heer Documentation

Release 0.1

klettgau

Nov 30, 2020

Contents:

CHAPTER 1

Introduction

`Zezima Ciphers` will be a simple API that will encode and decode messages using several popular ciphers.

The aim of the project is to produce an API that uses FLASK etc ,Cryptology course work and Sphinx documentation.The entire project is done in Python 3 and Tested on Linux Mint.

1.1 Motivation

The project will hopefully provided inspiration and reference material for future students of Cryptology.It is to combine two seperate projects and to furtheur understand the Flask framework. Did you ever hear the tragedy of Darth Plagueis the Wise?

I thought not. It's not a story the Jedi would tell you. It's a Sith legend. Darth Plagueis was a Dark Lord of the Sith, so powerful and so wise he could use the Force to influence the midichlorians to create life.He had such a knowledge of the dark side that he could even keep the ones he cared about from dying. The dark side of the Force is a pathway to many abilities some consider to be unnatural. He became so powerful.. the only thing he was afraid of was losing his power, which eventually, of course, he did. Unfortunately, he taught his apprentice everything he knew, then his apprentice killed him in his sleep. It's ironic he could save others from death, but not himself.

1.2 Limitations

- The Ciphers can not brute force decode the messages so the private keys or the shared inforamtion must be supplied to the api in order to function.
- Jefferson Wheel Cipher is currently always fixed so the private key is the wheel offset.
- Enigma is restricted to M3 device, and doesn;t account for Kriegsmarine.

CHAPTER 2

Caesar Cipher

class ciphers.Julius.**Julius**

Bases: flask_restful.Resource

decode(*encoded*, *key*)

encode(*plain_text*, *key*)

get()

methods = {'GET', 'POST'}

post()

CHAPTER 3

Affine module

class ciphers.Affine.**Affine**
Bases: flask_restful.Resource

A Simple Affine Cipher. y = mx+b

check_coprime(*proposed_value*, *modolus*)
Could just have a list of values but check if the user provided value is coprime with 26.

Parameters

- **proposed_value** – The user provided value
- **modolus** – Constant value of 26.

Returns None if it is coprime otherwise an error message with coprime value.

decode(*encoded*, *inverse*, *b*)
Decode the selected character based off the intercept and coefficient. unencoded = inverse *(encoded -m)yt

encoded
The user's message encoded

inverse
The inverse of the coefficient.

b
The intercept value.

Return
The decoded value of the given character.

egcd(*a*, *b*)
Extended Euclidean algorithm .. attribute:: a

The selected number

b
The selected modulus

Returns The Greatest Common Denominator.

encode(*user_input*, *m*, *b*)
Encode the selected character based off the intercept and coefficient. .. attribute:: user_input

The user's message

m
The coefficient of the picked character

b
The intercept value.

Return
The encoded value of the given character.

get()

Returns The Encoded/Decoded User's message in json format.

methods = {'GET'}

modinv(*a*, *m*)
https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python This is to find the multiplicative inverse in order to check if the user provided value is coprime. It only exists when the gcd of a,m are 1.

a
The selected number

m
The selected modulus,26 is the default here.

Returns None or the multiplicative inverse of the selected number.

CHAPTER 4

Vigenere module

class ciphers.Vigenere.**Vig**
Bases: flask_restful.Resource

decode(*cipher*, *keyword*)
Same as Encoding but subtract the key from the cipher text.

encode(*plainext*, *keyword*)
Encode the message using the supplied values. The key dictates the row to use and the plaintext dictates the column. :param plainext: The message to be encoded. :param keyword: The private key to be used for the message.

Returns The encoded message using the provided values.

get()

methods = {'GET'}

print_tabula()
Print out a reference tabula recta.

Returns A Tabula Recta in String format.

CHAPTER 5

Jefferson module

class ciphers.Jefferson.**Jefferson**
Bases: flask_restful.Resource

check_wheel_parameters(*wheel_order*)
This is to ensure that user provided wheel order is meets the criteria. The wheel order must be values [0-24],no repeats and comma seperated. :param wheel_order:

Returns True if the wheel order is valid otherwise False.

decode(*encode_input*, *shiftFactor*, *wheel_order*)
This performs the decoding of the message.The shiftfactor is negative for decoding.

Parameters

- **user_input** – The message provided to the cipher.
- **shiftFactor** – The line which to read the message
- **wheel_order** – the order of the 25 wheels.

Returns The decoded message and the wheel order used in json format.

encode(*user_input*, *shift*, *wheel_order*)
This performs the encoding of the cipher text.The shift factor is positive to encode the message.

Parameters

- **user_input** – The message provided to the cipher.
- **shift** – the line which to read from.
- **wheel_order** – the order of the 25 wheels.

