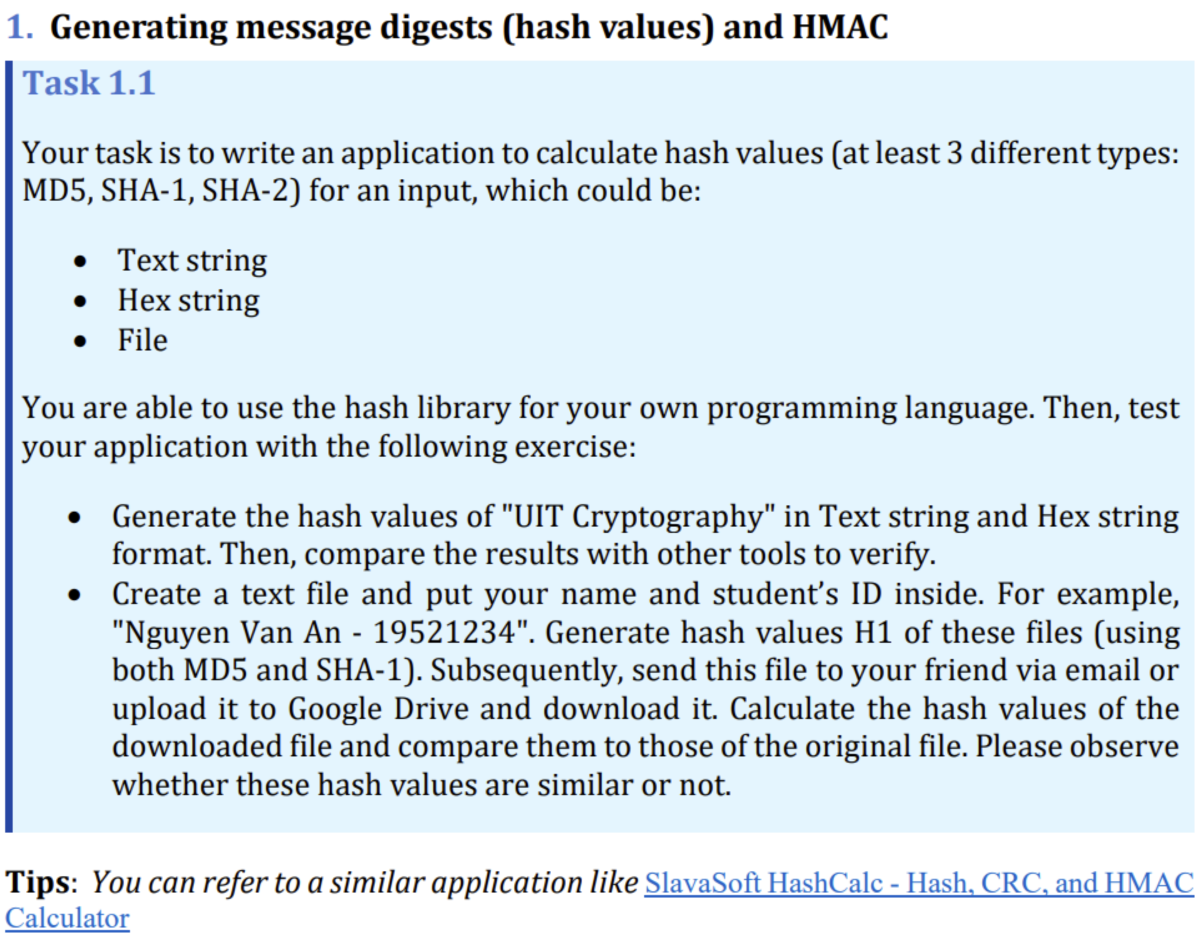
Exercise 1:



1. Generating message digests (hash values) and HMAC

Task 1.1

Your task is to write an application to calculate hash values (at least 3 different types: MD5, SHA-1, SHA-2) for an input, which could be:

- Text string
- Hex string
- File

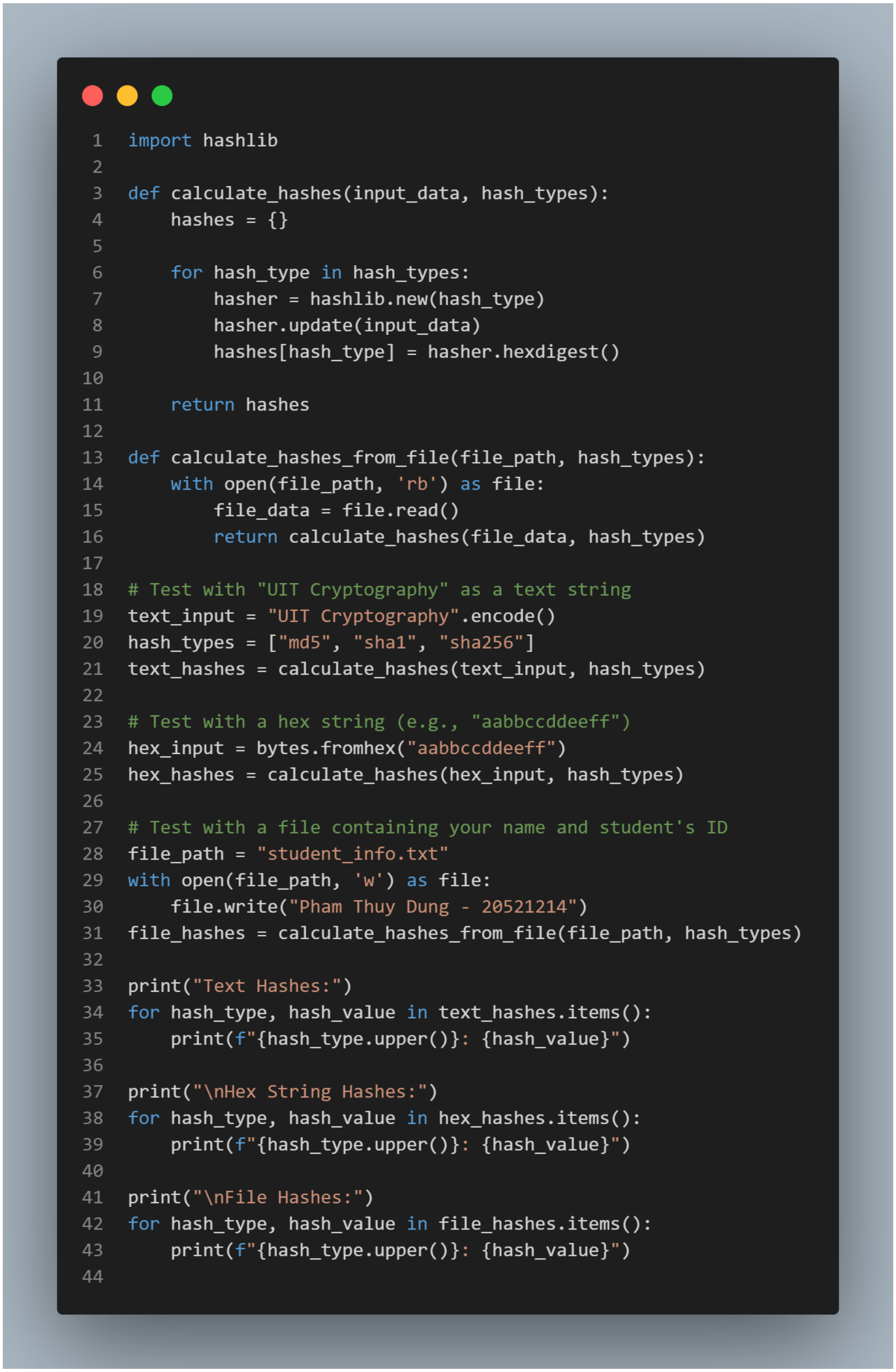
You are able to use the hash library for your own programming language. Then, test your application with the following exercise:

- Generate the hash values of "UIT Cryptography" in Text string and Hex string format. Then, compare the results with other tools to verify.
- Create a text file and put your name and student's ID inside. For example, "Nguyen Van An - 19521234". Generate hash values H1 of these files (using both MD5 and SHA-1). Subsequently, send this file to your friend via email or upload it to Google Drive and download it. Calculate the hash values of the downloaded file and compare them to those of the original file. Please observe whether these hash values are similar or not.

Tips: *You can refer to a similar application like* SlavaSoft HashCalc - Hash, CRC, and HMAC Calculator

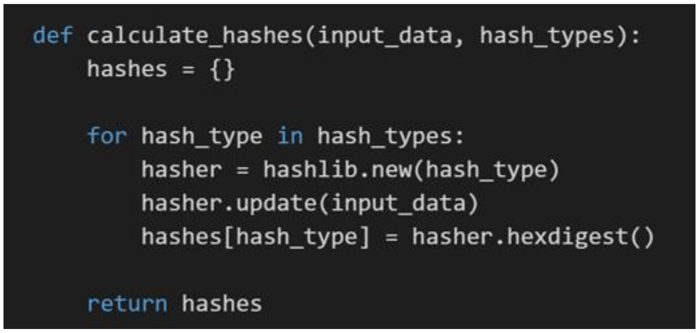
Here is my answer:

I use python the solve this problem:



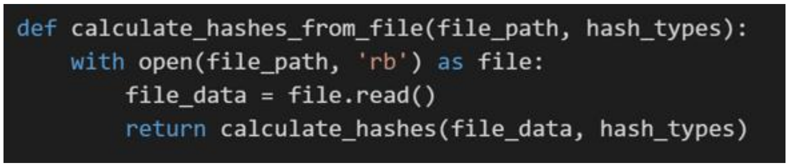
```python
import hashlib

def calculate_hashes(input_data, hash_types):
    hashes = {}

    for hash_type in hash_types:
        hasher = hashlib.new(hash_type)
        hasher.update(input_data)
        hashes[hash_type] = hasher.hexdigest()

    return hashes

def calculate_hashes_from_file(file_path, hash_types):
    with open(file_path, 'rb') as file:
        file_data = file.read()
        return calculate_hashes(file_data, hash_types)

# Test with "UIT Cryptography" as a text string
text_input = "UIT Cryptography".encode()
hash_types = ["md5", "sha1", "sha256"]
text_hashes = calculate_hashes(text_input, hash_types)

# Test with a hex string (e.g., "aabbccddeeff")
hex_input = bytes.fromhex("aabbccddeeff")
hex_hashes = calculate_hashes(hex_input, hash_types)

# Test with a file containing your name and student's ID
file_path = "student_info.txt"
with open(file_path, 'w') as file:
    file.write("Pham Thuy Dung - 20521214")
file_hashes = calculate_hashes_from_file(file_path, hash_types)

print("Text Hashes:")
for hash_type, hash_value in text_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nHex String Hashes:")
for hash_type, hash_value in hex_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nFile Hashes:")
for hash_type, hash_value in file_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")
```

Here is my explanation about the code:



```
def calculate_hashes(input_data, hash_types):
    hashes = {}

    for hash_type in hash_types:
        hasher = hashlib.new(hash_type)
        hasher.update(input_data)
        hashes[hash_type] = hasher.hexdigest()

    return hashes
```

