ILA Project – Encryption

Intro

Encryption is a crucial aspect of our everyday lives. From online banking to social media, encryption is used to protect our data and keep it private during its journey between senders and receivers. The modern world is built on encryption, and this has come from thousands of years of work between cipherers and decipherers both vying to claim permanent supremacy, in either creating an unbreakable code or creating a method/machine that can break every cipher. The Vigenère cipher, known as the indecipherable cipher, wasn't broken for nearly three hundred years and today the integrity of all online banking has, to this date, only been held up on the belief that public private key encryption cannot be deciphered by any means in existence. In this project the author will explore the history of encryption, looking at some important ciphers used in the past and how they were eventually broken.

Caesar cipher

One of the simplest ways to encrypt a phrase is to shift all the letters in the phrase. For example, with a positive shift of three, A would become D, F would become I and Y would become B. This is known as a Caesar cipher after Julius Caesar, who used it frequently in his personal messages. A Caesar cipher is very easily broken however, as all the decipherer needs to do is shift the letters back and there are only 25 different ways the letters could be shifted, so manually checking isn't that arduous. Below is the author's code for a Caesar cipher with a shift of one.

```python
input_string = input("")

# This will mean A maps to B.
shift1 = 1


# This is a loop that converts a string into a list.
def string_to_list(test_string):
    char_list = []
    for i in range(0, len(test_string)):
        char_list.append(test_string[i])
    return char_list


# This encodes our message.
def caesar_cipher(input_list):
    cipher_list = []
    for i in range(0, len(input_list)):
        if ord(input_list[i]) <= ord("Z"):
            cipher_list.append(chr((ord(input_list[i]) - ord("A")) % 26 + ord("A")
+ shift1))
    return cipher_list


# This converts a list back into a string.
def list_to_string(input_list):
    final_string = ""
    for i in range(0, len(input_list)):
        final_string += input_list[i]
    return final_string


# This runs the functions we have defined, and also prints our final message.
plain_text_1 = string_to_list(input_string)
cipher_text_1 = caesar_cipher(plain_text_1)
cipher_string_1 = list_to_string(cipher_text_1)
print(cipher_string_1)
```

With this code, if the message “attackatdawn” was encoded, the ciphertext would be “buubdlbuebxo” which appears unreadable, but upon inspection it becomes apparent that all the A’s becomes B and all the T’s become U and as a result finding the key (the shift of the alphabet) is not difficult.

Permutation Cipher

A permutation cipher involves mapping letters to other letters, for example A to G. This would mean all A’s in the plaintext become G in the cipher text. This has the benefit where not all possibilities can be tested, as just with rearranging the 26 letters in the alphabet there are 26! possible combinations of maps, which equates to 403,291,461,126,605,635,584,000,000 combinations. If one person could test 1 combination every second and everyone in the world worked together then it would take roughly 2 billion years to break. This is a significant improvement from the Caesar Cipher where there are 25 combinations. For the author’s permutation cipher they have included more characters in the cipher so there are a greater number of combinations (91!). This surprisingly doesn’t change the strength of the encryption much as the methods which are used to break it are still effective.

```python
# This is a cipher that encrypts a message

# This enables us to generate random values - by importing a prebuilt library.
import random
input_string = input("")


# This here is a loop that converts a string into a list.
def string_splitter(test_string):
    char_list = []
    for i in range(0, len(test_string)):
        char_list.append(test_string[i])
    return char_list


# This generates an alphabet.
alphabet = []
for i in range(0, ord("~") - ord(" ") + 1):
    alphabet.append(chr(ord(" ") + i))
# This creates a dictionary that maps each character to another character.
print(alphabet)
alphabet_perm = random.sample(alphabet, len(alphabet))
perm_dict = dict(zip(alphabet, alphabet_perm))
print(perm_dict)
# This creates our enciphered message
final_string = ""
for character in input_string:
    final_string += perm_dict[character]
print(final_string)
```

```python
# TThis deciphers our enciphered message using the key (the dictionary from letter
to letter).
decrypt_string = ""
decrypt_dict = dict(zip(alphabet_perm, alphabet))
for character in final_string:
    decrypt_string += decrypt_dict[character]
print(decrypt_string)
```

