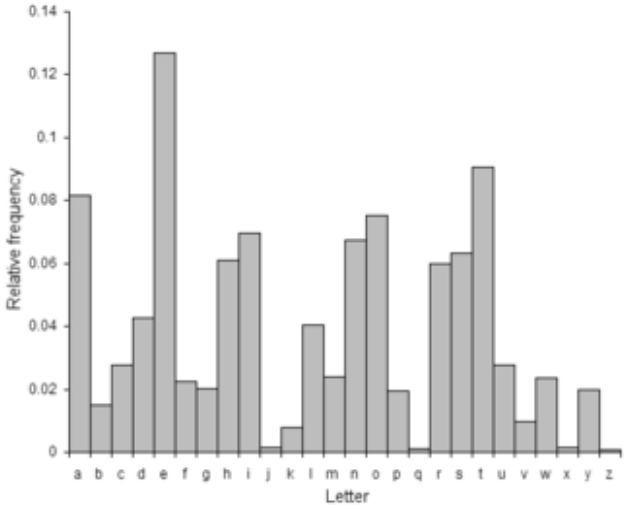
Hence, the total number of key is and is approximately equal to - a very large number indeed. And too many keys to check by brute-force alone.

1.8 Breaking the Cipher: Frequency Analysis

So far we have looked at the roles of the sender and receiver. Now, how will an an interceptor break a monoalphabetic cipher without the key?

Given a long piece of text, one can determine the frequency of each letter. In general, the frequencies of the letters in a given text do not change. Below is the expected frequency of each letter in English:

letter	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
frequency (%)	8.2	1.5	2.8	4.2	12.7	2.2	2.0	6.1	7.0	0.1	0.8	4.0	2.4	6.7	7.5	1.9	0.1	6.0	6.3	9.0	2.8	1.0	2.4	0.1	2.0	0.1



The letters ‘e’, ‘t’ and ‘a’ are the most common, with ‘j’, ‘q’, ‘x’ and ‘z’ the least common. Naturally, frequencies would be different in different languages. Also, frequencies are slightly different depending on the type of message it is, i.e. personal, military or other.

In a monoalphabetic cipher, if the plaintext letter ‘e’ becomes a different letter, say ‘w’, then ‘w’ will now be the most common letter in the ciphertext. The underlying frequencies of the letters remain unchanged, and this is a clue to help us break the cipher. This is called *frequency analysis*.

And there are other tricks we can use: Look for common words like ‘the’ or ‘and’. Look for letters at the ends of words that might be ‘s’. Look for double letters that might be ‘oo’ or ‘tt’; or certain pairs of letters, called *bigrams*, that might be ‘th’ or ‘qu’. Below are the ten most common bigrams and *trigrams*.

1	2	3	4	5	6	7	8	9	10
th	er	on	an	re	he	in	ed	nd	ha

1	2	3	4	5	6	7	8	9	10
the	and	tha	ent	ion	tio	for	nde	has	nce

Example. Let’s use frequency analysis to break:

VCIL NCI VWKIU WDI DSXLT AI HDIWQELB WU E FWYI NCI TIYQ WPSLI
WLT AZ IZI EL KWEL EU UIIQELB USAI BDIIL PIWJ NS DIUN XFSL
VCWN VSXPT LSN E BEKI NS VWLTID VCIDI AZ SPT YSAFWLESLU TVIPP
WHUILYI AWQIU NCI CIWDN BDSV JSLTID EUPI SJ HIWXNZ JWDI NCII VIPP

The ciphertext is 199 letter long, and the individual frequencies are given by:

letter	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
frequency (%)	3	3	4	6	5	2	0	2	18	2	2	8	0	6	0	5	2	0	7	4	5	5	9	2	2	2

The most common letter is 'I' which suggests it is the letter 'e' in the plaintext. The most common trigram is 'NCI', so maybe this is the word 'the'. Continuing in this way we will get the final message:

when the waves are round me breaking as i pace the deck alone
and my eye in vain is seeking some green leaf to rest upon
what would not i give to wander where my old companions dwell
absence makes the heart grow fonder isle of beauty fare thee well

1.9 Transposition Ciphers

So far we have looked at ciphers that substitutes a letter in the plaintext alphabet for a letter, number or symbol in the ciphertext alphabet. Collectively these are known as *substitution ciphers*. However, there is another type of cipher, known as *transposition ciphers*, this is when you mix-up the *position* of the letters of the message, to form an anagram.

For example, I am going to send the message 'this is a railfence cipher'. I write the message as two rows:

We now scramble up the order of the letters by taking the columns of the messages to make .
To decipher the message we need to reverse the procedure, in other words, we write the cipher in columns of length 2. The original message is then written in two rows.

In general, we can rearrange the positions of the letters any way we want. For a message of length , number the positions from to . We may then represent the transposition using a diagram. For example, we may rearrange the word 'eggs' as follows

to make the cipher .

Notice we may use the same ideas that we used for substitution ciphers for transposition ciphers. The only difference is substitution ciphers act on the plaintext letter's value (an alphabet of size , usually the letters a to z), while transposition ciphers act on the plaintext letter's position (from to).

