Movie Machine

Ivan Ocon

800451445

# Introduction

I have been working with a movie machine that will encrypt and decrypt movies requested by the user. Moreover, I have used the Python programming language to create and run my program on a Raspberry Pi microcontroller. The main motivation of my project was to put in practice what I have learned in the Cybersecurity course as well as to learn some Python programming. Likewise, my project has security authentication features where you can encrypt and decrypt movies. At the beginning, I started thinking to implement this with AES, DES, Caesar Cipher algorithm but end up working with Hill Cipher algorithm.

Hill Cipher was invented by Lester S. Hill in 1929. It was the first substitution cipher on more than three plaintext characters. It was a based on linear algebra. It never gained much popularity.

Here is a good reference for Hill Cipher:

<https://www.geeksforgeeks.org/hill-cipher/>

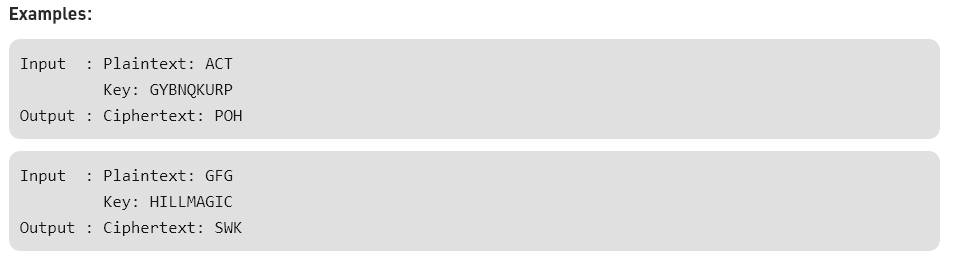
Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

For example, from this alphabetic table each letter will be represented by a number modulo 26:



In this example we can see how string “ACT” is converted to “POH” and “GFG” is converted to “SWK” with random keys.



In the below formulas we can see how the encryption and decryption are done:

K = Matrix which is our 'Secret Key'

P = Vector of plaintext (that has been mapped to numbers)

C = Vector of Ciphered text (in numbers)

C = E(K,P) = K\*P (mod X) -- X is length of alphabet used

P = D(K,C) = inv(K)\*C (mod X) -- X is length of alphabet used

Encryption

Here we can see that the encryption is done with this formula: C = E(K,P) = K\*P (mod X)

Decryption

Here we can see that the decryption is done with this formula: P = D(K,C) = inv(K)\*C (mod X)

# Tools Required

As mentioned before, my project utilizes a Raspberry Pi as hardware:

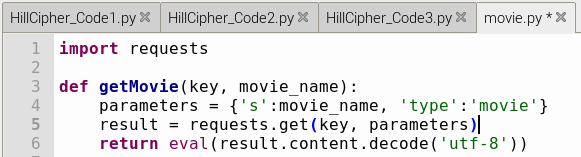


Therefore, my project will utilize Raspbian operating system as Software:



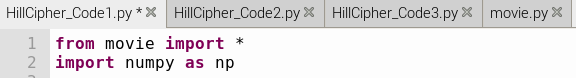
# Implementation and Details with code

As a security system my programs can convert a movie from plaintext to ciphertext and vice-versa. In other words, I am encrypting and decrypting the movies requested by the user for security purposes. In this movie machine the user will get the movie, year, type of most recent movie requested. Also, I have worked in the Hill Cipher implementation in three different programs using this movie script:



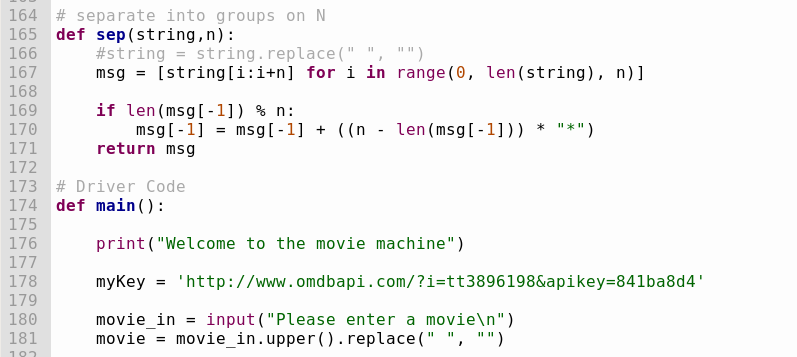
SET UP DETAILS:

For the setup of my programs I am using numpy library to use matrix algebra functions as well as a movie script to import requests from an open movie database for the execution of my programs:

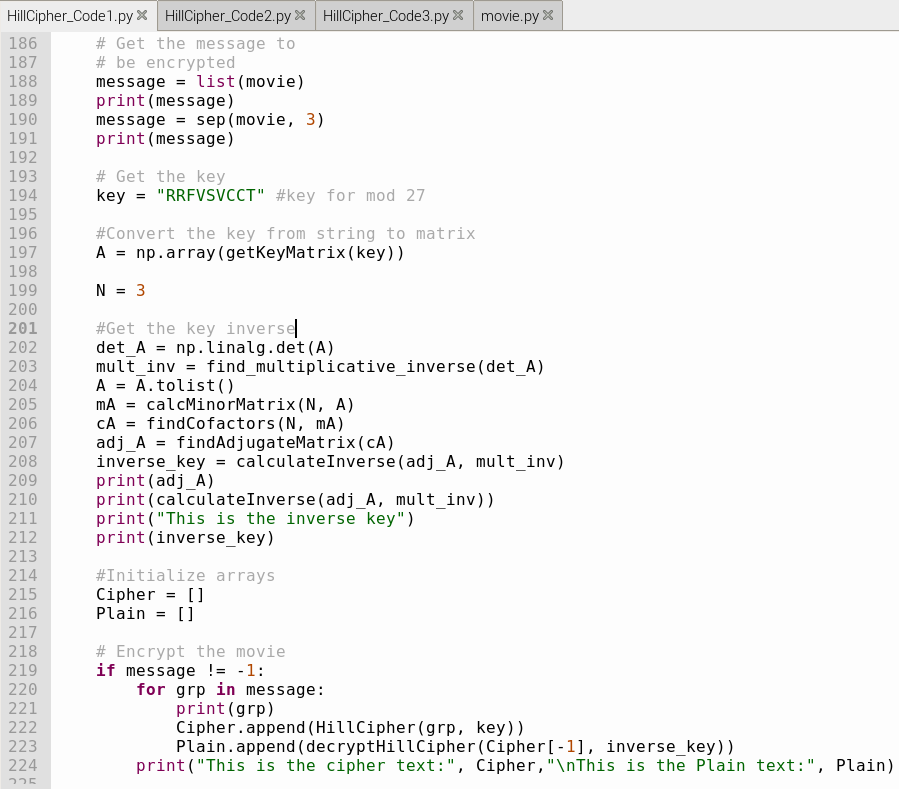


CODE 1

In my first approach from code 1, I am prompting the user to enter a movie. I will get the message to be encrypted by listing and separating each word in syllabus of three with a function called “sep” as below:

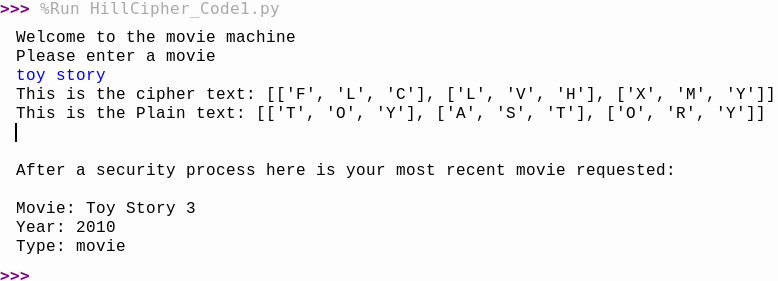


Here, I will get the key as a 3x3 matrix from a hardcoded nine letter string "RRFVSVCCT". Then, I will get the inverse key to decrypt the message in the same manner. I am encrypting the message into groups of three in a for loop with a function called HillCipher. Also, I am decrypting the message into groups of three in the same for loop but with a function called decryptHillCipher. For encryption we multiply the key matrix times the plaintext in vectors (mapped to numbers by dictionary). For decryption we multiply the inverse key matrix times the ciphertext in vectors (mapped to numbers by dictionary as well).



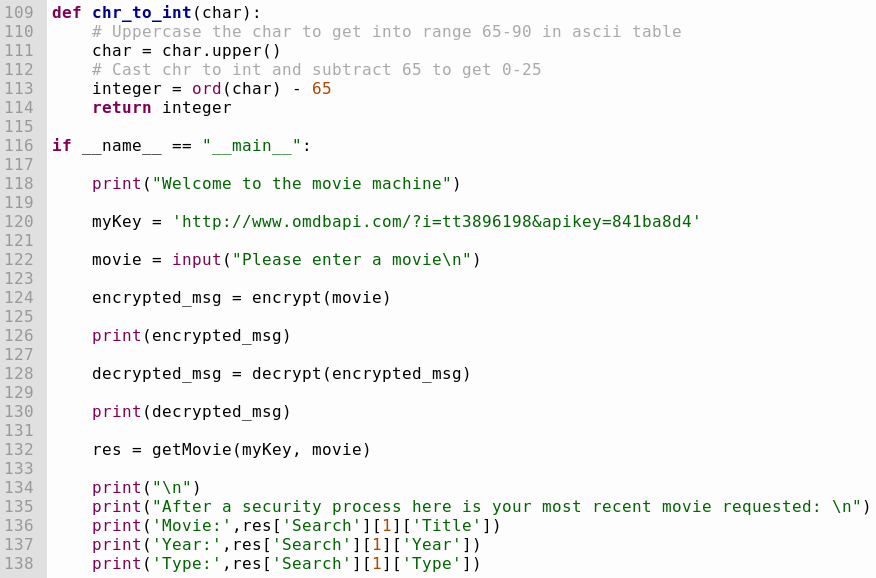
As mentioned before, I am getting the response from a script movie by calling a function to get movie requests from getMovie (myKey, movie) where “mykey” is the URL from the open movie database and “movie” corresponds to the movie selected by the user. As you can see, I have done a lot of debugging in this program to separate the movie string in syllabus of three. But after many tries, I have realized that the determinant of the key matrix "RRFVSVCCT" is equal to -939. Therefore, I won’t get an inverse for "RRFVSVCCT" which is equivalent to this class list [[17, 17, 5], [21, 18, 21], [2, 2, 19]]. Since I was working with mod 27 space I had to take a different key matrix and work with modulo 26 to can decrypt.