This is the author's code to encrypt a message with a permutation cipher. An input message of "Some are born great, others achieve greatness." would become "[`bN i)N e`)n \)NiqZ `q.N)f iI.QNdN \)NiqnNffh". This mess of characters is completely unreadable, and as spaces get encrypted too it is hard to determine where words are, making it further more difficult to decrypt.

However the method of decryption is simple and relies on a major flaw of this cipher - if one has determined that in one place an e becomes a Q then in all places of the cipher an e becomes a Q. This means letter frequency analysis can be used to decipher it. For instance, the most frequently used letter in the English alphabet is e, followed by t and then a. Double t and double e are commonly found together whereas double a is very rarely seen, so by finding the three most commonly used characters in the ciphertext, Q f and @, it becomes known that they are likely to be t a and e, and if one found many ff and QQ but few @@ it's likely @ maps to "a". Applying this method and using context of words much of the cipher can be decrypted. Looking for commonly-found short words like "and" and "the" can also assist decryption.

There are many clever ways of improving a simple permutation cipher by simply considering how one would go about breaking the encryption. The method of deciphering is to look at letter frequency. If it is known that Q appears as 12% of all letters in the text and in Standard English text E appears 12.8% of the time for each letter one could deduce Q was mapped to E. Then replace all Qs with Es and look for words where E's make up much of the word, such as a double E. These words with QQ would be obvious and so other letters could too be inferred. "And" is a very easy word to find as it occurs many times, the same 3 letters again and again. So to frustrate the decipherers the permutation cipher would have to be improved to stop this. Some methods include having one hundred characters, where if in Standard English "a" occurs 10% of the time then there would be ten characters that represent a, meaning every character appears equally in the text. Another method is to introduce more letters which mean certain things, such as a character which means the word "and". The Great Cipher excelled at this.

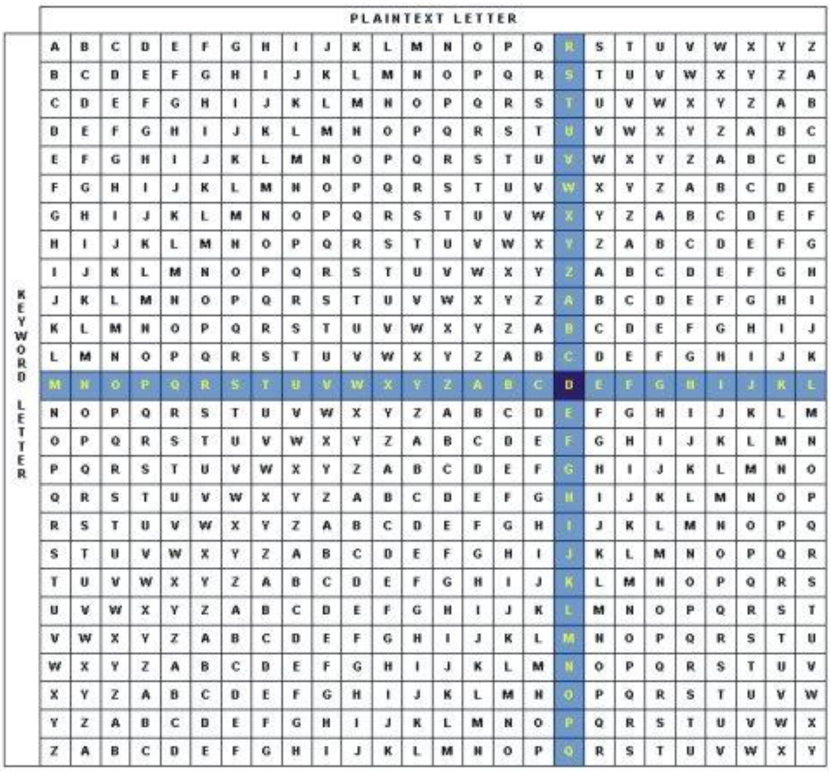
The Great Cipher was a famous French cipher used in the 17th century between its creator (Antoine Rossignol), his son and king Louis XIV. It fell out of use without the method of its deciphering becoming well-known, so all the documents in the French archives went unread until 1893 when it was deciphered. The Great Cipher was a permutation cipher, but it had over two hundred characters. Some had no meaning; some had subtle meanings like deleting the letter before it and another would indicate the letter before it was a double. This meant there were no obvious doubles and letter frequency would fail for another reason, if a letter appeared 12% of the time there would be twelve distinct characters that would mean that one letter. For instance E would be represented by 1, Q, x, $, t, T, e, 8, *, @, i , oo. The Great Cipher was special in that it referred to syllables, where each pair of numbers referred to a syllable. This made it incredibly hard to decipher until Étienne Bazeries realised the repeated numbers 124-22-125-46-345 stood for "les ennemis". It took him over 3 years to discover this, but once he did he unravelled the entire cipher.