1.10 Commuting Ciphers

We have seen that enciphering a message twice using two general substitution ciphers will make a new general substitution cipher - but not necessarily a more secure one! But does order matter? If I change which cipher I apply first, and which cipher I apply second, will the results be different? Earlier we saw this example:

Now consider what happens if we change the order of the two ciphers:

You see the results are . This is true in general.

On the other hand, if the order of the two ciphers do not matter (both orders give the same result) we say the ciphers *commute*. For example, two additive ciphers or two multiplicative ciphers will commute.

One of the major problems in cryptography is how to send the key. Traditionally both the sender and the receiver needs to know the key so they may encipher and decipher the message. This means the key needed to be sent secretly and securely. One solution to this problems is to use two ciphers that commute.

Imagine Alice wants to send a message to Bob. Alice and Bob each have their own cipher, called and respectively, and these two ciphers commute. First, Alice applies her cipher to the plaintext , written , and sends the message to Bob. Bob does not know how to decrypt this message, so encrypts it again using his cipher, written and sends it back to Alice. Alice now has a message that has been enciphered twice! But since the ciphers commute we have . If Alice applies her decryption key to this message it will remove her cipher, leaving only Bob's encryption . Finally, if Bob applies his decryption key he will be able to read the original message from Alice.

This idea is known as a *three-pass protocol*. Every time the message was sent it was enciphered, using either Alice's key, Bob's key, or both. Yet, the whole exchange took place without the two individuals

exchanging keys! If you can find two ciphers that commute, they may be used in this way as a solution to the key distribution problem.

Remember, each letter in the plaintext has two properties, *value* and *position*. Substitution ciphers act on value, while transposition ciphers act on position. But these two components do not interact, meaning if we use a substitution cipher and then rearrange the letters, we will get the same result as if we had rearranged the letters first, then applied the substitution cipher. In other words, substitution ciphers and transposition ciphers commute and can be used in the exchange described above.

For example, if Alice uses the general substitution cipher from section 1.7, then the word 'bead' becomes . Bob receives the message and uses the transposition cipher from in section 1.9, so becomes . Alice receives this message and applies her inverse substitution cipher, so becomes . Finally, Bob receives the message and applies his inverse transposition cipher, so becomes 'bead'.

Although no key is exchanged in this method, it is still only as strong as it's weakest link. With the first and third pass only encrypted once, standard techniques may be applied to break the cipher. In other cases all three passes may be used to determine the original message. For example, if we used additive ciphers to encrypt our message then if the someone intercepts the three passes and then that person may simply use as follows;

to recover the original message.

2 Polyalphabetic Ciphers

Monoalphabetic ciphers were used for thousands of years, but were vulnerable to statistical attacks, until a new idea in encryption began to be adopted in order to disguise letter frequencies. A *polyalphabetic cipher* is a rule that uses a different monoalphabetic cipher to encipher a plaintext letter p depending on its position in the plaintext. This means the same letter appearing in two different positions in the plaintext may be enciphered in two different ways, so a double letter in the plaintext may not necessarily be a double letter in the ciphertext.

2.1 Vigenère Cipher

The following cipher is named after the 16th century French diplomat Blaise de Vigenère. Imagine we want to encrypt the message: 'shaken not stirred'.

We start by constructing a Vigenère Square:

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

Notice, each line of the Vigenère Square is a Caesar Shift, starting with a shift of zero, and ending with a shift of 25.

Next we need a *keyword*. Let's use the word . We write that repeatedly above our message:

Keyword:
Plaintext: s h a k e n n o t s t i r r e d
Ciphertext:

To encrypt the first letter: go to row in the square - the first letter of the keyword; next find column - the first letter of the plaintext; and where they meet in the middle is the first letter of the ciphertext . For the next letter do the same again: go to row and column , and where they meet is the second letter of the ciphertext . Continuing in this way we get the ciphertext above.

Notice, the same letter in the plaintext may now be enciphered as different letters in the ciphertext depending on its position. Also notice, two letters that are the same in the ciphertext may come from two different letters in the plaintext.

If the keyword has length , let be the letters of the keyword. If is the plaintext letter in the th position, then the ciphertext letter in the same position, , is given by

Given the keyword, we may decrypt our ciphertext by simply reversing this algorithm.

The effect of the Vigenère cipher is to disguise the frequencies. The longer the keyword the more effect it has on frequency.

2.2 Breaking the Vigenère Cipher

If the keyword used in the Vigenère cipher has length , then every th letter will a result of the same monoalphabetic cipher. If we can determine the length of the keyword then, given a long enough message, we will be able to use standard cryptanalysis tools for monoalphabetic ciphers on every th letter to decrypt the whole message.

If monoalphabetic ciphers are broken by looking for common letters, polyalphabetic ciphers are broken by looking for common words.

Example. Below is ciphertext, 188 letters long, believed to have been enciphered using a Vigenère cipher:

FHVDWSEEMFQCZXVQRZAOBOCGOXPYIPQTZKQUPYMFZADMRMFKMFFHVAWJTVMBFHT
MBFUIGTDEEKVPIGTCYAKJZMIJMRQVZOSZEIMOZDZBKMSVDSZTLIZXYSZCWEEBVD
EVPIZDIMRKERZGXAKMFGSZVUFHVUSFHFLGPEMMZAPVLPKKRAWEKZIBPBRJPMGV

We begin our cryptanalysis by looking for any string of letters that appears more than once in the ciphertext. If we look closely, we can see the string appearing three times in the ciphertext. This is more than a coincidence. Since the keyword is repeated, we proceed by assuming this is a common trigram that just happens to appear under the same three letters of the keyword.

