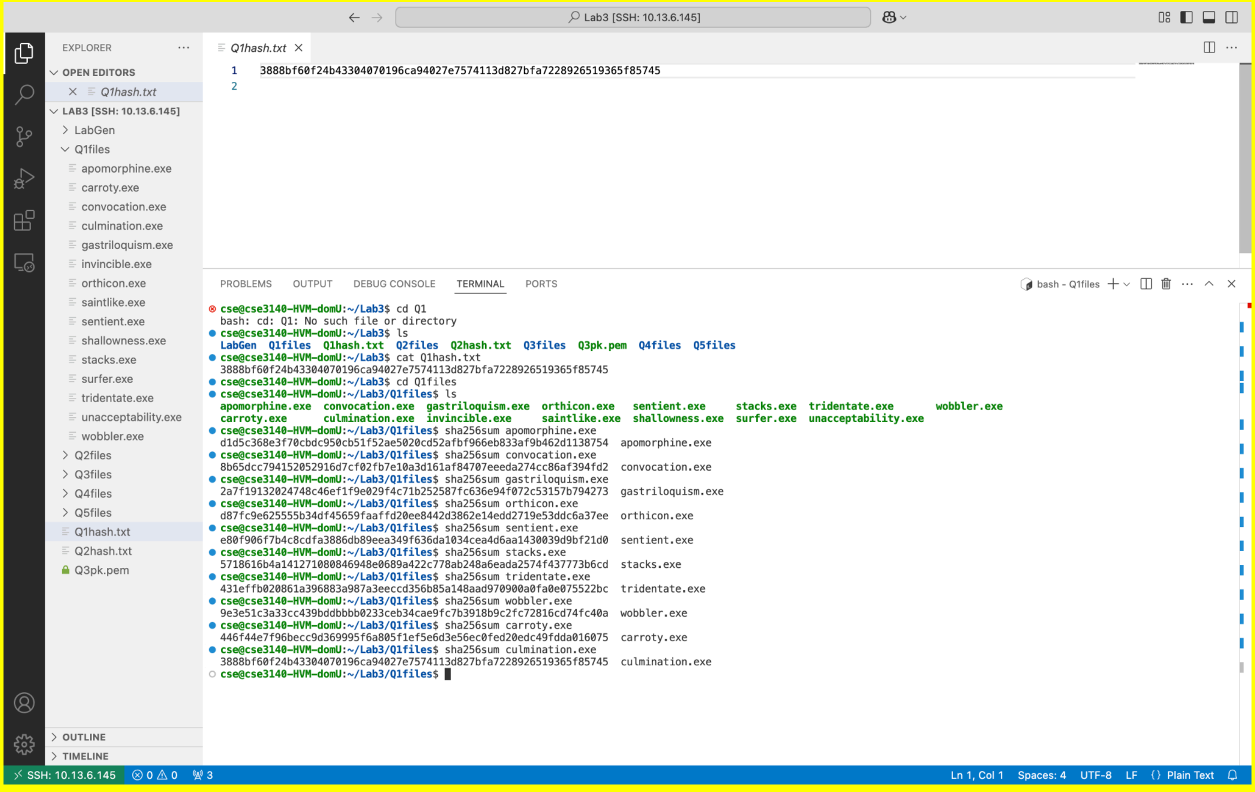
Layla Nassar
Section 003
CSE 3140
IP Address: **10.13.6.145**
NetID: LTN22001
March 11, 2025

