

2021

Powerhouse Documentation

### FARAI CHOLRIS MCCHARLES MATYUKIRA

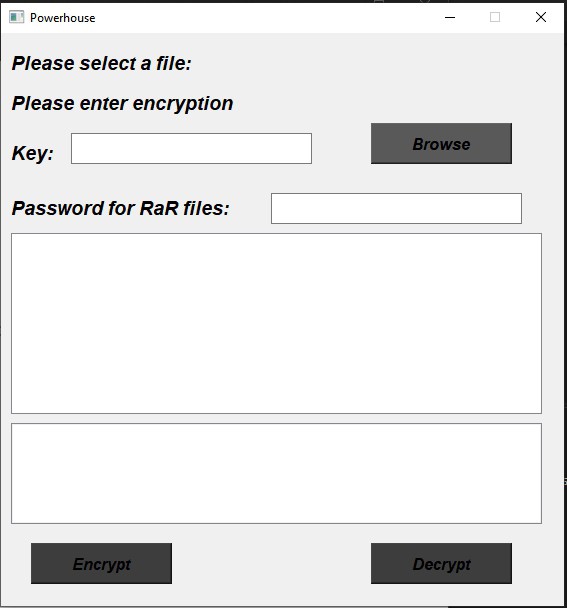
|  |  |
| --- | --- |
| Powerhouse Heavy | 3 - 6 |
| Powerhouse Lite | 7 - 9 |
| Powerhouse Calculation | 10 - 16 |

Powerhouse :

Powerhouse has two versions with is called Heavy and Lite with encrypt different information of files

Powerhouse Heavy:

Powerhouse Heavy encrypts and decrypts the internal information of files. It can encrypt data specifically from a .txt files, .jpeg, .jpg, .png and rar files but it doesn’t return the rar files into a rar format.

GUI.py – Powerhouse Heavy

UML diagram – Powerhouse Heavy

|  |
| --- |
| Mainwindow |
| + Dialog |
| + init ( self )  + browsefiles( self )  + encryptfiles( self )  + decryptfiles(self) |

GUI Explained – Powerhouse Heavy

The Graphical user interface contains the class Mainwindow with the 4 methods shown in the UML diagram above.

The GUI contains 3 buttons namely Encrypt, Decrypt and Browse.

The encrypt button initializes encryption on a file chosen by the user using the GUI, the decrypt button decrypts the encrypted file and the browse button allows the user to open a file dialog whereby they’ll choose a file to encrypt and decrypt using the encrypt and decrypt buttons.

There is a line edit where the user will be allowed to enter their own encryption key. The GUI also has two (2) list widgets.

The first list widget is used to display files chosen by the user to encrypted and second list widget displays encrypted files after the encrypt button is pressed and encryption is done.

Explanation:

file\_path\_finder()

**Parameters and arguments**: path\_array(array of characters in file paths of the files to be encrypted files) and the key(given by the user) , password (is a variable that has a defined value)

**explanation:** the function goes through a series of if and elif statements and for statements to go through the file path and file extension of the files loaded to powerhouse. The file path is split into head and tail using the split. Followed by checking the tail of the file and the function decides on the file type and also decides what function or class to use for encryption. Note that only .jpeg files

or .png files, .rar files, .txt

If tail ends with .rar: if tail.endswith(".rar"):

Function prints that an RAR is to be encrypted, requests the password for the .rar file password and extracts form the .rar file and they are store in the folder extract\_point .

If the file in extract\_point is a text file :

The information inside the file are read from the file and they are put through the ord() function and are saved in a array called array . A instance of Encrptor() is made and calls powerhouse and then array with key is sent to be encrypted .

If the file in extract\_point is a image :

If tail ends with .jpeg or .png:

Function starts of by printing photo to be encrypted, reads the image file, converts the image to bytes, stores the bytes in an array and a instance of Encrptor() is made and calls powerhouse and then array with key is sent to be encrypted .

elif tail.endswith(".jpeg") or tail.endswith(".png"):

If tail ends with .jpeg or .png:

Function starts of by printing photo to be encrypted, reads the image file, converts the image to bytes, stores the bytes in an array and a instance of Encrptor() is made and calls powerhouse and then array with key is sent to be encrypted .

*elif* tail.endswith(".txt"):

The information inside the file are read from the file and they are put through the ord() function and are saved in a array called array . A instance of Encrptor() is made and calls powerhouse and then array with key is sent to be encrypted .

