Lab 3 | CSE 3140 | Abdul Chowdhury (unable to communicate with partner asked TA said to work by myself) | amc20031 | ssh -L 127.0.0.1:8000:10.13.4.8:80 cse@10.13.6.41

Q1: First I ran Q1hash.text which gave me the file sum, then I went into the Q1 Files used the sha256sum command to check every file to see which sum would match the file sum, unfortunately my file was second to last so I spent a lot of time checking and finally got series.exe

```
cse@cse3140-HVM-domU:~/Lab3/Q1files$ cd
cse@cse3140-HVM-domU:~$ cd Lab3
cse@cse3140-HVM-domU:~/Lab3$ cat Q1hash.txt
37895d73546c363875c3c9a54b2db4a2b70ae0113327c4033e2d6b17f019a752
cse@cse3140-HVM-domU:~/Lab3$ cd Q1files
cse@cse3140-HVM-domU:~/Lab3/Q1files$ ls
absurdly.exe       cortina.exe    elaborately.exe  icepail.exe       secular.exe
authorization.exe  current.exe    ersatz.exe       lettered.exe      series.exe
blacken.exe        dissemble.exe  forum.exe        martinihenry.exe  tetragrammaton.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum series.exe
37895d73546c363875c3c9a54b2db4a2b70ae0113327c4033e2d6b17f019a752  series.exe
```

Q2: This python script (Q2.py) scans the Q2files in lab3 directory for all .exe files and computes their SHA-256 hash using sha256sum. It reads the expected hash from Q2hash.txt and compares it with the computed hashes. Once match is found it gets printed out

My match was: springiness.exe

```
cse@cse3140-HVM-domU:~/Lab3$ python3 Q2.py
Match found: springiness.exe
```

```python
import os
import subprocess

def compute_hash(file_name):  # Computes the SHA-256 hash of a given file
    os.chdir('/home/cse/Lab3/Q2files')
    output = subprocess.run(['sha256sum', file_name], stdout=subprocess.PIPE)
    hash_value = output.stdout.split()
    return hash_value

def main():
    executables = [f for f in os.listdir('/home/cse/Lab3/Q2files') if f.endswith(".exe")]  # Get all

    os.chdir('/home/cse/Lab3')
    with open('Q2hash.txt', 'r') as hash_file:
        expected_hash = hash_file.read().strip()  # Read expected hash

    for exe in executables:
        computed_hash, exe_name = compute_hash(exe)
        if computed_hash == expected_hash.encode():  # Ensure proper comparison
            print(f'Match found: {exe_name.decode()}')

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

Q3: This python script (Q3.py) verifies the executable file in Q3files is correctly using RSA digital signatures with SHA-256, a public key is loaded from Q3pk.pem, then computes the SHA-256 hash of each executable, verifying the .sign using PKCS#1. The correct signed file is printed upon verification. In this experiment I discovered a lot like key generation, signing, and verification times for different key lengths.

My match was: emotion.exe

```
cse@cse3140-HVM-domU:~/Lab3$ python3 Q3.py
Success: /home/cse/Lab3/Q3files/emotion.exe
```

```
import os
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA

def load_public_key(key_path):
    """Loads the RSA public key from a given file."""
    with open(key_path, 'r') as key_file:
        return RSA.importKey(key_file.read())

def verify_signature(executable_path, signature_path, verifier):
    """Verifies whether the signature matches the executable."""
    with open(executable_path, 'rb') as exe_file:
        hashed_data = SHA256.new(exe_file.read())

    with open(signature_path, 'rb') as sig_file:
        signature = sig_file.read()

    return verifier.verify(hashed_data, signature)

def main():
    key = load_public_key('/home/cse/Lab3/Q3pk.pem')
    verifier = PKCS1_v1_5.new(key)  # RSA verifier

    directory = '/home/cse/Lab3/Q3files'

    for entry in os.scandir(directory):
        if entry.path.endswith('.sign'):
            exe_path = entry.path[:-5]  # Remove '.sign' to get executable
path

            if verify_signature(exe_path, entry.path, verifier):
                print(f'Success: {exe_path}')

