Vigenere Cipher Encryption

This project contains a python script, vigenere_cipher.ipynb, designed with the purpose of taking a keyword and a string of plaintext as input to a function and using that information to transform the plaintext into ciphertext (an encoded message) using the **Vigenere Cipher**. This document provides information on what the Vigenere Cipher is, how it works, and how the code uses it to create ciphertext.

Note: that this code is designed to be run in an application that can process files with the .ipynb extension. For some code translators, you may have to change the extension name to .py to run the code properly. Additionally, the vigenere_cipher.ipynb file uses multiple code bubbles, AKA “kernels” to complete its task, so please ensure that you have an understanding of how these work and run all the kernels in order from top to bottom.

Table of Contents

What is the Vigenere Cipher, and how does it work?

The Vigenere cipher is a simple **Polyalphabetic Cipher**, a code in which each letter is represented by a different letter, so that a word or sentence looks completely different from before it was encoded. For instance, the word “hello” might become “fisjk” or any other arrangement of letters the same length. The encoded text, “fisjk” is then sent to the intended receiver of the message, who uses a **key** to decode it and return the word “fisjk” to its original form, “hello”.

	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

The Vigenere cipher selects which letters to switch with by using a keyword and the **Vigenere Square**, seen above. First the keyword is repeated until the full length is as long as the full message being encrypted.

Example: Create Repetitive Keyword

Let’s say that the message you want to encrypt is “salutations”. The word is eleven letters long. Now, imagine your keyword is “key”, which is only three letters long. You repeat the word “key” and string it together until you reach eleven letters. If you are only part way through the keyword when you reach elven, cut it off. This gives you the repeating keyword “keykeykeyke”, eleven letters long. Each letter from the first (number 1) to the last (number 11) is then matched to the letter with the same number in the original message:

SK, AE, LY, UK, TE, AY, TK, IE, OY, NK, SE

This is where the **Vigenere Square** comes into play. The square is a twenty-six by twenty-six grid of the letters in the alphabet. The first row across is the alphabet written in its normal order, then for each row after that, you can copy the previous row but move the letter at the start of it to the end.

As with any grid, we can use coordinates to find a specific spot (or, in this case, letter) within the grid. But with the Vigenere cipher we use letters for the coordinates rather than numbers. For each pair of letters seen above, we take the first letter (the one from the original message), and figure out where it is in the normal alphabet. In this case the first letter of the first set is S, which is the nineteenth letter in the English alphabet. Because it is the nineteenth letter, we count nineteen spaces across the square from left to right and end up on the nineteenth top-to-base column. We then find where the second letter of the first set is in the alphabet and count that many spaces down from the top of that column. The letter we land on is the first letter of the encoded message. We repeat this process for the second set of two letters, and the result becomes the second letter in the encoded message. We continue this process for all the sets in order until we have the full ciphertext message.

How the Code Replicates Vigenere Cipher Encryption

Sequence of Steps Included in the Vigenere Cipher

The code for encryption, which is stored in vigenere_cipher.ipynb, consists of four kernels, which function as follows:

1. Making Vigenere cipher 2D matrix
2. Making repetitive key
3. Making alphabet dictionary
4. Encrypting the plain text using Vigenere Cipher Encryption Algorithm

Diving a little deeper into that explanation, we have:

First kernel

The first segment of code defines an array using the **NumPy** python library, a python add-on that is used for mathematical processes. This array contains the lowercase versions of the twenty-six letters in the English alphabet. Then the code uses loops to iterate over each letter in th alphabet and the twenty-five letters after that letter. If the code reaches the end of the alphabet before it has gathered twenty-five letters after the first, it loops back to the start of the alphabet and continues counting until it reaches

twenty-five. Each iteration is stored as a list inside a larger array, creating the structured data that our code will use to simulate the Vigenere square. The last line of code in the first kernel prints the final array to the screen below the kernel.

Second kernel

Within the second kernel is defined a **function** (a repeatable and callable block of code) named a_function. This function takes a keyword and a string of plaintext as input and runs a process that repeats the keyword until its length matches the length of the plaintext, giving us our repeating keyword for encryption and storing it in the variable repeating_key.