If none of the tails for the ends of the files are found the function prints an error using exception handling.

The encrypted information is then save in the powerhouse specific file type (is a json file with a differentiated file extention .ph)

*with* open(*f*"endpool/{tail.split('.')[0]}.ph", "w") *as* outfile:

file\_content = json.dumps(

{

"reverse\_arrays": reverse\_arrays, "final\_text\_data": final\_text\_data, "file\_name": file\_path,

}

)

outfile.write(file\_content) print("Encrypted file saved")

Decryption area of Heavy

Pool\_search()

This function searches for files in the pool with the help of a key(That its takes in as input). Also consists of arrays: path\_arrays and file\_names that are related to the file paths of the files to be encrypted that are appended. Lastly this function calls the file\_path\_finder function.

def pool\_search():

key = input("Please enter in the key : ") path\_array =[]

file\_names = []

for directory ,subdirectories,filenames in os.walk("pool"): for files in filenames:

print (files) file\_names.append(files)

paths = os.path.join(directory,files) path\_array.append(paths)

file\_path\_finder(path\_array,key)

pool\_search searches through the end pool and retrevies the paths and file names of the files that have been encrypted and then the paths are appended into the array called path\_array and the key for the GUI is the sent to file\_path\_finder ()

File\_path\_finder() is also used for decryption on the boolean which checks if the file the file is powerhouse specific file type .ph

*elif* tail.endswith(".ph"):

.ph (.ph files are powerhouses own unreadable file type for saving encrypted files) files are expected if not then an error is returned.

Note .ph is only picked up if there are any encrypted files in the end pool.

Then the encrypted file is opened and the information inside is json loaded and assigned to the variable array .The variable array contains the reverse\_arrays and array of the file information and the original file name with are assigned to the variables final\_text\_data ,reverse\_arrays, filepath and then those variables are sent to the function decrypt().

*elif* tail.endswith(".ph"):

data\_array = []

*with* open(file\_path,"r") *as* fi: text\_file = fi.read()

array = json.loads(text\_file) final\_text\_data = array["final\_text\_data"] reverse\_arrays = array["reverse\_arrays"] filepath = array["file\_name"]

*for* values *in* final\_text\_data: data\_array.append(values)

print("MODE == DECRYPT")

decrypt(data\_array , reverse\_arrays, key, filepath) print("Encrypted file saved")

*def* decrypt(*data\_array*, *final\_text\_data*, *key*, *tail*): head, fileName = os.path.split(tail) export\_data = Decrptor()

decryptedDataArray = export\_data.powerhouse\_decrypt(data\_array, final\_text\_data, key,fileName)

decryptedData = bytearray(decryptedDataArray)

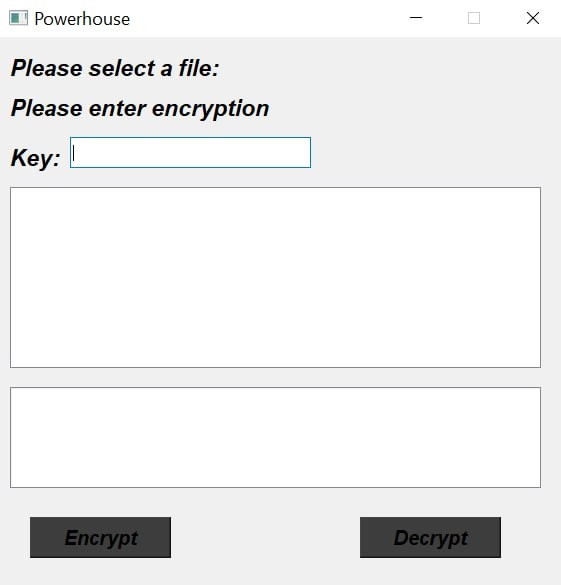
*with* open(*f*"reversed\_files/ph\_{fileName}", "wb") *as* outfile: outfile.write(decryptedData)

print("decryption completed")

A instance of Decrptor() is made and then the if of final\_text\_data and the other variables is then sent to be powerhouse\_decrypt for decryption

Powerhouse Lite:

This algorithm encrypts and decrypts the file itself instead of the internal information of the files. This algorithm unlike its sister program (Powerhouse Heavy) it can encrypt and decrypt every data type even audio files .