Vigenère Cipher

The Vigenère Cipher or indecipherable cipher was famous for its strength and seeming invincibility against attempts at deciphering it. It was first described by Giovan Battista Bellaso in 1553, however thirty three years later Blaise de Vigenère published a form of the cipher and it was misattributed to him during the 19th century. The cipher works with multiple Caesar ciphers, and a key word or phrase. To encrypt a phrase first decide on a key word. This key word will only be known between the sender, and the receiver. The first letter of the ciphertext is the intersection between the first letter of the plaintext and the first letter of the key, so if the plaintext was "royal" and the key was "Mexico", the first letter of the ciphertext would be D, as seen in Figure 1.Therefore royal would become "DUVJN".

If the plaintext is longer than the key then the key is repeated. One of the strengths of Vigenère encryption is that letters are not always encrypted to be the same letter, unlike permutation ciphers, which is very useful for deterring methods of decryption such as letter frequency, which were successful at decrypting permutation ciphers.

Despite its strength the Vigenère Cipher was mostly overlooked for two centuries after its inception and other improved versions of the Permutation Cipher were preferred due to the simplicity when encrypting and decrypting messages. The Vigenère Cipher was seen as unwieldy and so was sidelined until Black Chamber Operations[1] like those in Vienna were so adept at deciphering messages that a change was required. However there were some instances where old-fashioned permutation ciphers with a twist such as the Great Cipher or Beale's ciphers[2] were still indecipherable.



	PLAINTEXT LETTER																									
KEYWORD 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
	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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

Figure 1

The cipher, although occasionally broken, had no general method or solution which could reliably break it except to steal or discover the key words being used, for over three hundred years. However, the cipher had two weaknesses that made it decipherable. The first was that both parties required the key used to encrypt and decrypt which made the code breakable in real world scenarios, as this could be discovered with espionage. This was the case during the American Civil War, where the Confederate States of America used the Vigenère cipher to encipher their messages, yet their keys

[1] Black Chamber Operations consisted of a team of code-breakers working round the clock intercepting mail coming in and out of embassies. They were very successful (the most successful of which was situated in Vienna) and forced a change in encryption to something new.

[2] Beale's ciphers were a group of permutation ciphers that indicated the location of a large amount of gold. However as to this day they have still not been deciphered, many believe they are a hoax and meaningless.

were guessable, one such key was the phrase “complete victory”. As a result the Union regularly decrypted their messages. The second weakness with the Vigenère cipher was that occasionally due to repetition of both the key and the plaintext, the ciphertext will contain repeats. This was exploited first by Charles Babbage in 1854, but eventually Friedrich Kasaski published the general method in 1863 and gains the credit for deciphering the Vigenère cipher. As there are repeats in the Ciphertext, say after 30 characters there is a repeated phrase; this means the keys length is limited to 30, 15, 10, 6, 5, 3, 2 or 1. This gives insight into what the phrase could be and with an examination of lots of Ciphertext the key can be determined. The author has written a Vigenère cipher with the key words “flowers” and “obstructing”. The ciphertext is very difficult to decipher without knowledge of these key words.