Returns The encoded message,shift value and wheel order in json format.

get()
This is the method that handles GET Requests for Jefferson wheel Cipher :param None since a Custom Parser Object is imported from CustomParser:

Returns The output of the encode/decode functions.

methods = {'GET'}

stringify_wheel(*wheel_order*)

To provide the user a copy and paste method to share wheel orders :param wheel_order: The order of the provided wheels.

Returns The wheel order in a comma seperated string

CHAPTER 6

Engima module

6.1 M3 Enigma class

class ciphers.Engima.**M3**

Bases: flask_restful.Resource

Represents the Enigma Machine used by the Wehrmacht/Heer.

slow_rotor
This is the leftmost rotor,the rotor that steps the least.

medium_rotor
This is the middle rotor,the rotor that double steps.

fast_rotor
This is the rightmost rotor, steps for every character.

reflector
This is the chosen reflector, it defualts to B reflector.

stecker_board
This is the generated stecker board.

fast_counter
Counts the amount of times stepped for Fast Rotor

medium_counter
Counts the amount of times stepped for Middle Rotor.

slow_counter
Counts the amount of times stepped for Slowest Rotor.

right_rotor_ring
Ring setting for the Fastest Rotor.

middle_rotor_ring
Ring setting for the Slow Rotor.

left_rotor_ring
Ring setting for the Slowest Rotor.

rotor_choices
This is a dictionary of the five rotor's wiring and the stepping position for each rotor.

reflector
This is a dictionary that stores the two reflectors used.

check_stecker_restrictions(*stecker_pair*)
Checks if user provided stecker board pairs is valid.

Parameters **stecker_pair** – String that represent the stecker board pairs.

Returns True or False if the user provided values are valid.

check_valid_char(*suspected_char*)

create_stecker_board(*wire_pairing*)
Thanks to Brian Neal's Enigma project as a reference steckerboard. Creates the Stecker Board based off the pairs provided by the User. Max amount of Pairs allowed are 10.

Parameters **wire_pairing** – Space seperated String that contains max 10 pairs IOT populate the Steckerboard.Example "AB HG LK ZI".

Returns No return values but can raise aborts if the request is malformed.

forward(*user_input*)
The character is passed through the machine right to left.

Parameters **user_input** – The message to be passed through the machine.

Returns The char encoded/decoded is returned.

get()
This is for all get requests for Enigma.

Returns The message decoded/encoded.

methods = {'GET'}

reflector_result(*char*)

reverse(*user_input*)
The character is passed through the machine left to right.

Parameters **user_input** – The message to be passed through the machine.

Returns The char encoded/decoded is returned.

run_machine(*message*)
Runs through each character and encodes/decodes it. :param message: The user provided message.

Returns The outout of encoding/decoding the message.

set_ring_setting(*user_input*)

set_rotors(*slow*, *medium*, *fast*)
Sets the Rotors to use from User Input.

Parameters

- **slow** – This is the leftmost rotor,the rotor that steps the least.
- **medium** – This is the middle rotor,the rotor that double steps.
- **fast** – This is the rightmost rotor, steps for every character.

Returns No return value,updates internal reference of chosen rotor.

Raises `ValueError` – The passed in rotor string is not a valid integer

set_rotors_intial_position(*user_input*)
Sets the Rotors starting position.

Parameters **user_input** – The User Provided string of the rotors start position.

Returns No return values,updates the internal reference of the rotors position.

set_up()
Set up the machine according to user provided values.

Returns The user provided message.

stecker_board_output(*char_to_be_stecker*)
The result of the character passing through the steckerboard. :param char_to_be_stecker: The current character to pass through the stecker.

Returns Either returns a character that is self-steckered or the result of the steckerboard.

step_rotors()
Steps the User Chosen Rotors of the Enigma Machine for each character. The Middle Rotor does contain the double step flaw.

Returns No return value,updates internal position of the rotor.

Raises `KeyError` – The key doesn't exist in the rotor.

6.2 Rotor Dic

sadasdasdasd:

```
rotor_choices = {
    1: {
        'wiring': 'EKMFLGDQVZNTOWYHXUSPAIBRCJ',
        'step': 'Q'
    },
    2: {
        'wiring': 'AJDKSIRUXBLHWTMCQGZNPYFVOE',
        'step': 'E'
    },
    3: {
        'wiring': 'BDFHJLCPRTXVZNYEIWGAKMUSQO',
        'step': 'V'
    },
    4: {
        'wiring': 'ESOVPZJAYQUIRHXLNFTGKDCMWB',
        'step': 'J'
    },
    5: {
        'wiring': 'VZBRGITYUPSDNHLXAWMJQOFECK',
        'step': 'Z'
    },
}
```

6.3 Reflector Dict

These are the entries:

```
{
    'reflector_b': 'YRUHQSLDPXNGOKMIEBFZCWVJAT',  # b reflector
    'reflector_c': 'FVPJIAOYEDRZXWGCTKUQSBNMHL'  # c reflector
}
```

CHAPTER 7

clean

7.1 CustomParser module

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

R

S

V