Luke Pepin and Steven Chen

CSE 3140 - Lab 3 Report

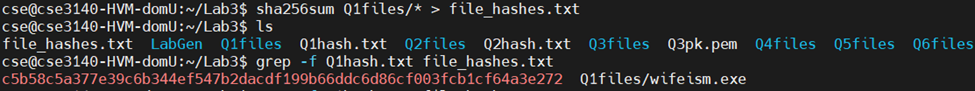
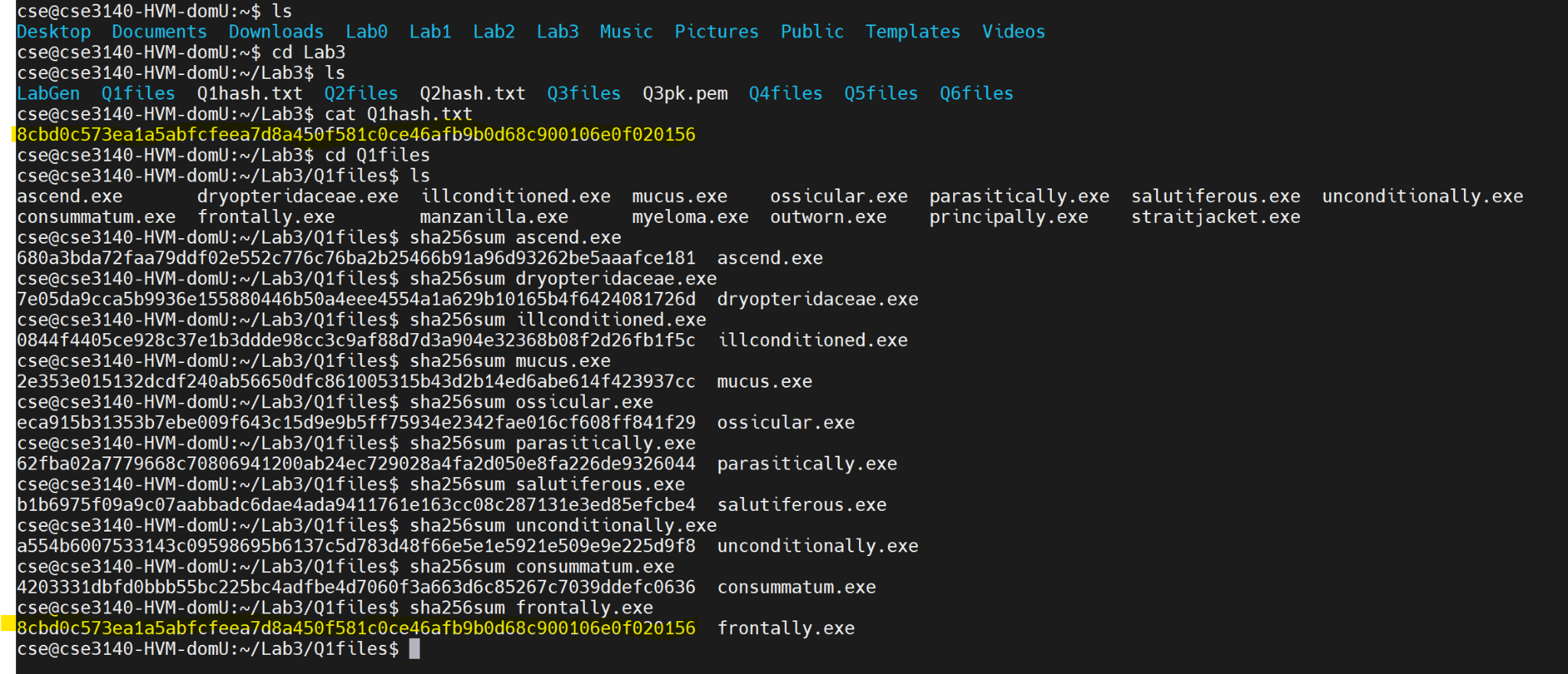
Question 1: [Pepin Name of matching file: frontally.exe]

[Chen Name of matching file: wifeism.exe]

Explanation:

In my VM in the lab3 directory the cat command was run Q1hash.txt to determine the SHA-256 hash function applied to the program file. Once it was obtained in the Q1files sub-directory the sha256sum command was run on each file until its hash was equal to the contents of Q1hash.txt.