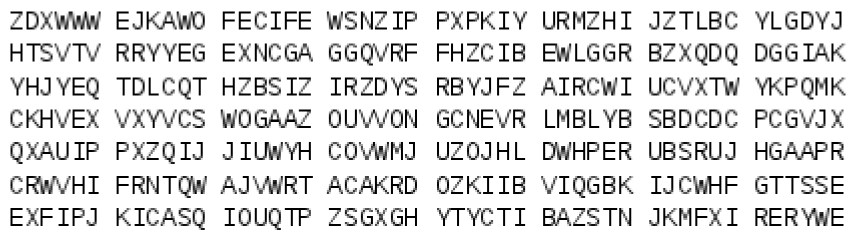
```python
# This function converts a string into a list.
def string_to_list(test_string):
    char_list = []
    for i in range(0, len(test_string)):
        char_list.append(test_string[i])
    return char_list


# This function converts a list into a string.
def list_to_string(input_list):
    final_string = ""
    for i in range(0, len(input_list)):
        final_string += input_list[i]
    return final_string


# These functions create a matrix of letters (26 by 26). The line below prints
this matrix if it is not commented out.
alphabet_matrix = [None] * 26

for n in range(0, 26):
    alphabet = [None] * 26

    for i in range(0, 26):
        added_letter = (chr(ord("a") + ((i + n) % 26)))
        alphabet[i] = added_letter
    alphabet_matrix[n] = alphabet

# print(*alphabet_matrix, sep='\n')

plain_text = input("")
# These keys will be used to encipher our message, if you don't know the key, you
cannot decipher the message.
key1 = ["f", "l", "o", "w", "e", "r", "s"]
key2 = ["o", "b", "s", "t", "r", "u", "c", "t", "i", "n", "g"]
key4 = key1 + key2
key3 = key4 + key4 + key4 + key4 + key4 + key4
plain_text_list = string_to_list(plain_text.lower())
cipher_text_list = []
# This enciphers our message, taking care to ignore values that aren't in the
simple english alphabet.
for i in range(0, len(plain_text_list)):
    if 96 < ord(plain_text_list[i]) < (ord("z") + 1):
        column_number = (ord(plain_text_list[i]) - ord("a"))
        row_number = (ord(key3[i]) - ord("a"))
        cipher_text_list.append(alphabet_matrix[column_number][row_number])
    else:
        cipher_text_list.append(plain_text_list[i])

cipher_text = list_to_string(cipher_text_list)
print(cipher_text)
deciphered_text_list = []

# This here is our decipher code using the keys.
for i in range(0, len(cipher_text_list)):
    if 96 < ord(cipher_text_list[i]) < ord("z") + 1:
        # print(cipher_text_list[i])
        # print(key3[i])
        decipher_value = (ord(cipher_text_list[i]) - (ord(key3[i]) - 1) +
ord("`"))
        # print(decipher_value)
        if decipher_value <= ord("`"):
            deciphered_text_list.append(chr((ord(cipher_text_list[i]) + 26 -
(ord(key3[i]) - 1) + ord("`"))))
        else:
            deciphered_text_list.append(chr((ord(cipher_text_list[i]) -
(ord(key3[i]) - 1) + ord("`"))))
    else:
        deciphered_text_list.append(cipher_text_list[i])


deciphered_text = list_to_string(deciphered_text_list)
print(deciphered_text)
```

This code creates a 26 by 26 matrix of letters like the one above and then encrypts the letter based on the Key words; here the author used flowers and obstructing. Using this code “attackatdawn” becomes encrypted to “fehwgbshespe”. The benefit of using words with prime number length is that the true key length (the number of characters before the key repeats itself) is the product of the lengths of the keys, 77 in this case. This makes the Kasaski test harder to perform.

