How two prime numbers help protect the internet

Introduction

What were you doing before you started this paper? You were most likely using the internet in some form - whether google searching or maybe an online purchase. Now as often as we indulge in the luxuries of modern technology, we don't realise how essential data protection is to our experience. I mean, I wouldn't be google searching if the search results had the possibility of being tampered with and being incorrect, and I definitely wouldn't be purchasing anything online if it meant gambling away my private bank details to the prying eyes of hackers. All in all, I hope this showcases how the complexity of the internet is built on the simple idea of *data protection* and in this essay, I hope to show you how data protection relies simply on just 2 prime numbers.

Example Scenario:

Alice and *Bob* are students who want to share a private message *Hello* in a classroom, they decide to encrypt their message with a key decided before the lesson, so that if their message is intercepted by their prying classmate *Carl* it cant be understood,

Early encryption

Alice and Bob aren't real people, however their dilemmas and solutions are representative of humans throughout history, tracing back most notably to Julius Caesar who used a *shift cypher* which involves shifting letters of the alphabet all by the same amount. Imagine we use a shift of 3:

A -> D

B -> E

Z -> A

Bob's message *Hello* becomes *Khoor* which would be unintelligible to the rest of their classmates if intercepted. Once *Alice* receives this message, she needs the *Key* (number of spaces each letter has been shifted) to decode it.

However, *Carl* could keep trying random keys up to 25 until the confusing string of letters became a word meaning he's broken *Alice* and *Bobs* cypher. This realisation led to the development of more complex ways of encrypting data such as mapping one letter to another one completely randomly:

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

Hello -> Psllf,

Carl could try to brute force decrypt the message (trying all different possibilities) however with their being over 400 septillion mapping table combinations it would be

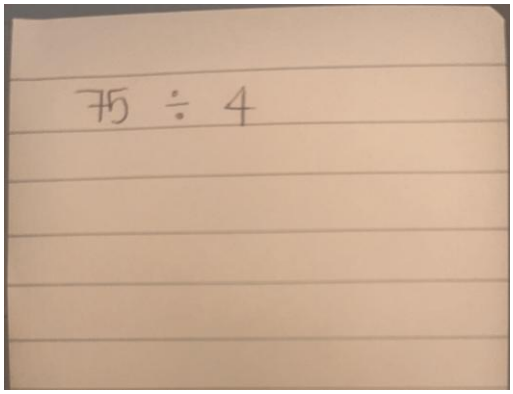
impossible. And this was a problem people had for many years until a solution was developed – frequency analysis

Frequency Analysis

Before I can explain Frequency Analysis, we need to think about these cyphers we've used in a maths way (this is a maths paper after all).

Redefining Cypher in mathematical terms

Before we can begin understanding the maths, I need to familiarise you with the *Mod* function. Think back to primary school doing bustop division before you knew about decimals, so you would write down the *Remainder* of a sum:



Well you can think of the *mod* function like asking just for the remainder of a sum.

a -> dividend (number going to be divided)
N -> divisor (what a is going to be divided by)
C -> remainder
C = aMod(N)
Example:
3 = 15Mod(12)

Using this we can redefine how we think about number shifts. Lets shift *H* by our key of 2.
H -> J (shift = 2)
numerically this can become:
7 -> 9 (shift = 2)

Intuitively, the general formula for encryption is this:
M + k = C
M = numerical version of plaintext character
k = key
C = numerical version of cyphertext character

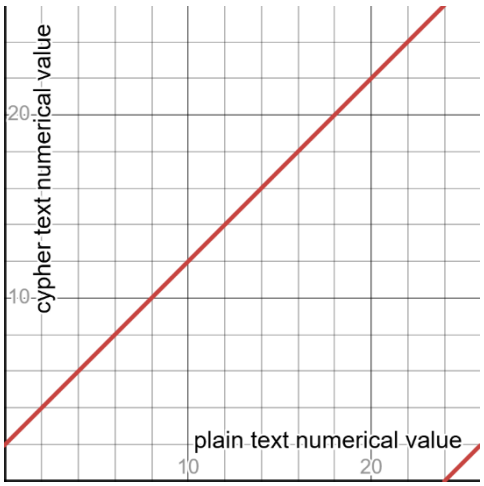
However, what happens when we get to Z, **25 + 2 = 27** and there is no 27^{th} letter of the alphabet. We actually want to know is how far 27 rolls over into the start of the alphabet (shift cypher is cyclical) as it's passed the end. What we want to do is 27 /

26 and use the remainder -> 1, to tell us the position of the cypher text character from the start which is 1 -> B.