Notice, the distances between the occurrences of is letters, followed by another letters.

The Kasiski Test: If a string of letters appear repeatedly in a polyalphabetic cipher, then the length of the keyword, , may be a common divisor of the distances between occurrences.

So in our example above, the keyword may have length or .

2.3 Probability

We start with some definitions. The *sample space* of an experiment or random trial is the set of all possible outcomes. An *event* is any subset of the sample space; a collection of outcomes within the sample space.

If all outcomes in the sample are equally likely, the *probability of* , written , is given by

Example. Let's pick a card from a pack of 52 cards. Each outcome is equally likely with a probability of . Then;

Given an event , the probability of is

Example.

Events, and can be thought of as regions in a *Venn diagram* of :

The union of and is the probability that occur. The probability is

The intersection is subtracted to avoid counting it twice. If we say and are *disjoint events*.

Example.

Since being a spade and a red card are disjoint events, and;

The joint probability is the probability of . The joint probability is given as follows:

where means the probability of given occurred. If then the outcome of has no effect on , and we say that and are *independent events.*

Example. The probability of picking two aces without replacement;

2.4 The Friedman Test

Now, let be the length of our ciphertext. Let be the number of 'a's in the ciphertext, be the number of 'b's and so on until , the number of 'z's in ciphertext. Picking letters at random, the probability of picking two 'a's is given by

If is large, then and , where is the frequency of 'a' in the ciphertext.

The *index of coincidence*, , is the probability of picking any two letters the same;

Example. Using the frequency table in section 1.8 we can calculate the index of coincidence for English plaintext .

Note, since a monoalphabetic cipher does not change the underlying frequencies, we can expect ciphertext encrypted with a monoalphabetic cipher to have the same value of .

On the other hand, if the frequencies of each letter in the ciphertext is then the distribution will be perfectly uniform, and the index of coincidence .

The index of coincidence is a measure of how strong your cipher is. A value close to suggests a monoalphabetic cipher, while the closer the value is to the closer the distribution is to being uniform, and suggests a polyalphabetic cipher.

Example. The Vigenère cipher above has length and the following frequencies

letter	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
frequency (%)	3.7	3.7	2.1	3.7	5.6	6.4	3.7	2.7	4.8	2.1	5.3	1.6	8.5	0	2.7	5.3	2.7	3.7	3.7	3.2	2.1	6.9	2.1	2.1	2.1	8.5

Giving an index of coincidence , suggesting it is a polyalphabetic cipher.

If the keyword in our polyalphabetic cipher has length , then writing the ciphertext in rows of length will make an array with approximately rows.

Each column will be the result of one monoalphabetic (Caesar) cipher. Now we can find by using a different method to calculate the index of coincidence:

Pick two letters at random from the ciphertext. The probability the first letter is from column 1 is . And, for large , the probability they both come from the column 1 is approximately .

So, picking two letters at random, the probability they both come from the same column is .

Now we can calculate the index of coincidence again as follows:

Notice, long keywords will give you a value closer to , while short keywords will give you a value closer to .

Rearranging above, we get the **Friedman Test** for the length of the keyword, namely

Example. For our Vigenère example we have already worked out , so using the Friedman Test we get

From the Kasiski test we already thought might be a divisor of , namely . So it seems most likely to be .

Try writing the ciphertext in rows of letters. We may now apply decryption methods, such as frequency analysis, on each column.

Some guesswork can also help us find the keyword. For example, the first three letters of the ciphertext, , is repeated three times. When the ciphertext is written in rows of , these three instances of appear in the same columns, meaning the letters have been enciphered in the same way.

Originally this may have been a common trigram such as the word . We may use the formula to piece together the keyword.

Eventually we discover the keyword was and decrypt the ciphertext as below:

'the Vigenère cipher is a polyalphabetic cipher named after the sixteenth century french diplomat blaise de Vigenère and it was eventually broken three hundred years later using the methods developed by kasiski and babbage'

2.5 One Time Pad Cipher

As we have seen, when using the Vigenère cipher, the fewer the repetitions and the longer the keyword the closer the frequencies are to a uniform distribution. The *one time pad cipher* uses a string of random letters, as long as the message itself, as its key.

Example.

Keyword:	**Q**	**F**	**D**	**X**	**S**	**I**	**J**	**L**	**C**	**K**	**S**	**Y**
Plaintext:	a	t	t	a	c	k	o	x	f	o	r	d
Ciphertext:	Q	Y	W	X	U	S	X	I	H	Y	J	B

If you are the interceptor with no knowledge of the key, then it can be proven that this cipher is unbreakable. We will not prove this here, however the idea is that all keys and ciphertexts are equally likely. So, given a piece of ciphertext, and by carefully choosing the key, we could decrypt that ciphertext to make any plaintext message we want. This message could be good, bad, or nonsense - but it doesn't make that message any more likely to be true than it was to begin with.

For example, we may believe there is a 60% chance the message says '*attack oxford*', a 30% chance it says '*attack london*' and a 1% chance the message says '*blah blah blah*' - and there are keys that can decrypt the ciphertext to each one of those messages. However, decryption will not increase the likelihood of any of those messages, to the point that we might as well not bother.