Third kernel

This code snippet uses a loop and the alphabet to make a dictionary where each letter in the alphabet is matched to a number equal to how far it is into the alphabet minus one, starting with ‘a’ as zero and ending with ‘z’ as twenty-five.

Fourth kernel

The final segment of code ties things together with yet another loop that looks at the repeating keyword, the plaintext, and the array that represents the Vigenere square and uses python’s built-in indexing function to find the correct letters to be added to the ciphertext. It then prints the newly-encrypted ciphertext to the screen one character at a time.

Using the Vigenere cipher to encrypt messages

To encrypt a message using the code file, start by running the first three kernels in order from the top. This will create a two-dimensional array that represents the Vigenere square and sets up the repeating keyword, which is conveniently stored in the variable repeating_keyword. Then, find the encrypt_text function in the fourth kernel (the place where it says encrypt_text()). Inside the parentheses at the end of the line of code that represents the function, type the text you want to encrypt in single or double quotations, followed by a comma and a space, then type the keyword you want to use for encryption. Once you have done this, run the fourth kernel. It should print the letters of the encrypted text in order from the top.

Example To Encrypt Text Using Vigenere Cipher

D, E = A, as seen below. Where D represents the plaintext char, and E represents the altered key chapter

	A	B	C	D	E	F	G
A	A	B	C	D	E	F	G
B	B	C	D	E	F	G	A
C	C	D	E	F	G	A	B
D	D	E	F	G	A	B	C
E	E	F	G	A	B	C	D
F	F	G	A	B	C	D	E
G	G	A	B	C	D	E	F

Decrypting the ciphertext using the Vigenere cipher

The current version does not have a decrypt feature.

Known issues

The code is imperfect and not yet built to handle certain features. Some of the downsides are:

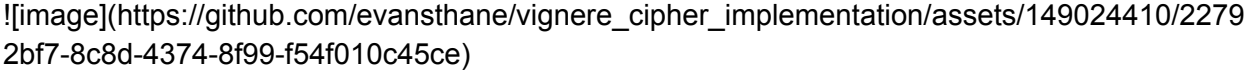
- **Inconvenient user interface**
- **No decrypt function**
- **Does not work on plaintext separated by spaces**

At some point I would like to update this project with a file that fixes the errors listed above and combines the project in one kernel, preferably in such a way that the user doesn't have to look at the code itself to use the algorithm. For now, however, these problems remain and should be taken into consideration.

Conclusion

This code is designed to speed up the relatively simple process of encrypting a message with the Vigenere cipher method. While still missing important features, it has proven useful tool for speedier encryption and experimentation.

The ‘tropes’

What is Vigenere Cipher?
The vigenere cipher is a polyalphabet encryption technique using the vigenere square as seen below.
Vigenere Square, you can check image below:


The cipher takes a keyword and a string of plaintext. It loops the letters in the keyword and strings them together until the length on the result equals the length of the plaaintext entered. then it aligns the repeating keyword with the plaintext and uses each pair of letters (one in the plaintext and the corresponding one in the repeating keyword) as coordinates to find the ciphertext, with the plaintext letter being across the top row and the repeating keyword letter on the side.

how ROT-13 works
Algorithm
Step1: The algorithm defines a function encrypt_text() which takes the plaintext and keyword as input.
Step2: once the inputs are assigned and the function is run, it starts using a for loop to repeat the index characters of the keyword in order and append them to a string variable named repeating_keyword. It continues to do this until the length of repeating_keyword equals the length of the plaintext that was input to the function. For instance, if the plaintext is "oranges" and the keyword is "yes", the code will identify the number of characters in "oranges", which is

seven, and will repeat the letters of the keyword in the order they are in the keyword, until the resulting string has seven characters. therefore, the final string would be "yesyesy".
Step3: the code uses a two-dimensional list named alphebet_dictionary to represent the vigenere square. the list is comprised of twenty-six alphabets, each one shifted one space forward using ROT-13. to find the ciphertext, the code uses the alphabet index of the first character in the plaintext as the index number for the second-dimension list it wants to find, then uses the alphabet index of the first letter in the repeating_key variable as the index for the letter inside that list. it adds that letter to a variable encrypted_vigenere_text. it then repeats this process but with the second letters from the plaintext and repeating_keyword, then third and so on.