Final Report – Key Escrow System

## Introduction & Overview

Nowadays, key escrow systems are becoming increasingly important as the key technique to protect data privacy ensures communication security between parties. The key escrow system is used to give the trusted third party the right to store and manage accesses to encryption keys, ensuring that only authorized individuals can access these keys. In this final project, we will have learned more in detail about key management, key exchange, and the required framework in a successful key escrow system implementation. Moreover, concepts related to key generation, storage, key distribution and destruction will also be studied. Therefore, it is necessary to use advanced encryption algorithms and permission control to ensure its confidentiality, integrity, availability and auditability. Using a secure protocol is also crucial to preventing key theft or tampering.

In a discussion regarding key escrow systems, it is not just said how they store keys or how secure they are. Key escrow systems can be used by authorities such as government entities for the purpose of law enforcement, criminal investigation and national security protection. In the case of criminal investigation, authorized entities can be granted access to keys that decrypt data even when the data owner is unavailable (non-cooperative, deceased, etc.). This leads to a higher chance of success in resolving said investigations and bringing perpetrators to justice. A government entity may also use key escrow systems to monitor organizations and ensure that they follow established regulations. This increases the security of user privacy and data protection. It also gathers evidence of those who abuse the laws to seek personal gain [1].

Although arguments exist claiming that governments may use this system to spy on its citizens [1], the importance of key escrow systems in maintaining social order can also be considered. Without the implementation of key escrow systems, it is more difficult to solve crimes when people are uncooperative. Furthermore, malicious organizations may deliberately leak critical information or user data should they have access rights to cryptographic keys.

Additional examples of real-world implementations of key escrow systems include:

* AWS Key Management Service (KMS) is a fully managed service that centrally controls and protects encryption keys for data. Users can manage Console, AWS CLI, SDK, or API through the AWS and build symmetric keys (for data encryption and decryption) and asymmetric keys (for signature and authentication). At the same time, you can customize detailed key policies and access control policies to specify which users can access and manage keys. After creating the key, the key will be stored encrypted inside the AMS KMS, only AWS account or AWS server who are authorized can access it, and AWS KMS will update automatically update the key to keep it safe [2].
* Microsoft Key Management Service (KMS) is primarily used for batch activate Windows and Office products. It uses symmetric encryption technology (AES-256) to encrypt data storage and dissymmetric encryption technology (RSA) to encrypt the key itself and stores these keys in the controlled Hardware Security Module (HSM) to keep them safe. KMS use multiple layers of security to limit and manage the access to keys, only authorized personnel and systems have access to the keys. Microsoft Key Management Service is used on Azure platform and Microsoft 365 [3].

Some other advantages and disadvantages that a key escrow system may include:

Advantages:

* Data recovery: Losing keys to encrypt and decrypt data is certainly one of the most disastrous events that could happen to any organization. Implementing key escrow is a good solution to prevent this from happening. Since key escrow stores copies of the keys, authorized individuals from organizations can always access and recover the keys [4].
* Compliance with established regulations: As mentioned above, key escrow systems can be used by authorities to oversee organizations’ activities [4].

Disadvantages:

* Privacy issues: Privacy issues arise when people are concerned about giving a third-party access to the cryptographic keys. This means the third-party might decrypt critical information without the knowledge or consent of the owner [4].
* Security issues: The third parties that are trusted to store the cryptographic keys can be seen as a great target for hacking because with all the stored keys, hackers can easily access and steal critical data. If the third-party is hacked, the encrypted data is now highly vulnerable and can lead to catastrophic incidents [4].

To implement a prototype of key escrow system, we used CentOS 9 virtual machine as the development platform. The prototype will consist of a Key Escrow VM that stores and manages keys and two client VMs that will encrypt and decrypt messages. These two client VMs will get the key from the Key Escrow VM. In the development process, we used Python, Flask, OpenSSH-server, and various cryptographic libraries to ensure a secure key escrow system. This final report will detail the process of planning and developing the key escrow system on Linux virtual machine platform. Through this project, we hope to demonstrate the effectiveness of key custody systems and provide references for related fields.

