MULTI-LAYER FILE ENCRYPTION TOOL USING RSA AND AES

A PROJECT REPORT

Submitted by

NAME OF THE CANDIDATES
(Register No)

TRILOK DHAKAD (20BCY10126)

in partial fulfillment for the award of the degree
of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING (SPECIALIZATION IN CYBER SECURITY AND DIGITAL FORENSICS)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
VIT BHOPAL UNIVERSITY
KOTHRIKALAN, SEHORE
MADHYA PRADESH – 466114

VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **“Multi-layer file encryption tool using RSA and AES”** is the bonafide work of “TRILOK DHAKAD (20BCY10126)**”** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/ research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR

Dr. Rakesh R
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

PROJECT GUIDE

Dr. Rizwan Ur Rahman
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination is held on _______________

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to Dr. Shishir Kumar Shandilya, Head of the Department, School of Cyber security and digital forensics for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide Dr. Rizwan Ur Rahman, for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Computer science engineering (BCY), who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

LIST OF FIGURES AND GRAPHS AND TABLES

FIGURE NO.	TITLE	PAGE NO.
1.	Organization of the project flowchart	9
2.	Software architectural design	15-16
3.	Working layout of Forms flowchart	30
4.	Initial content of plaintext files	31
5.	Output window	32
6.	Viewing a file	33
7.	Generating keys for odd.txt	34
8.	Generating keys for even.txt	34
9.	Content of public_key.json after generating keys	35
10.	Encryption of odd.txt	38
11.	Encryption of even.txt	40
12.	Decryption of odd.txt	40
13.	Decryption of even.txt	40
14.	Time taken for encryption and decryption using different algorithms	41

ABSTRACT

The purpose of the project is to provide secure encryption using multiple layers. Our project deals with higher level encryption techniques which allows us secure data exchange. Our aim is to create a hybrid cryptography system that makes it even tougher to break. Not only that, it reads the length of the name of the text file to be encrypted and if the length is odd it encrypts through AES, and if the length is even, it encrypts using RSA. Already existing cryptographic models use single encryption standard with single layer. This motivates us to move forward with our project. We have created this system using already existing encryption standards like RSA and AES. Our system can easily encrypt text files with a maximum word limit of 214 bytes (RSA). We have also worked upon the maximum file size that can be encrypted and the amount of time taken to encrypt i.e., time and space complexity. Utilizing all the modules and function we will create a multi-layer cryptosystem which will be giving maximum security to the user and their data.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	List of Abbreviations	iii
	List of Figures and Graphs	iv
	List of Tables	v
	Abstract	vi
1	**CHAPTER-1:**	
	PROJECT DESCRIPTION AND OUTLINE	1
	1. Introduction	
	1.2 Motivation for the work	.
	1.3 Problem statement	.
	1.4 Objective of the work	.
	1.5 Organization of the project	
2	**CHAPTER-2:**	
	RELATED WORK INVESTIGATION	
	2.1 Introduction	
	2.2 Core area of the project	
	2.3 Existing Approaches/Methods	
	2.3.1 Approach -1	
	2.3.2 Approach -2	
	2.3.3 Approach -3	
	2.4 Pros and cons of the stated Approaches	
	2.5 Issues/observations from investigation	
	2.6 Summary	

3	**CHAPTER-3:** **REQUIREMENT ARTIFACTS** 3.1 Introduction 3.2 Hardware and Software requirements 3.3 Specific Project requirements 3.3.1 Data requirement 3.3.2 Functions requirement 3.4 Summary	
4	**CHAPTER-4:** **DESIGN METHODOLOGY AND ITS NOVELTY** 4.1 Methodology and goal 4.2 Functional modules design and analysis 4.3 Software Architectural designs 4.4 Subsystem services 4.5 User Interface designs 4.6 Summary	
5	**CHAPTER-5:** **TECHNICAL IMPLEMENTATION & ANALYSIS** 5.1 Outline 5.2 Technical coding and code solutions 5.3 Working Layout of Forms 5.4 Prototype submission 5.5 Test and validation 5.6 Performance Analysis (Graphs/Charts) 5.7 Summary	

6	**CHAPTER-6:** **PROJECT OUTCOME AND APPLICABILITY** 6.1 key implementations outline of the System 6.2 Significant project outcomes 6.3 Project applicability on Real-world applications 6.4 Inference	
7	**CHAPTER-7:** **CONCLUSIONS AND RECOMMENDATION** 7.1 Outline 7.2 Limitation/Constraints of the System 7.3 Future Enhancements 7.4 Inference	
	References	

CHAPTER-1:

PROJECT DESCRIPTION AND OUTLINE

1. Introduction

The purpose of the project is to provide secure encryption using multiple layers. Our project deals with higher level encryption techniques which allows us secure data exchange. Utilizing all the modules and function we will create a multi-layer cryptosystem which will be giving maximum security to the user and their data.

2. Motivation for the work

Already existing cryptographic models use single encryption standard with single layer which can be decrypted easily but we are providing 2 layers. This motivates us to move forward with our project.

3. Problem Statement

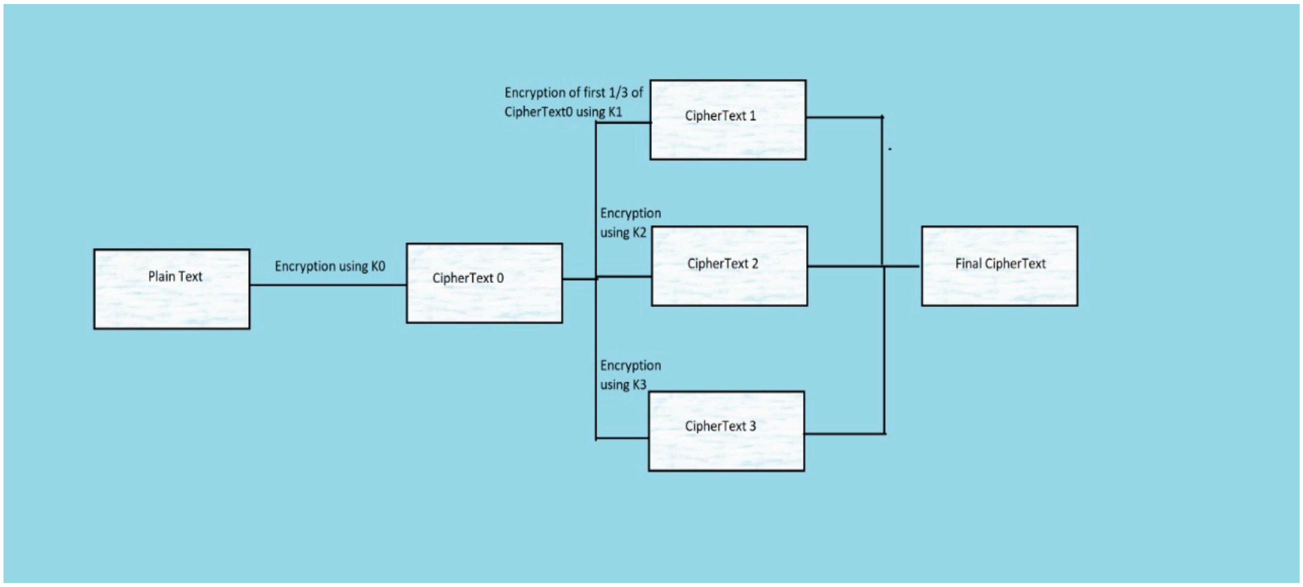
To create a system which can use multiple encryption algorithms.

1.4 Objective of the work

Our aim is to create a hybrid cryptography system that makes it even tougher to break. Not only that, it reads the length of the name of the text file and if the length is odd it encrypts through AES and if the length is even, it encrypts using RSA.

4. Organization of the project

Encryption in 2 layers -first using K0 and second using 3 different keys by dividing the cipher text obtained from K0 into 3 parts.



CHAPTER-2:

RELATED WORK INVESTIGATION

2.1 Introduction

There are many algorithms that are needed to be replaced and there also many which will be replaced in near future. So, we will try to make a secure algorithm which can replace most of the encryption algorithm in future.

2.2 The core idea of the project is to make the use of current technology and built an advanced technology.

2.3 Existing Approaches/Methods

2.3.1 Approach -1

The 1st existing approach/method is singly using AES or RSA algorithm to encrypt the file.

2.3.2 Approach -2

The 2nd existing approach/method is encrypting file with AES algorithm and then again encrypting the ciphertext from RSA algorithm.

2.4 The approach-1 is just simple one which will surely be broken in near future which needed to be replaced with another algorithm. Also, a similar kind algorithm to AES is DES which have been already broken.

As approach-2 is also simple with another layer of different encryption algorithm. As DES is already broken so surely AES will also be broken and using a algorithm which will be broken is not safe for the user for their data to use it in multi-layer encryption.

2.5 Issues/observations from investigation

As we are moving ahead in technology the cyber security is gaining more and more importance and digital security is becoming the most important aspect in everyday life. Every time we need more secure which should be better than the previous algorithms. As nothing remains forever, cryptanalyst always try to break the secure algorithm for which we need to have another secure algorithm. As we are still moving to the peak of Technology and Information Age the demand of secure algorithms would never end.

2.6 Summary

Encryption algorithm that are present today need an upgraded version of themselves or a new encryption algorithm to be secure enough to privacy and safe digital experience to users.

CHAPTER-3:

REQUIREMENT ARTIFACTS

3.1. Introduction

Our program works on python 3 and depends on some inbuilt and open-source modules to perform the desired functions. It needs a working computer that supports python 3. Also, it depends on the files named cryptRSA.py, cryptAES.py and read_write.py to access various functions that are used for encrypting text, viewing files, writing keys, etc.

3.2. Hardware and software requirements

3.2.1. Hardware requirements:

A Working computer that can run python3.

3.2.2. Software requirements:

3.2.2.1. Programming language: Python3

3.2.2.2. Inbuilt Modules:

- base64 (b64encode(), b64decode())
- hashlib (scrypt())
- json

3.2.2.3. Other Modules:

- Pycryptodome
- pycryptodomex

3.3. Specific project requirement

3.3.1. Data Requirement

3.3.1.1. Encryption(RSA):

For encryption in RSA, the file named public_key.json is required to access the public keys.

3.3.1.2. Encryption(AES):

For encryption in AES, the file named password.json is required to access the list of passwords for encryption.

3.3.1.3. Decryption(RSA):

For decryption in RSA, the file named private_key.json is required to access the private keys.

3.3.1.4. Decryption(AES):

For decryption in AES, the files named passwords.json and private_AES.json are required to access the passwords, salts, nonces and tags that are required to decrypt the ciphertext.

3.3.2. Functions Requirements

3.3.2.1. cryptRSA.py:

This file contains all the functions that deal with encryption, decryption and key generation in RSA.

3.3.2.2. cryptAES.py:

This file contains all the functions that deal with encryption, decryption and key input in AES.

3.3.2.3. read_write.py:

It contains the functions required to read and write data from the files. (Like reading/writing messages, keys, passwords, etc)

3.3.2.4. main.py:

main.py is the driver code that will be executed whenever the program is run. It imports the above files (cryptRSA.py, cryptAES.py and read_write.py) and use their functions to do the necessary operations depending on the user's choice.

3.4. Summary:

Our program should work on most modern-day computers and hence should be easy to use. To install the required modules (if not already installed), one can execute the command:

```
pip install -r requirements.txt
```

Our program is only made for and tested on windows operating system yet.

CHAPTER-4:
DESIGN METHODOLOGY AND ITS NOVELTY

4.1 Methodology and goal

We will be working on methodology of providing multi-layer encryption but not with one algorithm with another but with multi-layer of the same encryption algorithm.

The goal and whole motive of the project will be to give a next-level secure encryption cryptosystem for users.

4.2 Functional modules design and analysis

4.2.1 Programming language: Python3

4.2.2 Inbuilt Modules:

- **base64 (b64encode(), b64decode())**

  This module provides functions for encoding binary data to printable ASCII characters and decoding such encodings back to binary data. It provides encoding and decoding functions for the encodings specified in RFC 4648, which defines the Base16, Base32, and Base64 algorithms, and for the de-facto standard Ascii85 and Base85 encodings.

- **hashlib (scrypt())**

  This module implements a common interface to many different secure hash and message digest algorithms. Included are the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512 (defined in FIPS 180-2) as well as RSA’s MD5 algorithm (defined in internet RFC 1321).

- **json**

  JSON, or JavaScript Object Notation, is a minimal, readable format for structuring data. It is used primarily to transmit data between a server and web application, as an alternative to XML. Squarespace uses JSON to store and organize site content created with the CMS

4.2.3 Other Modules:

- **Pycryptodome**

  PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It supports Python 2.6 and 2.7, Python 3.4 and newer, and PyPy.

You can install it with:

```
pip install pycryptodome
```

All modules are installed under the Crypto package. PyCryptodome is a fork of PyCrypto that has been enhanced to add more implementations and fixes to the original PyCrypto library.

- **Pycryptodomex**

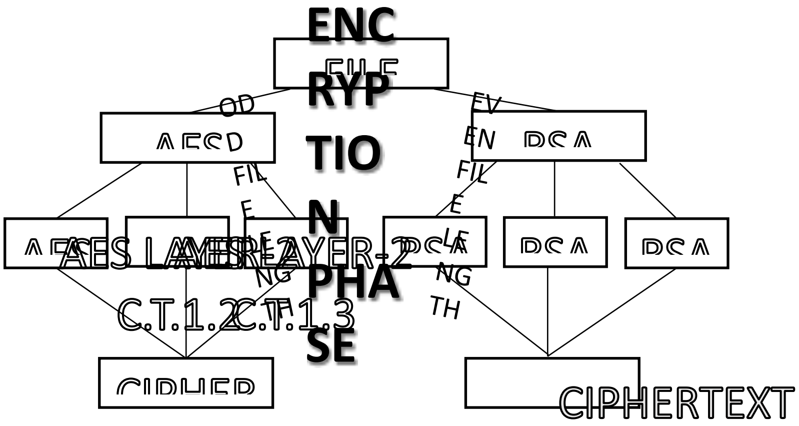
PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It supports Python 2.7, Python 3.5 and newer, and PyPy.

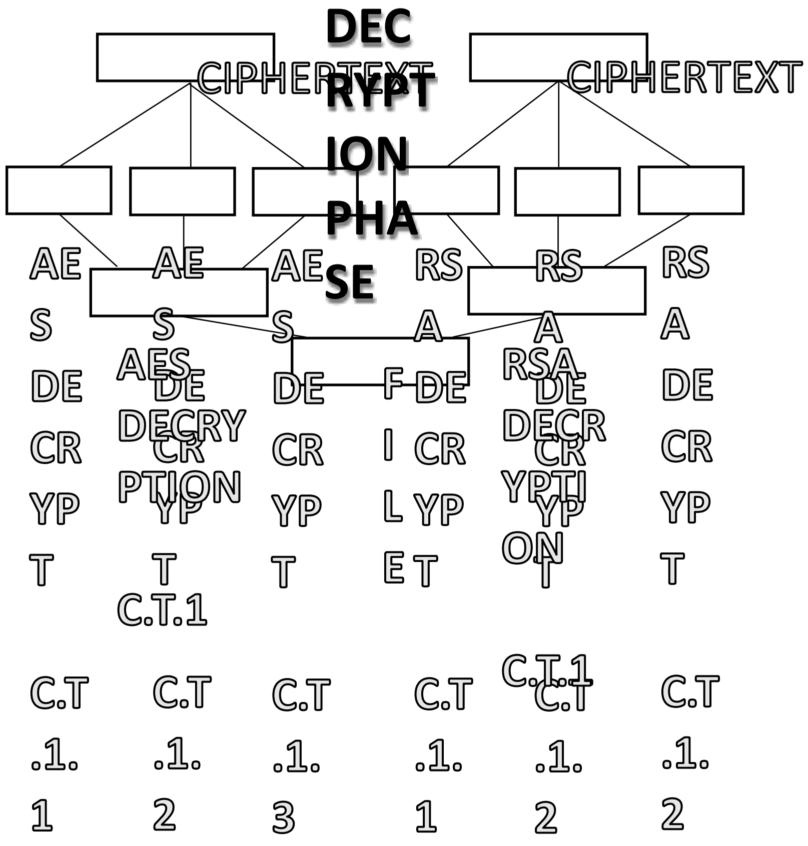
You can install it with:

```
pip install pycryptodomex.
```

All modules are installed under the Cryptodome package. PyCryptodome is a fork of PyCrypto.