GUI.py – Powerhouse Lite

UML diagram – Powerhouse Lite

|  |
| --- |
| Mainwindow |
| + Dialog |
| + init ( self )  + browsefiles( self )  + encryptfiles( self )  + decryptfiles(self) |

GUI Explained – Powerhouse Lite

The Graphical user interface contains the class Mainwindow with the 4 methods shown in the UML diagram above.

The GUI contains 2 buttons namely Encrypt, Decrypt.

The encrypt button initializes encryption on a file chosen by the user using the GUI, the decrypt button decrypts the encrypted file. Both buttons open a file dialog whereby they’ll choose a file to encrypt and decrypt. This allows for fast and easy access

There is a line edit where the user will be allowed to enter their own 12 character encryption key. The GUI also has two (2) list widgets.

The first list widget is used to display files chosen by the user to encrypted and second list widget displays a message to the user to inform them that the file has bee encrypted or decrypted. The second list widget also gives the user an error message whenever something goes wrong in the encryption and decryption process.

*def* browsefiles(*self*):

fname = QFileDialog.getOpenFileName(self, "open file") self.lstFile.addItem(fname[0]) self.files.append(fname[0])

*return*

*def* encryptfiles(*self*): self.browsefiles() print(self.files[0]) self.filename.setMaxLength(12) key = self.filename.text() print("[key] ->", key) self.encrypt(self.files[0], key) key = self.filename.clear()

*def* encrypt(*self*, *filePath*, *key*): fileDataArray = []

head, fileName = os.path.split(filePath) file = open(filePath, "rb")

fileData = file.read() file.close()

fileDataArray = bytearray(fileData) print("[head] -> ", head)

print("[fileName] -> ", fileName.split(".")) *# print("[fileDataArray] -> ", fileDataArray)* export\_data = Encrptor()

encryptedData = export\_data.powerhouse(fileDataArray, key, fileName) reverse\_arrays, final\_text\_data = encryptedData

*with* open(*f*"endpool/{fileName.split('.')[0]}.ph", "w") *as* outfile: file\_content = json.dumps(

{

"reverse\_arrays": reverse\_arrays, "final\_text\_data": final\_text\_data, "file\_name": fileName,

}

)

outfile.write(file\_content) print("encryption completed")

Code explanation:

def browsefiles(self): This function allows for files opened through file dialog to be returned in list widget (lstFile) and appended.

def encryptfiles(self): This function opens file dialog and sets key length to 12 characters then clears the line edit once encryption is complete after the encrypt button is pressed

def encrypt(self, filePath, key): encrypt takes the file path, and the key created from encryptfiles. This is given to function encrypt, encrypt opens and reads the bytes (“rb”) and sends it to FileDataArray. This is then given to Powerhouse to be encrypted.

Once encryption is complete the decryption process can commence which is the opposite of the powerhouse encryption algorithm .

The variable array contains the reverse\_arrays and array of the file information and the original file name with are assigned to the variables final\_text\_data ,reverse\_arrays, file\_name and self.file[0](is an array with the file to be decrypted )and then those variables are sent to the function decrypt().

*def* decryptfiles(*self*):

self.browsefiles() print(self.files[0]) self.filename.setMaxLength(12) key = self.filename.text() print("[key] ->", key)

*with* open(self.files[0], "r") *as* file: text\_file = file.read()

data = json.loads(text\_file) final\_text\_data = data["final\_text\_data"] reverse\_arrays = data["reverse\_arrays"] file\_name = data["file\_name"]

self.decrypt(self.files[0], final\_text\_data, reverse\_arrays, key, file\_name) key = self.filename.clear()

*def* decrypt(*self*, *filePath*, *data\_array*, *final\_text\_data*, *key*, *tail*): head, fileName = os.path.split(filePath)

export\_data = Decrptor()

decryptedDataArray = export\_data.powerhouse\_reverse( data\_array, final\_text\_data, key, tail

)

A instance of Decrptor() is made and then the if of final\_text\_data and the other variables is then sent to be powerhouse\_decrypt for decryption.