Therefore, this key matrix cannot get me the decryption result as I wanted. I had to take a key matrix where gcd of the determinant and 27 is equal to 1. Anyways at the very end, I am just printing the cipher text and the plain text as well as the movie, year, and type of most recent movie requested by the user

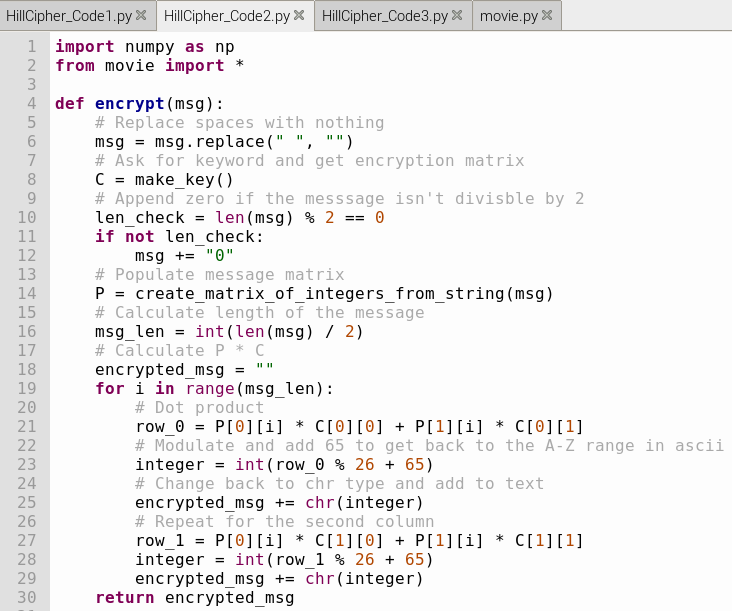


CODE 2

In my second approach from code 2, I am prompting the user to enter a movie. In this scenario, the encryption and decryption will be different. Now, I will get the key as a 2x2 matrix. I will get the message to be encrypted with a function called encrypt. Then, I will get the same message to be decrypted with a function called decrypt. After, I am getting the response from a movie script by calling a function to get movie requests such as approached before which are the movie, year, and type of most recent movie.

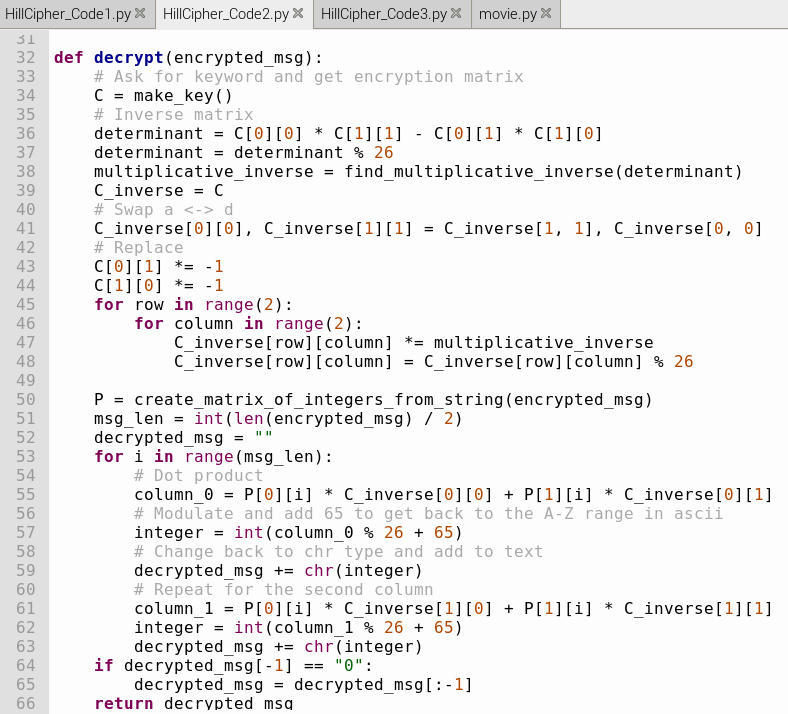


To encrypt, I will ask for keyword and get encryption matrix. Append zero if the message isn’t divisible by 2. Populate message matrix from string to integers. Calculate length of the message. Calculate the plaintext with dot product, modulate and add 65 to get back to the A-Z range in ascii. Change back to char type and add to text. Repeat for the second column in the same process and finally return the encrypted message.

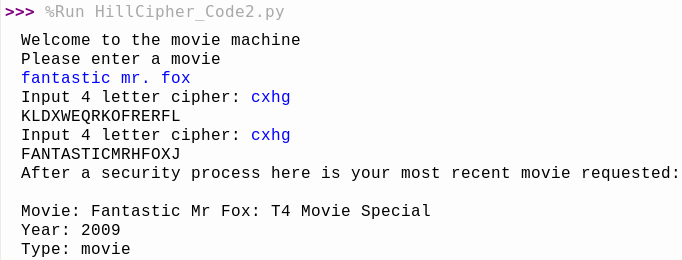


As you can see the inside of the program in the process is a little bit different now, because in this scenario we make sure cipher determinant is relatively prime to 26 and only a/A - z/Z are given. Likewise, we map string to a list of integers a/A <-> 0, b/B <-> 1 … z/Z <-> 25. Also, the encryption and decryption is done with a 2x2 key matrix and not with 3x3 matrix as in code 1.

To decrypt, I will ask for keyword and get decryption matrix. Get inverse matrix by getting the determinant with modulo 26. Swap a <-> d (diagonal numbers) and replace with negative values the other diagonal numbers. Calculate the ciphertext with dot product, modulate and add 65 to get back to the A-Z range in ascii. Change back to char type and add to text. Repeat for the second column in the same process and finally return the decrypted message.

