CS2413 – Assignment 1

Candice Inman

Partner: Chad Billings

2025.10.29

**Scenario 1**

**Question 1**

from Crypto.Hash import SHA256, MD5, HMAC

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

from Crypto.Cipher import AES

def key\_generation(password: str, shamd5: str) -> bytes:

# creating the AES Encryption key with your choice of either SHA256 or MD5

if shamd5.lower() == 'y': # use SHA256

return SHA256.new(password.encode('utf-8')).digest()

elif shamd5.lower() == 'n': # use MD5

return MD5.new(password.encode('utf-8')).digest()

else:

print("Error in input, neither SHA-256 or MD5 selected")

raise SystemExit(1)

def \_hash\_module(shamd5: str):

return SHA256 if shamd5.lower() == 'y' else MD5

def \_mac\_key(password: str, shamd5: str) -> bytes:

# create a MAC key from the provided password

Hash = \_hash\_module(shamd5)

return Hash.new(password.encode('utf-8') + b"|MAC|").digest()

def \_tag\_len(shamd5: str) -> int:

return 32 if shamd5.lower() == 'y' else 16

def encryption(message: str, password: str, shamd5: str) -> bytes:

# Encrypts the message with the provided AES key and Mac key.

key = key\_generation(password, shamd5)

mac\_key = \_mac\_key(password, shamd5)

Hash = \_hash\_module(shamd5)

iv = get\_random\_bytes(16)

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

ct = cipher.encrypt(pad(message.encode('utf-8'), 16))

tag = HMAC.new(mac\_key, iv + ct, digestmod=Hash).digest()

return iv + ct + tag

def decryption(blob: bytes, password: str, shamd5: str) -> str:

# Encrypts the message with the provided AES key and Mac key

key = key\_generation(password, shamd5)

mac\_key = \_mac\_key(password, shamd5)

Hash = \_hash\_module(shamd5)

tag\_len = \_tag\_len(shamd5)

if len(blob) < 16 + tag\_len:

raise ValueError("Ciphertext too short")

iv = blob[:16]

tag = blob[-tag\_len:]

ct = blob[16:-tag\_len]

# verifies integrity before decrypting/unpadding, both different password or altered text file

hmac\_obj = HMAC.new(mac\_key, iv + ct, digestmod=Hash)

hmac\_obj.verify(tag)

print("[INFO] Hash verified! Decrypting message...")

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

pt = unpad(cipher.decrypt(ct), 16)

return pt.decode('utf-8')

def main():

print("AES Encryption & Decryption with Integrity Check")

EorD = input("Do you want to (E)ncrypt or (D)ecrypt a message? ")

username = input("Enter your username: ")

password = input("Enter your password: ")

YorN = input("Use SHA-256 (Y) or MD5 (N)? ")

if EorD.lower() == 'e':

message = input("Enter message to encrypt: ")

try:

blob = encryption(message, password, YorN)

with open("encrypted\_output.txt", "w", encoding="utf-8") as f:

f.write(blob.hex())

print("[INFO] Encrypted message stored successfully")

except Exception as e:

print("[ERROR] Encryption failed:", e)

elif EorD.lower() == 'd':

try:

with open("encrypted\_output.txt", "r", encoding="utf-8") as f:

blob\_hex = f.read().strip()

blob = bytes.fromhex(blob\_hex)

except Exception as e:

print("[ERROR] Could not read encrypted\_output.txt:", e)

return

try:

plaintext = decryption(blob, password, YorN)

print("Decrypted message:", plaintext)

except ValueError as e:

print("[ERROR] Integrity check failed (wrong password or tampered data):", e)

except Exception as e:

print("[ERROR] Decryption failed:", e)

else:

print("ERROR in input, program execution terminated")

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

main()

A computer screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a computer

AI-generated content may be incorrect.A computer screen with white text

AI-generated content may be incorrect.A computer screen shot of a computer

AI-generated content may be incorrect.

**Scenario 2**

**Question 2**

The hash value that matches the private key is Option D, as can be seen in the screenshot below:

from Crypto.Hash import SHA256

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

import base64

with open("private\_key.pem", "rb") as f:

private\_key = f.read()

#print(private\_key[:50])

sha256\_hash = SHA256.new(private\_key).hexdigest()

print("\nSHA-256 hash from file: ", sha256\_hash, "\n")

hash\_options = {

"A": "1a4c8b9d847b3e2fa2d5f9d31c8e5f8b7c91a5d0f4e5b2f7d8c6e2a9b4d5e1c3",

"B": "9e427f6bea8af1fc9d2d332312338cf538759ebe5f71843af205c18d726623f9",

"C": "3f4b5c2d9a8e7f1c5d3b2a9c6e1d4f7b8a2c5e9f1d0b3a7c8e6d2f5b9a4c3e1d",

"D": "0d09a513353e632b068a1a49e6ecc0b2c753ccc1c95cb1751745ba576d1396c8"

}

print("Options:")

for key, value in hash\_options.items():

print(f"{key}: {value}")

if sha256\_hash.lower() == value.lower():

print(f"\nOption {key} is correct!\n")

A screenshot of a computer program

AI-generated content may be incorrect.

**Question 3**

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

import base64

with open("private\_key.pem", "rb") as f:

private\_key = RSA.import\_key(f.read())

with open("encrypted\_message.txt", "r") as f:

encrypted\_data = base64.b64decode(f.read().strip())

print("\nEncrypted cipher text loaded from file:\n", encrypted\_data)

rsa\_cipher = PKCS1\_OAEP.new(private\_key)

decrypted\_data = rsa\_cipher.decrypt(encrypted\_data)

print("\nDecrypted message: \n", decrypted\_data.decode(), "\n")

A computer screen with white text

AI-generated content may be incorrect.