MAIN ENCRYPTION CALCULATION CORE

powerhouse\_encrypt:

Encrptor **has method powerhouse**

UML Diagram for Encrptor:

|  |
| --- |
| Encrptor |
| + powerhouse ( bytearray1(array), key(string), file\_name(string)) |
|  |

**Parameters and arguments:** bytearray1(array) and key(string), file\_name(string)

**What it returns**: final\_data\_array

final\_data\_array is a variable that will receive the encrypted arrays of values of the file and return it to the GUI.py

final\_data\_array = step\_two(lpt,rpt,fixed\_key) print("powerhouse\_final\_array-->",final\_data\_array)

*return* final\_data\_array

## Explanation:

**final\_data\_array = holds two array with are the reverse\_arrays( calculation log that is dictionaries ) and final\_text\_data(which is the encrypted information of the file that was given )**

fixed\_key = is variable that calls and receives the more refined version of the key that will be used in calculations using encryption\_key\_refine() function.

byte\_array1 =is the array of data needs to be encrypted Calculation :

byte\_array1 is modulated by two to determine if the array of data is even or uneven.

-If even the array is split into equal array’s that will then be known as LPT and RPT

-If uneven the array will be split in to two uneven LPT and RPT arrays in which one in all cases LPT will be smaller and RPT will be bigger . The RPT will always take longer to be encrypted

Once LPT and RPT are made they are given to step\_two to be encrypted and the returned arrays are stored in **final\_data\_array**

*def* powerhouse(*self*,*bytearray1*,*key*,*file\_name*):

*try* :

lpt = []

rpt = []

print("Now in Powerhouse Heavy Encrypt\_Core ") final\_data\_array =[]

fixed\_key = encryption\_key\_refine(key)

*# uneven number array*

*if* len(bytearray1)% 2 == 1: print("Uneven")

extra\_value = len(bytearray1) x = len(bytearray1)

max = int(x/2) print("max -->",max) *for* i *in* range(max):

lpt.append(bytearray1[i])

*for* i *in* range (max,len(bytearray1)): rpt.append (bytearray1[i])

print("reverse\_third\_step lpt>>>:",lpt) print("reverse\_third\_step rpt>>>:",rpt)

final\_data\_array = step\_two(lpt,rpt,fixed\_key,file\_name)

*elif* len(bytearray1)% 2 == 0: print("Even")

max = int(len(bytearray1)/2)

*for* i *in* range (0,max): lpt.append(bytearray1[i])

*for* i *in* range (max,len(bytearray1)): rpt.append(bytearray1[i])

print("reverse\_third\_step lpt>>>:",lpt) print("reverse\_third\_step rpt>>>:",rpt)

final\_data\_array = step\_two(lpt,rpt,fixed\_key,file\_name)

*return* (final\_data\_array)

# encryption\_key\_refine():

**Parameters and arguments:** encrypt\_key

def encryption\_key\_refine(encrypt\_key)

**What function returns:** true\_key\_array

return true\_key\_array

**Explanation:** has two local arrays : true\_key\_array and temp\_storage(array ) and variable temp\_value (int). Function iterates through encryption\_key and values are stored in temp\_storage array, the fourth value of temp\_storage is orded and multiplied by 12 and saved into variable i and the same is done for the 8th value and saved into the variable y. After that 2 Boolean values are declared i\_big and y\_big, they remain true, while i\_big is true i is divided by 20 if i is greater than 50

and if i is not a decimal after being divided by 20 it is saved as temp\_value and added to the true\_key\_array and the same is done for variable y. If i is no an integer 10 is added to it and it is turned into an integer using int and added to true\_key\_array after i\_big is made false then the loop is broken. And true\_key\_array is returned to calling point

*def* encryption\_key\_refine(*encrypt\_key*):

*try* :

true\_key\_array = [] temp\_value = 0 temp\_storage =[]

*for* i *in* encrypt\_key: temp\_storage.append(i)

i = ord(temp\_storage[3])\*3 y = ord(temp\_storage[7])\*3 i\_big = True

y\_big = True z\_big = True

*while*( i\_big ):

*if* i > 50:

i= i/20

*if* isinstance(i, int)==True: temp\_value = i true\_key\_array.append(temp\_value) i\_big = False

*elif* isinstance(i, int)==False: temp\_value = int(i + 10) true\_key\_array.append(temp\_value)

i\_big = False

*while*(y\_big ):

*if* y > 50:

y= y/20

*if* isinstance(y, int)==True: temp\_value = y true\_key\_array.append(temp\_value) y\_big = False

*elif* isinstance(y, int)==False: temp\_value = int(y + 10) true\_key\_array.append(temp\_value) y\_big = False

*return* true\_key\_array

*except* Exception *as* e:

print("ERROR-in->encryption\_key\_refine>>>:", e)

# step\_two

**Parameters and arguments:** lpt, rpt and encrypty\_number

## What the function returns:

*return* final\_data\_array

## Explanation:

This is the core of the encryption which comprises of random selection of operation on each individual array(LPT and RPT) by utilizing the random function mainly the rand int and XOR operator in a three section calculation on the individual values of LPT and RPT .