Here the strength of the cipher does not lie in the individual monoalphabetic ciphers (which are additive) but in the randomness of the key. The key must be genuinely random; algorithmic or computer generated random keys will not do. Also, to prevent repetition, each key must be used to encrypt one message only, then discarded.

In reality, the one time pad cipher is impractical to use. When choosing which cipher to use we must balance security with practicality and ease of use. However, the one time pad cipher is used today to send messages between the President of the USA and the President of Russia.

3 Enigma

By the beginning of the 20th century it had become possible, and necessary, to mechanise encryption. In 1918 a German engineer named Arthur Scherbius patented the Enigma Machine. Originally it was sold to banks, railway companies and other organisations who needed to send secret information. By the mid-1920s the German military started to use the Enigma Machine, with some differences from the commercial version of the machine. Enigma was used by the German military throughout World War II, therefore breaking the enigma cipher became a top priority, first by the Polish, then later by the British and Americans.

3.1 Enigma Encryption

The Enigma Machine was an electro-mechanical machine, about the size of a typewriter, made with steel casing inside a wooden box. On the outside you will see two sets of letters which were the keyboard and the lampboard. You would type your message using the keyboard, the message would then be encrypted letter-by-letter, but instead of printing on paper the encrypted letters would light up on lampboard. The encrypted message would then be written down by the operator and would be transmitted by radio. The machine itself did not transmit.

Enigma is a polyalphabetic cipher, where each individual monoalphabetic cipher turns the letters of the alphabet into pairs, also known as a *product of transpositions*. A letter is encrypted as its paired partner.

For example, here are two monoalphabetic ciphers needed to send the message .

input:	a	b	c	d	e	f	g	h	i	j
output 1:	f	i	e	h	c	a	j	d	b	g
output 2:	h	g	j	f	i	d	b	a	e	c

Each cipher turns the letters into pairs, so if is encrypted as then is encrypted as . And the message becomes .

A product of transpositions makes decryption easy. Notice, encrypting a message twice will return the original message. So, starting again from the beginning, we may encrypt the message , and get back. A product of transpositions is *self inverse*.

In other words, if is the Enigma cipher for a given position, then and . This is how enigma decryption worked; by encrypting the message for a second time using the same settings you would get the original message back.

Next we will see how many settings the Enigma Machine had.

3.2 Combinations

We have already seen the idea of multiplying choices together, where is the product of all integers from to .

For example, imagine we want to list all the ways to arrange the five objects A, B, C, D, E. Starting from scratch, we have five blank spaces to fill:

For the first space we have choices, A, B, C, D, E. Then, having picked a letter for the first space, we would have choices left to fill the second space. After that, for the third space we would have choices. Then choices, then finally choice for the last letter.

You can think of your choices as nodes in a tree diagram. Multiplying choices together we find there are ways to arrange A, B, C, D, E. And in general there are ways to arrange objects.

Definition. *Permutations* are when you choose objects from and the order matters.

For example, imagine I want to choose objects from . First, list all ways to arrange A, B, C, D, E. Let the first three letters be the three objects I want, the last two letters are the objects I don't care about. So,

The order of the first three objects matter, but the order of the last two objects does not matter, so

There are ways to arrange the last two objects, so we need to divide our list of by . The total number of ways to choose objects from is .

Note, this is still the same as multiplying our choices together. We have five choices in the first place, four choices for the second place and three choices in the third place, so - and stop there.

In general, the number of permutations is given by

Definition. *Combinations* are when you choose objects from and the order doesn't matter.

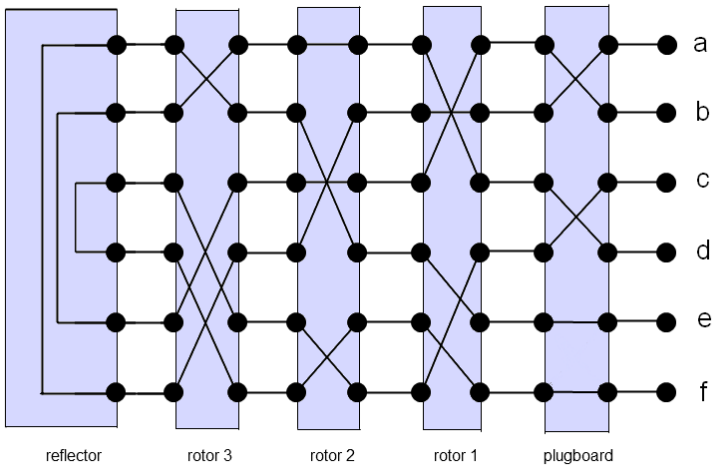
So, in the example above

In this case, we do not wish to distinguish between the different ways to arrange the first three objects. There are ways to arrange the first three objects, so we further divide the list of by to get .

In general, the number of combinations is given by

3.3 Enigma Settings

Inside each Enigma rotor is a criss-cross of wires. Pressing a letter on the Enigma Machine completes a circuit and causes a letter to light up on the lampboard. Here is a diagram of a small Enigma Machine:



First, the path goes through the plugboard, then the rightmost rotor; the middle rotor; the leftmost rotor; the reflector; the three rotors again in reverse; and finally through the plugboard again. That's a composition of nine ciphers.