## Planning

In the initial planning stage, we explored the core concepts of critical managed systems, security requirements and theoretical research of critical managed systems, and conducted the initial code implementation. To meet the security requirements of key escrow system, we decided to develop the prototype on Linux virtual machine – CentOS 9. With CentOS 9, we can configure, and control resources freely and flexibly, such as CPU, memory and storage space. Moreover, virtual machines are isolated environments from the host, which means we can run and test the prototype without having it interfere with said host. It is also easy to manage and increase security of the system. In case of failure, the code can always be modified or started over by creating new virtual machines. For the programming language in which this code would be written, Python was chosen for its readability, conciseness and flexibility. Python being common language, meaning it can be used for the development of various functions and easily integrated with other system components. In the development process, we also used Flask, OpenSSH-server, and various cryptographic libraries to ensure a secure key escrow.

## Design

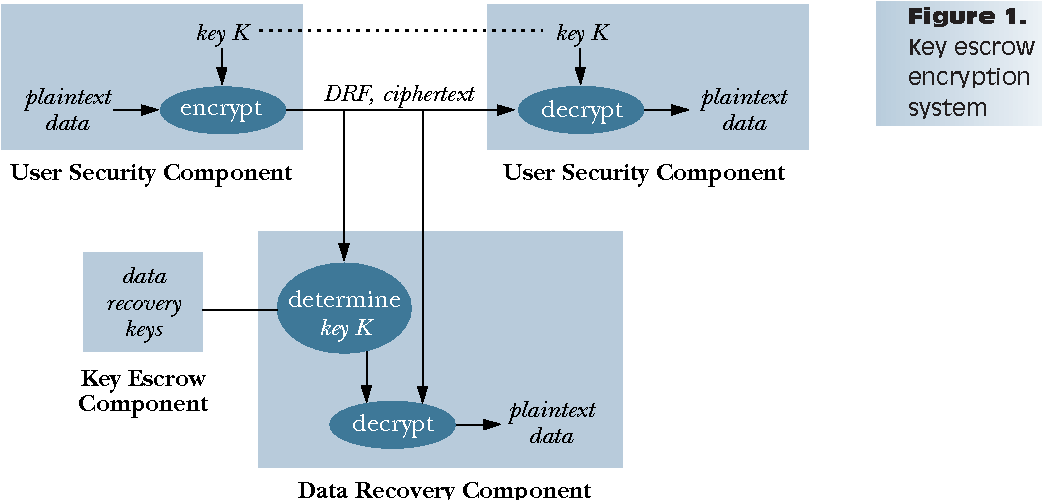
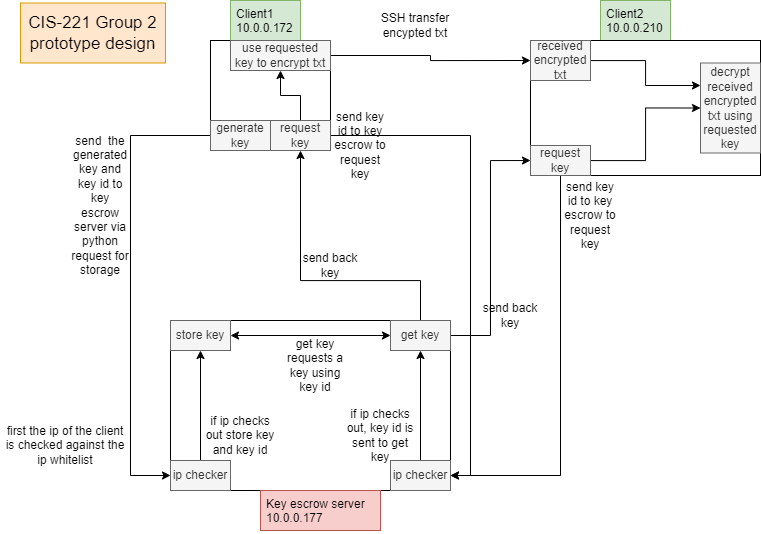


Fig. 1. Example of a key escrow encryption system. The need of the components of a key escrow system remains the same from initial conception to final implementation. [5]

As visualised in [5, Fig. 1], three components required for a key escrow system to function: a User Security Component (USC), a Key Escrow Component (KEC) and a Data Recovery Component (DRC). While each of these components are detailed in-depth within our first milestone report (research paper), the summary of each in their fundamentals are:

* USC: Responsible for encryption and decryption of data. Consists of hardware devices or software programs for encrypting and decrypting data in secure user communications, as well as supporting key escrow functions and data recovery functions [5], [6].
* KEC: Responsible for storing keys for data recovery. Consists of a key escrow agent, data recovery key, data recovery service, and escrow key protection. It is used to complete the operations of a key escrow agent and store and use data encryption [5], [6].
* DRC: Dedicated to decrypting data provided by the KEC. Consists of dedicated algorithms, protocols, and necessary equipment to recover plain text from the managed key K provided by cipher text and KEC [5], [6].