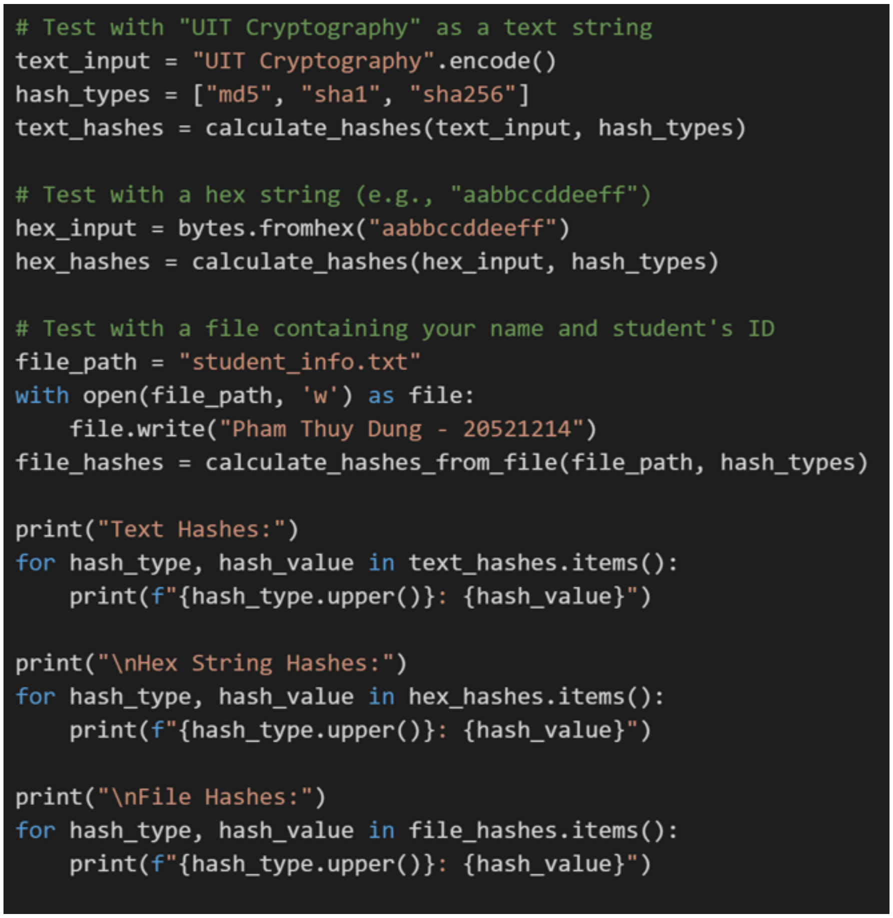
Purpose: To compute hashes of the provided data using specified hashing algorithms.



```
def calculate_hashes_from_file(file_path, hash_types):
    with open(file_path, 'rb') as file:
        file_data = file.read()
        return calculate_hashes(file_data, hash_types)
```

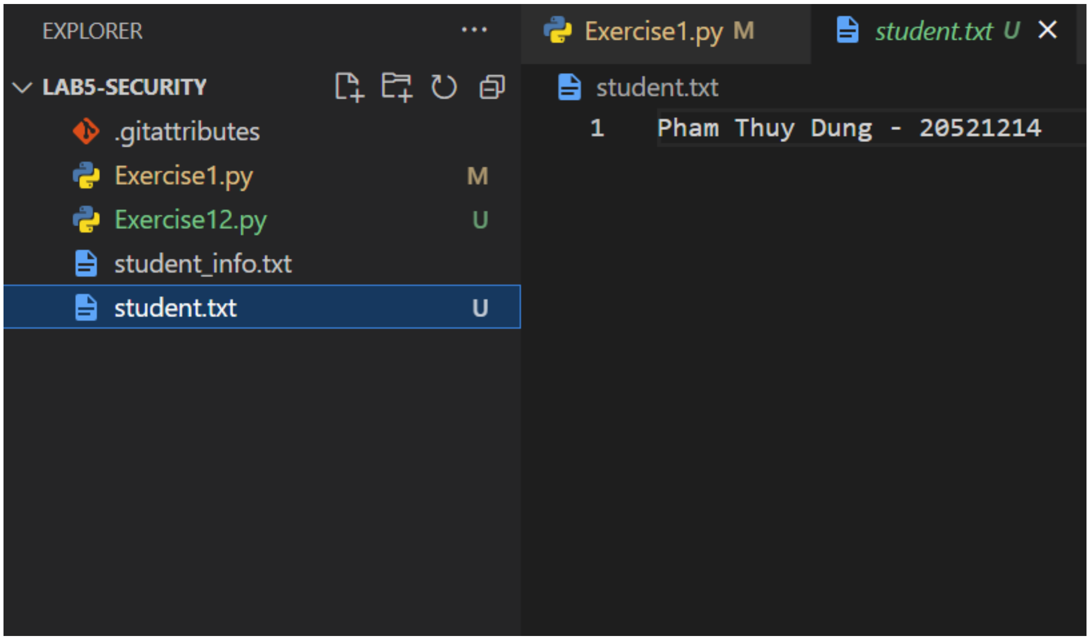
Purpose: To compute hashes of the contents of a specified file using different hashing algorithms.