4.3 Software Architectural designs





4.4 Subsystem services

4.4.1. cryptRSA.py:

This file contains all the functions that deal with encryption, decryption and key generation in RSA.

4.4.2. cryptAES.py:

This file contains all the functions that deal with encryption, decryption and key input in AES.

4.4.3 read_write.py:

It contains the functions required to read and write data from the files. (Like reading/ writing messages, keys, passwords, etc)

4.4.4. main.py:

main.py is the driver code that will be executed whenever the program is run. It imports the above files (cryptRSA.py, cryptAES.py and read_write.py) and use their functions to do the necessary operations depending on the user's choice**.**

4.5 User Interface designs

For now we will be working on the simple user choice program for this cryptosystem.

4.6 Summary

Utilizing all the modules and function we will create a multi-layer cryptosystem which will be giving maximum security to the user and their data.

5. CHAPTER-5:

TECHNICAL IMPLEMENTATION & ANALYSIS

5.1. Outline

The code for our program consists of 4 python files:

- **main.py**
- **read_write.py**
- **cryptRSA.py**
- **cryptAES.py**

1. **read_write.py** contains all the functions that are required to read or write data from various files like public or private keys (json), contents of a specified files, etc

2. **cryptRSA.py** contains 3 functions:

   - gen_key(keylen = 2048)
   - encrypt(message, pukey):
   - decrypt(message, prkey)

   The **gen_key()** function takes the key length as argument (Default = 2048) and generates four keypairs of the specified length and returns them as a list.

   The **encrypt()** function takes the plain text and the list of four exported public keys as the arguments and encrypts the plain text using the keys. First, the plain text is encrypted using K0, then it is split into 3 parts of nearly equal length. They are again encrypted separately using K1, K2, K3 and then the final ciphertext is returned.

   The **decrypt()** function takes the CipherTexts and list of private keys as arguments and returns the decrypted PlainText.

3. **cryptAES.py**

   - get_passwords()
   - encrypt(plain_text, passwords)

- decrypt(cipher_text, passwords, enc_dict)

The **get_passwords()** function prompts the user to enter 4 passwords that will be later used to generate the secret key and returns them in a list.

The **encrypt()** function generates lists of salts, private keys, ciphers and tags. The Cipher text is generated the same way as it was generated in cryptRSA.py and a list of 5 elements, the Cipher text and 4 encryption dictionaries (containing the salt, nonce and tag of each encryption pass).

The **decrypt()** function takes Cipher Text, list of passwords and list of encryption dictionaries as arguments to decrypt the ciphertext.

4. **main.py:**

main.py is the driver code that will be executed whenever the program is run. It imports the above files (cryptRSA.py, cryptAES.py and read_write.py) and use their functions to do the necessary operations depending on the user's choice.

0.) Introduction to the application

1.) View a file

2.) Get/Generate keys

3.) Encrypt file

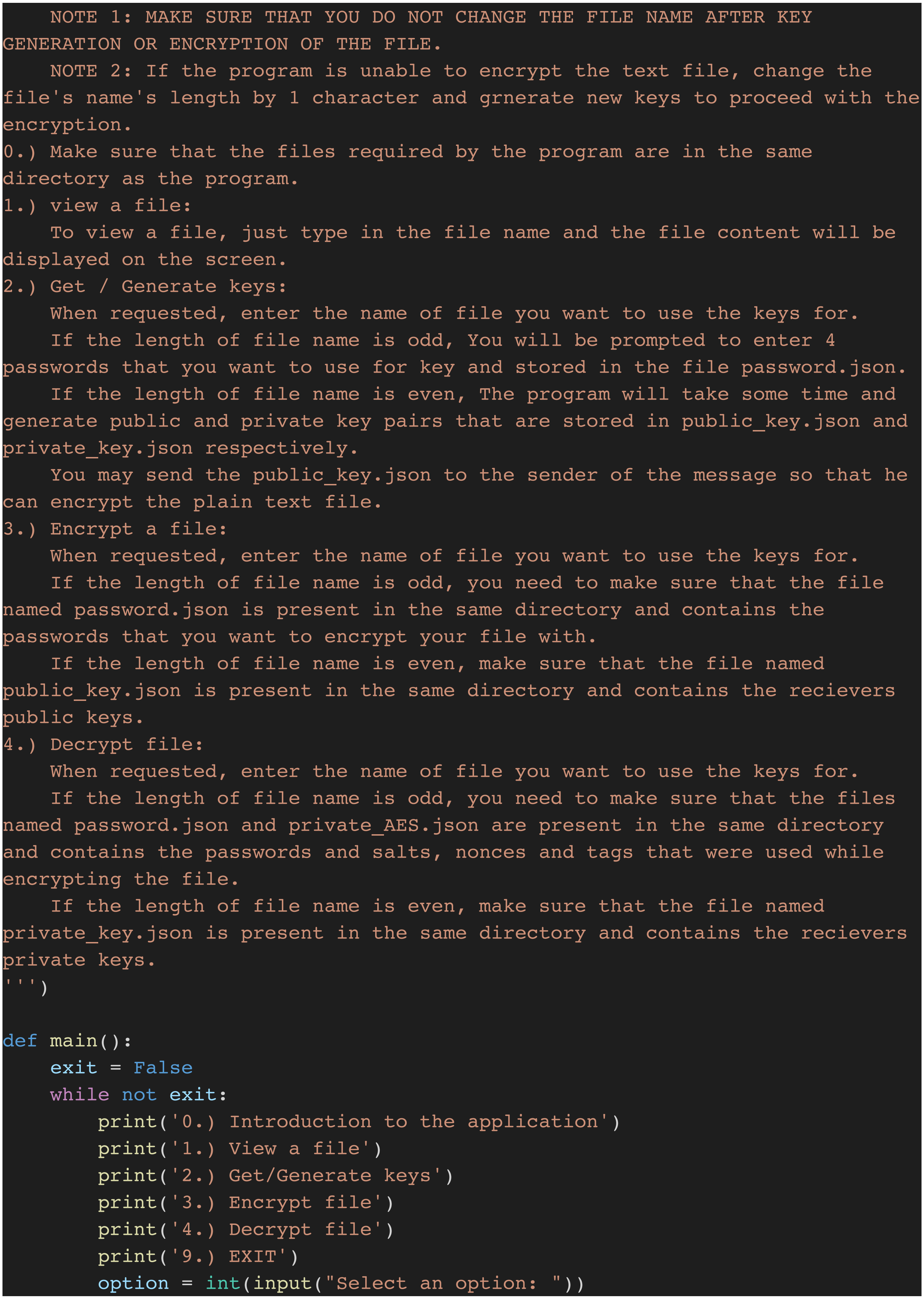
4.) Decrypt file

9.) EXIT

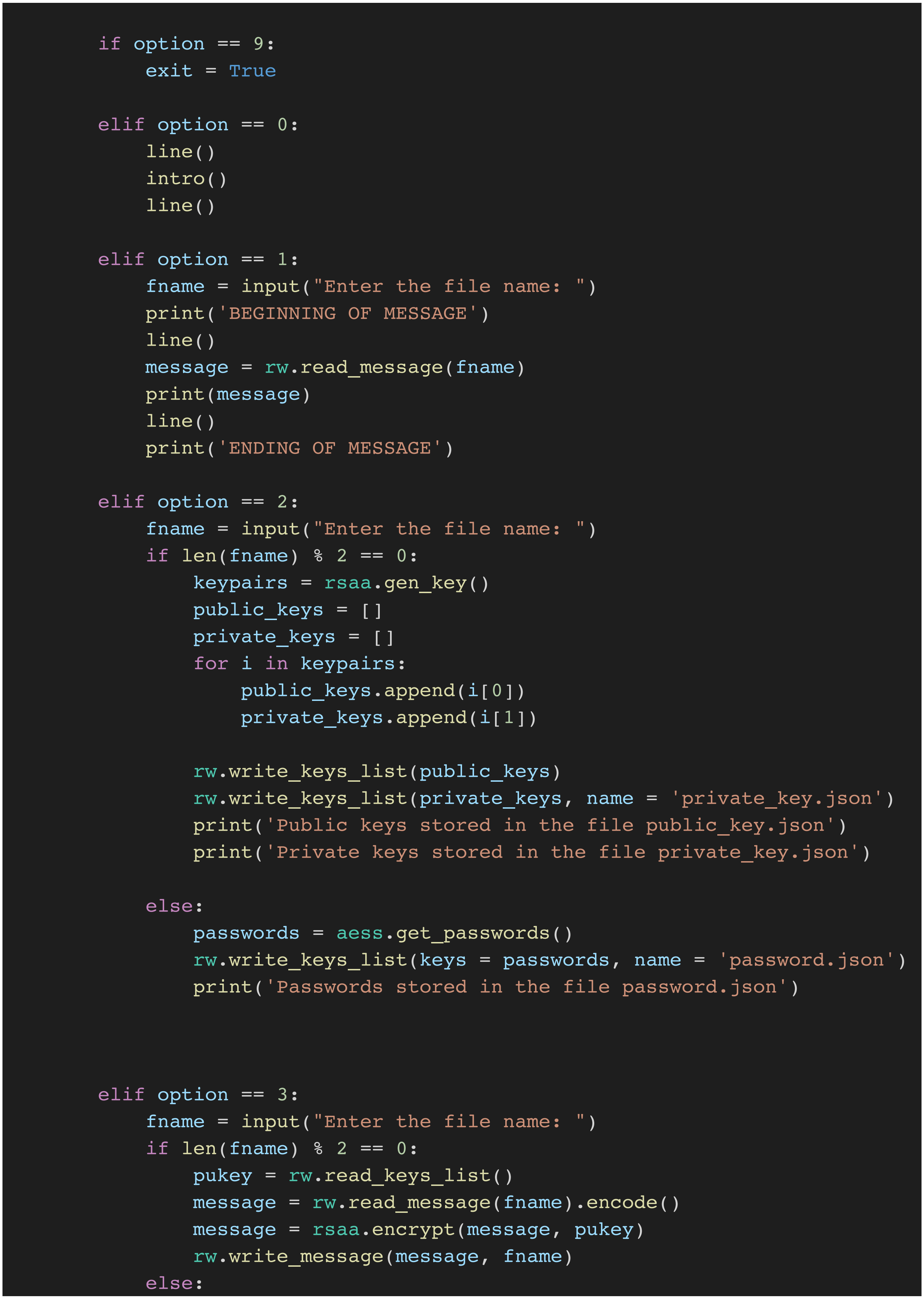
5.2. Technical coding and code solutions

main.py :

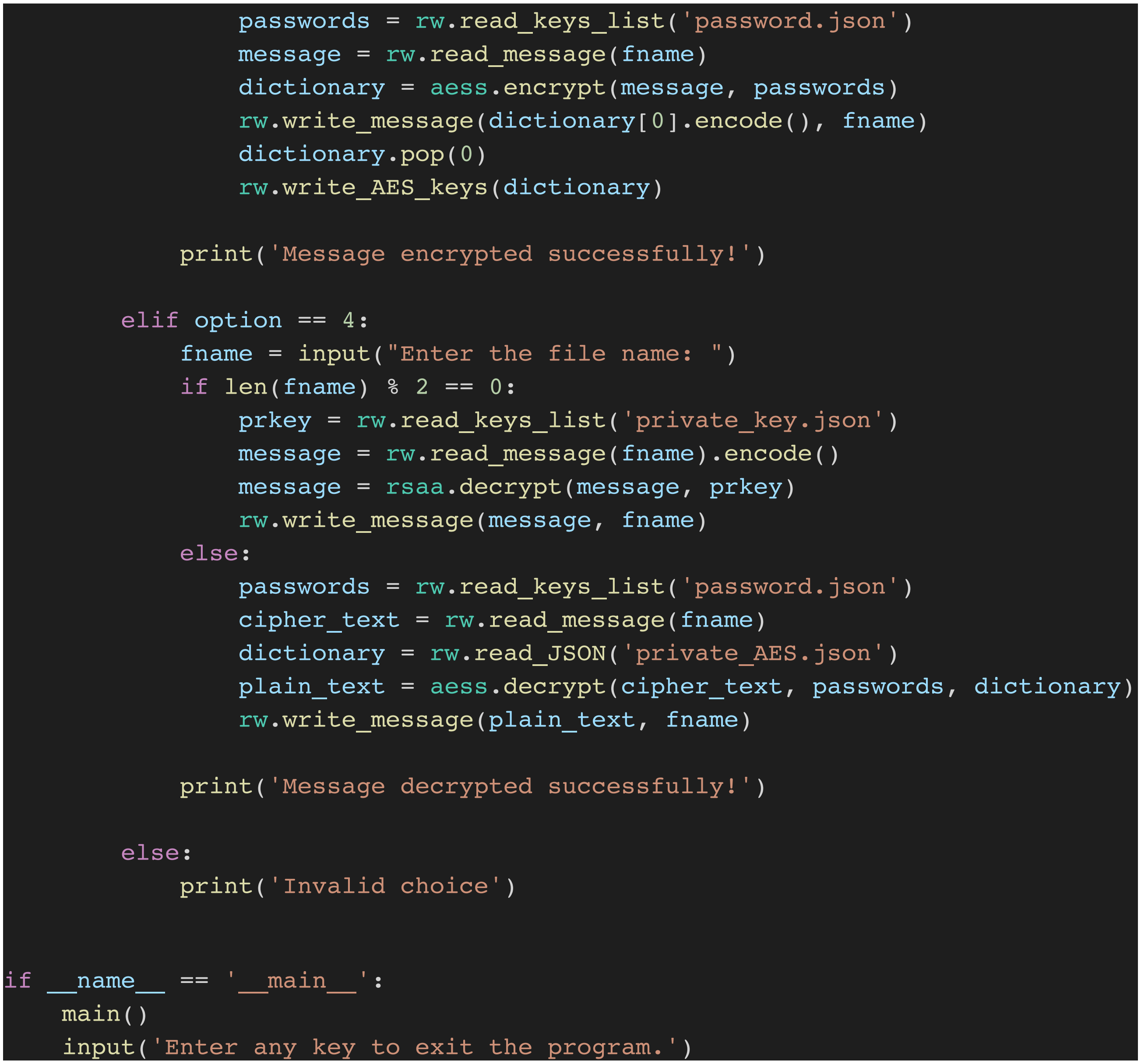
```
import read_write as rw
import cryptRSA as rsaa
import cryptAES as aess

def line(l = 80):
    print('_' * l)

def intro():
    print('''Instructions on how to use the program:
```



```
    NOTE 1: MAKE SURE THAT YOU DO NOT CHANGE THE FILE NAME AFTER KEY GENERATION OR ENCRYPTION OF THE FILE.
    NOTE 2: If the program is unable to encrypt the text file, change the file's name's length by 1 character and grnerate new keys to proceed with the encryption.
0.) Make sure that the files required by the program are in the same directory as the program.
1.) view a file:
    To view a file, just type in the file name and the file content will be displayed on the screen.
2.) Get / Generate keys:
    When requested, enter the name of file you want to use the keys for.
    If the length of file name is odd, You will be prompted to enter 4 passwords that you want to use for key and stored in the file password.json.
    If the length of file name is even, The program will take some time and generate public and private key pairs that are stored in public_key.json and private_key.json respectively.
    You may send the public_key.json to the sender of the message so that he can encrypt the plain text file.
3.) Encrypt a file:
    When requested, enter the name of file you want to use the keys for.
    If the length of file name is odd, you need to make sure that the file named password.json is present in the same directory and contains the passwords that you want to encrypt your file with.
    If the length of file name is even, make sure that the file named public_key.json is present in the same directory and contains the recievers public keys.
4.) Decrypt file:
    When requested, enter the name of file you want to use the keys for.
    If the length of file name is odd, you need to make sure that the files named password.json and private_AES.json are present in the same directory and contains the passwords and salts, nonces and tags that were used while encrypting the file.
    If the length of file name is even, make sure that the file named private_key.json is present in the same directory and contains the recievers private keys.
''')

def main():
    exit = False
    while not exit:
        print('0.) Introduction to the application')
        print('1.) View a file')
        print('2.) Get/Generate keys')
        print('3.) Encrypt file')
        print('4.) Decrypt file')
        print('9.) EXIT')
        option = int(input("Select an option: "))
```