Once the Vigenère cipher was widely broken (in the late 19th Century) there was no sturdy replacement, and so nations and organisations continued to use variations of the Vigenère cipher and other clever forms of permutation cipher, but nonetheless there was no assurance of security the way there was one hundred years prior. The addition of a onetime pad helped this a little but it wasn’t until 1919 and the formation of Enigma that there was a widely used code that couldn’t be deciphered. There were codes in existence that could have replaced the Vigenère cipher but, like the one-time pad cipher they were unwieldy and so didn’t become popular.

One time pad ciphers were revolutionary and to some extent still are as they have no known method of being broken. Using pre-agreed code pads (books of letters where the cipher text looks like pairs of numbers which refer to the row and column number of the letter on the pad), messages can be encrypted in the code on the pad and then after the first message is sent the first page of the pad is destroyed and the second sheet on the pad is used. The biggest issue with this method is one of synchronisation, if there are more than two people it’s incredibly difficult to synchronise which sheet in the pad is being used to encrypt and decrypt messages, and of course if the enemy steals a pad they can decipher every message. For this reason they are used in very rare circumstances, such as the US-

USSR hotline (now the US-Russia hotline), but from a codebreakers point of view they are unbreakable, unless with espionage, where one could obtain a copy of the pad.



```
ZDXWWW EJKAWO FECIFE WSNZIP PXPKIY URMZHI JZTLBC YLGDYJ
HTSVTV RRYYEG EXNCGA GGQVRF FHZCIB EWLGGR BZXQDQ DGGIAK
YHJYEQ TDLCQT HZBSIZ IRZDYS RBYJFZ AIRCWI UCVXTW YKPQMK
CKHVEX VXYVCS WOGAAZ OUVVON GCNEVR LMBLYB SBDCDC PCGVJX
QXAUIP PXZQIJ JIUWYH COVWMJ UZOJHL DWHPER UBSRUJ HGAAPR
CRWVHI FRNTQW AJVWRT ACAKRD OZKIIB VIQGBK IJCWHF GTTSSE
EXFIPJ KICASQ IOUQTP ZSGXGH YTYCTI BAZSTN JKMFXI RERYWE
```

This is an example of a one-time pad cipher, where (3, 4) would indicate the third letter across (X) and the fourth column (C)

Figure 2

Advanced Encryption

Encryption works on the basis that it is difficult for someone to know how the letters in the plaintext relate to the letters in the ciphertext. Using maths to define this relationship works well as it offers the ability to use complex[3] functions. The function is the key and if the function is known it is easy to input plaintext and create the ciphertext or, during decryption, input ciphertext to the inverse of the function and create the original plaintext. Without the function it is very difficult to take the ciphertext and turn it back into plaintext. This form of encryption was based on using maths to improve a Permutation Cipher so that the connections between the letters were uniquely generated for each letter. This would mean that unless the function that related the pairing of the initial letter to the cipher letter was known, the code couldn't be deciphered.

The famous Enigma used by the Germans during WW2 was a form of this, each letter was paired to another like in a permutation cipher but the pairings changed based on mechanical parts (that could have been modelled by an equation). Enigma was strong for two reasons, a letter would often be different in the cipher text (so "a" could become any letter) and the number of combinations of pairings was vast. Enigma started out as a civilian and military code, and due to the military nature of Germany at the time these diverged quickly until the military form of enigma was many times more secure than the civilian form. Poland succeeded in breaking the civilian and early forms of the military cipher during the 1930s, but as enigma was developed to be more secure by addition of a plug board and more rotors, the cost of deciphering the machine got too high and so it was abandoned. All the actions of Alan Turing and the British intelligence during WW2 were to break the naval form of the enigma, which was the most secure. All the other forms of the cipher had already been broken by the Polish, an achievement often overlooked by history. Manually it would have taken billions of years to test all combinations and every 24 hours the military settings of the code were scrambled and so the old key was useless.