The example of a key escrow system shown in [5, Fig. 1] was used as an inspiration to how we may design our own prototype, with it also showing the importance of the three components depicted. Ultimately, the prototype we created differed from the exact design of [5, Fig. 1], however, the relevance of the three components stays and remains the same from its initial conception to our implementation.

Fig. 2. Visualisation of the final design of our prototype.

The design of our prototype is illustrated in [Fig. 2]. In our implementation, the key escrow server VM acts as the KEC component, storing keys as described. Meanwhile, the client VMs act as both simultaneously the USC and DRC. Within a single client VM is python code that allows the VM to encrypt and decrypt a file, functionally making the VM the USC. The client VM also acts as the DRC, where it can encrypt and decrypt text as well as sending it back and forth. Our implementation is not limited to two client VMs, thanks to each client VM containing the same Python code. The system can have as many of these VMs as allowed by how much stress the key escrow server can handle. In fact, the number of keys that can be stored in the system is as much as memory allows (typically in the millions), which is owed to the feature of the Python dictionary.

## Development

Our goal was to build a key escrow system where the cryptographic keys are stored in the key server and only authorized individuals are allowed to access the keys. The stored keys are used to encrypt the plaintext and decrypt the ciphertext. With the three VMs, one key server and two clients, created and installed, we are now able to develop a key escrow system by using available tools and downloadable packages.

In our second milestone report, we highlighted various issues and hurdles encountered in the first attempts in the development of the system. The errors and lack of efficiency (and no effective solutions) with using Ubuntu VMs as the base to implement the system was noted, leading to instead switch the implementation to CentOS VMs. Other issues (save Python syntax) included connectivity errors between the client and escrow servers—specifically issues with configurations of the key escrow servers—migrating the entire implementation to the Oracle platform was also considered but dismissed in favour of committing to the CentOS VM infrastructure.

With further development of the system, we realized the main problem and solution to the connectivity issues encountered in our previous setup. A step missed in our previous report; setting up a bridge connection—an important step—it allows the VMs to connect and communicate over the network. [7] This is because all three VMs are on the same computer, therefore all VMs have the same IP address but with a bridged connection set up in settings of the VMs. Each VM has a unique private IP address that allows them to communicate between each other.

## Implementation

After solving all the problems, we started to run and test the prototype by having Client 1 VM send the generated key to the key escrow VM. The key escrow server is now responsible for storing the keys. Client 1 then sends a message to Client 2, which is encrypted into ciphertext. However, since the message is encrypted, Client 2 cannot read the message without the key. Client 2 now requests the key from the key escrow VM to decrypt the message.

As illustrated in [Fig. 2.], First, Client 1 generates a key and sends it to the key escrow server for storage. This step involves two actions: generating the actual key using the Fernet library and sending it to the key escrow server via a POST request. Additionally, the user creates a key ID, which will be used to request the key from the key escrow server in the future. Client 1 then encrypts the text by requesting the generated key from the key escrow server. Once encrypted, Client 1 sends the ciphertext to Client 2. To decrypt the encrypted text, Client 2 requests the key from the key escrow server using the same key ID that was used to generate the key that encrypted the text.

[Fig. 2] also shows that this process can be done from Client 2 to Client 1 as well. The two clients are used for example purposes only, as both clients use the same code. There can be as many clients as the key escrow system can handle, all generating and requesting keys from the key escrow server.

For security, the first level we added was the key ID, which can be a combination of letters and numbers. Without knowing the key ID, it is impossible to request the key from the key escrow system. Another level of security we added was an IP whitelist. This means that if a client's IP does not match the allowed IPs listed in the key escrow system, that client will not be able to perform any actions.

## Documentation

This section documents the information and sources used in developing and implementing the prototype.

<https://www.youtube.com/watch?v=vrlwIG33_l4>

Explanation on fernet ish

**Fernet Encryption**:

* **YouTube Video**: The YouTube video explains the principle of Fernet Encryption. Fernet Encryption is a symmetric encryption algorithm that can provides secure encryption and decryption of data. It uses AES-128-CBC for encryption and HMAC-SHA256 for authentication. Fernet Encryption is a state machine, which requires internal state to ensure correct encryption and decryption. Additionally, Fernet Encryption supports version control, allowing use different versions for encryption and decryption.
* **Main Points**: Fernet generates guarantee that a message encrypted using it cannot be manipulated or read without the key

<https://www.twilio.com/docs/usage/tutorials/how-to-set-up-your-python-and-flask-development-environment>

**Setting up Python and Flask Development Environment**:

* **Twilio Tutorial**: The tutorial covers steps on how to select, install the Python version, set up a text editor or IDE, create a new project using virtualenv, and how to create a Flask application.
* **Main Points**:
  + Installing Python and pip.
  + Setting up a virtual environment.
  + Installing Flask and other dependencies.
  + Running a simple Flask application.