CSE 3140 Lab 3 Report: Cryptography, Malware, and Ransomware

Question #1: culmination.exe

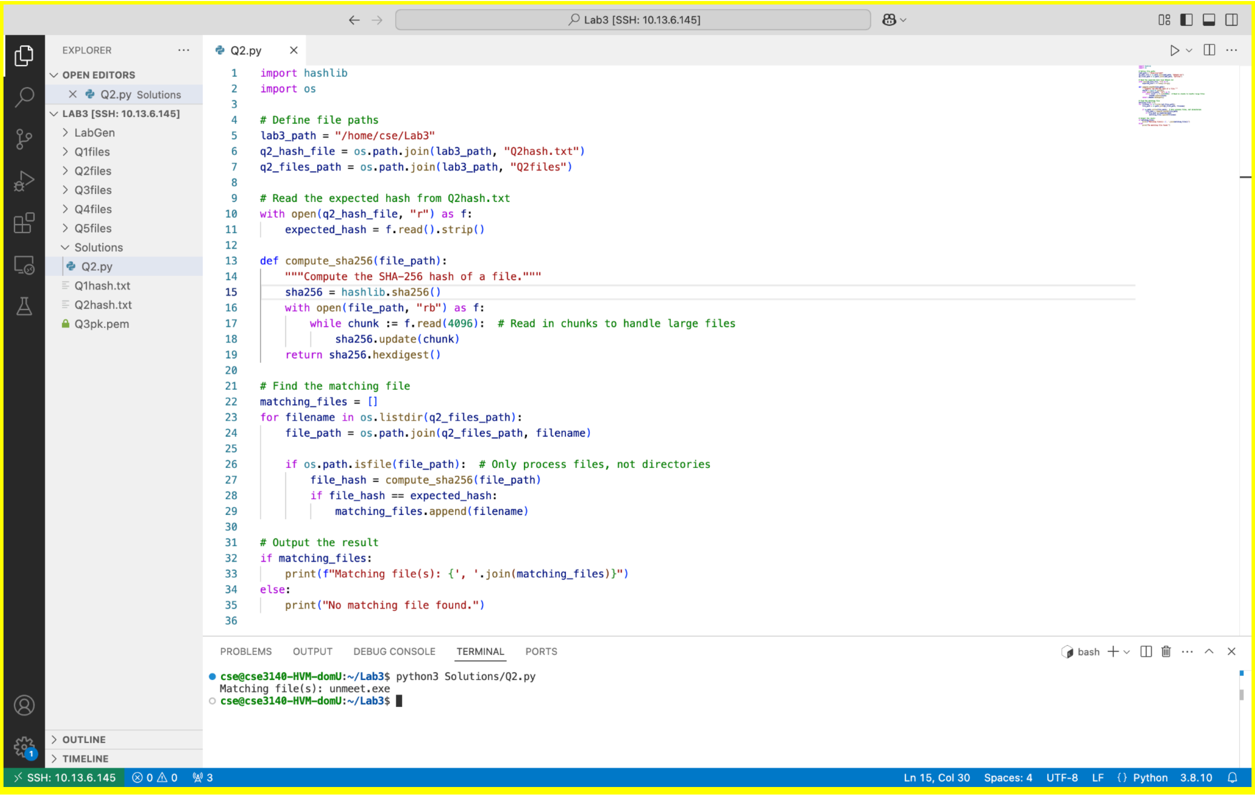


Q1hash.txt

```
3888bf60f24b43304070196ca94027e7574113d827bfa7228926519365f85745

```

```
cse@cse3140-HVM-domU:~/Lab3$ cd Q1
bash: cd: Q1: No such file or directory
cse@cse3140-HVM-domU:~/Lab3$ ls
LabGen  Q1files  Q1hash.txt  Q2files  Q2hash.txt  Q3files  Q3pk.pem  Q4files  Q5files
cse@cse3140-HVM-domU:~/Lab3$ cat Q1hash.txt
3888bf60f24b43304070196ca94027e7574113d827bfa7228926519365f85745
cse@cse3140-HVM-domU:~/Lab3$ cd Q1files
cse@cse3140-HVM-domU:~/Lab3/Q1files$ ls
apomorphine.exe  convocation.exe  gastriloquism.exe  orthicon.exe   sentient.exe     stacks.exe  tridentate.exe       wobbler.exe
carroty.exe      culmination.exe  invincible.exe     saintlike.exe  shallowness.exe  surfer.exe  unacceptability.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum apomorphine.exe
d1d5c368e3f70cbdc950cb51f52ae5020cd52afbf966eb833af9b462d1138754  apomorphine.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum convocation.exe
8b65dcc794152052916d7cf02fb7e10a3d161af84707eeeda274cc86af394fd2  convocation.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum gastriloquism.exe
2a7f19132024748c46ef1f9e029f4c71b252587fc636e94f072c53157b794273  gastriloquism.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum orthicon.exe
d87fc9e625555b34df45659faaffd20ee8442d3862e14edd2719e53ddc6a37ee  orthicon.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum sentient.exe
e80f906f7b4c8cdfa3886db89eea349f636da1034cea4d6aa1430039d9bf21d0  sentient.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum stacks.exe
5718616b4a141271080846948e0689a422c778ab248a6eada2574f437773b6cd  stacks.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum tridentate.exe
431effb020861a396883a987a3eeccd356b85a148aad970900a0fa0e075522bc  tridentate.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum wobbler.exe
9e3e51c3a33cc439bddbbbb0233ceb34cae9fc7b3918b9c2fc72816cd74fc40a  wobbler.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum carroty.exe
446f44e7f96becc9d369995f6a805f1ef5e6d3e56ec0fed20edc49fdda016075  carroty.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ sha256sum culmination.exe
3888bf60f24b43304070196ca94027e7574113d827bfa7228926519365f85745  culmination.exe
cse@cse3140-HVM-domU:~/Lab3/Q1files$ 
```