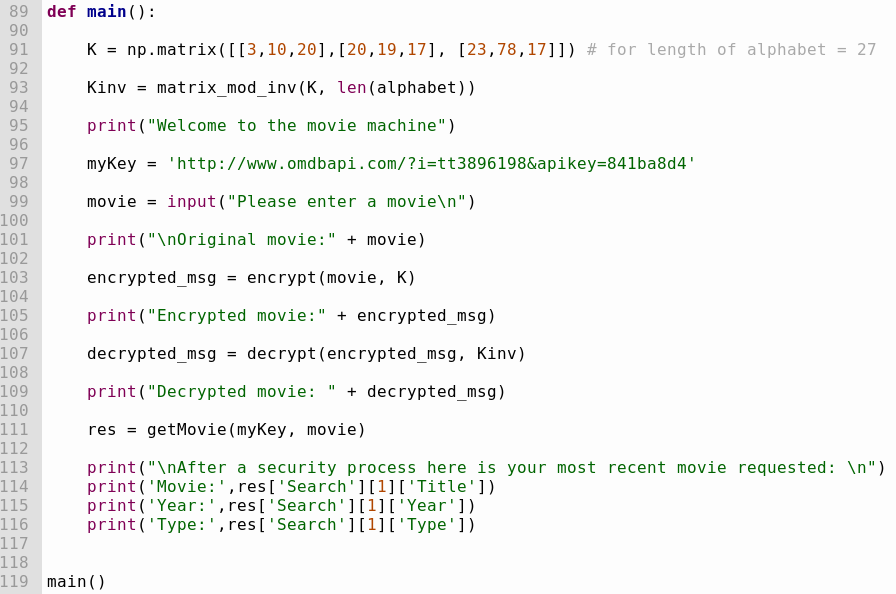


In this case, the encryption and decryption was done with a 4-letter string (2x2 key matrix) using mod 26

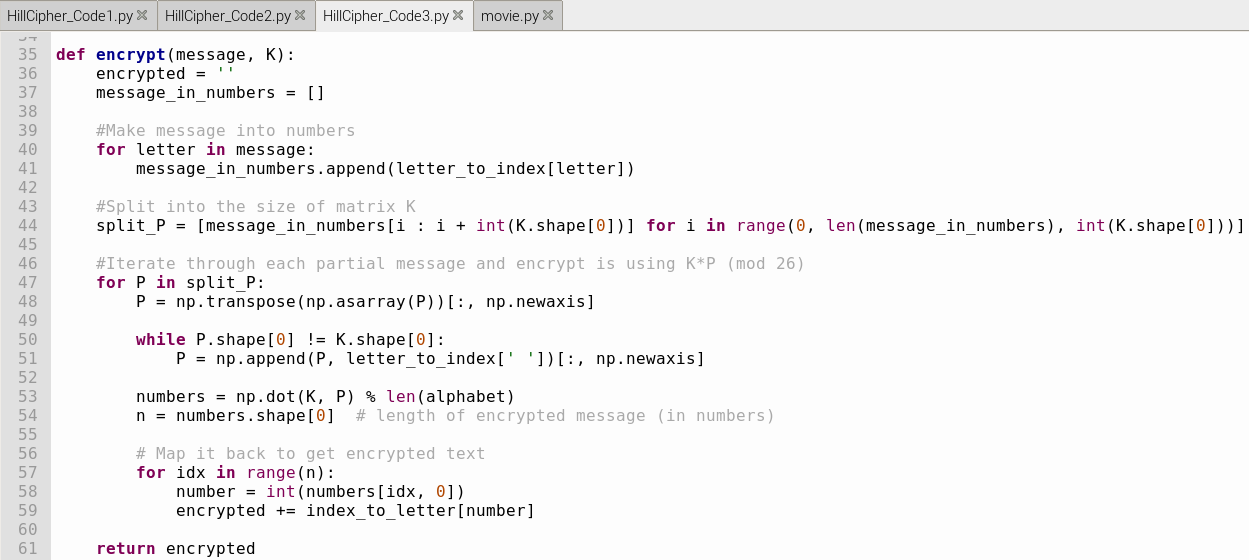


CODE 3

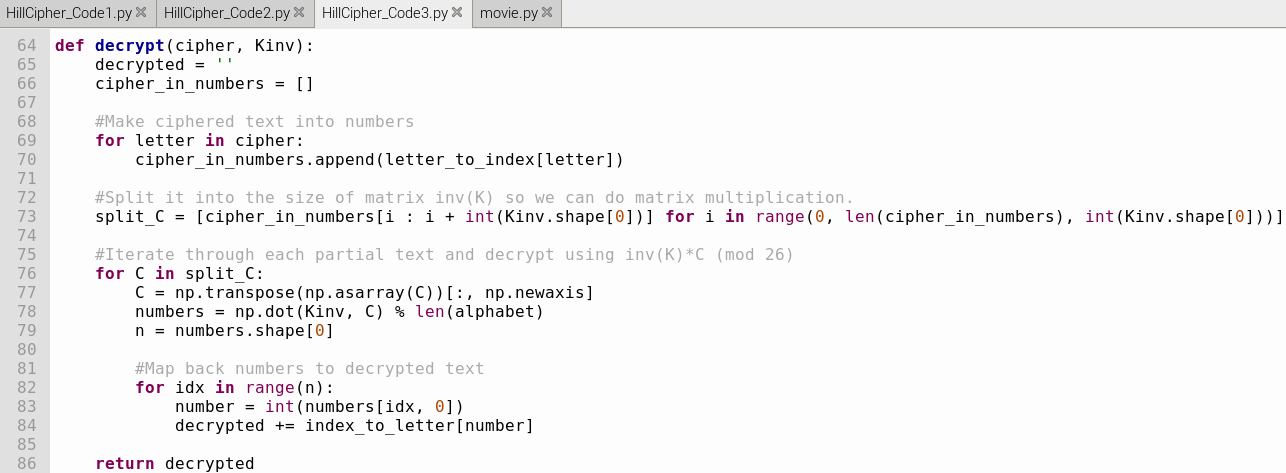
In my third approach from code 3, I am prompting the user to enter a movie. I will get the message to be encrypted for length of alphabet = 27. In this scenario, I am using mod 27 because I am taking movie spaces in consideration. Also, I will get the message to be decrypted for length of alphabet = 27.