We can use **M + k** from before, however if the result is greater than 26 it returns the remainder of the sum, So we substitute our **M + k** = dividend, divisor = 26 into the Mod formula.

(M + k)Mod(26) = C[1]

A graph of the formula:



Assuming k = 2

We can see a *one to one mapping*, each x value is assigned to one unique y value, meaning that each unique letter from the plain text becomes another unique letter from the cypher. In other terms, the frequency of the letters remains unchanged.

Frequency Analysis uses known frequencies of letters in a body of text to compare to a string of encrypted text. For the wordle players reading, your more likely to have the first word you guess to involve *A, E, T, this* subconscious intuition about letter frequencies makes frequency analysis possible, because certain letters appear more commonly than others, and we can infer that the most common letter in the plain text with correspond to the most frequent letter in the encrypted text. This means that we can take a string of encrypted text, count the most common letter appearing in it, and compare to this table of known English letter 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
Freq %	8.1	1.4	2.7	4.2	12.7	2.2	2.0	6.0	6.9	0.1	3.8	4.0	2.4	6.7	7.5	1.9	0.1	5.9	6.3	9.2	2.7	1.0	5.3	0.2	4.0	0.1

[1] https://dn710701.ca.archive.org/0/items/strategic_intelligence_network/communication/encryption/introduction_to_modern_cryptography.pdf

This means that both these encryption methods only guarantee data security for a short period of time.[5] Our problem is that we now need even stronger methods of encryption.

Key distribution problem

So we now have a more complicated method to encrypt our secret message, however, in order for *Alice* to decrypt the message she needs the secret key. So *Bob* needs to send this data across the internet to get to *Alice*, however, *Carl* can interfere this message, gaining access to the secret key, meaning when *Bob* sends the encrypted data, *Carl* has the opportunity to gain access to that as well, and he has everything he needs to decrypt the data. And this is the key distribution problem.

Public key encryption

Public key encryption uses different keys for encryption and decryption, encryption key is public and decryptions private. This way, anyone can encrypt a message but only people with access to a private key can decrypt the message. This is where RSA encryption and 2 prime numbers get involved.

RivestShamirAdleman encryption

RSA encryption relies on the main idea that if I gave you 2 prime numbers 53, 59 and I told you to multiply them together, you could get the answer 3,127 but finding the prime numbers that created it would be harder.[2]

How does RSA work

RSA encryption involves 2 formulas and 3 variables.(b, n, a) The 2 formulas are public knowledge to everyone as we are about to find out, however only 2/3 of the variables are public, meaning someone can encrypt a message, but only the person with the remaining variable can decrypt the message.

X = numerical plain text character

The encryption formula:

$$e(x) = x^a \text{ Mod } (n)$$

substituting x = L = 11:

$$e(11) = 11^a \text{ Mod}(n)$$

Defining variables *a, n* randomly:
a = 3
n = 200
Subbing in:

$$e(11) = 11^3 \text{ Mod}(200) = \underline{131(\text{cypher text})}$$

Bob sends the cypher text 131 to *Alice,* along with sharing *publicly* the encryption formula, a, n.

[2] https://www.rtings.com/vpn/learn/what-is-rsa

So now how will she decrypt it?

Decryption formula:

$$x = \text{Cypher text}^b \text{ Mod}(n)$$

To prove this works im going to tell you b = 27:

$$131^{27}\text{Mod}(200) \rightarrow 11$$

However, for x = 18

Encrypted text = 18^3 Mod(200) = 32

Sub into decryption formula:

$$\text{Plain text} = 32^{27} \text{ Mod}(200) = 188 \text{ (incorrect)}$$

This shows that b = 27, only works when x = 11, so we need to find a value of b that works for all values of x.

Lets do this by thinking of our two equations for cypher text and plain text

Cypher text(y) = x^aMod(n)

Plain text(x) = y^bMod(n)

As simultaneous equations, we can sub our equation for y into our x equation:

$x = (x^a\text{Mod}(n))^b\text{Mod}(n)$

-> $\underline{(x^{ab}) \text{ Mod}(n) = x}$

We have 2 unknowns *a x* in the above equation, that we can't find, meaning we need to learn extra maths!

Eulers Totient Theorem

Numbers are *Coprime* if the highest factor they have in common is **1,**.