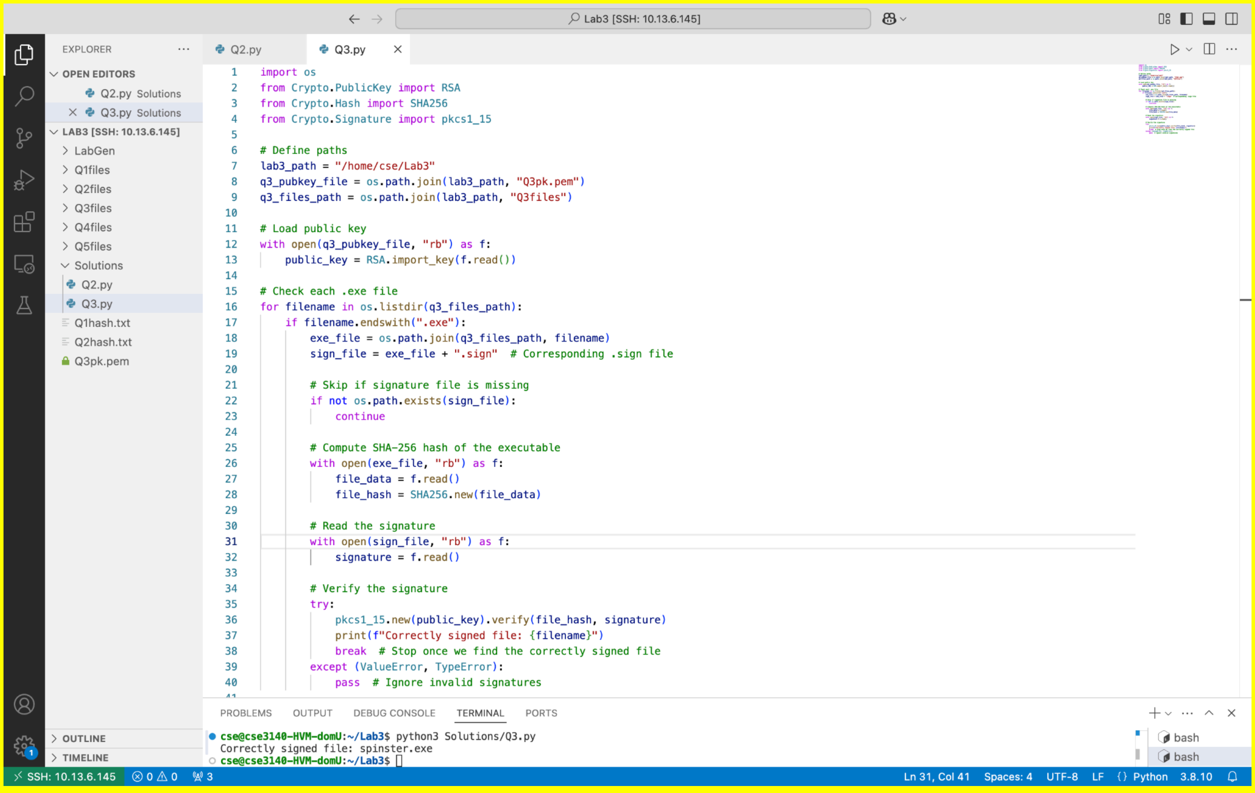
Question #2: unmeet.exe



```python
import hashlib
import os

# Define file paths
lab3_path = "/home/cse/Lab3"
q2_hash_file = os.path.join(lab3_path, "Q2hash.txt")
q2_files_path = os.path.join(lab3_path, "Q2files")

# Read the expected hash from Q2hash.txt
with open(q2_hash_file, "r") as f:
    expected_hash = f.read().strip()

def compute_sha256(file_path):
    """Compute the SHA-256 hash of a file."""
    sha256 = hashlib.sha256()
    with open(file_path, "rb") as f:
        while chunk := f.read(4096):  # Read in chunks to handle large files
            sha256.update(chunk)
    return sha256.hexdigest()

# Find the matching file
matching_files = []
for filename in os.listdir(q2_files_path):
    file_path = os.path.join(q2_files_path, filename)

    if os.path.isfile(file_path):  # Only process files, not directories
        file_hash = compute_sha256(file_path)
        if file_hash == expected_hash:
            matching_files.append(filename)

# Output the result
if matching_files:
    print(f"Matching file(s): {', '.join(matching_files)}")
else:
    print("No matching file found.")
```

```
cse@cse3140-HVM-domU:~/Lab3$ python3 Solutions/Q2.py
Matching file(s): unmeet.exe
cse@cse3140-HVM-domU:~/Lab3$ 
```