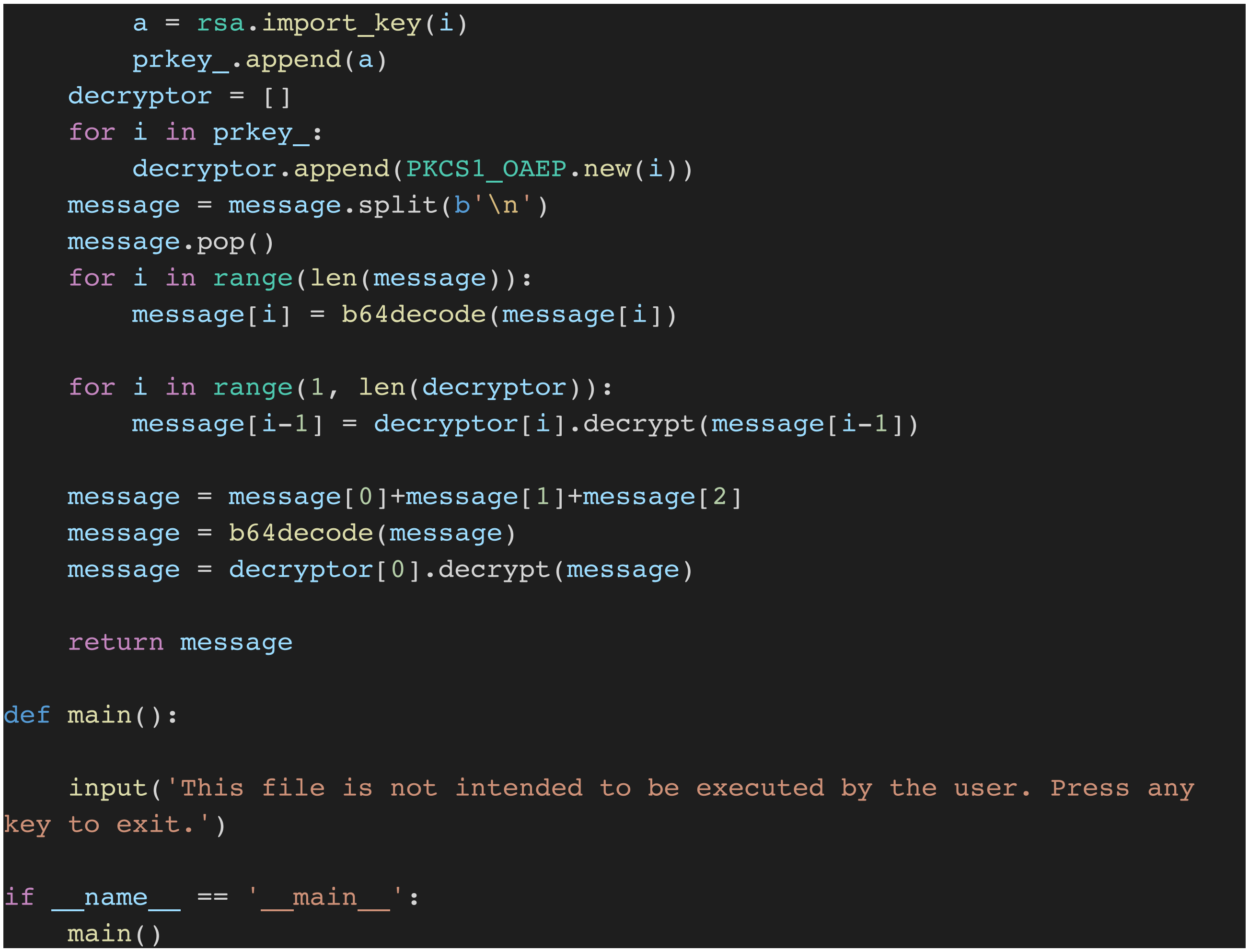
```
        if option == 9:
            exit = True

        elif option == 0:
            line()
            intro()
            line()

        elif option == 1:
            fname = input("Enter the file name: ")
            print('BEGINNING OF MESSAGE')
            line()
            message = rw.read_message(fname)
            print(message)
            line()
            print('ENDING OF MESSAGE')

        elif option == 2:
            fname = input("Enter the file name: ")
            if len(fname) % 2 == 0:
                keypairs = rsaa.gen_key()
                public_keys = []
                private_keys = []
                for i in keypairs:
                    public_keys.append(i[0])
                    private_keys.append(i[1])

                rw.write_keys_list(public_keys)
                rw.write_keys_list(private_keys, name = 'private_key.json')
                print('Public keys stored in the file public_key.json')
                print('Private keys stored in the file private_key.json')

            else:
                passwords = aess.get_passwords()
                rw.write_keys_list(keys = passwords, name = 'password.json')
                print('Passwords stored in the file password.json')



        elif option == 3:
            fname = input("Enter the file name: ")
            if len(fname) % 2 == 0:
                pukey = rw.read_keys_list()
                message = rw.read_message(fname).encode()
                message = rsaa.encrypt(message, pukey)
                rw.write_message(message, fname)
            else:
```



```python
                passwords = rw.read_keys_list('password.json')
                message = rw.read_message(fname)
                dictionary = aess.encrypt(message, passwords)
                rw.write_message(dictionary[0].encode(), fname)
                dictionary.pop(0)
                rw.write_AES_keys(dictionary)

            print('Message encrypted successfully!')

        elif option == 4:
            fname = input("Enter the file name: ")
            if len(fname) % 2 == 0:
                prkey = rw.read_keys_list('private_key.json')
                message = rw.read_message(fname).encode()
                message = rsaa.decrypt(message, prkey)
                rw.write_message(message, fname)
            else:
                passwords = rw.read_keys_list('password.json')
                cipher_text = rw.read_message(fname)
                dictionary = rw.read_JSON('private_AES.json')
                plain_text = aess.decrypt(cipher_text, passwords, dictionary)
                rw.write_message(plain_text, fname)

            print('Message decrypted successfully!')

        else:
            print('Invalid choice')


if __name__ == '__main__':
    main()
    input('Enter any key to exit the program.')
```

cryptRSA.py:

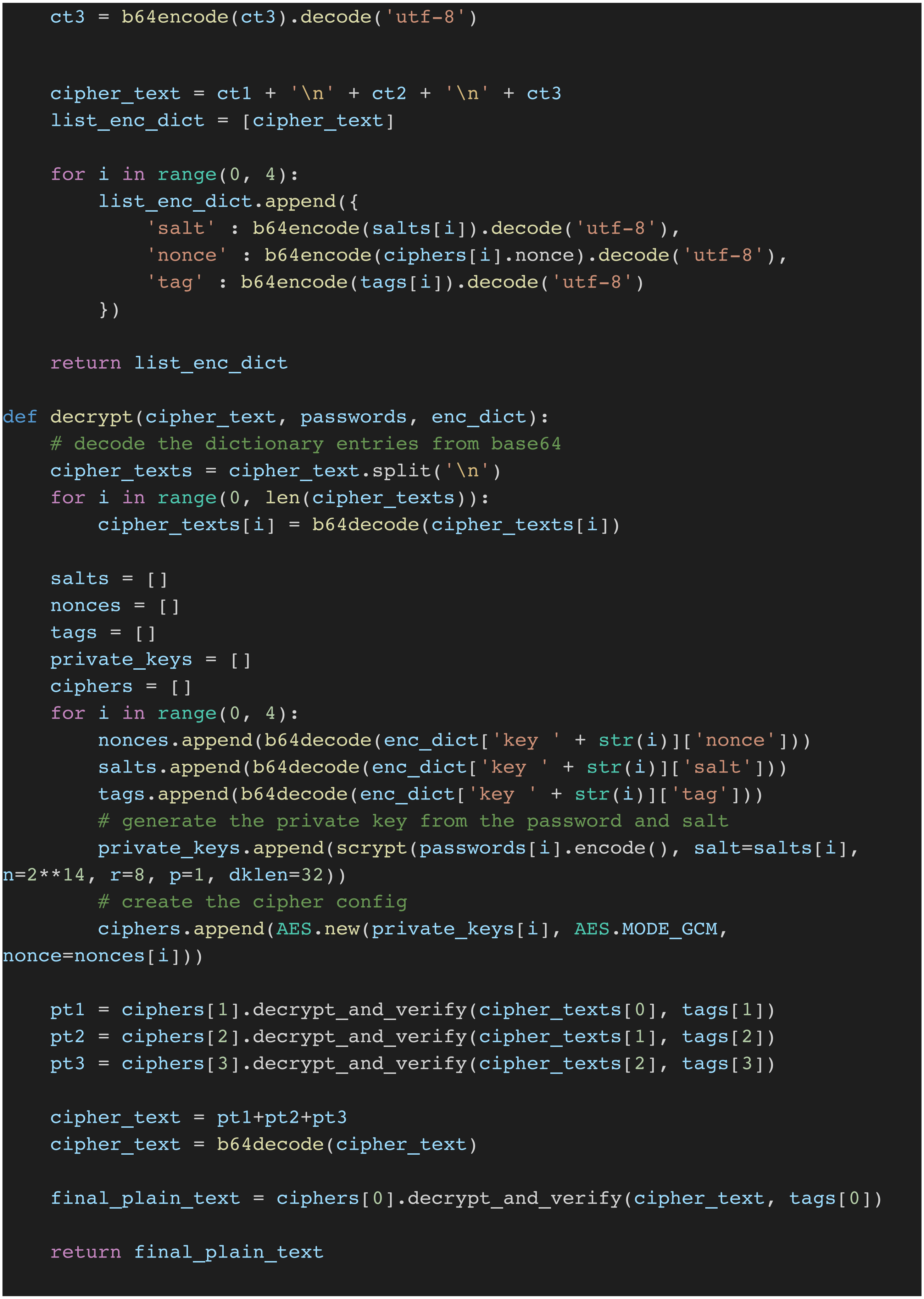
```
from Crypto.PublicKey import RSA as rsa
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64encode, b64decode

def gen_key(keylen = 2048):
    key_pair = []
    for i in range(0, 4):
        n = rsa.generate(keylen)
        key_pair.append([n.publickey().exportKey(), n.exportKey()])
 #generates 4 public key / private key pairs and returns them
    return key_pair

def encrypt(message, pukey):
    pukey_ = []
    for i in pukey:
        a = rsa.import_key(i)
        pukey_.append(a)
    encryptor = []
    for i in pukey_:
        encryptor.append(PKCS1_OAEP.new(i))

    encmessage = []

    encmessage.append(encryptor[0].encrypt(message))
    encmessage[0] = b64encode(encmessage[0])
    ct = encmessage[0].decode()
    length_ct = len(ct)
    encmessage[0] = ct[:int(length_ct/3)]                           # first 1/3
of CT
    encmessage.append(ct[int(length_ct/3):int(length_ct*2/3)])     # second
1/3 of CT
    encmessage.append(ct[int(length_ct*2/3):])                      # third 1/3
of CT

    encmessage[0] = encryptor[1].encrypt(encmessage[0].encode())
    encmessage[1] = encryptor[2].encrypt(encmessage[1].encode())
    encmessage[2] = encryptor[3].encrypt(encmessage[2].encode())
    final_ct = b''
    for i in range(3):
        encmessage[i] = b64encode(encmessage[i])
        final_ct = final_ct+encmessage[i]+b'\n'
    return final_ct

def decrypt(message, prkey):
    prkey_ = []

    for i in prkey:
```



```
        a = rsa.import_key(i)
        prkey_.append(a)
    decryptor = []
    for i in prkey_:
        decryptor.append(PKCS1_OAEP.new(i))
    message = message.split(b'\n')
    message.pop()
    for i in range(len(message)):
        message[i] = b64decode(message[i])

    for i in range(1, len(decryptor)):
        message[i-1] = decryptor[i].decrypt(message[i-1])

    message = message[0]+message[1]+message[2]
    message = b64decode(message)
    message = decryptor[0].decrypt(message)

    return message

def main():

    input('This file is not intended to be executed by the user. Press any
key to exit.')

if __name__ == '__main__':
    main()
```

cryptAES.py

```
from base64 import b64encode, b64decode
from hashlib import scrypt
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes

def get_passwords():
    passwords = []
    for i in range(0, 4):
        x = input('Enter password (' + str(i+1) + '/4)').encode()
        passwords.append(x)
    return passwords


def encrypt(plain_text, passwords):
    # generate a random salt
    salts = []
    private_keys = []
    ciphers = []
    tags = []
    for i in range(0, 4):
        salts.append(get_random_bytes(AES.block_size))
        # use the Scrypt KDF to get a private key from the password
        private_keys.append(scrypt(passwords[i].encode(), salt=salts[i],
n=2**14, r=8, p=1, dklen=32))
        # create cipher config
        ciphers.append(AES.new(private_keys[i], AES.MODE_GCM))

    cipher_text, tag = ciphers[0].encrypt_and_digest(bytes(plain_text,
'utf-8'))
    cipher_text = b64encode(cipher_text).decode('utf-8')
    tags.append(tag)
    length = len(cipher_text)
    ct1 = cipher_text[:int(1/3 * length)]
    ct2 = cipher_text[int(1/3 * length):int(2/3 * length)]
    ct3 = cipher_text[int(2/3 * length):]

    ct1, tag = ciphers[1].encrypt_and_digest(bytes(ct1, 'utf-8'))
    tags.append(tag)
    ct1 = b64encode(ct1).decode('utf-8')

    ct2, tag = ciphers[2].encrypt_and_digest(bytes(ct2, 'utf-8'))
    tags.append(tag)
    ct2 = b64encode(ct2).decode('utf-8')

    ct3, tag = ciphers[3].encrypt_and_digest(bytes(ct3, 'utf-8'))
    tags.append(tag)
```

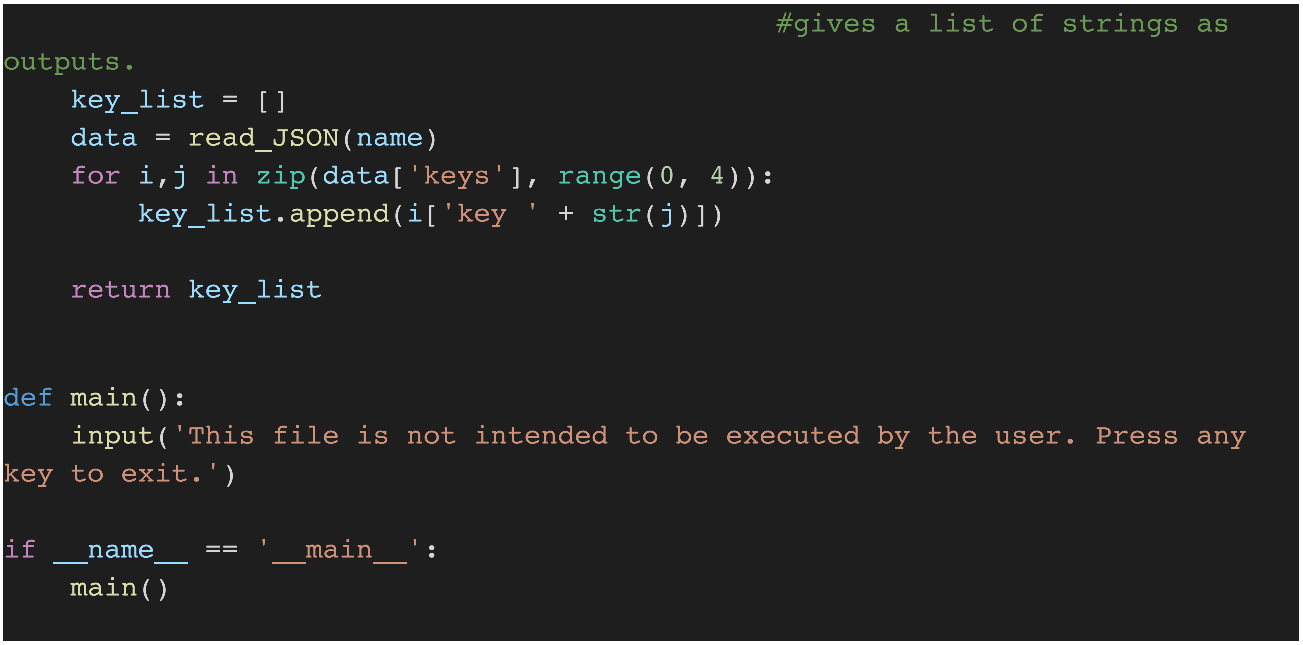


```
    ct3 = b64encode(ct3).decode('utf-8')


    cipher_text = ct1 + '\n' + ct2 + '\n' + ct3
    list_enc_dict = [cipher_text]

    for i in range(0, 4):
        list_enc_dict.append({
            'salt' : b64encode(salts[i]).decode('utf-8'),
            'nonce' : b64encode(ciphers[i].nonce).decode('utf-8'),
            'tag' : b64encode(tags[i]).decode('utf-8')
        })

    return list_enc_dict

def decrypt(cipher_text, passwords, enc_dict):
    # decode the dictionary entries from base64
    cipher_texts = cipher_text.split('\n')
    for i in range(0, len(cipher_texts)):
        cipher_texts[i] = b64decode(cipher_texts[i])

    salts = []
    nonces = []
    tags = []
    private_keys = []
    ciphers = []
    for i in range(0, 4):
        nonces.append(b64decode(enc_dict['key ' + str(i)]['nonce']))
        salts.append(b64decode(enc_dict['key ' + str(i)]['salt']))
        tags.append(b64decode(enc_dict['key ' + str(i)]['tag']))
        # generate the private key from the password and salt
        private_keys.append(scrypt(passwords[i].encode(), salt=salts[i],
n=2**14, r=8, p=1, dklen=32))
        # create the cipher config
        ciphers.append(AES.new(private_keys[i], AES.MODE_GCM,
nonce=nonces[i]))

    pt1 = ciphers[1].decrypt_and_verify(cipher_texts[0], tags[1])
    pt2 = ciphers[2].decrypt_and_verify(cipher_texts[1], tags[2])
    pt3 = ciphers[3].decrypt_and_verify(cipher_texts[2], tags[3])

    cipher_text = pt1+pt2+pt3
    cipher_text = b64decode(cipher_text)

    final_plain_text = ciphers[0].decrypt_and_verify(cipher_text, tags[0])

    return final_plain_text
```

```python
def main():
    input('This file is not intended to be executed by the user. Press any
key to exit.')

if __name__ == '__main__':
    main()
```