Rotor 1 moved after every letter. When rotor 1 had done a full revolution it would kick rotor 2 one place. When rotor 2 had done a full revolution it would kick rotor 3 one place. So, there was a fast moving rotor, a middle rotor and a slow moving rotor.

The Enigma settings changed every day. These *daily settings* were written down on a keysheet, which themselves were changed once a month.

Geheim! **Sonder - Maschinenschlüssel BGT**

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Grundstellung
31.	IV II I	F T R	HR AT IW SK UY DF GV LJ EO MA	vyj
30.	III V II	Y V P	OR KI JV OU ZK MU BF YC DS GP	cqr
29.	V IV I	O H R	UX JC FD BN TA ED ST DS LU FI	vhf

These settings include:

- **Rotor order:** There are three rotors in the machine. The army could choose three rotors from a choices of five, (labelled I, II, III, IV, V). As we have seen, there are ways to choose rotors from .
- **Starting position:** Each rotor has starting positions, giving starting positions.

- **Plugboard:** The plugboard uses ten wires to connect letters into pairs, leaving letters unpaired.

The number of ways to choose letters from is .
We then form pairs. Pair order does not matter, i.e. is the same as , so we further divide this number by .
Finally, the order of the two letters within the pairs do not matter, i.e. is the same pair as , so we further divide by for each pair.
This means that the total number of ways to connect letters into pairs is .

Altogether, the total number of individual ciphers for the Enigma Machine was

One other important setting was the **Ring Setting**. There is a ring around the outside of each rotor, which labels the rotor starting positions from 1 to 26. However, the ring could rotate in relation to the internal wiring, which moves the kickover point. This does not affect the ciphers but changes the pattern in which the machine moves between ciphers. Due to a quirk of the middle rotor called 'double stepping', the rotors actually had a period of .

Finally, there was the **Reflector**. The reflector is a product of transpositions and, because it is in the middle of a sandwich, is responsible for the total cipher being a product of transpositions. However, the reflector was not a choice, it was fixed and unchanged.

As you can imagine, breaking the enigma code was quite a task! For comparison, a Vigenère cipher is a polyalphabetic cipher that uses individual ciphers and relatively short key lengths, while Enigma is a polyalphabetic cipher that uses individual ciphers and has a key length of .

3.4 Breaking Enigma

The Polish and British codebreakers eventually came up with a number of ways to determine enigma settings, these exploited various procedural mistakes and human errors made by the enigma operators. However, one such method was due to an inherent flaw in the design of the Enigma Machine itself.

Each monoalphabetic cipher consists of thirteen pairs, as a consequence no letter can be paired with itself. The cryptanalysts could then take some common phrase, slide it underneath the ciphertext and find where that phrase might fit in the ciphertext; two letters were *not* allowed to match. These phrases were known as *cribs*.

For example, there is only one place the phrase 'wetterbericht' ('weather report') may fit in the ciphertext below. Moving the crib to the left or right will result in two letters coinciding.

position:	
ciphertext:	⋯ j x a t q b g g y w c r y b g d t ⋯
crib:	

Let's assume this portion of the ciphertext does correspond to our crib phrase. For example, from the crib we can see that, in position 2, becomes . This is the result of the plugboard, ; followed by the combined result of the three rotors and reflector in the second position, ; followed by the plugboard again:

Remember, because the Enigma Machine acts as a product of transpositions, this diagram also works in the opposite direction.

Alan Turing realised that, for a given rotor setting, it is possible to deduce certain plugboard settings. For example, let's assume is a pair on the plugboard. So after the first use of the plugboard, input becomes .

This letter will now go through the combined rotors.

Starting from some initial position, we have the following successive outputs from an enigma machine without its plugboard, i.e. this is a table of , .

input:	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
output 1:	j	f	q	x	h	b	s	e	k	a	i	y	z	t	v	u	c	w	g	n	p	o	r	d	l	m
output 2:	p	n	s	k	u	z	o	w	v	l	d	j	r	b	g	a	t	m	c	q	e	i	h	y	x	f
output 3:	k	d	p	b	i	q	t	m	e	o	a	n	h	l	j	c	f	t	g	r	x	z	y	u	w	v
output 4:	x	o	d	c	h	z	l	e	p	y	u	g	q	w	b	i	m	v	t	s	k	r	n	a	j	f
output 5:	o	f	y	e	d	b	z	x	l	w	q	i	n	m	a	s	k	v	p	u	t	r	j	h	c	g
output 6:	v	w	p	t	m	x	k	u	o	l	g	j	e	z	i	c	r	q	y	d	k	a	b	f	s	n
output 7:	d	r	t	a	g	u	e	m	z	k	j	x	i	v	y	w	s	b	q	c	f	n	p	l	o	i
output 8:	v	z	e	j	c	q	u	n	l	d	y	i	r	h	w	x	f	n	t	s	g	a	o	p	k	b
output 9:	h	x	i	z	g	p	e	a	c	y	o	v	s	t	k	f	w	u	m	n	r	l	q	b	j	d
output 10:	h	v	x	m	z	i	k	a	f	s	g	w	d	q	u	r	n	p	j	y	o	b	l	c	t	e
output 11:	o	w	m	p	y	l	t	z	k	x	i	g	c	u	a	d	r	q	v	f	n	s	b	j	e	h
output 12:	r	u	z	l	j	y	i	t	f	e	m	d	k	x	q	r	o	a	p	h	b	w	v	n	f	c
output 13:	t	l	s	p	o	h	x	f	q	k	j	b	w	r	e	d	i	n	c	a	y	z	m	g	u	v

This works under the assumption that the second and third rotors do not move. The probability that the second rotor will move for a crib of length is , so cribs were kept to 10-14 letters.