Question #3: spinster.exe

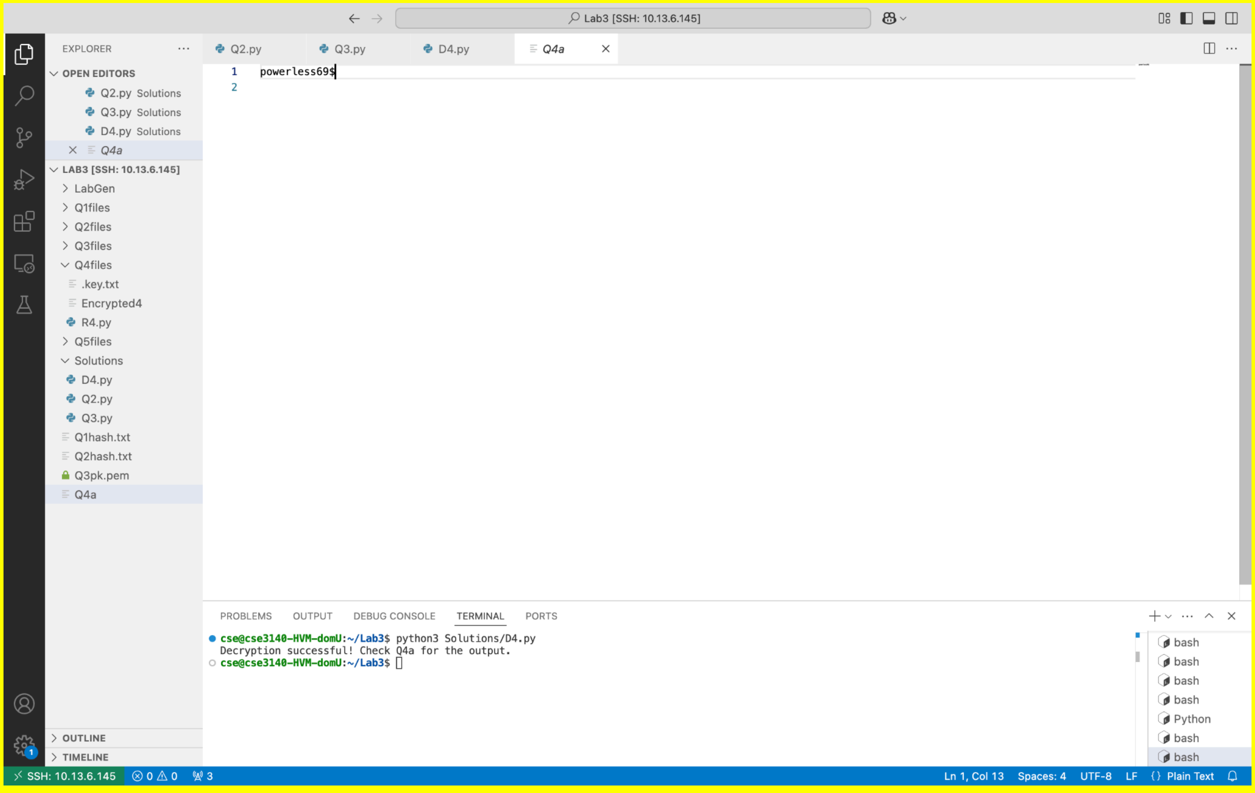


```python
import os
from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from Crypto.Signature import pkcs1_15

# Define paths
lab3_path = "/home/cse/Lab3"
q3_pubkey_file = os.path.join(lab3_path, "Q3pk.pem")
q3_files_path = os.path.join(lab3_path, "Q3files")

# Load public key
with open(q3_pubkey_file, "rb") as f:
    public_key = RSA.import_key(f.read())

# Check each .exe file
for filename in os.listdir(q3_files_path):
    if filename.endswith(".exe"):
        exe_file = os.path.join(q3_files_path, filename)
        sign_file = exe_file + ".sign"  # Corresponding .sign file

        # Skip if signature file is missing
        if not os.path.exists(sign_file):
            continue

        # Compute SHA-256 hash of the executable
        with open(exe_file, "rb") as f:
            file_data = f.read()
            file_hash = SHA256.new(file_data)

        # Read the signature
        with open(sign_file, "rb") as f:
            signature = f.read()

        # Verify the signature
        try:
            pkcs1_15.new(public_key).verify(file_hash, signature)
            print(f"Correctly signed file: {filename}")
            break  # Stop once we find the correctly signed file
        except (ValueError, TypeError):
            pass  # Ignore invalid signatures
```

```
cse@cse3140-HVM-domU:~/Lab3$ python3 Solutions/Q3.py
Correctly signed file: spinster.exe
cse@cse3140-HVM-domU:~/Lab3$ 
```