Lab3\_Q1\_0.png:



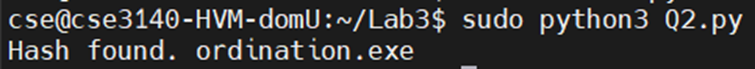
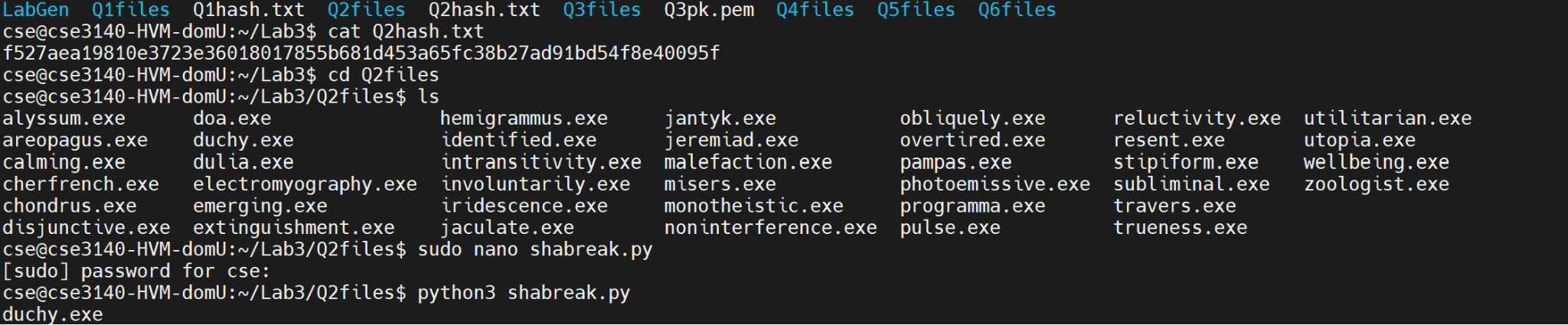
Question 2: [Pepin Name of matching file: duchy.exe]

[Chen Name of matching file: ordination.exe]

Explanation:

In my VM in the lab3 directory the cat command was run Q2hash.txt to determine the SHA-256 hash function applied to the program file. Once it was obtained in the Q2files subdirectory shabreak.py was added and ran to determine which file has the same hash as in Q2hash.txt. The shabreak.py file works by running sha256sum on each file in the same directory as itself and breaks the loop if the same hash is found.

Lab3\_Q2\_0.png:



Q2.py:

import hashlib

import os

# The target SHA-256 checksum

target\_hash = "f527aea19810e3723e36018017855b681d453a65fc38b27ad91bd54f8e40095f"

# Directory containing the files

directory = os.getcwd()

for filename in os.listdir(directory):

if os.path.isfile(os.path.join(directory, filename)):

with open(os.path.join(directory, filename), "rb") as file:

# Calculate the SHA-256 checksum

sha256 = hashlib.sha256()

while True:

data = file.read(65536) # Read the file in 64k chunks

if not data:

break

sha256.update(data)

file\_hash = sha256.hexdigest()

# Check if the checksum matches the target checksum

if file\_hash == target\_hash:

print(filename)

break

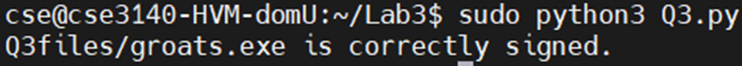
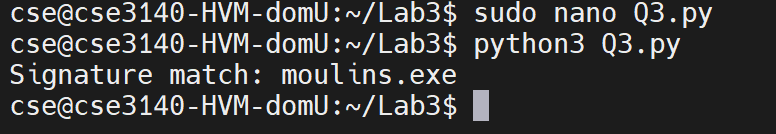
Question 3: [Pepin Name of matching file: moulins.exe]

[Chen Name of matching file: groats.exe]

Explanation:

The code utilizes the RSA.import\_key function to load the contents of the "Q3pk.pem" file, interpreting it as an RSA key. It subsequently scans the files within the "Q3 Files" directory, calculating the SHA256 hash of each file's data and verifying whether the "Q3pk.pem" key was employed in signing or encrypting those files, effectively confirming their authenticity.

Lab3\_Q3\_0:



Q3.py:

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

from Crypto.Hash import SHA256

import os

with open("Q3pk.pem", "rb") as public\_key\_file:

public\_key = RSA.import\_key(public\_key\_file.read())

folder = "Q3files"

for filename in os.listdir(folder):

if filename.endswith(".sign"):

program\_file = os.path.join(folder, filename.replace(".sign", ""))

signature\_file = os.path.join(folder, filename)

with open(signature\_file, "rb") as sign\_file:

signature = sign\_file.read()

with open(program\_file, "rb") as file:

program\_data = file.read()

h = SHA256.new(program\_data)

try:

verifier = PKCS1\_v1\_5.new(public\_key)

if verifier.verify(h, signature):

print(f"{program\_file} is correctly signed.")

break

except (ValueError, TypeError):

pass

else:

print("No correctly signed files found.")