From this table we see the input in the second position becomes .

This leaves the second use of the plugboard. However, since we know the final output is , we can deduce that is the other plugboard setting.

In the same way, by considering what happens in the third, fourth and thirteenth positions we get three further plugboard settings, namely , and .

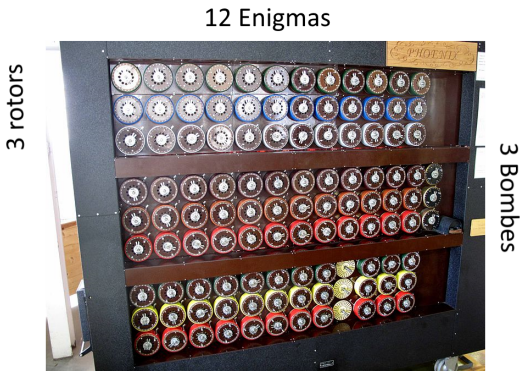
But here we have a contradiction, and cannot both be pairs on the plugboard. So our original assumption that is paired on the plugboard is wrong.

If we can show that all plugboard settings for a given letter (like 't') lead to a contradiction, then our rotor setting must be wrong, and we should try another.

We can speed up the process in a couple of ways. First, because Enigma is self-inverse, if we had assumed was a plugboard setting it would imply , and so this assumption is equally false.

Secondly, the plugboard settings , and will also lead to the same contradiction, so are equally false. As are all other consequences of these deductions.

Finally, these deductions can be made instantaneously with electrical circuits. Turing used this principle in his 'Bombe Machine'. Acting like simultaneous Enigma Machines, plugboard deductions from one machine fed into the others, with the aim of finding as many *false* plugboard settings as possible.



After making an initial plugboard assumption, either;

- the current flows through all plugs for a given letter. This means rotor position must be wrong, and the rotors move forwards one step;
- the current flows through one plug for a given letter, or all but one plug, and the machine stops;
- the current flows through some other number of plugs, meaning further checking is required;
- we get a false stop due to coincidence.

3.5 Reducing False Stops

Instead of the crib, Bombe operators would receive a diagram called a *menu*. The menu was another way to show the inputs and outputs at each position. For example, we have the crib

position:	1	2	3	4	5	6	7	8	9	10	11	12	13
ciphertext:													
crib:	w	e	t	t	e	r	b	e	r	i	c	h	t

To draw a menu, connect the inputs and outputs with an edge, and label that edge with its position:

Menus with loops were more successful because they reduced the number of inconclusive results and false stops. For example, our menu contains a cycle connecting the letters , , and . The correct rotor settings will make the following loop:

In other words, applying successive Enigma functions, we have the relation

Because the plugboard is a product of transpositions it is self-inverse. This means, if we write , consecutive plugboards will cancel each other out, and the relation can be rewritten as

This provides a much tougher condition to satisfy. The probability of satisfying this condition at random is . The probability of satisfying such cycles at random, and causing a false stop, is .

When the machine stops the rotor and plugboard settings are recorded. The remaining settings can be deduced by hand. Gordon Welchman later added a 'diagonal board' to the Bombe, increasing the machine's connectivity and speed. Now the Bombe Machine could check the logical implications of all 17576 rotor positions in 20 minutes.

With all this in place, the bombe machines became a highly successful attack on the Enigma throughout the Second World War.

4 Public Key Cryptography

A cipher system is *symmetric*, or *one-key*, if it is easy to deduce the decryption key from the encryption key. In practice, these two keys are often identical. This creates the problem of exchanging keys without the keys being stolen.

To solve this problem, two schemes were developed that don't require any secret channels, known as *public key cryptography*. The first was devised by Diffie and Hellman in 1976, and the second was devised by Rivest, Shamir and Adleman in 1977. We will describe the two routines, along with some small examples.

4.1 Diffie-Hellman

The idea of *Diffie-Hellman key exchange* is to create a *shared key*. Starting with a shared base, each participant makes a partial-key, which are then exchanged and used to complete the final shared key.

Here is an example using exponents, where all exchanges can be made on an insecure channel:

Let be prime. Then there exists an integer such that the powers of generate all values modulo .

Step 1: Alice and Bob share the modulus and a generator .
Step 2: Alice has a secret integer and calculates .
Bob has a secret integer and calculates .
Step 3: Alice and Bob exchange these partial-keys.
Step 4: Finally, Alice calculates and Bob calculates .

This procedure creates a shared key because . All participants may now this key to encrypt and decrypt messages. For example, ElGamel encryption uses the shared key as a multiplicative key, modulo .

Example. Let and . Starting with this base, the two participants use their secret integers and to create two partial-keys:

The two participants exchange keys and perform their secret operation again:

The two participants now have a shared key of .