The Enigma was finally broken due to the fact that any letter couldn't be enciphered to be itself, and that some key phrases were known before the message was intercepted, such as a weather report containing phrases like "6 am" and "weather report" and at the end of most messages, "heil Hitler". This was used to build a deciphering machine unlike any other, a mechanical decipherer where the machine tested combinations at a rate previously unheard of. Alan Turing developed this machine, known as the Bombe, and it took the Bombe roughly 20 minutes to decipher the code every day. This

[3] The author refers to complicated functions, not functions using complex numbers.

was the first time in history electronics were used instead of a human brain to decipher a message, and it revolutionised decryption. The author’s code doesn’t have the same relationship (mathematical function) that Enigma had; however the principles are the same. Additionally, their code doesn’t have the flaw of Enigma, a letter can be itself.

```python
# These access prebuilt libraries which enable me to use powerful maths functions
such as trig.
import random
import math

input_string = input("")


# This converts a list into a string.
def list_to_string(test_list):
    final_string = ""
    for i in range(0, len(test_list)):
        final_string += test_list[i]
    return final_string


# This converts a string into a list.
def string_to_list(test_string):
    char_list = []
    for i in range(0, len(test_string)):
        char_list.append(test_string[i])
    return char_list


# This encodes a message based on a mathematical function, without the function
they cannot decipher the message.
def string_encoder(test_string):
    n = 0
    encoded_list = []
    for i in range(0, len(test_string)):
        n = (ord(" ") + ord("~")) % (math.floor(5*abs(math.cos(i) + 2)))
        encoded_list.append(perm_dict[chr(ord(input_list[i]) + n)])
    return encoded_list


# This prints our final enciphered message.
input_list = string_to_list(input_string)
alphabet = []
for i in range(0, ord("~") - ord(" ") + 1):
    alphabet.append(chr(i + ord(" ")))
alphabet_perm = random.sample(alphabet, len(alphabet))
perm_dict = dict(zip(alphabet, alphabet_perm))
final_list = string_encoder(input_list)
print(list_to_string(final_list))
```

Using this code, the phrase “He who would valiant be, let him come hither.” becomes “`6YeX@Yt/ubJYejSJ6{u#O$<#{6e&X44Y$K{X#J*_aa/m”. This cipher is very difficult to break, however it has some issues, such as being unwieldy and hard to compute. Additionally, you must take care to ensure the function you use is bijective otherwise the ciphertext will not be recoverable into plaintext form. Enigma was bijective due to a clever part of the machinery called a reflector, which assured that if the settings were correct you could both encipher messages and decipher messages with the same one machine, which is massively useful during wartime where efficiency and low cost is arguably the most important factor.

RSA encryption

RSA encryption is a form of public private key encryption used across the world today and it is an example of the type of encryption that protects all the data that is possessed, sent and received over the internet and across the world. It was developed in 1977[4] by Rivest, Shamir and Adleman, who worked for over a year attempting to form a public-key encryption. This meant that the key was public (N and e), they just couldn't decrypt the encrypted message with it. This became declassified and used in 1997 when it became important for encryption on the internet. It is often used in conjunction with a symmetric key algorithm[5] due to its inefficiency at encoding large amounts of data.

RSA encryption is a form of encrypting numerical data where one relies on the difficulty of factorising a large number into its products of primes to encode the data. The method of encryption is as follows:

Take two primes p and q, normally these would be very large but for this example we will make them superficially small. p = 5 and q = 7

Compute N where N = pq, so N = 7 * 5 = 35

Find λ (N), the lowest common multiple of (p-1)*(q-1). So λ (35) = LCM (4, 6) = 12

Now one must choose an encryption key, it must be smaller than λ (N) and coprime with λ (N) and N (meaning they share no common factors). 11 would work as 11 < 12 and 11 is prime so it's easy to check if they have any common factors. So e = 11

To encrypt any number (we will call this number M) we compute M^e mod N (where Mod means the remainder left after division)

So here if we want to encrypt the number 2 we find 2^{11} mod 35 which is 2048 mod 35 = 18

So our encrypted message is 18.