With Main Script: it create to get input and give the result of the output



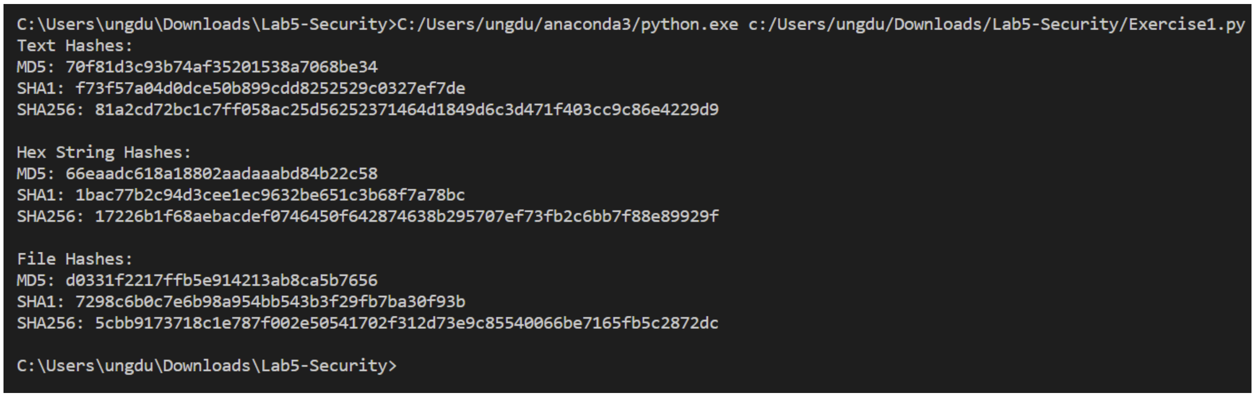
```python
# Test with "UIT Cryptography" as a text string
text_input = "UIT Cryptography".encode()
hash_types = ["md5", "sha1", "sha256"]
text_hashes = calculate_hashes(text_input, hash_types)

# Test with a hex string (e.g., "aabbccddeeff")
hex_input = bytes.fromhex("aabbccddeeff")
hex_hashes = calculate_hashes(hex_input, hash_types)

# Test with a file containing your name and student's ID
file_path = "student_info.txt"
with open(file_path, 'w') as file:
    file.write("Pham Thuy Dung - 20521214")
file_hashes = calculate_hashes_from_file(file_path, hash_types)

print("Text Hashes:")
for hash_type, hash_value in text_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nHex String Hashes:")
for hash_type, hash_value in hex_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nFile Hashes:")
for hash_type, hash_value in file_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")
```

Here is my file text created:



```
Pham Thuy Dung - 20521214
```

Here is my Result when run the code:



```
C:\Users\ungdu\Downloads\Lab5-Security>C:/Users/ungdu/anaconda3/python.exe c:/Users/ungdu/Downloads/Lab5-Security/Exercise1.py
Text Hashes:
MD5: 70f81d3c93b74af35201538a7068be34
SHA1: f73f57a04d0dce50b899cdd8252529c0327ef7de
SHA256: 81a2cd72bc1c7ff058ac25d56252371464d1849d6c3d471f403cc9c86e4229d9

Hex String Hashes:
MD5: 66eaadc618a18802aadaaabd84b22c58
SHA1: 1bac77b2c94d3cee1ec9632be651c3b68f7a78bc
SHA256: 17226b1f68aebacdef0746450f642874638b295707ef73fb2c6bb7f88e89929f

File Hashes:
MD5: d0331f2217ffb5e914213ab8ca5b7656
SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b
SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc

C:\Users\ungdu\Downloads\Lab5-Security>
```

Text Hashes:

MD5: 70f81d3c93b74af35201538a7068be34

SHA1: f73f57a04d0dce50b899cdd8252529c0327ef7de

SHA256: 81a2cd72bc1c7ff058ac25d56252371464d1849d6c3d471f403cc9c86e4229d9

Hex String Hashes:

MD5: 66eaadc618a18802aadaaabd84b22c58

SHA1: 1bac77b2c94d3cee1ec9632be651c3b68f7a78bc

SHA256: 17226b1f68aebacdef0746450f642874638b295707ef73fb2c6bb7f88e89929f

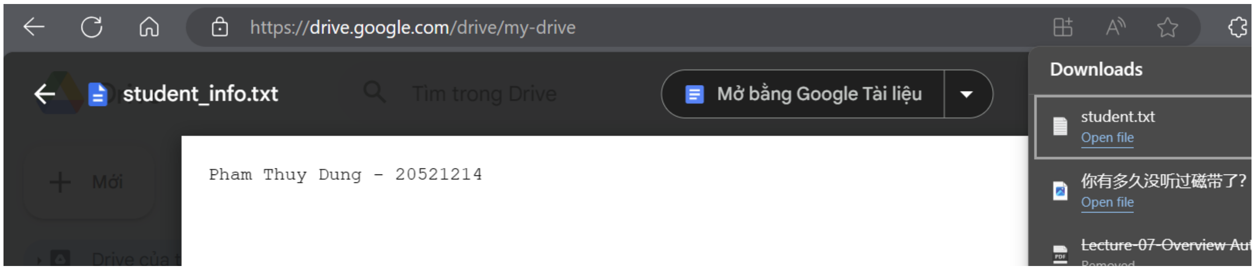
File Hashes:

MD5: d0331f2217ffb5e914213ab8ca5b7656

SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b

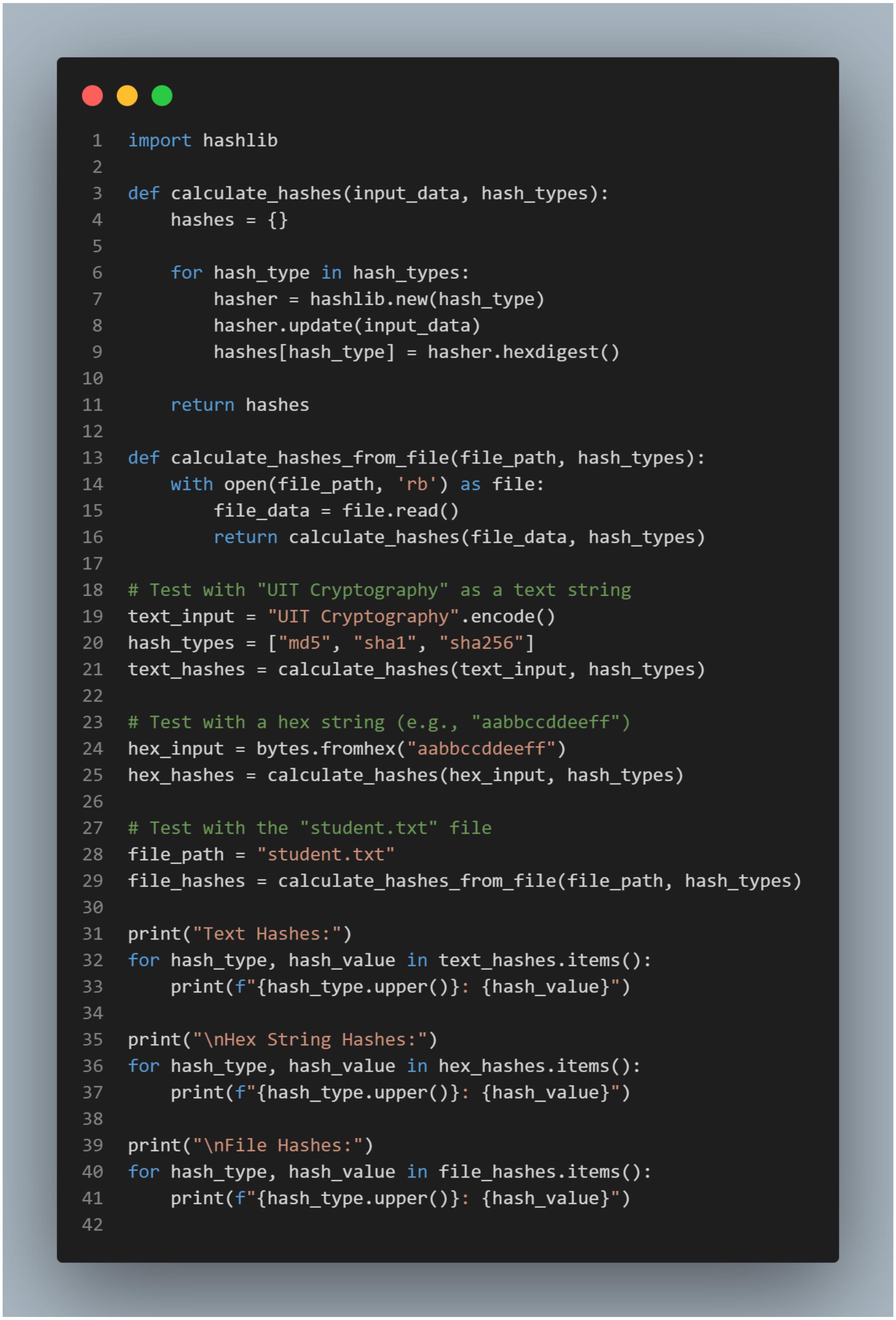
SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc

Here is illustration upload and download file from google drive



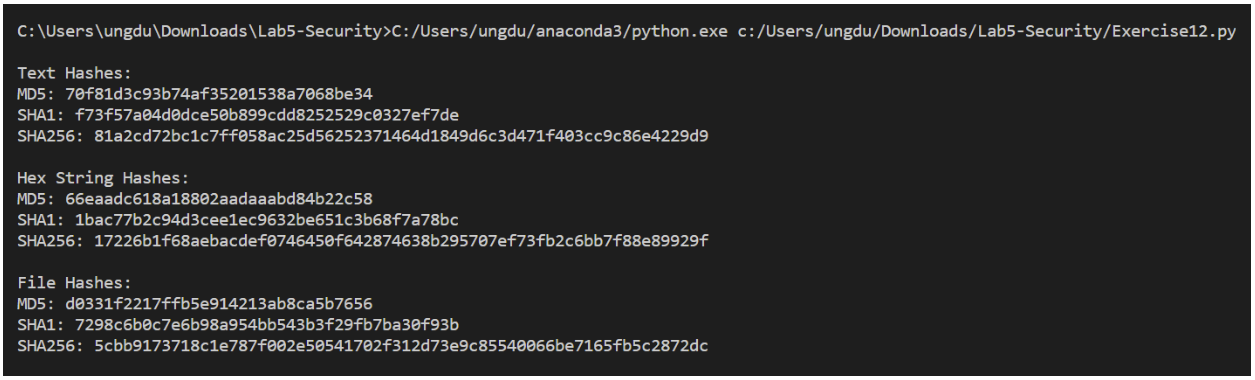
```
Pham Thuy Dung - 20521214
```

And next, is my code running to check the result again:



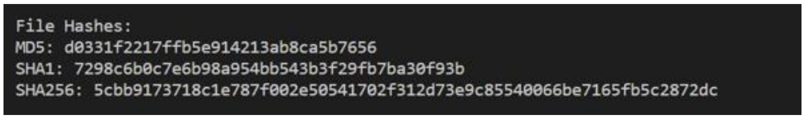
```python
import hashlib

def calculate_hashes(input_data, hash_types):
    hashes = {}

    for hash_type in hash_types:
        hasher = hashlib.new(hash_type)
        hasher.update(input_data)
        hashes[hash_type] = hasher.hexdigest()

    return hashes

def calculate_hashes_from_file(file_path, hash_types):
    with open(file_path, 'rb') as file:
        file_data = file.read()
        return calculate_hashes(file_data, hash_types)

# Test with "UIT Cryptography" as a text string
text_input = "UIT Cryptography".encode()
hash_types = ["md5", "sha1", "sha256"]
text_hashes = calculate_hashes(text_input, hash_types)

# Test with a hex string (e.g., "aabbccddeeff")
hex_input = bytes.fromhex("aabbccddeeff")
hex_hashes = calculate_hashes(hex_input, hash_types)

# Test with the "student.txt" file
file_path = "student.txt"
file_hashes = calculate_hashes_from_file(file_path, hash_types)

print("Text Hashes:")
for hash_type, hash_value in text_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nHex String Hashes:")
for hash_type, hash_value in hex_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

print("\nFile Hashes:")
for hash_type, hash_value in file_hashes.items():
    print(f"{hash_type.upper()}: {hash_value}")

```

Here is result of the code with downloaded file:

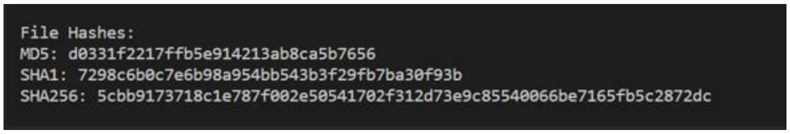


```
C:\Users\ungdu\Downloads\Lab5-Security>C:/Users/ungdu/anaconda3/python.exe c:/Users/ungdu/Downloads/Lab5-Security/Exercise12.py

Text Hashes:
MD5: 70f81d3c93b74af35201538a7068be34
SHA1: f73f57a04d0dce50b899cdd8252529c0327ef7de
SHA256: 81a2cd72bc1c7ff058ac25d56252371464d1849d6c3d471f403cc9c86e4229d9

Hex String Hashes:
MD5: 66eaadc618a18802aadaaabd84b22c58
SHA1: 1bac77b2c94d3cee1ec9632be651c3b68f7a78bc
SHA256: 17226b1f68aebacdef0746450f642874638b295707ef73fb2c6bb7f88e89929f

File Hashes:
MD5: d0331f2217ffb5e914213ab8ca5b7656
SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b
SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc
```

Comparing 2 results:



```
File Hashes:
MD5: d0331f2217ffb5e914213ab8ca5b7656
SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b
SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc
```



```
File Hashes:
MD5: d0331f2217ffb5e914213ab8ca5b7656
SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b
SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc
```

So, you can see the hash values of downloaded file is the same with original file:

For the file hashes, ideally, they remain the same before and after uploading/downloading, assuming the file content hasn't changed. This is because hash functions are designed to produce the same output for the same input and are sensitive to even minor changes in input.

Exercise 2:

2. Hash properties: One-way vs Collision-free

Task 2.1

It is now well-known that the cryptographic hash function MD5 and SHA-1 has been clearly broken (in terms of collision-resistance property). We will find out about MD5 and SHA-1 collision in this task by doing the following exercises:

1. Consider two HEX messages as follows:

   Message 1

   ```
   d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
   55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8
   823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f
   33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
   ```

   Message 2

   ```
   d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8
   955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd728037
   3c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d
   248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965a
   b6ff72a70
   ```

   How many bytes are the difference between two messages?

   Let's generate MD5 hash values for each message. Please observe whether these MD5 are similar or not and describe your observations in the lab report.

2. Consider two executable programs named hello and erase:
   - If you are using Windows, you can download these .exe files here.
   - If you are using Linux, you can download the similar: hello and erase.

   Run these programs and observe what happens. Note these programs must be run from the console. Let's generate MD5 hash values for these programs and report your observations

3. Download two PDF files: shattered-1.pdf and shattered-2.pdf. Open these files to check the difference. Then generate SHA-1 hash for them, and observe the result.

Draw the conclusion based on your observations, explain the reasons for the existence of collision in MD5 and SHA-1.

1. Consider two HEX messages as follows:

Message 1

d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89

55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8

823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f

33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70

Message 2

d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8

955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd728037

3c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d

248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965a

b6ff72a70

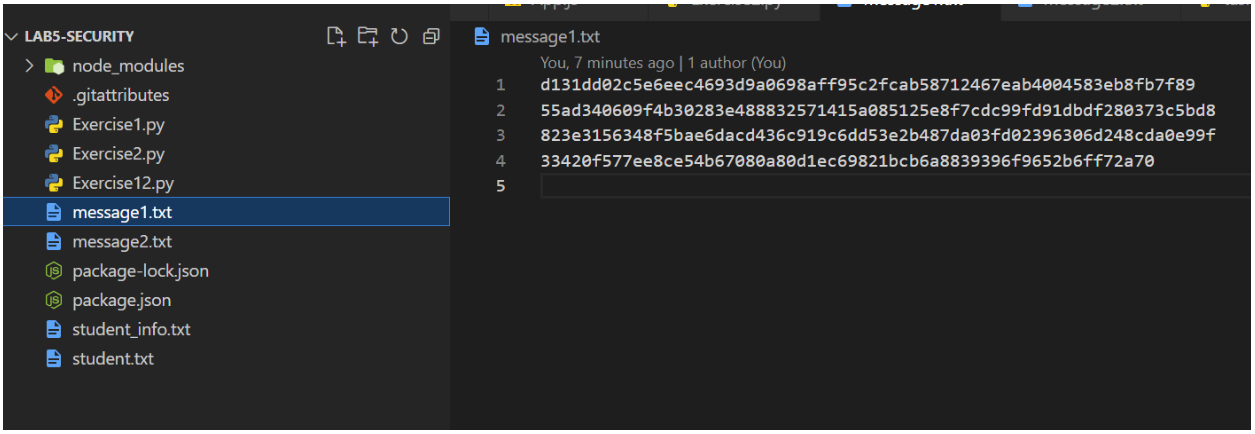
How many bytes are the difference between two messages?

To do it we determine the byte difference between two given hexadecimal messages.