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

Q4: This python script (D4.py) decrypted file Encrypted4 using AES in CBC mode. Initially it reads the first 16 bytes as the IV and the rest as encrypted text, then loads decryption key from.key.txt, using AES cipher it decrypts text, removes padding and then prints decrypted content

```
cse@cse3140-HVM-domU:~/Lab3$ python3 D4.py
content: b'diluent93@\n'
```

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

with open('/home/cse/Lab3/Q4files/Encrypted4', 'rb') as et:
    iv = et.read(16)
    encrypted_text = et.read()

with open('/home/cse/Lab3/Q4files/.key.txt', 'rb') as key_file:
    key = key_file.read()

cipher = AES.new(key, AES.MODE_CBC, iv)
content = unpad(cipher.decrypt(encrypted_text), AES.block_size)

print('content:', content)
```

Q5: This python script (D5.py) decrypts the file Encrypted5 using AES in CBC mode. Initially generates a decryption key by computing MD5 hash of R5.py. The first 16 bytes of Encrypted5 is read as the IV and the rest as encrypted stuff. Using AES cipher, content is decrypted and padding is removed before printing the decrypted message.

```
cse@cse3140-HVM-domU:~/Lab3$ python3 D5.py
ginkgoaceae39!
```

```python
from Crypto.Cipher import AES
from Crypto.Hash import MD5
from Crypto.Util.Padding import unpad

with open("/home/cse/Lab3/Q5files/R5.py", "rb") as f:
    newf = MD5.new(f.read()).digest()

with open("/home/cse/Lab3/Q5files/Encrypted5", "rb") as file_2:
    iv = file_2.read(16)
    encoded = file_2.read()

msg_decrypt = unpad(AES.new(newf, AES.MODE_CBC, iv).decrypt(encoded), AES.block_size)

print(msg_decrypt.decode())
```

Q6: Video posted on huskyct

KG6.py: Python script generates public and private key pairs and it gets sent to e.key, d.key

```python
import os
from Crypto.PublicKey import RSA

# Generate a 2048-bit RSA key pair
print("Generating RSA key pair...")
private_key = RSA.generate(2048)  # Generate 2048-bit key pair

# Export the private and public keys
private_key_bytes = private_key.export_key()  # Private key in bytes
public_key_bytes = private_key.publickey().export_key()  # Public key in bytes

# Define the directory path
key_directory = '/home/cse/Lab3/Q6files/Solutions/'

# Ensure the directory exists
os.makedirs(key_directory, exist_ok=True)
print(f"Directory {key_directory} ensured.")

# Write the keys to the specified files in the directory
with open(os.path.join(key_directory, 'e.key'), 'wb') as public_key_file:
    public_key_file.write(public_key_bytes)
    print("Public key saved to e.key")

with open(os.path.join(key_directory, 'd.key'), 'wb') as private_key_file:
    private_key_file.write(private_key_bytes)
    print("Private key saved to d.key")
```

Public Key:

```
cse@cse3140-HVM-domU:~/Lab3/Q6files/Solutions$ cat e.key
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsoAzl7MZa0Ov4kS2xIgT
aKwPyfsa0ToTdFmBInAsnLh3ygBEI0oUhglomKRqNnejjx3zIvLkhSc7yU1DVih3
6imB7YslygQXIENgDbYntbyGTO+X2s0lm5c4zgKO3WgYLADCjSD2WuVF/ljgzqZ1
INzGfc8TyU6dDB213a0GuJItUibE69wQU0VFhp3HI5Z0+9fBMEYu6S6wzaXIV5P8
kQRZKduJTJz+0gCQINeHfUNNxFb4eATimyPpa6M0wDqrDADI++9Meey3Xqxuf+xW
RszfGPsxJLasPpF4Cvb0Hs2Zk1rKlxe7fu3f2jrpLReAe9fxGqueN3mwMa5Q9cu3
CwIDAQAB
-----END PUBLIC KEY-----cse@cse3140-HVM-domU:~/Lab3/Q6files/Solutions$
```

Private Key:

```
cse@cse3140-HVM-domU:~/Lab3/Q6files/Solutions$ cat d.key
-----BEGIN RSA PRIVATE KEY-----
MIIEoQIBAAKCAQEAsoAzl7MZa0Ov4kS2xIgTaKwPyfsa0ToTdFmBInAsnLh3ygBE
I0oUhglomKRqNnejjx3zIvLkhSc7yU1DVih36imB7YslygQXIENgDbYntbyGTO+X
2s0lm5c4zgKO3WgYLADCjSD2WuVF/ljgzqZ1INzGfc8TyU6dDB213a0GuJItUibE
69wQU0VFhp3HI5Z0+9fBMEYu6S6wzaXIV5P0kQRZKduJTJz+0gCQINeHfUNNxFb4
eATimyPpa6M0wDqrDADI++9Meey3Xqxuf+xWRszfGPsxJLasPpF4Cvb0Hs2ZK1rK
lxe7fu3f2jrpLReAe9fxGqueN3mwMa5Q9cu3CwIDAQABAoH/HqDM96dW4zRkXu5L
CsrnGduU9zsPQayDP3nbXAce9Sytro/mAtvZ4yXh2CmVZJyBy9iXez3m9Q9CrTLq
YTcP5k0NGFp2aqAdKcG6cG20zTEbMPoIxxfTvl9Wwykz/3LmdkhlF3pvAWyYooQe
HYMyppA6SYXbZhStmfKhm/gKBnAjloDnOVdqXwJ+tqw38DVpVr96ooNvIEvoKlcY
VSno7NNv3M+a3JyCr6bDDs9qn7+s8iZ78dgx93Sp0B/tioXUQ/P0PVQcEsxmc/4B
aTD9tnD5rMwbmngJiWCdZPpMKPdiIIqUDjqezUdWeOnpkx/b3AV8PbcnOjZ1jC4b
0TVRAoGBALlN2YZ6KLCb/kUrH64Kh0qRA5LJU24pr7GlrHVNvVEeUw1cCNqKrvo7
LsTj2pZOkZHqM0ddBlDHfb2onUa6obrZXzF1Pt6aRYzaMlB8NpNs7Z2GysoWhwLg
X9ILKGD8HDM+eYa5X+4AqURW298le8ZskuCM9AjpVRLMVGu4QYVdAoGBAPaZ5AAv
z5HSccYgcz4SDN2OlGv+L1edbr9g0Br5lRPtkmc9FTQRBeVJnQ/9Qd9zmEuqlIfT
24xQhZL6tS7dAZwPIDGX26M3m3YxsYiKRQuCjNfxeYGZHw8ZJhvpNGt850ACulQ5
4A+DhOA8NVuIiQxvHgSzM56/vGWz+jpRlr+HAoGBAI1pyd2EkFTnVdE+rbiGNGVF
cbKtjcv+aCRsSMZkgivKhpslZ2RQfeRdEf4T3Rbj8temIWgOp+kxtNXuPchiWVZc
2DbHgSWkFnu9cLRXdtbfRB75ZP0AYBxj2vxSoUgQHfiFajzUb08UlhifKMCMvH7b
EOPo7bt3RgNIiAyE2wvJAoGkdO+Si3gggLB/HQ1kPGnIygMDZJZbErvnuT6KUFBL
nfCI/QWdGVx9JDYdEWjxpKkm7bKMzj/rGwVP4duasQdB7SPLC1lMYIBM6uYznmCa
zxNwADJUgN0AOmOdi5yxK8tFyTfB7ZCoLBMcnkxOrQH4ovnKDUaA0uVuy/dlT+Rg
5d8CgYBt3VzknuQHjztTTLUmiwOpQQ37e4WzPcvGVnKhLhUVaOXbQytzHEpGvV/T
8TOiJjjAlfJ/mEnA+FuEYABYZXHFi8RXUKHmTMgCcohzdbFC2aLkLKfRWtrKZziI
mh/eLEbTgoFpn/yjUYQWlfNO00N9M7E4kWcUfJRG3zyi7c+ZMw==
-----END RSA PRIVATE KEY-----cse@cse3140-HVM-domU:~/Lab3/Q6files/Solutions$
```