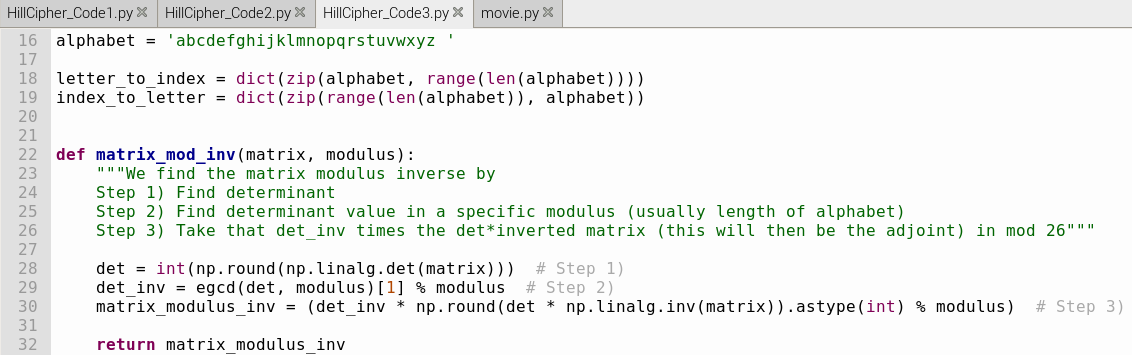
To encrypt, I will make the message into numbers, split into the size of matrix, iterate through each partial message and encrypt using formula K\*P (mod 26) and map it back to get the encrypted text.



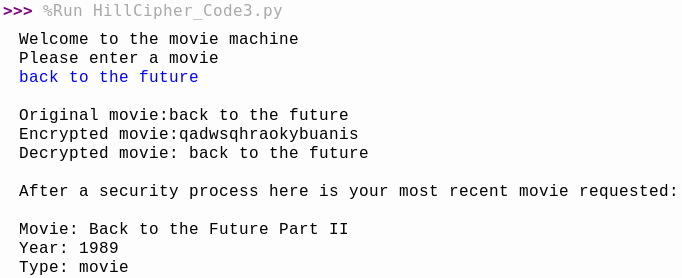
To decrypt, I will make the ciphered text into numbers, split it into the size of matrix so we can do matrix multiplication. Iterate through each partial text and decrypt using formula inv(K)\*C (mod 26) and map back numbers to decrypted text.



As shown below, we need to find the matrix modulus inverse in a 3x3 matrix following numpy library:



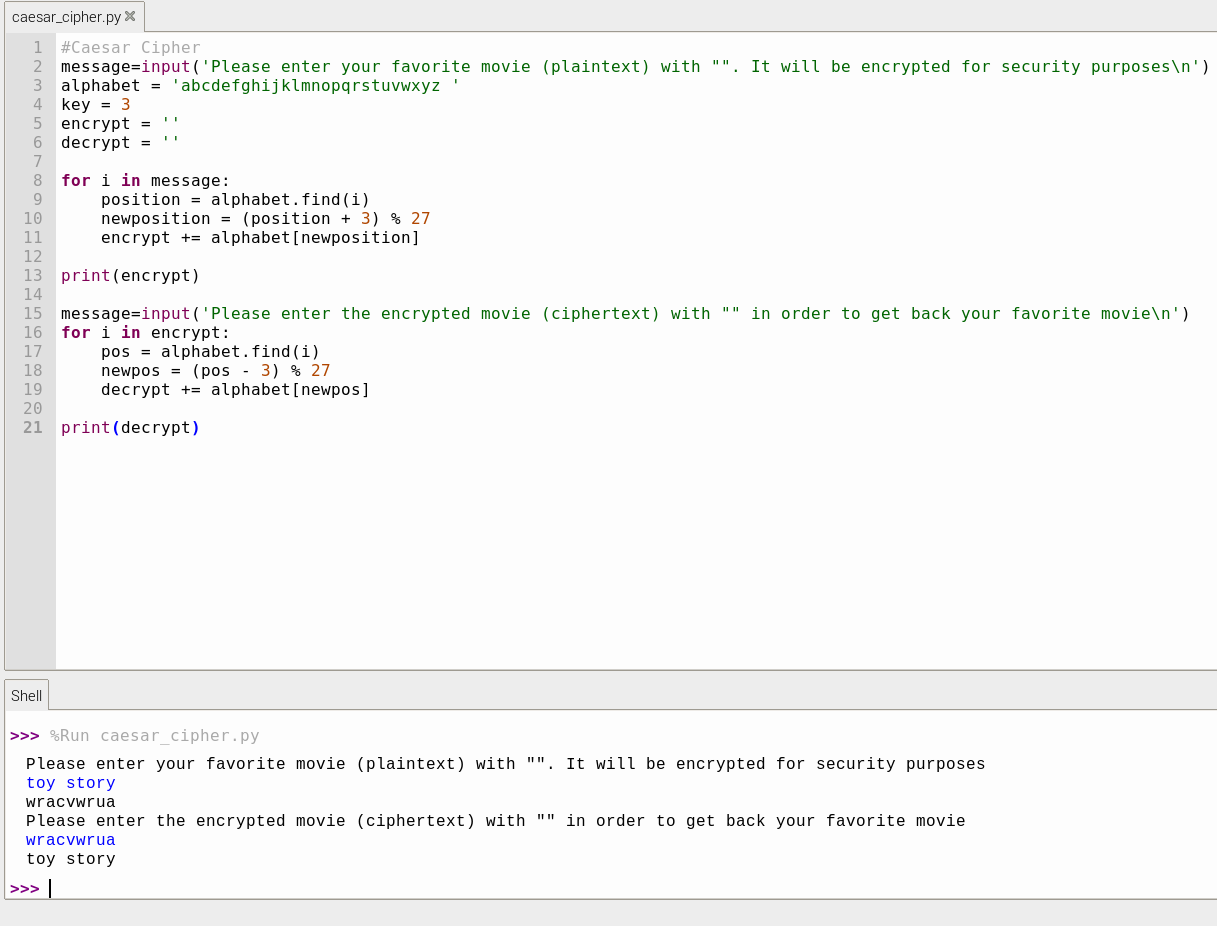
As you can see below here is the encryption “qadwsqhraokybuanis” and decryption “back to the future”:



# Project progress details

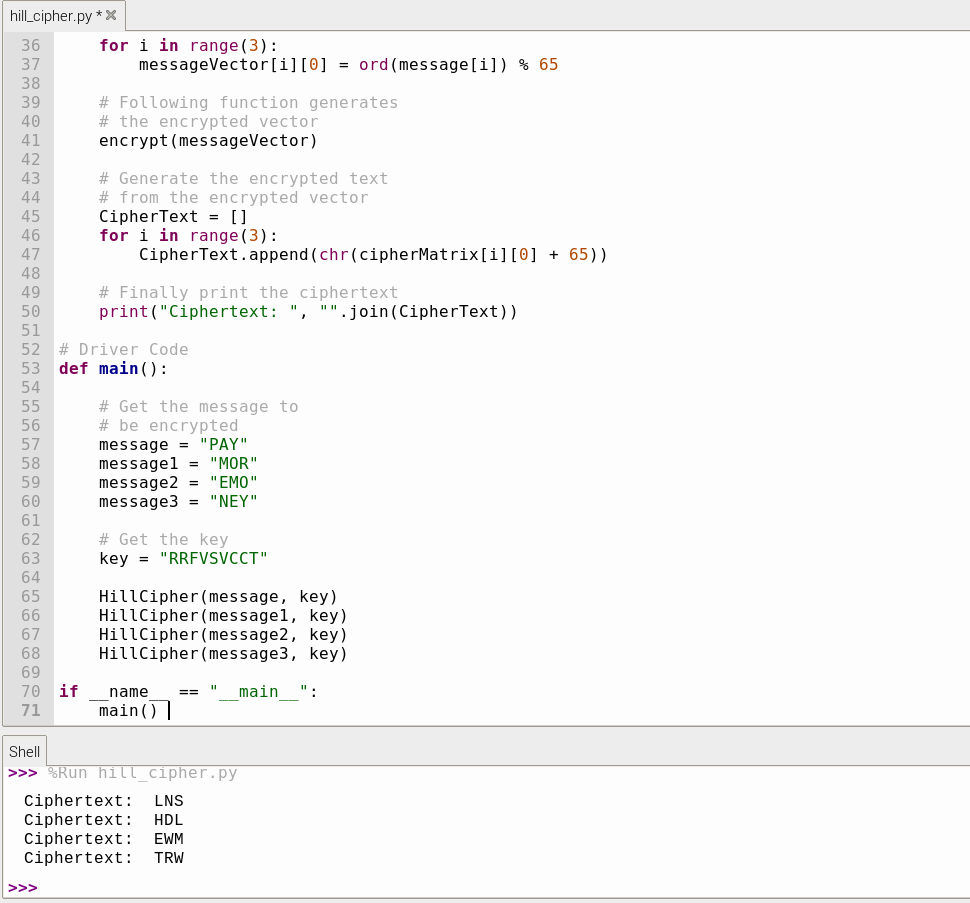
I started working to create a movie machine. I have been practicing with Raspberry Pi microcontroller and python programming language where I can use security authentication features. I started with Caesar Cipher algorithm but end up working with Hill Cipher algorithm. I have worked to convert a plaintext to ciphertext and vice versa by encrypting and decrypting the movies selected by the user.