<https://www.linuxjournal.com/article/8904>

Ssh transfer

**SSH Transfer**:

* **Linux Journal Article**: The article explains how to install SSHFS and FUSE, configure user permissions, and mount remote directories.
* **Main Points**:
  + Using commands like scp (secure copy) and sftp (SSH File Transfer Protocol) for transferring files.
  + Examples of how to use these commands.
  + Benefits of using SSH for secure file transfers.

<https://www.w3schools.com/python/gloss_python_string_concatenation.asp>

python

**Python String Concatenation**:

* **W3Schools Guide**: This guide provides an overview of different methods to concatenate strings in Python.
* **Main Points**:
  + Strings using ‘+’ Operator
  + Strings using ‘join()’ Method
  + Strings using F-string

Create an IP Address Allow List with CIDR Block Support in Python (2022, July 26) <https://nickjanetakis.com/blog/create-an-ip-address-allow-list-with-cidr-block-support-in-python>

CentOS Documentation Home: CentOS Docs site. (n.d.). <https://docs.centos.org/en-US/docs>

Welcome to Flask — Flask Documentation (2.0.x). (n.d.). <https://flask.palletsprojects.com/en/2.0.x/>

3.12.4 documentation. (n.d.). <https://docs.python.org/3/>

OpenSSH: Manual pages. (n.d.). <https://www.openssh.com/manual.html>

pip documentation v24.1.2. (n.d.). <https://pip.pypa.io/en/stable/>

Welcome to pyca/cryptography — Cryptography 43.0.0.dev1 documentation. (n.d.). <https://cryptography.io/en/latest/>

Requests: HTTP for HumansTM — Requests 2.32.3 documentation. (n.d.). <https://requests.readthedocs.io/en/latest/>

"Cryptography and Network Security: Principles and Practice" by William Stallings (Book)

"Applied Cryptography: Protocols, Algorithms, and Source Code in C" by Bruce Schneier (Book)

Ellingwood, J. (2015b, March 20). How To Serve Flask Applications with uWSGI and Nginx on CentOS 7. DigitalOcean. [https://www.digitalocean.com/community/tutorials/how-toserve-flask-applications-with-uwsgi-and-nginx-on-centos-7](https://www.digitalocean.com/community/tutorials/how-to%02serve-flask-applications-with-uwsgi-and-nginx-on-centos-7)

NGINX Product Documentation. (n.d.). <https://docs.nginx.com/>

QuickStart — Flask Documentation (2.0.X). (n.d.-b). <https://flask.palletsprojects.com/en/2.0.x/quickstart/>

RSA — Cryptography 43.0.0.dev1 documentation. (n.d.). <https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/>

Admin. (2023, April 27). What is OpenSSH? SSH Academy. <https://www.ssh.com/academy/ssh/openssh>

Fernet (symmetric encryption) — Cryptography 43.0.0.dev1 documentation. (n.d.). <https://cryptography.io/en/latest/fernet/>

S, S. (2024b, January 23). How to install and use Pip3. ActiveState. <https://www.activestate.com/resources/quick-reads/how-to-install-and-use-pip3/#:~:text=Install%20Pip3-,pip3%20is%20the%20official%20package%20manager%20and%20pip%20command%20for,PyPI%20(Python%20Package%20Index>

requests. (2024, May 29). PyPI. <https://pypi.org/project/requests/>

Ashi. (2024, February 7). Understanding Flask Framework: Installation, features & Expert Insights. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/10/flask-python/#:~:text=Flask%20is%20a%20popular%20Python,web%20applications%20quickly%20and%20easily>

## References

[1] “Key Escrow: Keeping Secrets Safe: Public Key Escrow Mechanisms Explained,” FasterCapital, Jun. 28, 2024. <https://fastercapital.com/content/Key-Escrow--Keeping-Secrets-Safe--Public-Key-Escrow-Mechanisms-Explained.html#The-Role-of-Governments-in-Key-Escrow>