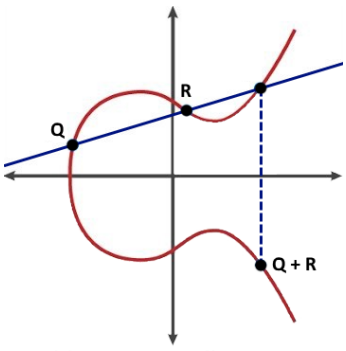
In real life, the values of , and are much larger. To make these large calculation more manageable, we break them into smaller parts. We are hoping that some of the parts will have a small remainder, ideally 0, 1, or 2.

Example. To calculate notice that . Then

Even though , , and are sent in the clear, it is still very difficult to work out from this information. This is called the *discrete logarithm problem.*

The procedure created a shared key because the order of exponents didn't matter (i.e. they commute). However, we can replace the use of exponents with any operation that commutes.

For example, *Elliptic Curve Cryptography* uses curves with their own kind of addition. Adding two points on the curve is defined to be the operation of drawing a straight line through the points; finding where the line intersects the curve; and then reflecting that point in the x-axis. Importantly, this definition of addition commutes. Multiples of a point can be found quickly by a combination of doubling and addition.



Addition on an elliptic curve

To create a key, the participants share a starting point, . Each participant then creates a secret multiple of , say and , which are then exchanged. Both participants multiply the exchanged points by their secret value to create the point . The coordinates of the final point may now be used as the shared key.

It is very difficult to work out the final point from the information that is exchanged. This is even harder if the calculations are done in modular arithmetic, as it makes the points seem totally unrelated.

4.2 RSA

If it is practically impossible to deduce the decryption key from the encryption key, then the system is called *asymmetric*. This allows you to freely publish the encryption key, known as the *public key*, while keeping the decryption key secret, known as the *private key*. RSA is an example of an asymmetric system. Let's describe the general routine using the smallest possible numbers.

RSA setup: Choose two prime numbers , . Set , and choose a number which is less than and coprime to .

To encrypt the plaintext as ciphertext we use the formula

RSA encryption is similar to additive or multiplicative encryption, except this time we encrypt using exponents. The numbers and are published and form our public key to encipher the message.

Example. Let , . Then , and , and let's choose . The numbers we work with must be less than the value of , so in this example I will send the message

plaintext:	b	a	d	e	g	g
in numbers:	1	0	3	4	6	6
:						
modulo :						

4.3 RSA Decryption

To decipher, we need a decryption key , such that

In other words, we need to find a number such that

The next three results will show that exists and how to find it.

Theorem 4.3.1 (Fermat's Little Theorem) *If is prime and an integer coprime to , then*

.

Proof. If and are coprime then there exists integers , such that . Equivalently, . This means has multiplicative inverse modulo . So if we may multiply both sides by and see .

Therefore no two items in the list can be congruent modulo , because they all have different coefficients of . This means the list is simply modulo , although possibly in a different order. If both lists are the same, multiplying all terms in each list gives us;

Since is prime, it is coprime to all integers , therefore we may cancel the term on both sides as before. This leaves as required. ∎

Example. Take and , then . This conclusion is guaranteed by Fermat's Little Theorem.

Theorem 4.3.2 (Chinese Remainder Theorem) *If and with , coprime integers, then* .

Proof. If and , then there exist integers and such that and . So .

Therefore , and since , are coprime, this means that . This means there exists an integer such that . We conclude that , which is the definition of . ■

A *corollary* is an easy consequence of a theorem or proposition. RSA decryption is a corollary to Fermat's Little Theorem and the Chinese Remainder Theorem.

Corollary 4.3.3 (RSA Decryption) *Recall the RSA setup: If , are primes, then , , and is an integer coprime to .*
Then there is a decryption key , such that , for all integers . The decryption key is the multiplicative inverse of modulo , i.e. .

Proof. If and are coprime then . Equivalently, . Therefore, the multiplicative inverse of modulo is . Let .

To show that , for all integers , we must first prove the congruence is true mod and true mod . We can then use the Chinese Remainder Theorem to prove it's true modulo .

If divides then . Similarly, , and so .
If does not divide , we can use Fermat's Little Theorem:

And since we know , for some integer , we have

Either way the congruence is true mod . Similarly it is true mod . Therefore, since , are both prime, and by the Chinese Remainder Theorem, the congruence must also be true mod . ■

So the decryption key is the multiplicative inverse of modulo . In other words, it is the coefficient of in . This can be found by observation (if small enough) or by using Euclid's Algorithm.

Example. If and as above, then and we may use as our decryption key.

ciphertext:						
:						
modulo :	1	0	3	4	6	6
plaintext:	b	a	d	e	g	g

[NB. I could have used here, ( ), but did not want you to think the encryption key and the decryption key had to be the same.]

In practice, RSA is a relatively slow algorithm, so is rarely used to encrypt plaintext messages directly. Instead, RSA is used to encrypt the keys for much faster symmetric key cryptography.

4.4 Breaking RSA

RSA encryption is incredibly strong provided we protect ourselves from some obvious attacks.

Frequency analysis: Encrypting one letter at a time still leaves you vulnerable to a simple frequency attack. This problem is overcome by grouping the letters into *blocks*.

Example. Let and . Numerically, our message 'bad egg' has values '103466'. We may split this into three blocks of two letters '10 34 66' and it is these groups that are acted on by the cipher.

blocks: 10 34 66

ciphertext:

Block frequencies become more independent and more uniform as blocks get longer. In English, blocks of ten letters are approximately equally likely.