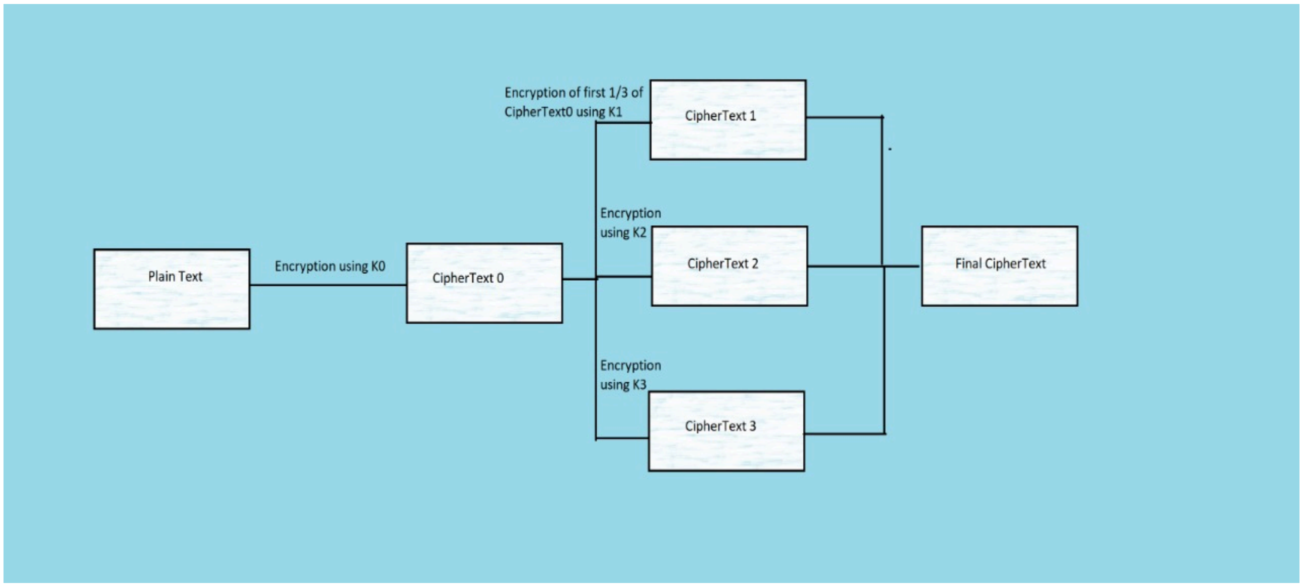
read_write.py:

```
import json

def write_JSON(data, name):
    with open(name, 'w') as outfile:
        json.dump(data, outfile, indent=4, ensure_ascii=False)

def read_JSON(name):
    with open(name, 'r') as readfile:
        data = json.load(readfile)
    return data

def read_keys_AES():
    print("read AES keys!")

def write_AES_keys(list_dicts):
    final_dict = {
        'key 0' : list_dicts[0],
        'key 1' : list_dicts[1],
        'key 2' : list_dicts[2],
        'key 3' : list_dicts[3]
    }
    write_JSON(final_dict, 'private_AES.json')

def write_keys_list(keys, name = 'public_key.json', n = 3):  # public or
private key as a bool(0 = Public, 1 = Private), keys as a list
    data = {}                                 # Takes a list of bytes as
input.
    data['keys'] = []
    for i in range(0, n+1, 1):
        data['keys'].append({
            'key '+ str(i): keys[i].decode()
        })

    write_JSON(data, name)

def read_message(filename):  # name of file as a string
    file1 = open(filename, 'r')
    message = file1.read()
    file1.close()
    return message

def write_message(message, fname):
    with open(fname, 'w') as file:
        file.write(message.decode())

def read_keys_list(name = 'public_key.json'): # reads keys from json file
(0 = Public key file, 1 = Private key file)
```



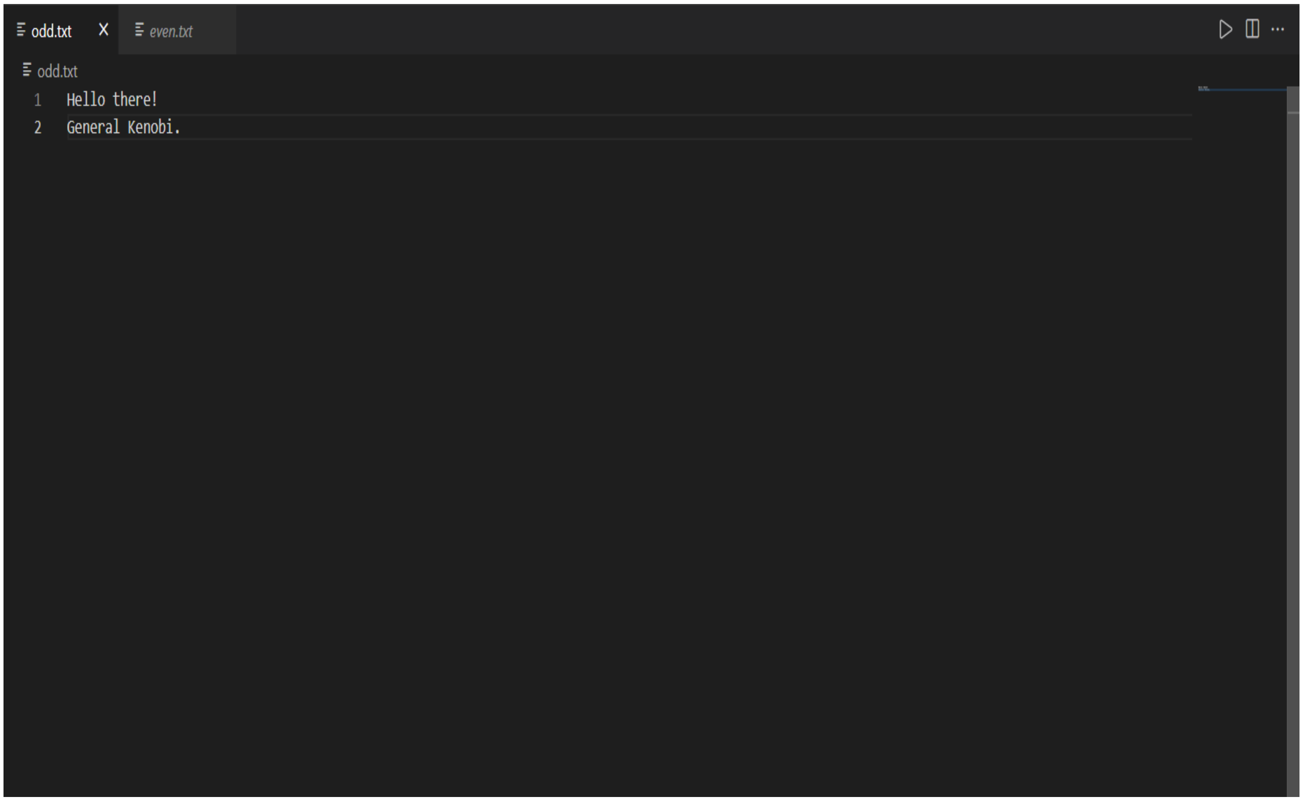
```
                                              #gives a list of strings as
outputs.
    key_list = []
    data = read_JSON(name)
    for i,j in zip(data['keys'], range(0, 4)):
        key_list.append(i['key ' + str(j)])

    return key_list


def main():
    input('This file is not intended to be executed by the user. Press any
key to exit.')

if __name__ == '__main__':
    main()
```

5.3 Working Layout of Forms:



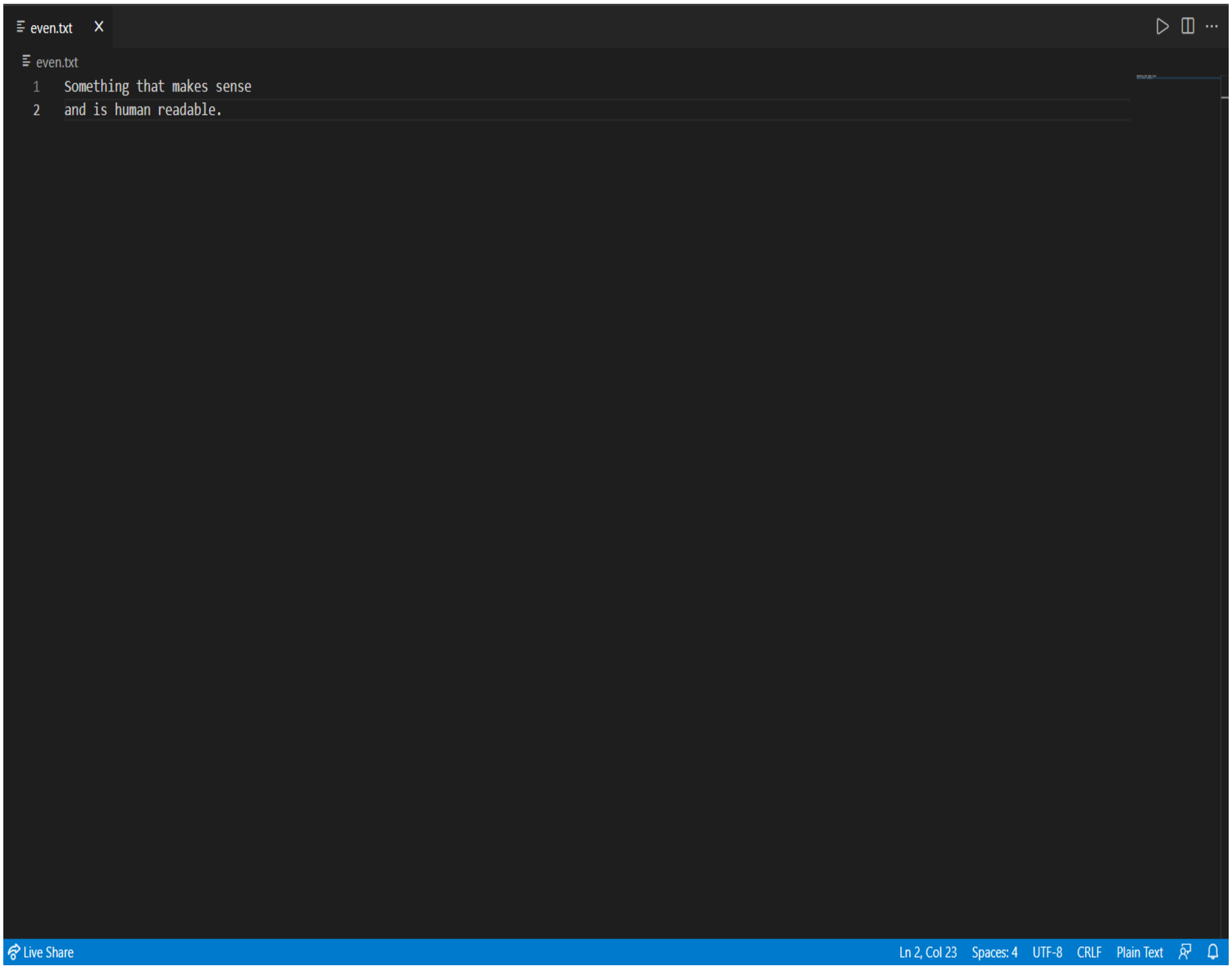
5.4 Test and validation

1. Initial content of plaintext files:



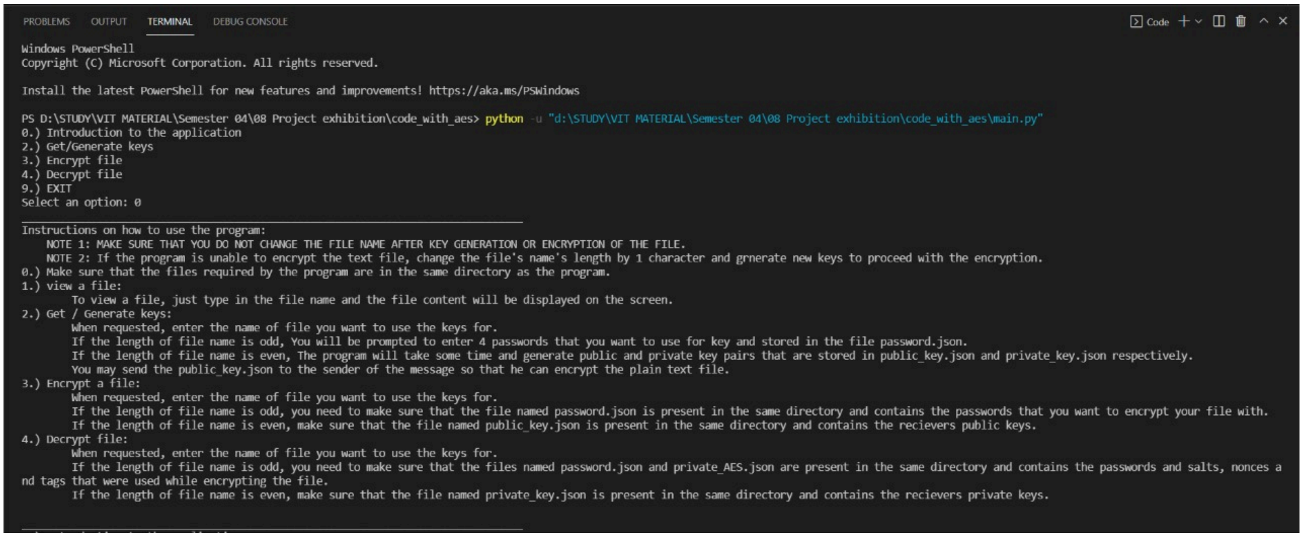
```
Hello there!
General Kenobi.
```

Odd.txt



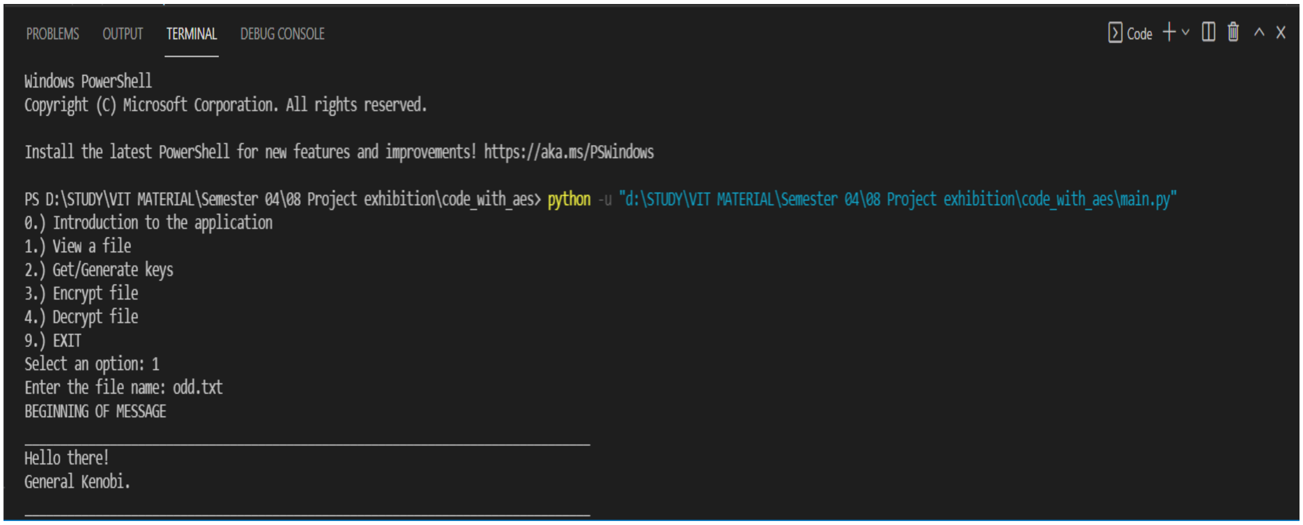
```
Something that makes sense
and is human readable.
```