Then to decrypt we find d (the decrypt key) which is the inverse of e mod N, or in other words the value that would satisfy de mod (N) = 1. In this case if we had 11d Mod (35) = 1, d would be 16 as 11*16 = 176 and 176 mod 35 = 5 remainder 1 so it equals 1.

If you take the encrypted message, 18 and put it to the power of 16 mod 35 you get 18^{16} mod 35 = 2. This was the initial message.

```
# Here we define a class.
class Key:
    def __init__(self, prime1, prime2, encryption_key, our_message):
```

[4] https://www.comparitech.com/blog/information-security/rsa-encryption/

[5] A symmetric key encryption algorithm is a very efficient way of encrypting a document or file with data inside, and the key of this data is sent along with the file. The key is encrypted with RSA to ensure high levels of security.

```
        self.prime1 = prime1
        self.prime2 = prime2
        self.product_of_primes = 0
        self.phi = 0
        self.encryption_key = encryption_key
        self.decryption_key = 0
        self.our_message = our_message
        self.cipher_text = 0
        self.decrypted_text = 0

    # This is where we compute N.
    def product_of_primes_function(self):
        self.product_of_primes = self.prime1 * self.prime2

    # This is where we compute phi.
    def phi_function(self):
        self.phi = ((self.prime1 - 1) * (self.prime2 - 1))

    # This is where we find the decipher key used to decipher the enciphered
message.
    def modulo_inverse(self):
        for i in range(1, self.product_of_primes):
            if (self.encryption_key * i) % self.phi == 1:
                self.decryption_key = i
                print(self.decryption_key)
                print("this is the decryption key")
                return self.decryption_key
        return "Group needed"

    # This is where we encipher our message.
    def cipher_text_function(self):
        self.cipher_text = (self.our_message ** self.encryption_key) %
self.product_of_primes
        print(self.cipher_text)
        print("this is the enciphered message")

    # This is where we decipher the enciphered message using the decryption key.
    def decrypted_text_function(self):
        self.decrypted_text = (self.cipher_text ** self.decryption_key) %
self.product_of_primes
        print(self.decrypted_text)
        print("this is the decrypted message")


# Here we initialise a class and run the functions within the class.
Key1 = Key(prime1=3,
           prime2=11,
           encryption_key=3,
           our_message=4
           )

Key1.product_of_primes_function()
Key1.phi_function()
Key1.modulo_inverse()
Key1.cipher_text_function()
Key1.decrypted_text_function()
```

This is the author's code for performing an RSA encryption. It uses different numbers to those used in the earlier example and encodes the number 4 to 31. It must be remembered that RSA is not generally used to encrypt a message and instead that is done by symmetric-key encryption, and RSA is used to encrypt the key of the symmetric-key encryption. Using the decrypt key this code proceeds to generate 4. This shows the relationship between inverses in modulo arithmetic is successful at retrieving our initial plaintext.

The future of encryption and quantum encryption

Encryption is very important to the modern world, more so than ever before. Online banking and cryptocurrencies are here to stay and only 34% of all transactions in the UK were with physical cash in the last decade[6]. This means that a breakdown in encryption would be a global catastrophe as cyber theft could occur without people having any means to protect their online assets. However the prospect of a breakdown in encryption has become a more pressing threat since the creation of quantum computers. Due to the nature of a quantum computer, they excel at the same process that makes RSA strong, factorising a large number. RSA is used to encrypt everything in our lives and normally the value of N is 2048 digits[7] long, which so far remains unbroken. But quantum computers are in very early stages of creation and testing and yet they have already broken 831 bit RSA, which means N is 831 digits long. This gives rise to a scenario where RSA cannot keep up with Quantum factorising and so new encryption methods like quantum encryption must become widely used.