[2] Amazon Web Services. (n.d.). *AWS Key Management Service (KMS) features*. Retrieved July 30, 2024, from <https://aws.amazon.com/cn/kms/features/>

[3] Microsoft. (n.d.). *Microsoft compliance assurance: Encryption*. Retrieved July 30, 2024, from <https://learn.microsoft.com/en-us/compliance/assurance/assurance-encryption>

[4] “The Importance of Key Escrow in Cybersecurity,” Blue Goat Cyber. <https://bluegoatcyber.com/blog/the-importance-of-key-escrow-in-cybersecurity/>

[5] D. E. Denning and D. K. Branstad, “A taxonomy for key escrow encryption systems,” Communications of the ACM, vol. 39, no. 3, pp. 34–40, Mar. 1996, doi: <https://doi.org/10.1145/227234.227239>.

[6] D. E. Denning, “Descriptions of Key Escrow Systems,” Feb. 26, 1997. <https://faculty.nps.edu/dedennin/publications/Descriptions%20of%20Key%20Escrow%20Systems.htm>

[7] “How to Set up a Network Bridge for Virtual Machine in Linux,” GeeksforGeeks, Oct. 24, 2023. <https://www.geeksforgeeks.org/how-to-set-up-a-network-bridge-for-virtual-machine-in-linux/>

## Appendix

Python code for key escrow system prototype:

#Class: CIS-221

#Group: 2

#Project: Key escrow system

#Group members: Anoop, Pranav, Lisa, Evan, Teny

#Imports library Flask, request, and jsonify

from flask import Flask, request, jsonify

#flask syntax

app = Flask(\_\_name\_\_)

#pyhton dictionary that stores keys and key id's in key: value pairs

keys = {}

#a temp variable that stores the key while the user is makeing a key id

key = "null"

#holds ip and only IP's on this list can ascess this key escrow

IP\_whitelist = ['10.0.0.172', '10.0.0.210']

#activates when the 'get\_key' url is used in a client

@app.route('/get\_key', methods=['POST'])

def get\_key():

#checks request ip against ip whitelist

if request.remote\_addr not in IP\_whitelist:

return jsonify("3"), 403

else:

#gets the key\_id from the client

key\_id = request.get\_json()

#checks if the key id matches a key in the keys, if it does it returns the key if not

#it returns 1

if key\_id in keys:

return jsonify(keys[key\_id]), 200

else:

return jsonify('1'), 500

#activates when the 'store\_key' url is used in a client

@app.route('/store\_key', methods=['POST'])

def store\_key():

#checks request ip against ip whitelist

if request.remote\_addr not in IP\_whitelist:

return jsonify("3"), 403

else:

#gets the key\_id from the client

key\_id = request.get\_json()

#checks if the key id matches a key in the keys, if it does return 1 otherwise store the

#key id with the key already gotten from the 'store\_key\_value function in the python

#dictionary

if key\_id in keys:

return jsonify("1"), 500

else:

keys[key\_id] = key

return jsonify("2"), 200

#activates when the 'store\_key\_value' url is used in a client

@app.route('/store\_key\_value', methods=['POST'])

def store\_key\_value():

#checks request ip against ip whitelist

if request.remote\_addr not in IP\_whitelist:

return jsonify("3"), 403

else:

#gets the key from the client

key\_value = request.get\_json()

#stores the key in the global value

global key

#checks to see if it was stored succsessfully, if yes then return 2 but if no then return 1

key = key\_value

if key == key\_value:

return jsonify('2')

else:

return jsonify('1')

#flask syntax to start the sever will be running on port 5000

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

Client code:

#Class: CIS-221

#Group: 2

#Project: Key escrow system

#Group members: Anoop, Pranav, Lisa, Evan, Teny

#imports library requests, and Fernet

import requests

from cryptography.fernet import Fernet

#urls to connect to the key escrow and request it to do things

url1 = 'http://10.0.0.177:5000/store\_key'

url2 = 'http://10.0.0.177:5000/get\_key'

url3 = 'http://10.0.0.177:5000/store\_key\_value'

def generate\_key():

# generated a key

key = Fernet.generate\_key()

#turns the key into utf-8 format

key\_str = key.decode('utf-8')

#sends the key to the key escrow server

response = requests.post(url3, json=key\_str)

#checks to see if the key was recieved based on the response or if your ip is authorized

data = response.json()

if data == '1':

print("key not received")

elif data =='2':

print("key received")

elif data == '3':

print("not authorized")