True\_operation = is a random number created using the randint() function and is limited to a range of 1 to 3 to by the used to reverence a invisible order of operators

*if* true\_operation == 1 :

value = i - round\_encryption\_number2

*elif* true\_operation == 2:

value= i + round\_encryption\_number2

*elif* true\_operation == 3:

value = i \* round\_encryption\_number2

## Local functions variables :

round\_control\_number\_lpt = 0 cal1lpt = []

cal2lpt = []

**How :**

1. encrypt\_number is a array of int value .Those values of encrypt\_number are assigned to three variables which are containers that will be used in the round calculations **:**

round\_encryption\_number1 = encrypt\_key[0] round\_encryption\_number2 = encrypt\_key[1]

1. The LPT array, RPT array are accessed and each value is put through three sections of calculations .LPT is calculated first and RPT is calculated after :

\*Note the code For LPT and RPT calculation is similar in the Example below LPT is the array that is being accessed but same processes is done on RPT

cal1lpt = []

cal2lpt = []

print("lpt data receiving check") *#round one* round\_control\_number\_lpt += 1 *for* i *in* lpt:

*# XOR calculation*

value = 0

value = i ^ round\_encryption\_number1 cal1lpt.append(value)

round\_control\_number\_lpt += 1

*for* i *in* cal1lpt:

true\_operation = randint(1,2) reverse\_encryption\_key\_lpt.append(true\_operation) value = 0

*if* true\_operation == 1 :

value = i + round\_encryption\_number2

*elif* true\_operation == 2:

value= i \* round\_encryption\_number2

*elif* true\_operation == 3:

value = i - round\_encryption\_number2 cal2lpt.append(value)

* First section the LPT array is accessed and each value is XOR with the round\_encryption\_number1 and the value are saved in the cal1 array
* Second section the cal1 array is accessed and the a random number is created for each accessed value and that value passes through nested Boolean to determine what operator will be used for its calculation so each value has a different micro calculation done to it and the true\_operation number is save into the reverse\_encryption\_key\_lpt array and that array will later be added to the new dictioctionary called reverse\_arrays reverse\_arrays will have 2 keys and 2 values, first key being lpt\_reverse\_array and second key rpt\_reverse\_array, values being: the reverse\_encryption\_key\_lpt array value lpt, rpt are then sent to third\_step and are returned back to the variable final\_text\_data which will be returned back and then saved in the .ph file for representing an encrypted file.

reverse\_arrays = {} reverse\_arrays = {

"lpt\_reverse\_array":reverse\_encryption\_key\_lpt, "rpt\_reverse\_array":reverse\_encryption\_key\_rpt

}

final\_text\_data = third\_step(cal2lpt,cal2rpt) print("Round LPT: ",round\_control\_number\_lpt," :") print("Round RPT: ",round\_control\_number\_rpt," :") *return* (reverse\_arrays, final\_text\_data)

Libraries used:

# Random

The random library was used to generate random numbers between 1 and 5 for unpredictability, if a normal sequence of numbers from 1 to 5 was used the encryption would have been more predictable.

**Seed -** Seed is used to initialize random numbers so that the same random numbers are generated on multiple execution of the code for use in decryption.

# third\_step:

*def* third\_step(*lpt*, *rpt* ):

*try*:

*if* lpt !=[]:

print("lpt recived ")

*if* rpt !=[]:

print("rpt recived ") textdata\_array = []

*for* i *in* lpt:

textdata\_array.append(i)

*for* j *in* rpt:

textdata\_array.append(j)

*# print("third step array -->", textdata\_array) return* textdata\_array

print ("Encryption has been completed")

*except* Exception *as* e:

print("ERROR-in->third\_step>>>:", e)

## Parameters and arguments: LPT and RPT

**What function returns: textdata\_array (a combination of the LPT and RPT)**

**Explanation: T**he LPT and RPT arrays are joined by being added to the textdata\_array and then the textdata\_array is returned to calling statement

Libraries used:

# PYQT5

Is a python binding of the cross-platform GUI toolkit Qt, implemented as a python plug-in.

**Sys –** a module in python that provides various functions and variables that are used to manipulate different parts of the python runtime environment.

**Os –** provides function for interacting with the operating system.