Quantum encryption is a method of encryption that, although has been around since the 1980's, has become more relevant recently due to the rise of quantum computers and the threat they pose to current encryption methods. Quantum encryption uses the laws of quantum physics to protect the value of a key, by using Heisenberg's uncertainty principle.

$$\Delta x \Delta p \geq \frac{h}{4\pi}$$

The encryption method relies on 4 properties of quantum effects:

- Particles are uncertain in their positions (equation above), and simultaneously exist in 2 places
- Randomly-generated photons exist in one of 2 quantum states
- This property, known as spin, cannot be measured without disturbing or changing said property
- You can clone some quantum properties of a particle, but not the whole particle

The photons are sent from the sender to the recipient and they represent 1s and 0s. This is based on their polarisation, where if they are vertical they are a 1, if they are horizontal they are a 0, if they are 45° left they are a 1 and if they are 45° right they are a 0. To check the orientation of the photon one must use either a rectilinear or a diagonal polariser. Now, let's assume there is a sender, Alice, and a receiver, Bob.

Alice sends Bob multiple photons and he uses a random selection of rectilinear and diagonal polarizers to check their orientations, and therefore their values (either 1 or 0). Afterwards Alice tells

[6] https://www.goodhousekeeping.com/uk/consumer-advice/consumer-rights/a30470116/cashless-society/
[7] https://crypto.stackexchange.com/questions/22971/what-prime-lengths-are-used-for-rsa

Bob which polarizers to use for the different photons and he deletes all the incorrect entries. This final string of binary is the key they will use to encrypt their message. If another person, Eve, intercepts the message she will inevitably look at the message (to try and obtain the key). If she doesn't continue to send the message to Bob he will know the key is not safe, if she does she has observed the photons and as a result she will have changed all the orientations. This will be obvious to Bob when Alice confers with him and so he will know the key is unsafe to use. This is useful for transferring safe keys to use, instead of using RSA to encrypt the key.

Conclusion

Encryption is so important to the world, more so today than at any other point in history. This is due to the use of the internet in all aspects of our lives, from socializing to banking to leisure. For this reason the need to keep our data safe and our transactions secure are greater than ever before. However the author does not believe there is any immediate risk to encryption being widely broken, as not only are there multiple options and replacements for pre-existing methods of encryption, but also because the risk posed to RSA by the new quantum computers is not severe in the short-term. Quantum computers are only in their preliminary stage and are yet to break encryption used across the world, but also they are so large, expensive and unwieldy that no hackers and fraudsters have access to them.

Bibliography

The science of Secrecy – Simon Singh

The Code Book – Simon Singh

Sciencia – Matthew Watkins

Euclidian distance based encryption – Fuchun Guo.

https://webcache.googleusercontent.com/search?q=cache:L4Yq_psxIVEJ:https://www.uow.edu.au/~fuchun/publications/ccsposter14.pdf+&cd=2&hl=en&ct=clnk&gl=uk

RSA encryption, how does it work - comparitech

https://webcache.googleusercontent.com/search?q=cache:7oIptZqXqFQJ:https://www.comparitech.com/blog/information-security/rsa-encryption/+&cd=18&hl=en&ct=clnk&gl=uk

An Overview of Cryptography – Gary C. Kessler

https://www.garykessler.net/library/crypto.html

Quantum Encryption explained.

https://quantumxc.com/quantum-cryptography-explained/

Google images - https://www.google.com/search?q=one-time+pad+cipher&sxsrf=ALeKk01d_LBg51cCZo2a0elP0Gkz5PfYjQ:1594027712002&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjLyqT8p7jqAhWJXsAKHe7DBiIQ_AUoAnoECBYQBA&biw=1920&bih=911#imgrc=0Kulr5Bt-bQd3M