In this experiment, we used RSA digital signatures with SHA-256 hashing to verify the authenticity of `.exe` files in the `Q3files` directory. The public key from `Q3pk.pem` was used to check the validity of each file's signature. The process involved computing the SHA-256 hash of each `.exe` file and comparing it with its corresponding `.sign` file using PKCS#1 v1.5 verification. The correctly signed file was identified as spinster.exe, confirming that it was signed by the legitimate software vendor. This experiment highlighted the role of digital signatures in ensuring both integrity and authenticity, as the verification process successfully filtered out improperly signed or modified files.

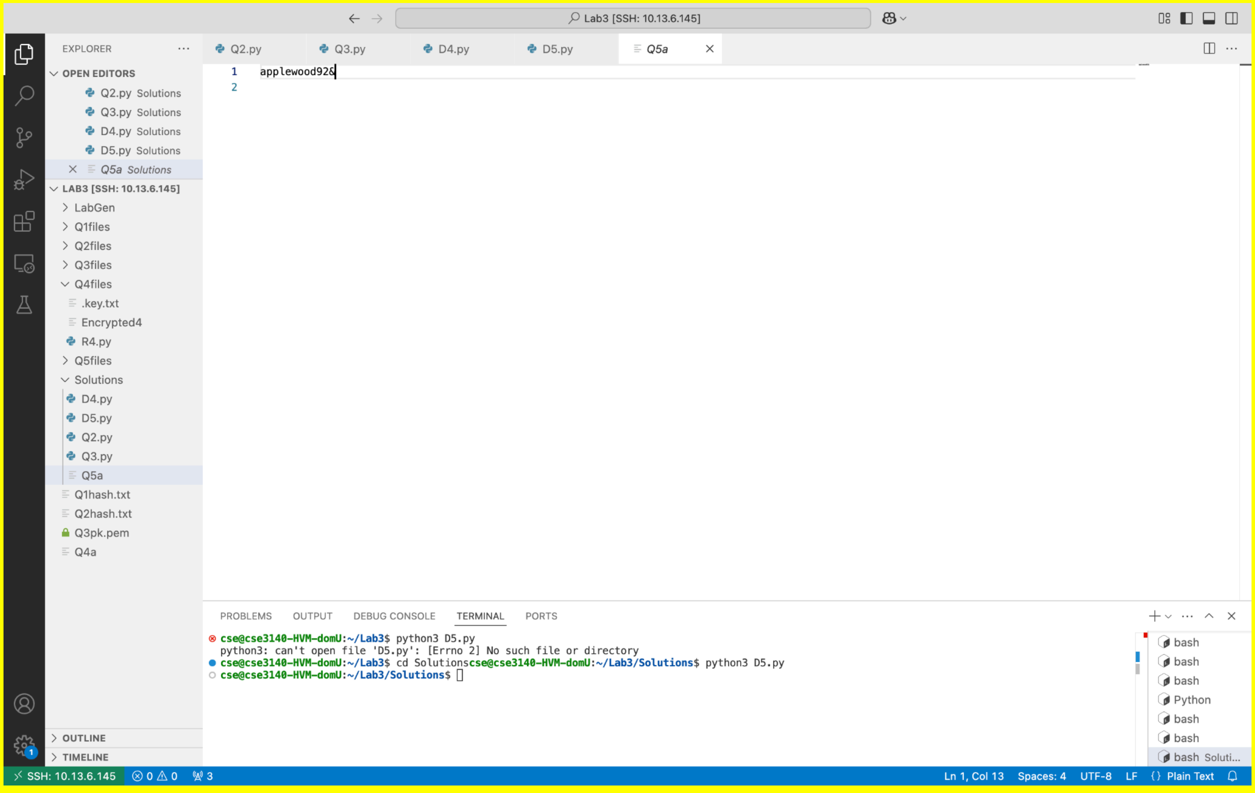
Question #4: powerless69$



```
powerless69$
```

```
cse@cse3140-HVM-domU:~/Lab3$ python3 Solutions/D4.py
Decryption successful! Check Q4a for the output.
cse@cse3140-HVM-domU:~/Lab3$
```

Question #5: applewood92&



```
applewood92&
```

```
cse@cse3140-HVM-domU:~/Lab3$ python3 D5.py
python3: can't open file 'D5.py': [Errno 2] No such file or directory
cse@cse3140-HVM-domU:~/Lab3$ cd Solutionscse@cse3140-HVM-domU:~/Lab3/Solutions$ python3 D5.py
cse@cse3140-HVM-domU:~/Lab3/Solutions$ 
```

Question #6: All files and screen recording located in ZipFile.

For this lab project, I chose to use RSA as the public key cryptosystem because it is widely used for secure encryption and key exchange. RSA ensures that the symmetric key, which encrypts the actual files, can only be decrypted by the private key, making it an effective method for ransomware-like encryption. I selected a key size of 2048 bits because it provides strong security while maintaining reasonable computational efficiency. A smaller key, such as 1024 bits, would be vulnerable to modern cryptographic attacks, while larger keys would increase processing time without significant security benefits.

To test my implementation, I first generated a public and private key pair using my key generation program. Then, I ran my ransomware program, which hardcoded the public key and used it to encrypt a randomly generated symmetric key. This symmetric key was stored in EncryptedSharedKey and was used to encrypt all `.txt` files in the working directory, renaming them with the `.txt.encrypted` extension.

Next, I acted as the attacker and used my decryption script to take EncryptedSharedKey and decrypt it using the private key, recovering the original symmetric key and saving it as DecryptedSharedKey. Finally, I ran the victim's decryption script, which used this decrypted key to restore all encrypted files to their original `.txt` format.

The process worked as expected, demonstrating how ransomware typically operates by encrypting files in a way that requires the attacker's private key for decryption. I recorded my screen while testing the program to document each step, ensuring that the key generation, encryption, and decryption processes functioned correctly.