Chosen-plaintext attack: Another way to break RSA is to encrypt likely plaintext messages using the public key and seeing if they are equal to the ciphertext. To prevent this, RSA typically embeds some form of structured, randomised *padding* into the plaintext. The padding ensures that the plaintext will encrypt to a large number of different possible ciphertexts.

If we cannot distinguish between two ciphertexts, even if they know the original plaintexts, then we say the the cipher is *semantically secure*. In practice, this makes the cipher's security equivalent to that of a one-time-pad cipher. RSA with appropriate padding is semantically secure.

Chosen-ciphertext attack: To decrypt , we could (sneakily) ask the holder of the private key to decrypt , for some chosen value . The decryption will be , which can then be multiplied by the multiplicative inverse of . However, current padding schemes, such as Optimal Asymmetric Encryption Padding (OAEP), are chosen-ciphertext secure.

Factorisation: Finally, RSA can be broken if we can find the original primes and . Once we have the primes, we can calculate , and use and to find the multiplicative inverse . Unfortunately, because we can find the original primes by factorising .

Example. If and , then it is not hard to work out that . Once we have the primes, we can calculate , and using Euclid's Algorithm calculate . This means the decryption key .

Since and were small here, was quickly factorised, and the private key was found from the public key. In reality the primes used are much larger, and carefully chosen, to make factorisation a very difficult task. Current RSA keys would take modern computers *thousands of years* to factorise.

4.5 Signatures

The intention of RSA is to encipher using E and decipher using D. But in fact the roles of E and D are interchangeable. That is to say, it is also possible encipher a message with D and decipher a message with E. This has a very important function in authenticating messages.

Consider two correspondents, John and Mary. Each have their own public key, which are known to everyone, and each have their own private key, known only to themselves.

Let's say, Mary's public key is $E = 8023$, $m = 24257$, and John's public key is $E = 8993$, $m = 11413$.

John wants to send the message 'meet you at six john'. Written in blocks this becomes:

me et yo ua ts ix jo hn

or, in numbers:

1204 0419 2414 2000 1918 0823 0914 0713

How can he let Mary know that the message really is from him? One way is for John to use his private key.

John enciphers his signature using his private key so it reads **2981 6024**:

1204 0419 2414 2000 1918 0823 **2981 6024**

John now enciphers the message completely using Mary's publicly key:

5141 12103 10542 3905 15868 21143 **0124 15283**

On receiving the message, Mary decrypts it using her private key to produce:

1204 0419 2414 2000 1918 0823 **2981 6024**

Mary extracts the signature **2981 6024**. Knowing that the message is supposed to be from John, Mary applies John's public key to his signature to get **0914 0713**.

Finally, putting all this together, Mary gets the message: 'meet you at six **john**'. And the only one who could have made that possible is John himself.

Note, it is more secure to use different RSA keys for encryption and signing, and padding should be used for message signing as it is for message encryption.

5 The Future

As we have seen, public key cryptography allows two people, who might never have met before, to communicate securely. This is done by publishing a public key, and keeping a private key that cannot be determined from the public key. In the case of RSA, this is due to how hard it is to factorise large numbers.

The two fastest known algorithms for factorising numbers are the *quadratic sieve* and the *general number field sieve*. Current RSA keys are 2048-bits (i.e. 617 digits) long, which can take years, or even thousands of years, to factorise. However, future advances in *quantum computing* may change that.

Traditional computers store information as 1s and 0s. However, quantum computers store information as 1s and 0s simultaneously. This allows quantum computers to perform algorithms that exploit the probabilistic rules of quantum physics. One such algorithm is *Shor's algorithm* that can efficiently determine prime factors, thus breaking RSA encryption.

Other forms of public key encryption, such as *Diffie-Hellman key exchange* and *elliptic curve cryptography*, are also quantum-breakable. So the future of cryptography will depend on devising problems that are provably hard for quantum computers.

One candidate for a quantum-secure system is *lattice-based cryptography*. Security here is related to the difficulty of finding the nearest point in a lattice with hundred of spatial dimensions (associated with the private key) given an arbitrary location in space (associated with the public key).

Other potentially quantum-secure systems include *code-based cryptography* and *multivariate cryptography*. Both rely on problems that are (presumably) difficult for quantum computers.

Quantum computers are still in their early days, and do not yet exist in any practical form. However, there are other technologies, available today, that do use the properties of quantum physics for sending secret information. These methods are collectively known as *quantum cryptography*.

The most well-known example of quantum cryptography is *quantum key exchange*. Here, the key is transmitted by particles of light through fibre optic cables. However, according to quantum physics, measuring particles of light changes them. So, a portion of the key is compared before and after transmission. Then, if someone has attempted to spy on the transmission, this act can be detected and the key is not used.

Once the key has been established, the message is encrypted using classical techniques. For example, the encryption method could be a one-time pad, which would make decryption impossible. Practical problems with quantum key exchange include transmission distance and key generation rate limitations.

Another example of quantum cryptography, and one that is purely quantum-based, is *Kak's three-stage protocol*, a form of three-pass protocol that uses particles of light to encrypt and transmit a message directly.

Cryptology used to be a problem for linguists. Later it became a problem for mathematicians and computer scientists, and now optical engineers. Who will be the next code breakers?