Even.txt



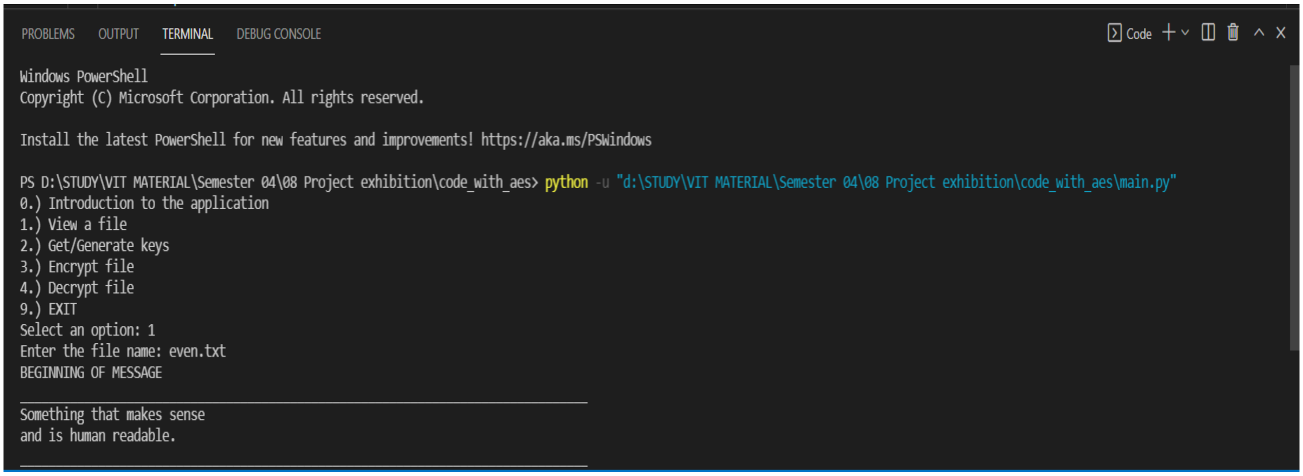
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 0
________________________________________________________________________________
Instructions on how to use the program:
    NOTE 1: MAKE SURE THAT YOU DO NOT CHANGE THE FILE NAME AFTER KEY GENERATION OR ENCRYPTION OF THE FILE.
    NOTE 2: If the program is unable to encrypt the text file, change the file's name's length by 1 character and grnerate new keys to proceed with the encryption.
0.) Make sure that the files required by the program are in the same directory as the program.
1.) view a file:
        To view a file, just type in the file name and the file content will be displayed on the screen.
2.) Get / Generate keys:
        When requested, enter the name of file you want to use the keys for.
        If the length of file name is odd, You will be prompted to enter 4 passwords that you want to use for key and stored in the file password.json.
        If the length of file name is even, The program will take some time and generate public and private key pairs that are stored in public_key.json and private_key.json respectively.
        You may send the public_key.json to the sender of the message so that he can encrypt the plain text file.
3.) Encrypt a file:
        When requested, enter the name of file you want to use the keys for.
        If the length of file name is odd, you need to make sure that the file named password.json is present in the same directory and contains the passwords that you want to encrypt your file with.
        If the length of file name is even, make sure that the file named public_key.json is present in the same directory and contains the recievers public keys.
4.) Decrypt file:
        When requested, enter the name of file you want to use the keys for.
        If the length of file name is odd, you need to make sure that the files named password.json and private_AES.json are present in the same directory and contains the passwords and salts, nonces a
nd tags that were used while encrypting the file.
        If the length of file name is even, make sure that the file named private_key.json is present in the same directory and contains the recievers private keys.


________________________________________________________________________________
```

2. Viewing a file:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 1
Enter the file name: odd.txt
BEGINNING OF MESSAGE
________________________________________________________________

Hello there!
General Kenobi.
________________________________________________________________
```

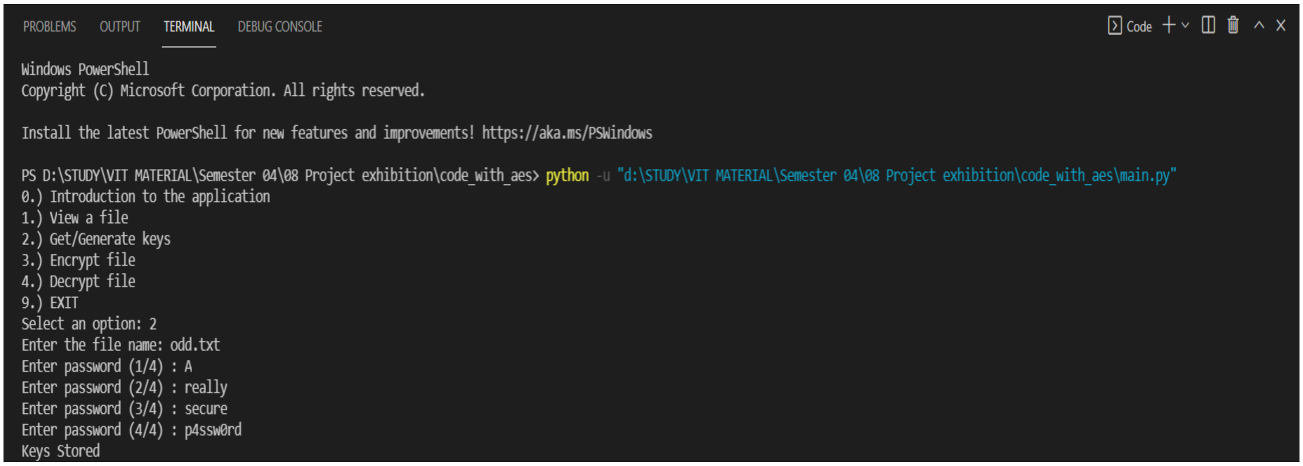
Odd.txt



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 1
Enter the file name: even.txt
BEGINNING OF MESSAGE
________________________________________________________________

Something that makes sense
and is human readable.
________________________________________________________________
```

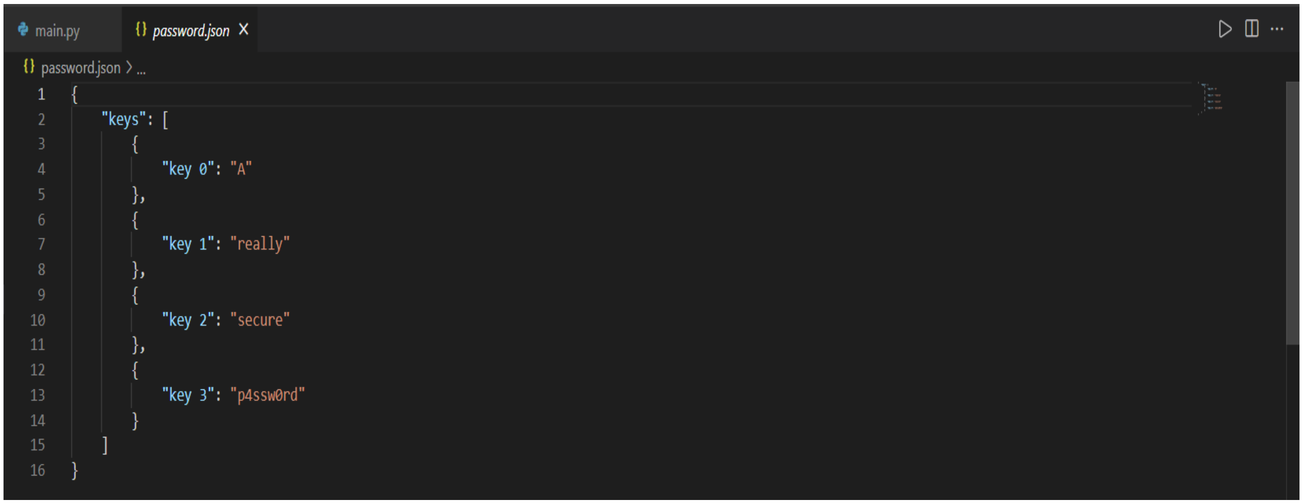
Even.txt

3. Generating keys for odd.txt:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 2
Enter the file name: odd.txt
Enter password (1/4) : A
Enter password (2/4) : really
Enter password (3/4) : secure
Enter password (4/4) : p4ssw0rd
Keys Stored
```

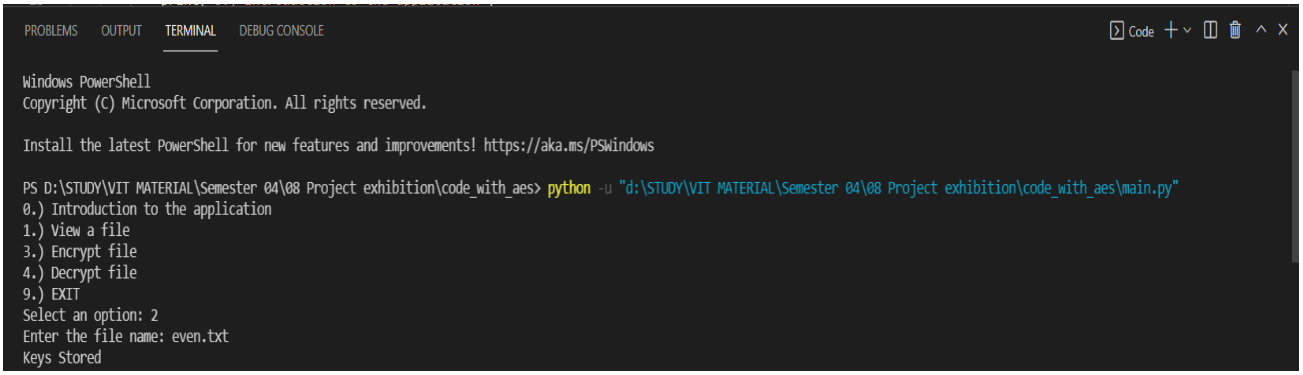
Asking user to enter password that are to be used to encrypt odd.txt



```
{
    "keys": [
        {
            "key 0": "A"
        },
        {
            "key 1": "really"
        },
        {
            "key 2": "secure"
        },
        {
            "key 3": "p4ssw0rd"
        }
    ]
}
```

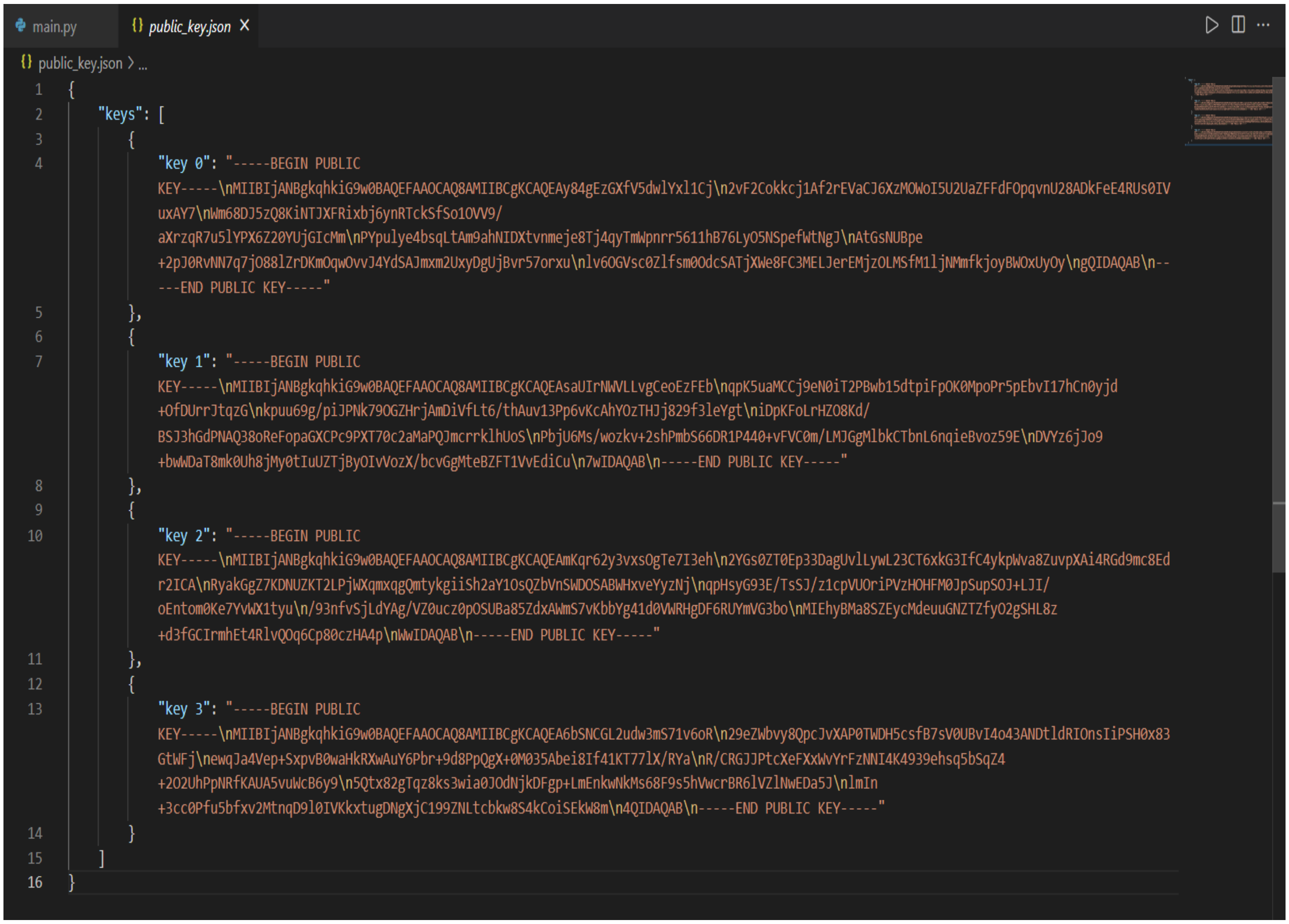
Content of password.json after storing passwords for odd.txt