Eulers Totient function is asking 'for a number *n* how many numbers leading up to that number are *coprime* to it' and the notation is:

Φ(n)

So to see it in action lets use n = 3:

Φ(3)

The numbers leading up to 3 are 1,2,3, writing as prime factors:

1 = 1 x 1

2 = 2 x 1

3 = 3 x 1

Coprime to 3	Not coprime to 3	
1		
2		
3		

ϕ(3) = 3

We can use this to define *Eulers Theorem* to find *b*

Eulers Theorem

Eulers Theorem states that,

If a and n are coprime:

$$\mathbf{x^{\phi(n)} \equiv 1 \bmod n}$$

So we now have 2 equations:

$$\mathbf{(x^{ab}) \bmod n = x}$$
$$\mathbf{x^{\phi(n)} \equiv 1 \bmod n}$$

and 2 unknowns – meaning we can substitute one equation into the other to find *b.*

Multiplying the second equation equations by **ϕ(n)**:

$$\mathbf{x^{\phi(n)} \cdot x^{\phi(n)} = x^{\phi(n)} \cdot 1 \bmod n}$$

$$\mathbf{x^{\phi(n)} \cdot x = 1}$$

$$\mathbf{x^{2\phi(n)} = 1 \bmod n}$$

Multiplying by **x^ϕ(n)** again:

$$\mathbf{x^{3\phi(n)} = 1 \bmod n}$$

We can repeat this *k* times getting:

$$\mathbf{x^{k\phi(n)} = 1 \bmod n \ \#1}$$

Rearranging our earlier equation we get:

$$\mathbf{X^{ab} = x \bmod n \ \#2}$$

Multiplying #1 by *x* can turn it into:

$$\mathbf{x^{k\phi(n)+1} = x \bmod n \ \#1}$$
$$\mathbf{X^{ab} = x \bmod n \ \#2}$$

In order to find b we need to see when **kϕ(n)+1** = **ab:**

a = 3

n = 200

ϕ(n) = 80

Substituting in we get:

$$3b = 80 \times (k + 1)$$

In order to find k we use the **Extended Euclidean Algorithm** which says that:

$$tx + zy = GCF(t, z)$$

x = b and y = k.

Were going to say:
t = a
z = ϕ(n)

GCF(a, *φ(n)) = 1*

$$3x + 80y = 1$$

In order to save words I'm not going to get into how this works, but that gives us:

$$x = 27$$
$$y = -1$$

*This only works when the greatest common factor of *a* and *φ(n)* is 1 -> limitation to be aware of.

Using RSA encryption in practice

Lets recap, we have 2 keys – one for encryption:

$$e(x) = x^a \text{ Mod } (n)$$

Decryption:

$$d(x) = y^b \text{ Mod}(n)$$

We have our three variables, *a b n*

Bob is going to reveal publicly the variables *a* and *n* meaning *Alice* can substitute these values into the encryption formula, to encrypt a message to send to *Bob* that only he can decrypt as he is the only one with access to variable b.
However, *carl* decides that he is going to try and compute *b*, the same way we have – so we need to find a way to make that infeasible for him.

Let's think back to our formula for calculating *b,* one thing we used was *z = φ(n),* if we in theory made *φ(n)* impossible to compute than b would be impossible to figure out, lets get into how we do that know ->

When *p* is a prime number: ϕ(p) = p – 1
Another fact to note is that ϕ(g x h) = ϕ(g) x ϕ(h) -> *multiplicative*

If we say that *g* and *h* are both primes that multiply to make *n* than:
n = g x h
ϕ(g) = g – 1
ϕ(h) = h – 1

ϕ(n) = (g – 1)(h – 1)

This links back to what we said before that its much easier to find *n* when you have *g* and *h*, but it's a lot harder to find *g* and *h* given you only have n. But if someone were to find *g* and *h,* than our RSA encryption would be broken, which is why its important to pick very large *prime* values of g and h, making n almost impossible to factorise

In the real world, *g* and *h* are 1024 bits long, making n 2048 bits long.

And I hope this showcases to you how two prime numbers are fundamental to internet security and protection![3]

Here is a link to download my full article without a word count ->

[3] https://youtu.be/nvcssTsiavg?si=v0vL5zEFkzudb4Q7

https://docs.google.com/document/d/e/2PACX-1vSWspkiyumZQBToqZQFEUSrEj7JKyHbBObk3yf4hCn_QsMGGEnx7OU8k8_62Fo9-f4r7uPd8zFN7Rw5/pub