Experiment:

We wrote 3 python scripts to conduct this experiment. The first one generated keys from lengths 1024 to 2048 and used the time module to find out how long “key = RSA.generate(key\_length)” took to run. The next tested how long it would take to sign and verify a file using the same 1024 to 2048 length keys. The last is similar to the previous python script, but it doesnt use hashing. The results are that generating longer keys took more time than generating shorter keys, signing a file with hashing is faster than without hashing since it reduces the amont of data that needs to be processed during the signature operation, signing with longer keys took more time than signing with shorter keys, and finally verification without hashing was faster than verification with hashing.

Question 4: [Pepin plaintext: landside89!]

[Chen plaintext: declaratory56&]

Explanation:

In order to decrypt the given ciphertext, the following python script uses the AES encryption key found in the found encryption python script, ‘R4.py’, in order to decrypt the file using the PyCryptodome module. The initialization vector (IV) is found in the encrypted file “Encrypted” as the first 16 bytes, while the rest is the actual encrypted data. It the uses the encryption key, IV, and decrypted data to find the original data using cypher.decrypt. The decrypted data is then printed to the console and saved to a new file.

Lab3\_Q4\_0:



D4.py:

from Crypto.Cipher import AES

from Crypto.Util.Padding import unpad

variable = b'\xd9\xfeFK\xdb\xb2\xaa\xd18o\xb6-\xf1\xf2\xed\xbf'

with open('Encrypted4', 'rb') as encrypted\_file:

iv = encrypted\_file.read(16)

encrypted\_data = encrypted\_file.read().strip()

cipher = AES.new(variable, AES.MODE\_CBC, iv=iv)

decrypted\_data = unpad(cipher.decrypt(encrypted\_data), AES.block\_size).strip()

decrypted\_string = decrypted\_data.decode('utf-8')

print(f'{decrypted\_string} is the plaintext')

with open('decrypted\_plaintext.txt', 'wb') as output\_file:

output\_file.write(decrypted\_data)

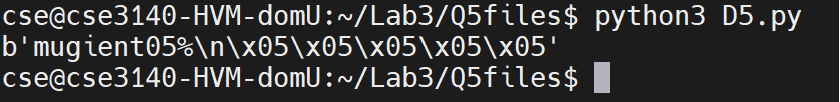
Question 5: [Pepin Name of matching file: mugient05%]

[Chen Name of matching file: unhinge99%]

Explanation:

Similar to the previous question the following Python script decrypts the given ciphertext. It opens and reads an encrypted file, "Encrypted5”. This time however R5.py is obfuscated meaning written intentionally in a way to make it difficult to understand. By reading thought the R5.py file, we found that the key is stored in bird, so we used that as the key and reused most of the previous question’s code.

Lab3\_Q5\_0:





D5.py:

from Crypto.Cipher import AES

from Crypto.Util.Padding import unpad

from Crypto.Hash import MD5

BLOCKSIZE = 4096

h = MD5.new()

count = 0

with open( 'R5.py' , 'rb') as afile:

buf = afile.read(BLOCKSIZE)

while len(buf) > 0:

count = count + 1

h.update(buf)

buf = afile.read(BLOCKSIZE)

hf = h.digest()

bird = hf

with open('Encrypted5', 'rb') as encrypted\_file:

iv = encrypted\_file.read(16)

encrypted\_data = encrypted\_file.read()

cipher = AES.new(bird, AES.MODE\_CBC, iv=iv)

decrypted\_data = unpad(cipher.decrypt(encrypted\_data), AES.block\_size).strip()

decrypted\_string = decrypted\_data.decode('utf-8')

print(f'{decrypted\_string} is the plaintext')

with open('decrypted\_plaintext.txt', 'wb') as output\_file:

output\_file.write(decrypted\_data)

Question 6: [Pepin Name of approval code: 062RFG]

[Chen Name of approval code: WF4J7X]

Explanation:

KG6.py generates a keypair of a public key and a private key and are saved in e.key and d.key respectively. Next, in R6.py a random shared key is generated *k* which is then encrypted with the previously generated public key, the output is another file *EncryptedSharedKey*. The folder is then searched and any files with a .txt extension will be encrypted with the shared key and replaced with a .encrypted version. In AD6.py, the script takes the file *EncryptedSharedKey* from the R6.py file and decripts it with the private key and saves it in *DecryptedSharedKey*. Finally, the last Python file, D6.py, is a victims decryption program which requires the *DecryptedSharedKey* files and decrypts all the files in the current directory using it. We also generated a key size 2048 since it was large enough to have a good amount of security since nobody should be able to guess it and computers would take way to long to try to figure it out, and it wasn’t that large that it would take a significant amount of time to generate.