4. Generating keys for even.txt:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 2
Enter the file name: even.txt
Keys Stored
```

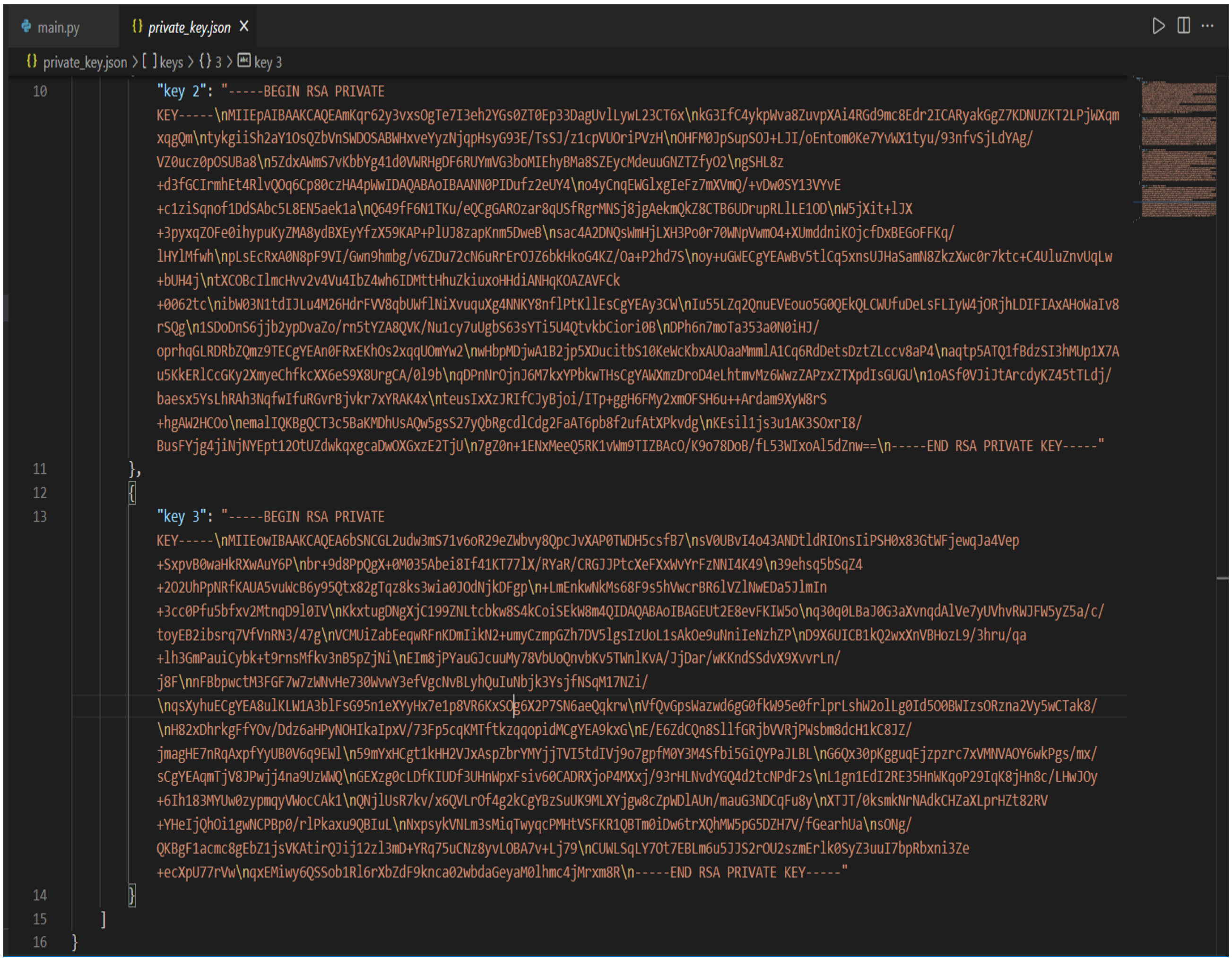
Generating random RSA key pairs for encryption/decryption



```
{
    "keys": [
        {
            "key 0": "-----BEGIN PUBLIC
            KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy84gEzGXfV5dwlYxl1Cj\n2vF2Cokkcj1Af2rEVaCJ6XzMOWoI5U2UaZFFdFOpqvnU28ADkFeE4RUs0IV
            uxAY7\nWm68DJ5zQ8KiNTJXFRixbj6ynRTckSfSo1OVV9/
            aXrzqR7u5lYPX6Z20YUjGIcMm\nPYpulye4bsqLtAm9ahNIDXtvnmeje8Tj4qyTmWpnrr5611hB76LyO5NSpefWtNgJ\nAtGsNUBpe
            +2pJ0RvNN7q7jO88lZrDKmOqwOvvJ4YdSAJmxm2UxyDgUjBvr57orxu\nlv6OGVsc0Zlfsm0OdcSATjXWe8FC3MELJerEMjzOLMSfM1ljNMmfkjoyBWOxUyOy\ngQIDAQAB\n--
            ---END PUBLIC KEY-----"
        },
        {
            "key 1": "-----BEGIN PUBLIC
            KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsaUIrNWVLLvgCeoEzFEb\nqpK5uaMCCj9eN0iT2PBwb15dtpiFpOK0MpoPr5pEbvI17hCn0yjd
            +OfDUrrJtqzG\nkpuu69g/piJPNk79OGZHrjAmDiVfLt6/thAuv13Pp6vKcAhYOzTHJj829f3leYgt\niDpKFoLrHZO8Kd/
            BSJ3hGdPNAQ38oReFopaGXCPc9PXT70c2aMaPQJmcrrklhUoS\nPbjU6Ms/wozkv+2shPmbS66DR1P440+vFVC0m/LMJGgMlbkCTbnL6nqieBvoz59E\nDVYz6jJo9
            +bwWDaT8mk0Uh8jMy0tIuUZTjByOIvVozX/bcvGgMteBZFT1VvEdiCu\n7wIDAQAB\n-----END PUBLIC KEY-----"
        },
        {
            "key 2": "-----BEGIN PUBLIC
            KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmKqr62y3vxsOgTe7I3eh\n2YGs0ZT0Ep33DagUvlLywL23CT6xkG3IfC4ykpWva8ZuvpXAi4RGd9mc8Ed
            r2ICA\nRyakGgZ7KDNUZKT2LPjWXqmxqgQmtykgiiSh2aY1OsQZbVnSWDOSABWHxveYyzNj\nqpHsyG93E/TsSJ/z1cpVUOriPVzHOHFM0JpSupSOJ+LJI/
            oEntom0Ke7YvWX1tyu\n/93nfvSjLdYAg/VZ0ucz0pOSUBa85ZdxAWmS7vKbbYg41d0VWRHgDF6RUYmVG3bo\nMIEhyBMa8SZEycMdeuuGNZTZfyO2gSHL8z
            +d3fGCIrmhEt4RlvQOq6Cp80czHA4p\nWwIDAQAB\n-----END PUBLIC KEY-----"
        },
        {
            "key 3": "-----BEGIN PUBLIC
            KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6bSNCGL2udw3mS71v6oR\n29eZWbvy8QpcJvXAP0TWDH5csfB7sV0UBvI4o43ANDtldRIOnsIiPSH0x83
            GtWFj\newqJa4Vep+SxpvB0waHkRXwAuY6Pbr+9d8PpQgX+0M035Abei8If41KT77lX/RYa\nR/CRGJJPtcXeFXxWvYrFzNNI4K4939ehsq5bSqZ4
            +2O2UhPpNRfKAUA5vuWcB6y9\n5Qtx82gTqz8ks3wia0JOdNjkDFgp+LmEnkwNkMs68F9s5hVwcrBR6lVZlNwEDa5J\nlmIn
            +3cc0Pfu5bfxv2MtnqD9l0IVKkxtugDNgXjC199ZNLtcbkw8S4kCoiSEkW8m\n4QIDAQAB\n-----END PUBLIC KEY-----"
        }
    ]
}
```

Content of public_key.json after generating keys

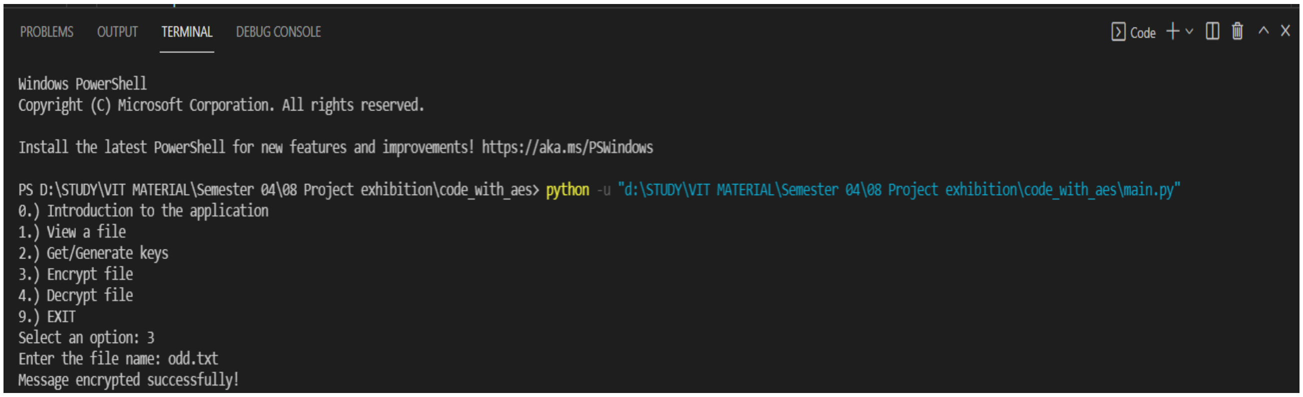
```
{
    "keys": [
        {
            "key 0": "-----BEGIN RSA PRIVATE
            KEY-----\nMIIEowIBAAKCAQEAy84gEzGXfV5dwlYxl1Cj2vF2Cokkcj1Af2rEVaCJ6XzMOWoI\n5U2UaZFFdFOpqvnU28ADkFeE4RUs0IVuxAY7Wm68DJ5zQ8KiNTJXFRixbj6
            ynRTc\nkSfSo1OVV9/aXrzqR7u5lYPX6Z20YUjGIcMmPYpulye4bsqLtAm9ahNIDXtvnmej\ne8Tj4qyTmWpnrr5611hB76LyO5NSpefWtNgJAtGsNUBpe
            +2pJ0RvNN7q7jO88lZr\nDKmOqwOvvJ4YdSAJmxm2UxyDgUjBvr57orxulv6OGVsc0Zlfsm0OdcSATjXWe8FC\n3MELJerEMjzOLMSfM1ljNMmfkjoyBWOxUyOygQIDAQABAoIB
            AGAAD7bK0N+CkGqm\nvERY/V34CFDNKehP9u4uBh6JeP0NRouy6U7N3N9fSRaTBCRf+0ap62PV/eciWNI+\nUB8xCks+RwEwr/3WlB63wDJVR9q02ODKP112oy9BA2N5s/
            yVQ0wX3sLRcGaAXUPI\nCmBD8uLhLtevo544dVcQkXMCcxm516S0DGfkDHi6pjL6ONSHlGBuNEClciI2tfoJ\nY7JBI4R7sZEO7ewbI4jwXNg1KjpByKCjmNPGtEFnFF/
            4qy3ykAdFpDATe0jHtMTq\nFYFW3LZHk6Q9OOv7xw/sLpUPiKVcF65PXZqX+DSp4XU5QT
            +IkT6COcmY1PTBNtU0\npl2dQGMCgYEA0TmRZFSGaSSvnQOU1rPZpKE5vUT867AJ0ZBLbYz/UtHpZK6Klof9\nGd2vvEmMIVEqCfnJcq4Bi5fe7DtOZYTlIa
            +WOYJezvbWgODd06qvLIJo/qoqKu8q\nTEy/BVrccL/ssutZwfC8lqX2LG9TmaiIryGDmkb13Q2KqmLOpUeXaWsCgYEA+V5g\nEyC446WPm58dS7JRHptNT/0I8Vsa2EN6owROK
            +WO3OWBFBRRa+fsMnGi2ZtHnQh+\n5MNALtOCycpi5CV3nwY5GBHUPvgCdpCsdmZiGbKpWwrkXEHrKc57JKfV/16L+DGs\nFLLEocXTLjTwJGji5Yy7ttxVLfU4beMLs
            +EGssMCgYAAr7apZJmYSLWGx1c6qRZi\nsuC0nUBTKZ2uoVGOB3Ni+ytP2D6+Ja4P4qlL7dXW2iRQKhCeHrIEmjfyAHPOY3Q4\n+xF8GNBkz2x
            +ub4gjqPKepcLltVSlLxHycyqoNSuTgLhx2HhAAAuYlb/aFIW4/
            Wm\n1W5QGEGAOyYkTvSsmCKIBwKBgBKXXKZQbDhcoYZCqJZ6Jt4XSf5KqR4FYsz4yu3Y\nZlOTe8EwQx1SHnfjBJMGMdPzZbVparx3ahOTpU6SDilgObjbZ
            +pJef9v0g8w3gq9\nu1abIVEuULQoHfYx9LBY6vVtARZx7Zhlc490BjiNrdGpGDFw+pW5e4g/ngNjR/4T\nKsKtAoGBAKQTBtuHkLQs2aTqLQFw
            +rSHCybC5dJ1G6FesNU3wPd70MhmTnMMTOwf\nD3rg27M2rQR5WSh0+LmLjkYF4vCaEuxCmfFqGmiaPtwomQZhUPj5FkL1s2b/8lDh\nBOc0EhnFtMxbbGaW
            +iG39jBrw6WXVa8YmzCmQM633Arnvi09k+c0\n-----END RSA PRIVATE KEY-----"
        },
        {
            "key 1": "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEAsaUIrNWVLLvgCeoEzFEbqpK5uaMCCj9eN0iT2PBwb15dtpiF\npOK0MpoPr5pEbvI17hCn0yjd
            +OfDUrrJtqzGkpuu69g/piJPNk79OGZHrjAmDiVf\nLt6/thAuv13Pp6vKcAhYOzTHJj829f3leYgtiDpKFoLrHZO8Kd/
            BSJ3hGdPNAQ38\noReFopaGXCPc9PXT70c2aMaPQJmcrrklhUoSPbjU6Ms/wozkv+2shPmbS66DR1P4\n40+vFVC0m/LMJGgMlbkCTbnL6nqieBvoz59EDVYz6jJo9
            +bwWDaT8mk0Uh8jMy0t\nIuUZTjByOIvVozX/bcvGgMteBZFT1VvEdiCu7wIDAQABAoIBAD+O5IOyByVyMr5Z\nVE4l1tg4luqsP9aaxYLSHNPJcvDlunHSMD7GPeK87E/
            PyhxH3LPIkcMjJAgiI+g9\nnF4XgxSBTxxiyqMOM3ki2caMEalo4LS++kcyuzTQpWkij2FWPZ/H3ImSYI52n5tz\ns96/skrDUC6tuKFkcjYpMGZi1l7+qAbq1Bow/
            vrS7GYcUlfk0Zo4xgMhyYLyIyYH\n3Ig4SMECNt+DnZRXqUh3+4yivInabAQb5HAhY9Bn92LaDgE/37QZNPP9YnPzVP5p\ntR033QxUIttQm0mA/u3aPIGF+icKzIo/
            ymvHaxsT4MH1L94y8mbKJbnpySf72c3L\nJm6pxYECgYEA0wZRFWc2yEOo/wJt2bGar3lYmbIMvXExd1OKlC+V4xo50swOxGXq\nICa87eFfKWSMxIxU1BxOSyZf9vkJEf
            +g3ipY04fOrrkiSmEp/ZWI5EEyL/xawEJ4\ntZbPkMhAzIrJqJRO9rwZYz70dfHUa3ssNmT6r9ykIifGfGLp4yegdxkCgYEA14F5\nypxBw54kkBLIABKg0z4ui0o/
            wHb6BzimHHhTfFXF2BTvhX53pZcq76zK/o+gU8CW\nahVJqqkzNeA2JGcuaM7ZOj7codnaNi6KpFEXZcc8kUE3Zr4H3BbPcqjdQD7DlBHD\nWPM40FJ/
            uJwSfh7shON40tHAT6x6B9ER2Epkv0cCgYA4C89xPc6A9edmg0ClttP8\nkuADQhS8I9oddSSzIshOXuLIp8WsR90c2O0IyPi7R6dTdGn7Q+FPeEu4jn/
            oNdqF\n5v1cgQ9VvHntA01hult3NBGeSQnZUk44/Gr4axc6vAm/KlSml2Kux0OqJ6ZhPX+K\nmtsx5gn5wFKpKqQv3pBx0QKBgQDS5UNR0x7Ml8uCLs8IuzIl5/lWmJ/
            Q8t32CSsO\nO3a/+5R0xPAtvLjMFSaQjJHPErE8pQDN6e1AlPqno7Z4Y31YVezhgjkaLv7L/Tny\nWeVC1fpgz+iupBM/ABedRsHWqFVvZv674rHbtJCHEj1KrWqbFZzc
            +yVvEwQM+D21\nZAA56wKBgCnC/UtvyRM8+8TagMtpUjEAubwGxL8RklmYrZ8hfFbLjr7anuh1bDPT\nRXrvoAQri3ALFkdpomkTpLXcqYvcF91zkzyDx7JKLEfY7Gz8t1B
            +AEJ/uOa2xSfk\nnOry7vapLAC5zqF1QrayILl/FIfV5rrV9Gt+JGBNwBOkOENvNLtv\n-----END RSA PRIVATE KEY-----"
        },
        {
```