As you can see in the below picture I can encrypt and decrypt “TOY STORY” for Caesar Cipher:



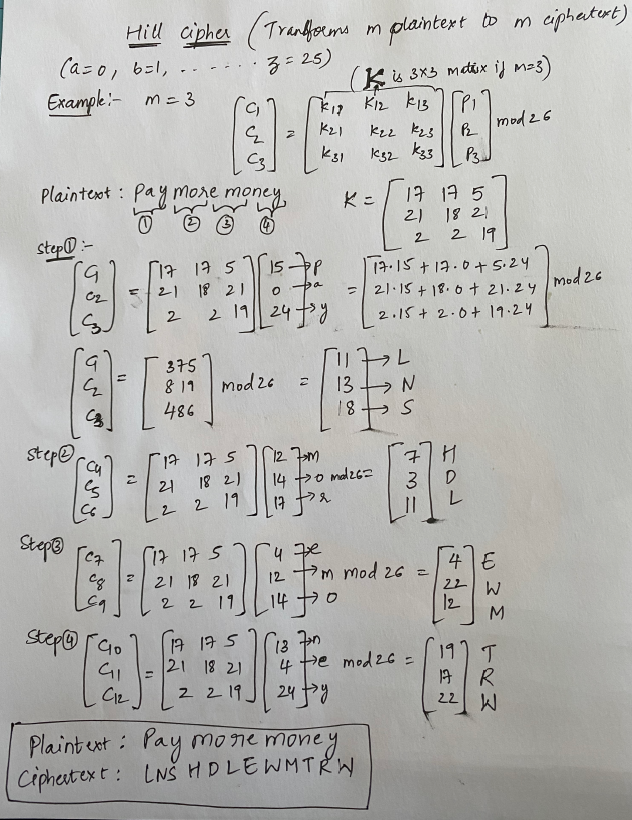
Moreover, I started to do a lot of testing with different strings to see if my program was working correctly or not. In fact, my program was working correctly with different movies. Also, I created three different scripts (3 programs) for Hill Cipher algorithm to test more security authentication features to encrypt/decrypt. Also, I started to ask me a lot of questions on how to implement some linear algebra functions in python. So, I started by hardcoding the programs with different scenarios. Likewise, I started to ask a lot of questions to my professor for guidance.

As you can see in the below picture I can encrypt for Hill Cipher by hardcoding the plaintext “PAYMOREMONEY” to ciphertext “LNSHDLEWMTRW”:

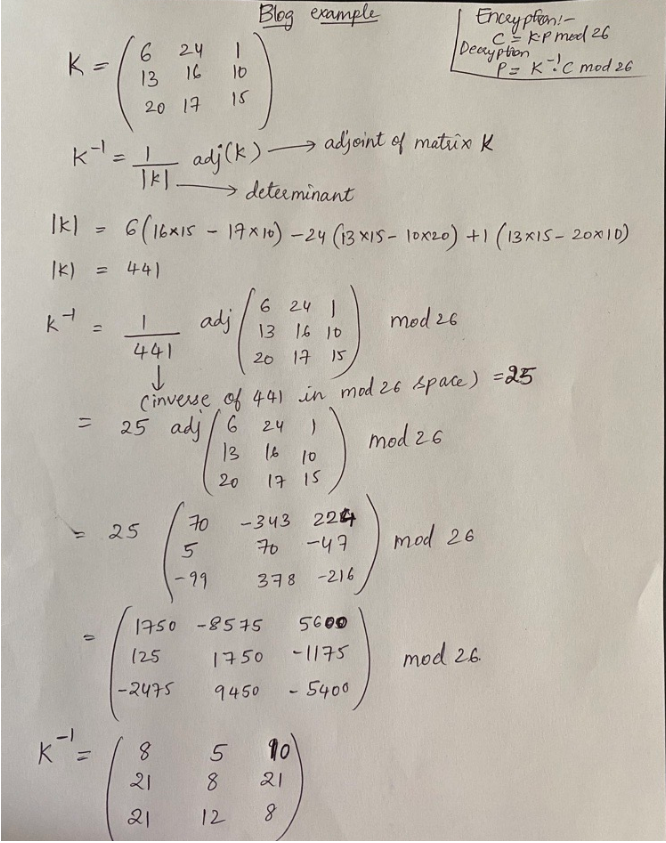


Then, my implementation plan was to identify a key that help me identify the title of my favorite movie. I could print the most recent movie as well as the year and type of the movie. Basically, the main idea was to make requests from the server and encrypt the movies. My plan was to get an API key to authenticate the request from the server. My main goal always was to get a ciphertext by dividing the movie (plaintext) given into syllables and then multiply it by a matrix to come up with a movie name encrypted (ciphertext) and then to decrypt the movie in the same manner to get back the same movie requested.

Likewise, I am verifying that my movie machine will encrypt the movie name (plaintext) and provide you with a movie name encrypted (ciphertext) as shown in the cybersecurity class example:



As mentioned before, I started to ask a lot of questions to my professor for guidance on how to get the inverse of the key matrix. Because the way numpy library does the inverse with linear algebra functions is quite different. I have learned that this is the correct manner to get the inverse of the key matrix:



# Pitfalls and Challenges

I had a lot of issues in hardware and software. For example, when I was trying to create a remote connection from my PC to the Raspberry pi I was getting errors and could not connect. At the end of the day, I successfully connected via remote desktop by putting the IP address of the raspberry pi:

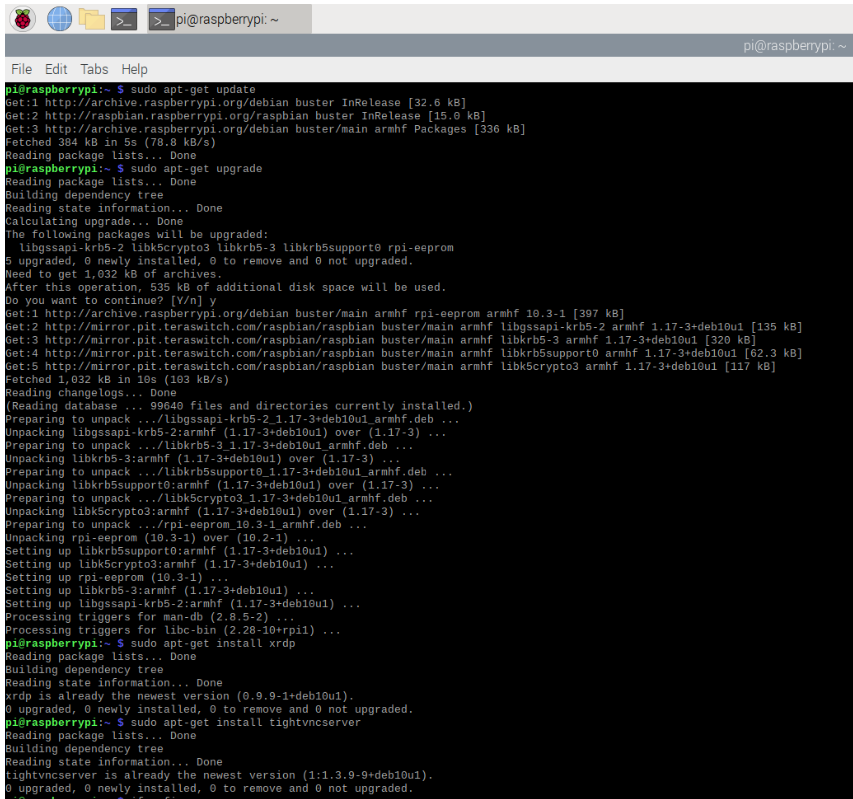
sudo apt-get update

sudo apt-get upgrade

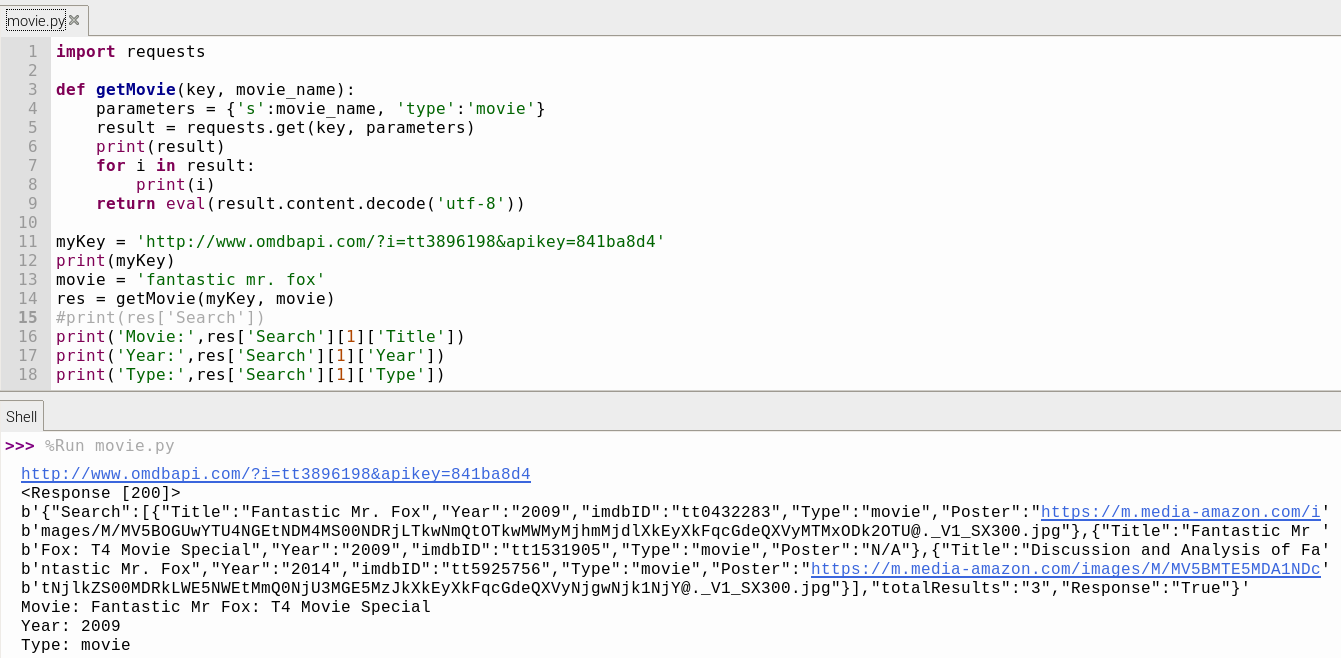
sudo apt-get install xrdp

sudo apt-get install tightvncserver

ifconfig



Also, I had some challenges when trying to connect to the database to retrieve the movie requested by the user. I had to learn a lot of python to get the movie requested by the user. As you may know we need to get a movie from the web and the way I am approaching this is by getting a URL with an open movie database as a key. I am requesting to the open movie database with an API key the movie selected by the user with python built in functions such in the below:



In summary, the steps to encrypt and decrypt using hill cipher algorithm are:

1. Choose secret key matrix k
2. Convert message into vectors that fit the size of K
3. Encrypt using C = E(K,P) = K\*P
4. Find inverse of matrix K in mod 26 or mod 27 (tricky part)
5. Decrypt using P = D(K,C) = inv(K)\*C

In conclusion, I had a lot of issues in this project but I could find out a solution for each of them. As mentioned before, I have worked with three different programs that provide you the same solution using three different approaches. It was challenging but satisfactory at the same time to end up with a finished product.

# References

<https://en.wikipedia.org/wiki/Hill_cipher>

<https://www.geeksforgeeks.org/hill-cipher/>

<https://gist.github.com/EppuHeilimo/0a901056f9e48a451e0c30a55537ad1b>