#asks the user to make a key id for the key that was generated

creation\_key\_id = input("please make a key id\n")

#sends the user created key id to key escrow

response = requests.post(url1, json=creation\_key\_id)

#checks to see if the key id was recieved based on the response or if your ip is authorized

data = response.json()

if data == '1':

print("key id already exists")

elif data =='2':

print("key id received")

elif data == '3':

print("not authorized")

def view\_keys():

#asks the user to enter a key id

key\_id1 = input("key id: \n")

#requests the key that matches the key id from the key escrow

response = requests.post(url2, json=key\_id1)

#checks to see if key was found based on the response or if your ip is authorized

data = response.json()

if data == "1":

print("key not found")

elif data == '3':

print("not authorized")

else:

#prints the key that was received

print("key found: ", data)

#takes key\_id and requests a key from key escrow and returns the result

def get\_key(key\_id):

response = requests.post(url2, json=key\_id)

data = response.json()

return data

def encrypt\_file():

#asks the user to input the key id for the key they want to use to encrypt

key\_id = input("enter the key id for the key you want to use to encrypt:\n")

#sends that key id to the get\_key function and puts the response in key

key = get\_key(key\_id)

#checks the response(if response == 1 then key not found) or if your ip is authorized

if key == "1":

print("key not found")

elif key == '3':

print("not authorized")

else:

#after the key is found asks the user to input the file path for the for the file they want to encrypt

file\_path = input("enter the file path for the file you want to encrypt")

#trys to accsess that file however if an error happens it prints file not found

try:

#opens the file and puts its content in file\_content variable

with open(file\_path, 'r') as file:

file\_content = file.read()

print("file content is: ")

print(file\_content)

#uses fernet to encrypt the infrmation on the file using thekey previously retreaved

msg = file\_content.encode()

fernet\_key = Fernet(key)

cipher\_text = fernet\_key.encrypt(msg)

cipher\_text\_str = cipher\_text.decode('utf-8')

print("cipher text is:")

print(cipher\_text\_str)

#asks user to input the file path fo where they want to put the encrypted information

encrypt\_file\_path = input("enter the file path for the file you")

#places that encrypted information in that file

with open(encrypt\_file\_path, 'w') as file1:

file1.write(cipher\_text\_str)

except FileNotFoundError:

print("file not found")

def decrypt\_file():

#asks user what key was used to encrypt this information that you want to decrypt

key\_id = input("enter the key that was used to decrypt this file:\n")

#sends that key id to the get\_key function and puts the response in key

key = get\_key(key\_id)

#checks the response(if response == 1 then key not found) or if your ip is authorized

if key == "1":

print("key not found")

elif key == '3':

print("not authorized")

else:

#after the key is found it asks the user to input the file path for the for the file they want to decrypt

file\_path = input("enter the file path for the file you want to decrypt")

#trys to accsess that file however if an error happens it prints file not found

try:

#opens the file and puts its content in file\_content variable

with open(file\_path, 'rb') as file:

file\_content = file.read()

print("file content is: ")

print(file\_content)

#uses fernet to decrypt the information on the file using the key previously retreaved

fernet\_key = Fernet(key)

decrypted\_info = fernet\_key.decrypt(file\_content)

#prints the decrypted info

print("decrypted info:")

print(decrypted\_info)

except FileNotFoundError:

print("file not found")

#starts user input at '99' just to initialize the variable

user\_input = '99'

#While loop for menu

while user\_input != '5':

#asks user toinput a vlue for the menu

user\_input = input("please enter\n1 to generate key\n2 view keys\n3 to encrypt ")

#if user inputs 1 it runs the generate\_key() function

if user\_input == '1':

generate\_key()

#if user inputs 2 it runs the view\_keys() function

elif user\_input == '2':

view\_keys()

#if user inputs 3 it runs the encrypt\_file() function

elif user\_input == '3':

encrypt\_file()

#if user inputs 4 it runs the decrypt\_file() function

elif user\_input == '4':

decrypt\_file()

#if user inputs 5 it ends the program

elif user\_input == '5':

print("program end")

break

## Performance Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| Group 2 | Individual Contribution | Team Grade | Individual Grade |
| Anoop Gill | 110% |  |  |
| Evan Pham | 92% |  |  |
| Lisa Nham | 95% |  |  |
| Pranav Garg | 108% |  |  |
| Teny Zhang | 95% |  |  |
| Average | 100% |  |  |