```
            "key 2": "-----BEGIN RSA PRIVATE
            KEY-----\nMIIEpAIBAAKCAQEAmKqr62y3vxsOgTe7I3eh2YGs0ZT0Ep33DagUvlLywL23CT6x\nkG3IfC4ykpWva8ZuvpXAi4RGd9mc8Edr2ICARyakGgZ7KDNUZKT2LPjWXqm
            xqgQm\ntykgiiSh2aY1OsQZbVnSWDOSABWHxveYyzNjqpHsyG93E/TsSJ/z1cpVUOriPVzH\nOHFM0JpSupSOJ+LJI/oEntom0Ke7YvWX1tyu/93nfvSjLdYAg/
            VZ0ucz0pOSUBa8\n5ZdxAWmS7vKbbYg41d0VWRHgDF6RUYmVG3boMIEhyBMa8SZEycMdeuuGNZTZfyO2\ngSHL8z
            +d3fGCIrmhEt4RlvQOq6Cp80czHA4pWwIDAQABAoIBAANN0PIDufz2eUY4\no4yCnqEWGlxgIeFz7mXVmQ/+vDw0SY13VYvE
            +c1ziSqnof1DdSAbc5L8EN5aek1a\nQ649fF6N1TKu/eQCgGAROzar8qUSfRgrMNSj8jgAekmQkZ8CTB6UDrupRLlLE1OD\nW5jXit+lJX
            +3pyxqZOFe0ihypuKyZMA8ydBXEyYfzX59KAP+PlUJ8zapKnm5DweB\nsac4A2DNQsWmHjLXH3Po0r70WNpVwmO4+XUmddniKOjcfDxBEGoFFKq/
            lHYlMfwh\npLsEcRxA0N8pF9VI/Gwn9hmbg/v6ZDu72cN6uRrErOJZ6bkHkoG4KZ/Oa+P2hd7S\noy+uGWECgYEAwBv5tlCq5xnsUJHaSamN8ZkzXwc0r7ktc+C4UluZnvUqLw
            +bUH4j\ntXCOBcIlmcHvv2v4Vu4IbZ4wh6IDMttHhuZkiuxoHHdiANHqKOAZAVFCk
            +0062tc\nibW03N1tdIJLu4M26HdrFVV8qbUWflNiXvuquXg4NNKY8nflPtKllEsCgYEAy3CW\nIu55LZq2QnuEVEouo5G0QEkQLCWUfuDeLsFLIyW4jORjhLDIFIAxAHoWaIv8
            rSQg\n1SDoDnS6jjb2ypDvaZo/rn5tYZA8QVK/Nu1cy7uUgbS63sYTi5U4QtvkbCiori0B\nDPh6n7moTa353a0N0iHJ/
            oprhqGLRDRbZQmz9TECgYEAn0FRxEKhOs2xqqUOmYw2\nwHbpMDjwA1B2jp5XDucitbS10KeWcKbxAUOaaMmmlA1Cq6RdDetsDztZLccv8aP4\naqtp5ATQ1fBdzSI3hMUp1X7A
            u5KkERlCcGKy2XmyeChfkcXX6eS9X8UrgCA/0l9b\nqDPnNrOjnJ6M7kxYPbkwTHsCgYAWXmzDroD4eLhtmvMz6WwzZAPzxZTXpdIsGUGU\n1oASf0VJiJtArcdyKZ45tTLdj/
            baesx5YsLhRAh3NqfwIfuRGvrBjvkr7xYRAK4x\nteusIxXzJRIfCJyBjoi/ITp+ggH6FMy2xmOFSH6u++Ardam9XyW8rS
            +hgAW2HCOo\nemalIQKBgQCT3c5BaKMDhUsAQw5gsS27yQbRgcdlCdg2FaAT6pb8f2ufAtXPkvdg\nKEsil1js3u1AK3SOxrI8/
            BusFYjg4jiNjNYEpt12OtUZdwkqxgcaDwOXGxzE2TjU\n7gZ0n+1ENxMeeQ5RK1vWm9TIZBAcO/K9o78DoB/fL53WIxoAl5dZnw==\n-----END RSA PRIVATE KEY-----"
        },
        {
            "key 3": "-----BEGIN RSA PRIVATE
            KEY-----\nMIIEowIBAAKCAQEA6bSNCGL2udw3mS71v6oR29eZWbvy8QpcJvXAP0TWDH5csfB7\nsV0UBvI4o43ANDtldRIOnsIiPSH0x83GtWFjewqJa4Vep
            +SxpvB0waHkRXwAuY6P\nbr+9d8PpQgX+0M035Abei8If41KT77lX/RYaR/CRGJJPtcXeFXxWvYrFzNNI4K49\n39ehsq5bSqZ4
            +2O2UhPpNRfKAUA5vuWcB6y95Qtx82gTqz8ks3wia0JOdNjkDFgp\n+LmEnkwNkMs68F9s5hVwcrBR6lVZlNwEDa5JlmIn
            +3cc0Pfu5bfxv2MtnqD9l0IV\nKkxtugDNgXjC199ZNLtcbkw8S4kCoiSEkW8m4QIDAQABAoIBAGEUt2E8evFKIW5o\nq30q0LBaJ0G3aXvnqdAlVe7yUVhvRWJFW5yZ5a/c/
            toyEB2ibsrq7VfVnRN3/47g\nVCMUiZabEeqwRFnKDmIikN2+umyCzmpGZh7DV5lgsIzUoL1sAkOe9uNniIeNzhZP\nD9X6UICB1kQ2wxXnVBHozL9/3hru/qa
            +lh3GmPauiCybk+t9rnsMfkv3nB5pZjNi\nEIm8jPYauGJcuuMy78VbUoQnvbKv5TWnlKvA/JjDar/wKKndSSdvX9XvvrLn/
            j8F\nnFBbpwctM3FGF7w7zWNvHe730WvwY3efVgcNvBLyhQuIuNbjk3YsjfNSqM17NZi/
            \nqsXyhuECgYEA8ulKLW1A3blFsG95n1eXYyHx7e1p8VR6KxSOg6X2P7SN6aeQqkrw\nVfQvGpsWazwd6gG0fkW95e0frlprLshW2olLg0Id5O0BWIzsORzna2Vy5wCTak8/
            \nH82xDhrkgFfYOv/Ddz6aHPyNOHIkaIpxV/73Fp5cqKMTftkzqqopidMCgYEA9kxG\nE/E6ZdCQn8SllfGRjbVVRjPWsbm8dcH1kC8JZ/
            jmagHE7nRqAxpfYyUB0V6q9EWl\n59mYxHCgt1kHH2VJxAspZbrYMYjjTVI5tdIVj9o7gpfM0Y3M4Sfbi5GiQYPaJLBL\nG6Qx30pKgguqEjzpzrc7xVMNVAOY6wkPgs/mx/
            sCgYEAqmTjV8JPwjj4na9UzWWQ\nGEXzg0cLDfKIUDf3UHnWpxFsiv60CADRXjoP4MXxj/93rHLNvdYGQ4d2tcNPdF2s\nL1gn1EdI2RE35HnWKqoP29IqK8jHn8c/LHwJOy
            +6Ih183MYUw0zypmqyVWocCAk1\nQNjlUsR7kv/x6QVLrOf4g2kCgYBzSuUK9MLXYjgw8cZpWDlAUn/mauG3NDCqFu8y\nXTJT/0ksmkNrNAdkCHZaXLprHZt82RV
            +YHeIjQhOi1gwNCPBp0/rlPkaxu9QBIuL\nNxpsykVNLm3sMiqTwyqcPMHtVSFKR1QBTm0iDw6trXQhMW5pG5DZH7V/fGearhUa\nsONg/
            QKBgF1acmc8gEbZ1jsVKAtirQJij12zl3mD+YRq75uCNz8yvLOBA7v+Lj79\nCUWLSqLY7Ot7EBLm6u5JJS2rOU2szmErlk0SyZ3uuI7bpRbxni3Ze
            +ecXpU77rVw\nqxEMiwy6QSSob1Rl6rXbZdF9knca02wbdaGeyaM0lhmc4jMrxm8R\n-----END RSA PRIVATE KEY-----"
        }
    ]
}
```

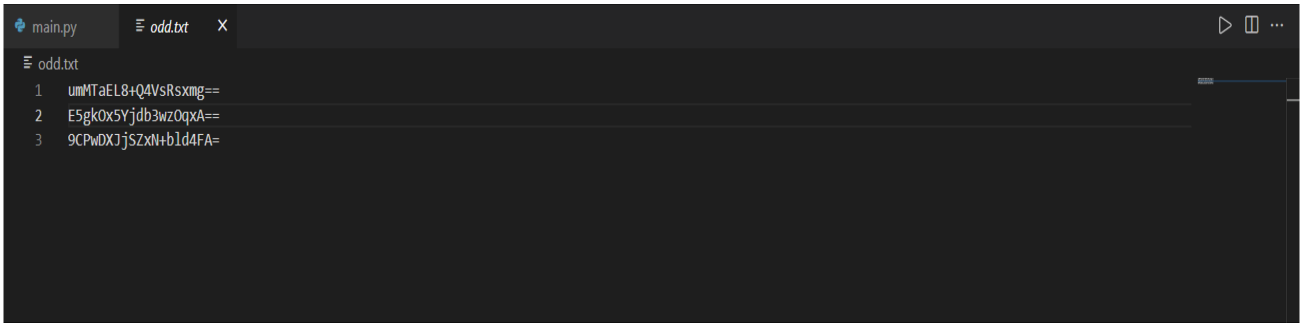
Content of private_key.json after generating keys

5. Encryption of odd.txt:



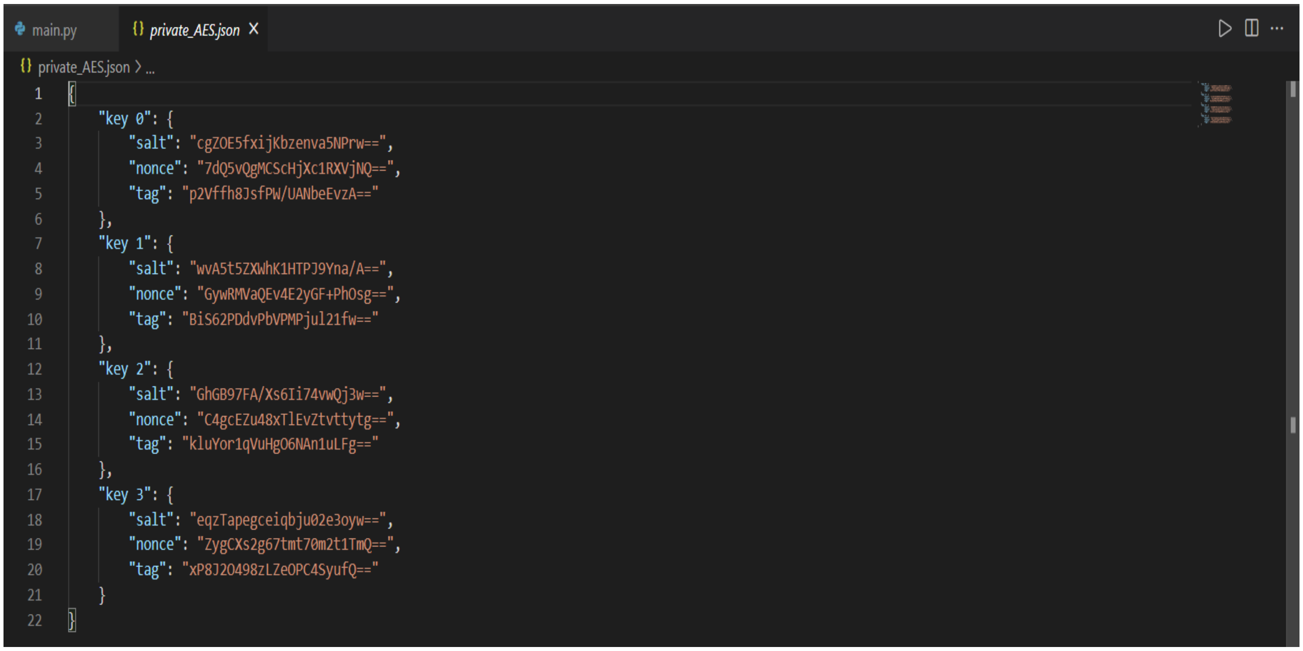
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 3
Enter the file name: odd.txt
Message encrypted successfully!
```

Encrypting odd.txt



```
umMTaEL8+Q4VsRsxmg==
E5gkOx5Yjdb3wzOqxA==
9CPwDXJjSZxN+bld4FA=
```

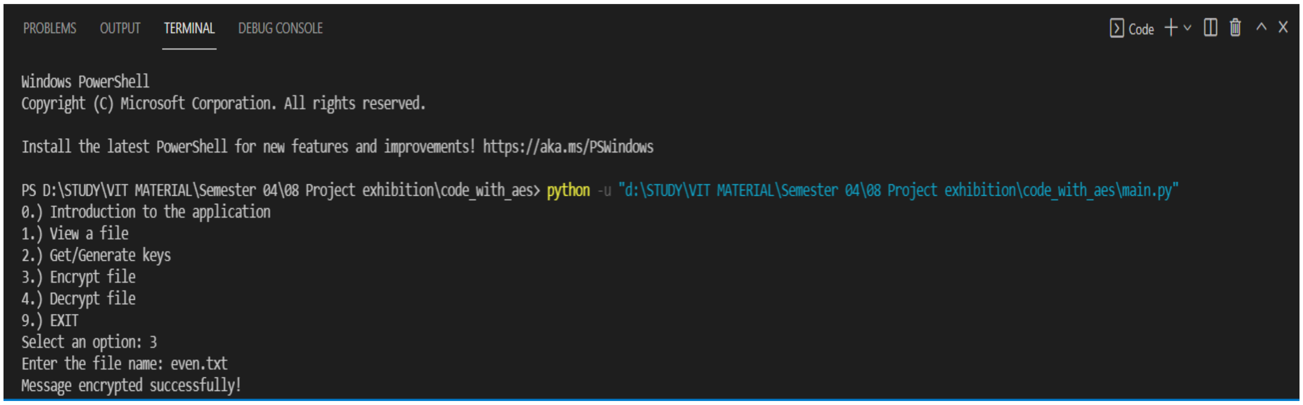
Content of odd.txt after encryption



```
{
    "key 0": {
        "salt": "cgZOE5fxijKbzenva5NPrw==",
        "nonce": "7dQ5vQgMCScHjXc1RXVjNQ==",
        "tag": "p2Vffh8JsfPW/UANbeEvzA=="
    },
    "key 1": {
        "salt": "wvA5t5ZXWhK1HTPJ9Yna/A==",
        "nonce": "GywRMVaQEv4E2yGF+PhOsg==",
        "tag": "BiS62PDdvPbVPMPjul21fw=="
    },
    "key 2": {
        "salt": "GhGB97FA/Xs6Ii74vwQj3w==",
        "nonce": "C4gcEZu48xTlEvZtvttytg==",
        "tag": "kluYor1qVuHgO6NAn1uLFg=="
    },
    "key 3": {
        "salt": "eqzTapegceiqbju02e3oyw==",
        "nonce": "ZygCXs2g67tmt70m2t1TmQ==",
        "tag": "xP8J2O498zLZeOPC4SyufQ=="
    }
}
```

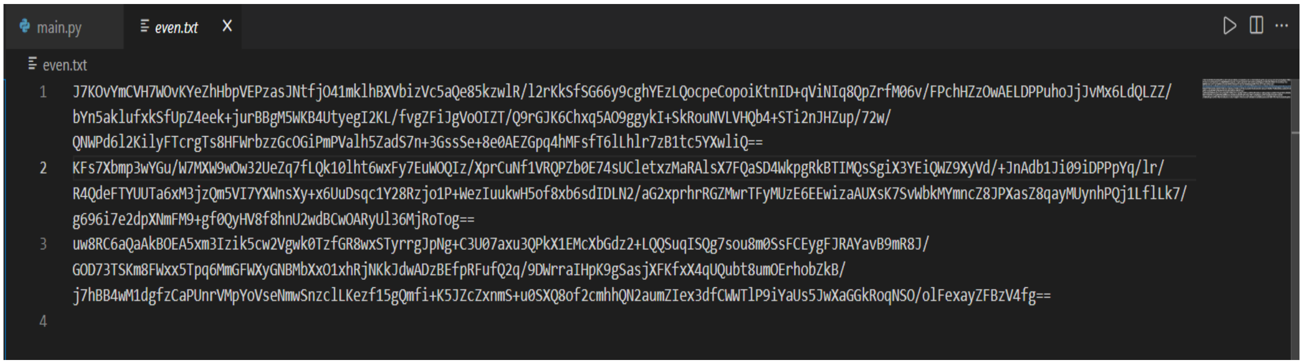
Private_AES.json containing salts, nonces and tags used for encryption/decryption using AES