Lab3\_Q6\_0:

KG6.py:

from Crypto.PublicKey import RSA

key = RSA.generate(2048)

public\_key = key.publickey()

private\_key = key

with open("e.key", "wb") as public\_key\_file:

public\_key\_file.write(public\_key.export\_key())

with open("d.key", "wb") as private\_key\_file:

private\_key\_file.write(private\_key.export\_key())

print("Public and private keys have been saved to e.key and d.key files.")

R6.py:

from Crypto.PublicKey import RSA

from Crypto.Cipher import AES, PKCS1\_OAEP

from Crypto.Random import get\_random\_bytes

import os

public\_key\_data = """

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzcF6Nk8/lu5nwbd+W3hD

kS2b3t+mA5TrRTola8UCRPXVrW+uPBvORau9gUmC8GLyZYnkhHmMH4YlIFUe8dNo

Ajo3mumF+fWsclTKjInnF5DsudC+9uGM9aNY3goI5t+Af+VQ4oILcPUDlgnHVSRP

d67batl2GkD9uIW2vqfUs/jZPiKvjmNRBRRJZidd9+kFM+o+YvZN3HAtvcLhJzim

1f05sJYPZncBrxCFwURddzV+tyGva6dZFEdq9FnKVIOgEYR4olGNJOw0YkkmXY9/

GJTf9fG2/3cRzOnfE+5v/NFWIWkQlY2C//X2TmwPA6QWnSLtrovMTuqC0lKF5KWd

PQIDAQAB

-----END PUBLIC KEY-----

"""

shared\_key = get\_random\_bytes(16)

print(shared\_key)

public\_key = RSA.import\_key(open("e.key").read())

cipher\_rsa = PKCS1\_OAEP.new(public\_key)

encrypted\_shared\_key = cipher\_rsa.encrypt(shared\_key)

with open("EncryptedSharedKey", "wb") as key\_file:

key\_file.write(encrypted\_shared\_key)

def encrypt\_file(file\_path, shared\_key):

cipher = AES.new(shared\_key, AES.MODE\_EAX)

with open(file\_path, "rb") as f:

plaintext = f.read()

cipheredtext, tag = cipher.encrypt\_and\_digest(plaintext)

encrypted\_file\_path = file\_path + ".encrypted"

with open(encrypted\_file\_path, "wb") as f:

f.write(cipher.nonce)

f.write(tag)

f.write(cipheredtext)

os.remove(file\_path)

for filename in os.listdir():

if filename.endswith(".txt"):

encrypt\_file(filename, shared\_key)

print("Files encrypted with shared key and EncryptedSharedKey saved.")

AD6.py:

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

private\_key\_data = """

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEAzcF6Nk8/lu5nwbd+W3hDkS2b3t+mA5TrRTola8UCRPXVrW+u

