2021

Powerhouse Documentation

[COMPANY NAME] | [Company address]

Group Members :

Farai Chorlis McCharles 34855467

Bethel Zvomuno 31709974

Charmain Lebelo 34687602

Seisaphoko Satekge 32395620

Tembelani Tshaka 35560940

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

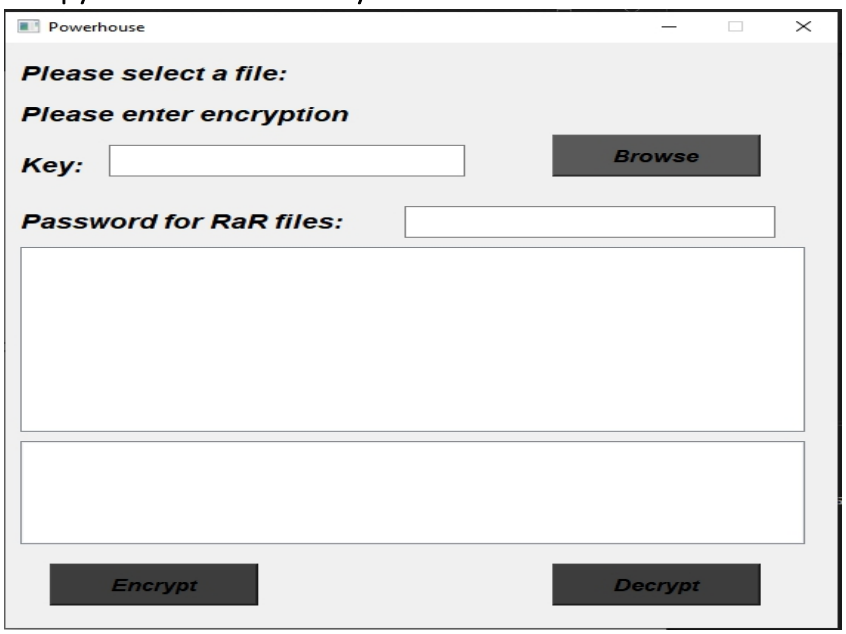
Powerhouse :

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")
```

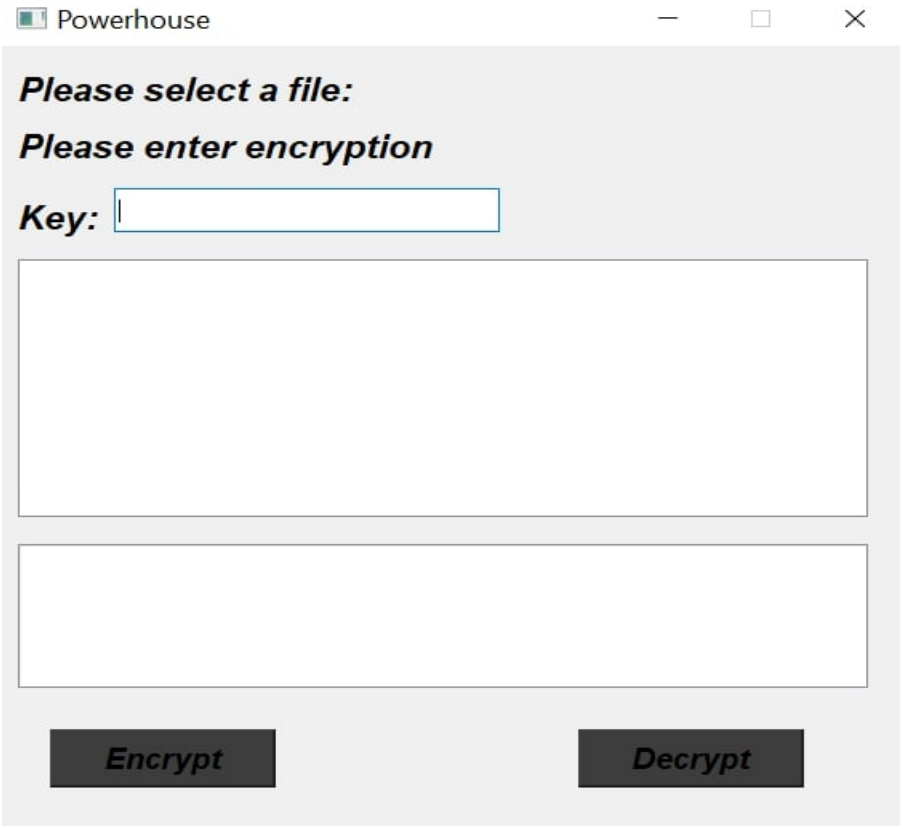
```
def decrypt(data_array, final_text_data, key, tail):
        head, fileName = os.path.split(tail)
        export_data = Decrptor()
        decryptedDataArray = export_data.powerhouse_decrypt(data_array, final_text_data, key,fileName)

        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]


```

2) 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: The 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.