6. Encryption of even.txt



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 3
Enter the file name: even.txt
Message encrypted successfully!
```

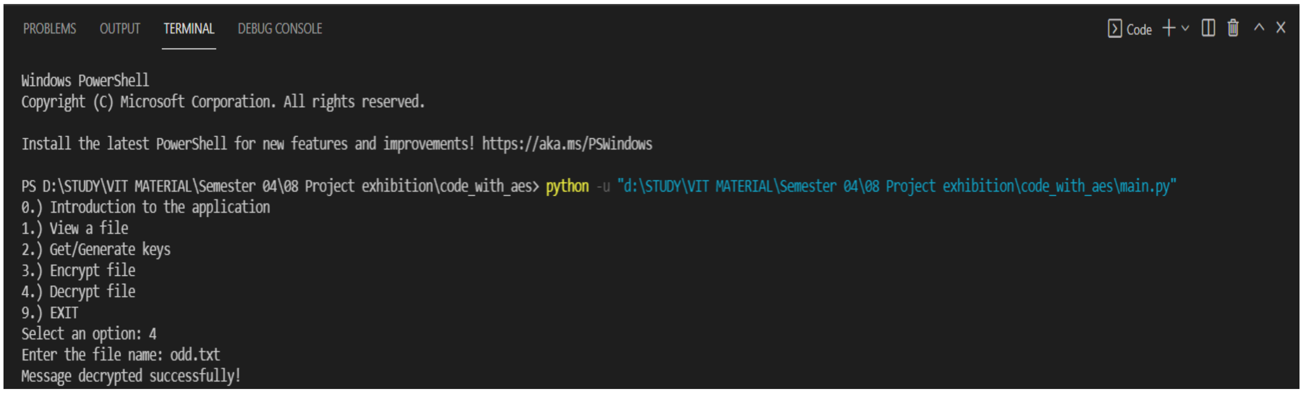
Encrypting even.txt



```
J7KOvYmCVH7WOvKYeZhHbpVEPzasJNtfjO41mklhBXVbizVc5aQe85kzwlR/l2rKkSfSG66y9cghYEzLQocpeCopoiKtnID+qViNIq8QpZrfM06v/FPchHZzOwAELDPPuhoJjJvMx6LdQLZZ/
bYn5aklufxkSfUpZ4eek+jurBBgM5WKB4UtyegI2KL/fvgZFiJgVoOIZT/Q9rGJK6Chxq5AO9ggykI+SkRouNVLVHQb4+STi2nJHZup/72w/
QNWPd6l2KilyFTcrgTs8HFWrbzzGcOGiPmPVaIh5ZadS7n+3GssSe+8e0AEZGpq4hMFsfT6lLhlr7zB1tc5YXwliQ==
KFs7Xbmp3wYGu/W7MXW9wOw32UeZq7fLQk10lht6wxFy7EuWOQIz/XprCuNf1VRQPZb0E74sUCletxzMaRAlsX7FQaSD4WkpgRkBTIMQsSgiX3YEiQWZ9XyVd/+JnAdb1Ji09iDPPpYq/lr/
R4QdeFTYUUTa6xM3jzQm5VI7YXWnsXy+x6UuDsqc1Y28Rzjo1P+WezIuukwH5of8xb6sdIDLN2/aG2xprhrRGZMwrTFyMUzE6EEwizaAUXsK7SvWbkMYmncZ8JPXasZ8qayMUynhPQj1LflLk7/
g696i7e2dpXNmFM9+gf0QyHV8f8hnU2wdBCwOARyUl36MjRoTog==
uw8RC6aQaAkBOEA5xm3Izik5cw2Vgwk0TzfGR8wxSTyrrgJpNg+C3U07axu3QPkX1EMcXbGdz2+LQQSuqISQg7sou8m0SsFCEygFJRAYavB9mR8J/
GOD73TSKm8FWxx5Tpq6MmGFWXyGNBMbXxO1xhRjNKkJdwADzBEfpRFufQ2q/9DWrraIHpK9gSasjXFKfxX4qUQubt8umOErhobZkB/
j7hBB4wM1dgfzCaPUnrVMpYoVseNmwSnzclLKezf15gQmfi+K5JZcZxnmS+u0SXQ8of2cmhhQN2aumZIex3dfCWWTlP9iYaUs5JwXaGGkRoqNSO/olFexayZFBzV4fg==
```

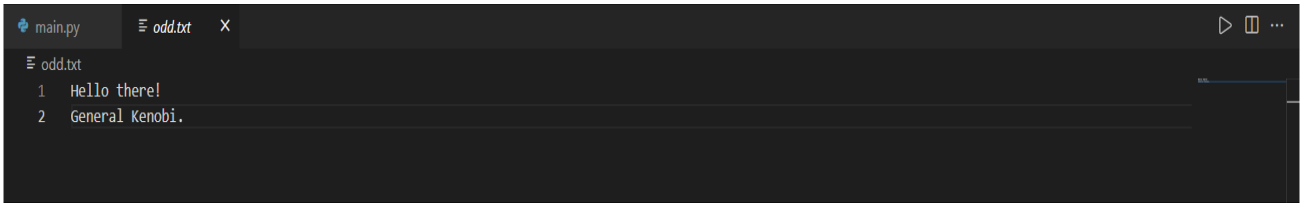
Content of even.txt after encryption

7. Decryption of odd.txt:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 4
Enter the file name: odd.txt
Message decrypted successfully!
```

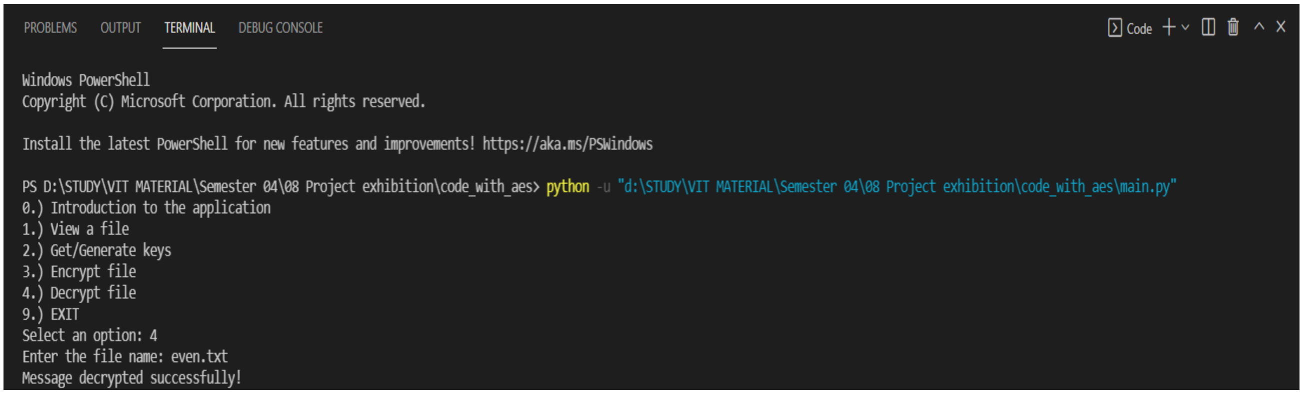
Decrypting odd.txt



```
Hello there!
General Kenobi.
```

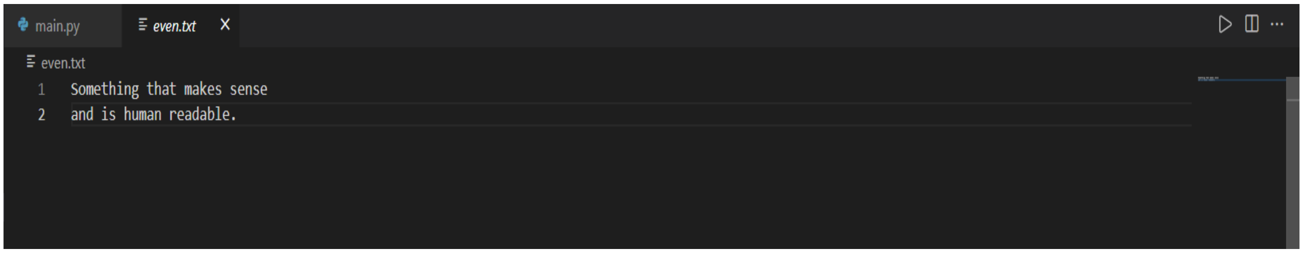
Content of odd.txt after decryption

8. Decryption of even.txt:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes> python -u "d:\STUDY\VIT MATERIAL\Semester 04\08 Project exhibition\code_with_aes\main.py"
0.) Introduction to the application
1.) View a file
2.) Get/Generate keys
3.) Encrypt file
4.) Decrypt file
9.) EXIT
Select an option: 4
Enter the file name: even.txt
Message decrypted successfully!
```

Decrypting even.txt



```
Something that makes sense
and is human readable.
```

Content of even.txt after decryption

5.5. Performance Analysis (Graphs/Charts)

5.5.1. Time complexity:

5.5.1.1. For RSA

Encryption: - 4O(n^2) n-bits
Decryption: - 4O(n^3)
Total time complexity for RSA: - 4(O(n^2) + O(n^3))

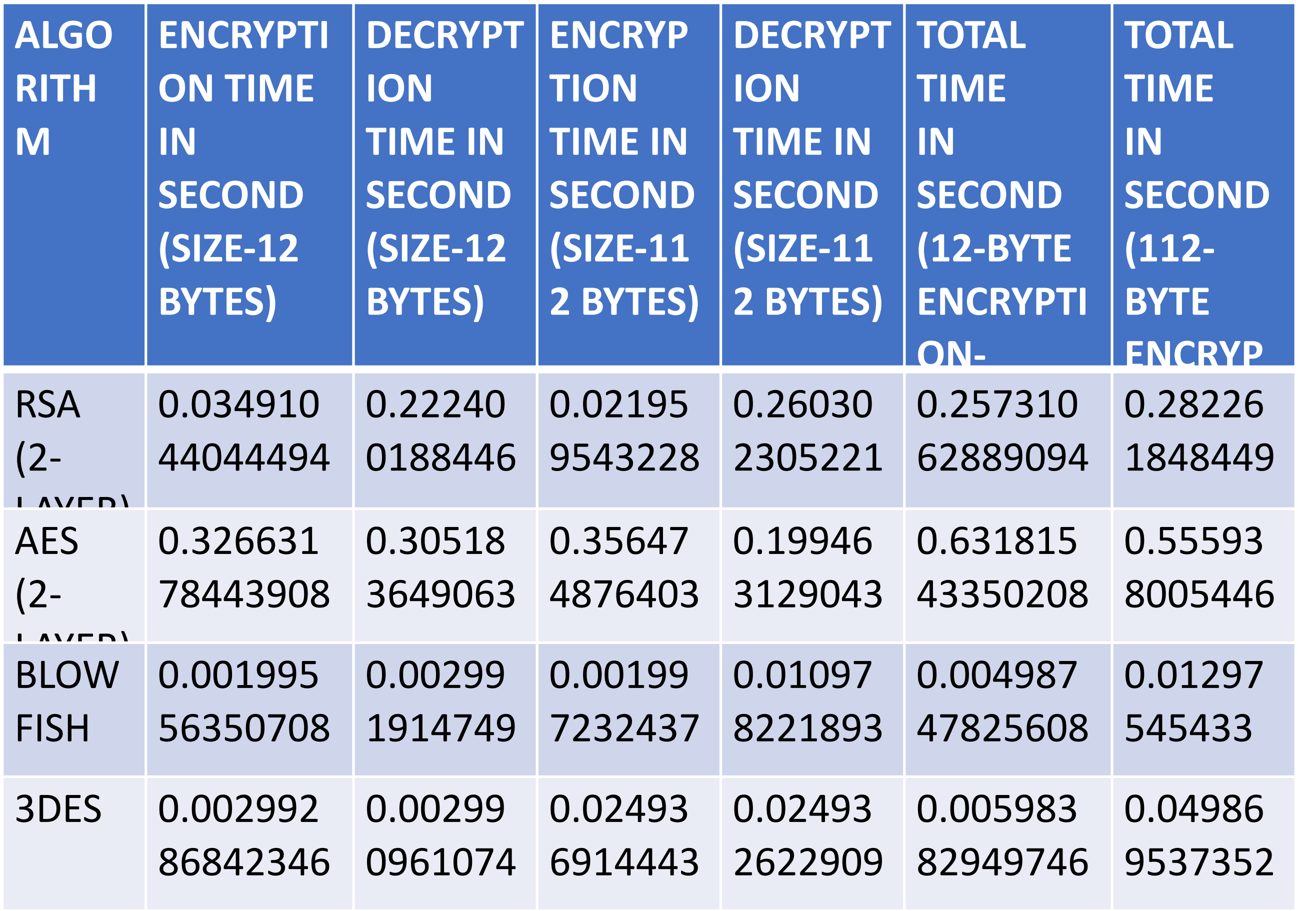
5.5.1.2. For AES

Time complexity for AES: - 4(2^48)

5.5.2. Space complexity:

As more advance hardware and web servers are present today, space complexity will not be a concern for this project.

TIME TAKEN FOR ENCRYPTION AND DECRYPTION USING DIFFERENT ALGORITHMS



ALGORITHM	ENCRYPTION TIME IN SECOND (SIZE-12 BYTES)	DECRYPTION TIME IN SECOND (SIZE-12 BYTES)	ENCRYPTION TIME IN SECOND (SIZE-112 BYTES)	DECRYPTION TIME IN SECOND (SIZE-112 BYTES)	TOTAL TIME IN SECOND (12-BYTE ENCRYPTION-	TOTAL TIME IN SECOND (112-BYTE ENCRYP
RSA (2-LAYER)	0.03491044044494	0.222400188446	0.021959543228	0.260302305221	0.25731062889094	0.282261848449
AES (2-LAYER)	0.32663178443908	0.305183649063	0.356474876403	0.199463129043	0.63181543350208	0.555938005446
BLOWFISH	0.00199556350708	0.002991914749	0.001997232437	0.010978221893	0.00498747825608	0.01297545433
3DES	0.00299286842346	0.002990961074	0.024936914443	0.024932622909	0.00598382949746	0.049869537352

CHAPTER-6:

PROJECT OUTCOME AND APPLICABILITY

6.1 Key implementations outline of the System

Our program may be used to securely encrypt and store /transmit text files containing confidential content. Even if, in the future, the RSA or AES encryptions are broken, files encrypted by our algorithm will be secure as not only we are encrypting the files twice, we are also encrypting the different sections using different keys after first layer of encryption. Also, the encryption algorithm that is used depends on the length of the file name.

6.2 Significant project outcomes

We have made a program that encrypts the files in such a way that they are incredibly hard to decode by a 3rd party interceptor.

6.3 Project applicability on Real-world applications: -

This project holds the ability to provide a secure experience to the world also it will provide the user the ability to secure their data and share it safely without damaging the confidentiality of the data. As it is wholly a new concept it will open many different paths for cryptographer to think on this new concept of encryption.

6.4 Inference

While working on the project, we learnt a lot of new things that enabled us to create a working prototype that demonstrates our idea.

Not only is our idea unique, it is more secure than the existing alternatives as it will be able to keep the data confidential even if the basic algorithms are broken in the future.

CHAPTER-7:

CONCLUSIONS AND RECOMMENDATION

7.1 Outline

7.2 Limitation/Constraints of the System

The limitation of our system is that it cannot encrypt text file larger than 214 bytes. As we know that the maximum word limit to encrypt in RSA is 214 bytes and is infinite for AES. The maximum file size that can be encrypted without changing the length of text file is up to 214 bytes.

7.3 Future Enhancements

We are working on it to enhance the power of our system to make it compatible for encrypting files with larger size.

7.4 Inference

Our project is unique and provide security more than already existing algorithms. It can also be a building block for powerful encrypting softwares. Future enhancements in it will help us achieve encryption on larger size files.

REFERENCES

1. https://www.pythonpool.com/rsa-encryption-python/

2. https://gist.github.com/lkdocs/6519378

3. https://qvault.io/cryptography/aes-256-cipher-python-cryptography-examples/