PBvORau9gUmC8GLyZYnkhHmMH4YlIFUe8dNoAjo3mumF+fWsclTKjInnF5DsudC+

9uGM9aNY3goI5t+Af+VQ4oILcPUDlgnHVSRPd67batl2GkD9uIW2vqfUs/jZPiKv

jmNRBRRJZidd9+kFM+o+YvZN3HAtvcLhJzim1f05sJYPZncBrxCFwURddzV+tyGv

a6dZFEdq9FnKVIOgEYR4olGNJOw0YkkmXY9/GJTf9fG2/3cRzOnfE+5v/NFWIWkQ

lY2C//X2TmwPA6QWnSLtrovMTuqC0lKF5KWdPQIDAQABAoIBAD6ZZbhORheCcxjR

3eSwY0YxLSISwyyFy0qllSsDplwauyboOodZIlm6FY7XMbo3MG5KFLdlT4drxbHf

k+mZGyTKV46PC87vusaSG4nEjwVxd0RVSTRNjRhS2T+8p9auGydOOlyfmiGEudGV

KgsyX+cW0PTiMndsXIUeUWoSH7mdKQKnEURzxU7YNlopi3QAibZbTMYL/zNSojbh

53d/aSOyjr3Jy9Obb6QvIA3UWY5pl9DRNqQJgrA1nw1Qawde51kUtDuO+nw4PrK1

g5KfQ6mL1UTRk3jtdI9TOJVRtOSj55ep/Bp8x78mtmRq6UEgJAKQe/WlEQkpq02V

dGIZF5UCgYEAz0sBMR79T8QLZY4com93zAIA5xjHV1EoeDSjZqxFLY43/kF52eZ/

6bbRWWrGvxK3tO0VAibx5bpkNI3sD6YCqBoxGFfu3ieWevp7aIPJs8gir/Ht8VX4

Zma9T5ZuaEV3eV+6Td9gBNVi+QW5+Ur73u+ma/GNhWeQbU172JGx41cCgYEA/hoB

vmFdYUu/rbjYFkXsNqDknU9i4aaQwc2hlgPek4WvTBhFpVl1CueB68HpJ5G1fpJ7

NAmva3IfuFiYRheKtE3PQdUskOEUT02ifuYF9e4KVX9mqx34+2CHE+gJJk6255x3

pSMRb69C0JBB2WB1uCZsxaffXUeIqJaV3YpzG4sCgYAz9vyEgN1FmsK0oayB9+mT

JcluJWMLe5fmkmOIAJWS9v0IaweN3V+hiifu/3p1Oc79txU2JecsDM7D8fQEdDxk

QbSfAqQp4sixaYt4RtlQjVJMwxxADKopmYXJps9kqk1qCtpcXSvn6kpWXN4jJW4A

lBknJDWzBKRLEkqCnvt2HwKBgH4IdDjznhd2NNp7xmI1yFXIM9x61bbKoR4fWjhI

LwvI0QvTNigvWMhRz7UZ9wUzVmwzR3ymGSYX8vfCqbSKVLfdqCElFPe/TA2RmjmK

6gXzTZjemhPhx+8XD6il/HPNsxbqGsYDwNA+g8ti4eZp27m9BVJ8U2O26WeFWUwL

MdDFAoGBAJFBe7Ymlz/+xXbI8vFyYl47u8Mo9kQ0DuurxUFpzUd/JZ6VK5DYCWYk

fl0tocsXsWxFVpxhPDreJDSzydWyhhenqynaHPwQiP75uYAkyZ4sDUYo/aGKxtyL

+Q+ZZ7eauaBhCQ5Nm3EdHyoiY8VXClAb+tpJcTJCMkdPL6eNq3mh

-----END RSA PRIVATE KEY-----

"""

def decrypt\_shared\_key(encrypted\_key, private\_key\_data):

private\_key = RSA.import\_key(open('d.key').read())

cipher = PKCS1\_OAEP.new(private\_key)

shared\_key = cipher.decrypt(encrypted\_key)

return shared\_key

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) != 2:

print("Usage: python AD6.py EncryptedSharedKey")

else:

encrypted\_key\_path = sys.argv[1]

with open(encrypted\_key\_path, "rb") as key\_file:

encrypted\_shared\_key = key\_file.read()

shared\_key = decrypt\_shared\_key(encrypted\_shared\_key, private\_key\_data)

print("Decrypted Shared Key (k):", shared\_key)

with open("DecryptedSharedKey", "wb") as key\_file:

key\_file.write(shared\_key)

D4.py:

from Crypto.Cipher import AES

import os

import sys

def decrypt\_file(file\_path, shared\_key):

encrypted\_file\_path = file\_path + ".encrypted"

if not os.path.exists(encrypted\_file\_path):

print(f"Encrypted file not found: {encrypted\_file\_path}")

return

with open(encrypted\_file\_path, "rb") as f:

nonce = f.read(16)

tag = f.read(16)

ciphertext = f.read()

cipher\_aes = AES.new(shared\_key, AES.MODE\_EAX, nonce=nonce)

try:

plaintext = cipher\_aes.decrypt\_and\_verify(ciphertext, tag)

with open(file\_path, "wb") as f:

f.write(plaintext)

os.remove(encrypted\_file\_path)

print(f"Decrypted {file\_path}")

except ValueError:

print(f"Decryption failed for {file\_path}. Incorrect key or file tampering.")

if \_\_name\_\_ == "\_\_main\_\_":

if len(sys.argv) != 2:

print("Usage: python D6.py DecryptedSharedKey")

else:

shared\_key\_path = sys.argv[1]

with open(shared\_key\_path, "rb") as key\_file:

shared\_key = key\_file.read()

for filename in os.listdir():

if filename.endswith(".encrypted"):

decrypt\_file(filename[:-10], shared\_key)

print("Decryption complete.")