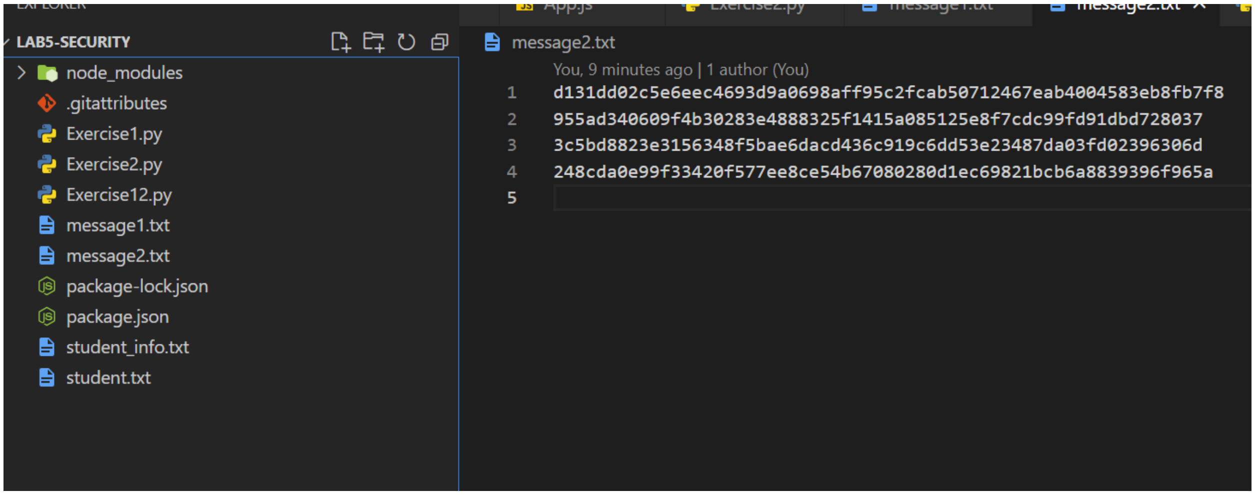
And then compare the two messages byte by byte

I use tool WinHex to compare two messages

First, I create 2 file text

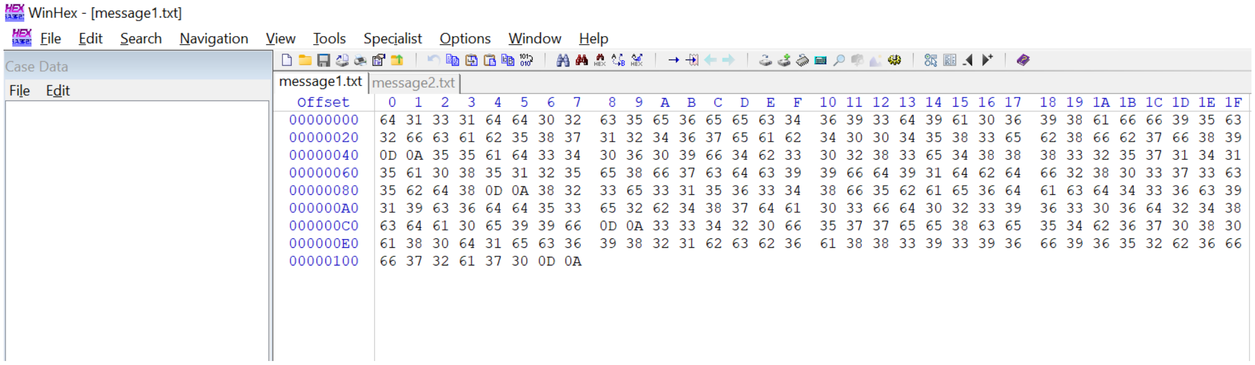


```
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8
823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f
33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

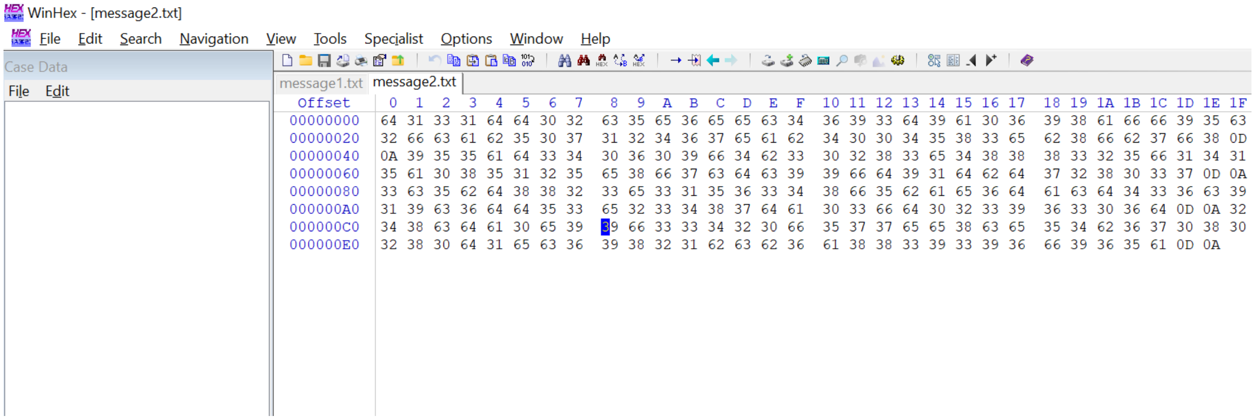


```
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8
955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd728037
3c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d
248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965a
```

And the open it in WinHex

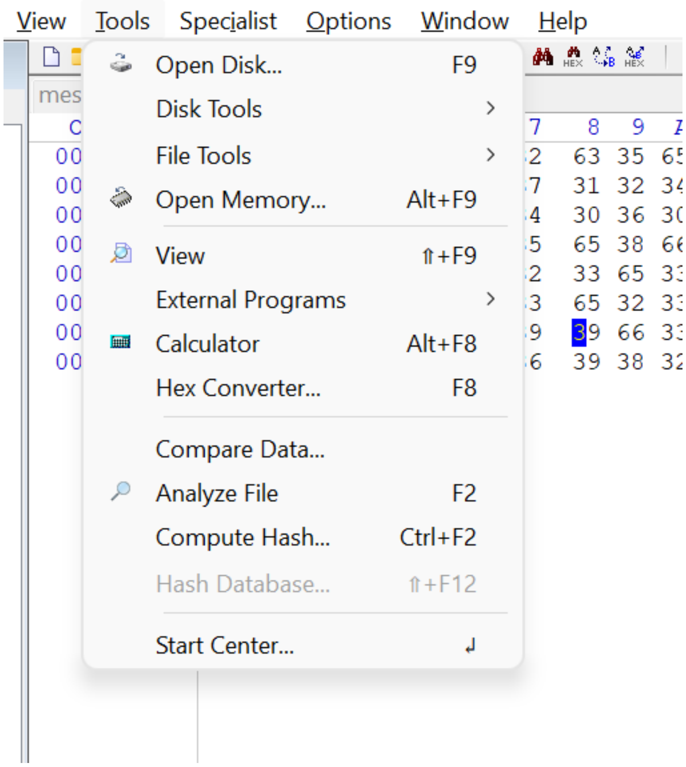

WinHex - [message1.txt]
File Edit Search Navigation View Tools Specialist Options Window Help
Case Data
File Edit
message1.txt
message2.txt

```
  Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F
00000000  64 31 33 31 64 64 30 32  63 35 65 36 65 65 63 34  36 39 33 64 39 61 30 36  39 38 61 66 66 39 35 63
00000020  32 66 63 61 62 35 38 37  31 32 34 36 37 65 61 62  34 30 30 34 35 38 33 65  62 38 66 62 37 66 38 39
00000040  0D 0A 35 35 61 64 33 34  30 36 30 39 66 34 62 33  30 32 38 33 65 34 38 38  38 33 32 35 37 31 34 31
00000060  35 61 30 38 35 31 32 35  65 38 66 37 63 64 63 39  39 66 64 39 31 64 62 64  66 32 38 30 33 37 33 63
00000080  35 62 64 38 0D 0A 38 32  33 65 33 31 35 36 33 34  38 66 35 62 61 65 36 64  61 63 64 34 33 36 63 39
000000A0  31 39 63 36 64 64 35 33  65 32 62 34 38 37 64 61  30 33 66 64 30 32 33 39  36 33 30 36 64 32 34 38
000000C0  63 64 61 30 65 39 39 66  0D 0A 33 33 34 32 30 66  35 37 37 65 65 38 63 65  35 34 62 36 37 30 38 30
000000E0  61 38 30 64 31 65 63 36  39 38 32 31 62 63 62 36  61 38 38 33 39 33 39 36  66 39 36 35 32 62 36 66
00000100  66 37 32 61 37 30 0D 0A
```

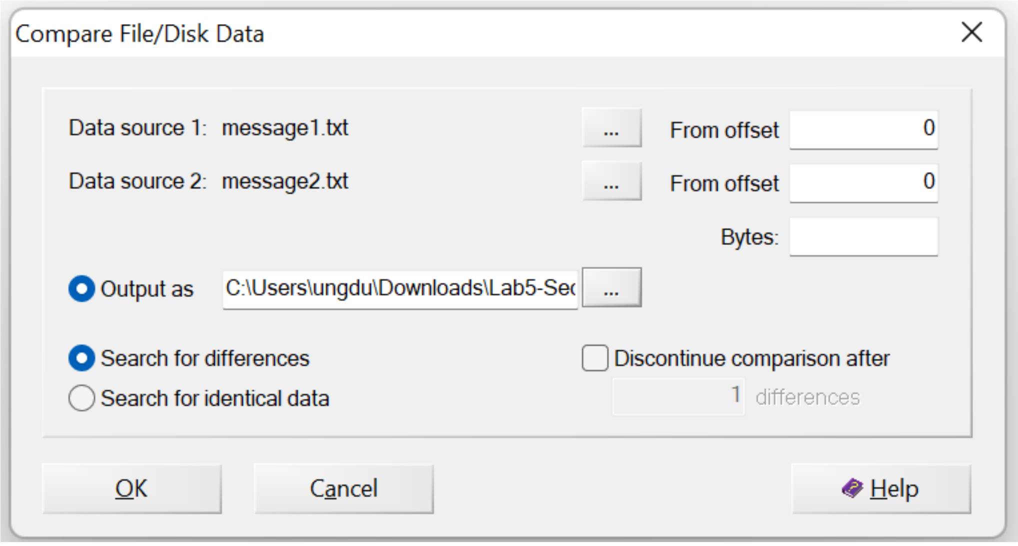

WinHex - [message2.txt]
File Edit Search Navigation View Tools Specialist Options Window Help
Case Data
File Edit
message1.txt
message2.txt

```
  Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F  10 11 12 13 14 15 16 17  18 19 1A 1B 1C 1D 1E 1F
00000000  64 31 33 31 64 64 30 32  63 35 65 36 65 65 63 34  36 39 33 64 39 61 30 36  39 38 61 66 66 39 35 63
00000020  32 66 63 61 62 35 30 37  31 32 34 36 37 65 61 62  34 30 30 34 35 38 33 65  62 38 66 62 37 66 38 0D
00000040  0A 39 35 35 61 64 33 34  30 36 30 39 66 34 62 33  30 32 38 33 65 34 38 38  38 33 32 35 66 31 34 31
00000060  35 61 30 38 35 31 32 35  65 38 66 37 63 64 63 39  39 66 64 39 31 64 62 64  37 32 38 30 33 37 0D 0A
00000080  33 63 35 62 64 38 38 32  33 65 33 31 35 36 33 34  38 66 35 62 61 65 36 64  61 63 64 34 33 36 63 39
000000A0  31 39 63 36 64 64 35 33  65 32 33 34 38 37 64 61  30 33 66 64 30 32 33 39  36 33 30 36 64 0D 0A 32
000000C0  34 38 63 64 61 30 65 39  39 66 33 33 34 32 30 66  35 37 37 65 65 38 63 65  35 34 62 36 37 30 38 30
000000E0  32 38 30 64 31 65 63 36  39 38 32 31 62 63 62 36  61 38 38 33 39 33 39 36  66 39 36 35 61 0D 0A
```

Then I click Tool > Compare data to compare 2 text



And then set up the output file with name report with search for differences engines and click OK



Compare File/Disk Data

Data source 1: message1.txt ... From offset 0

Data source 2: message2.txt ... From offset 0

Bytes:

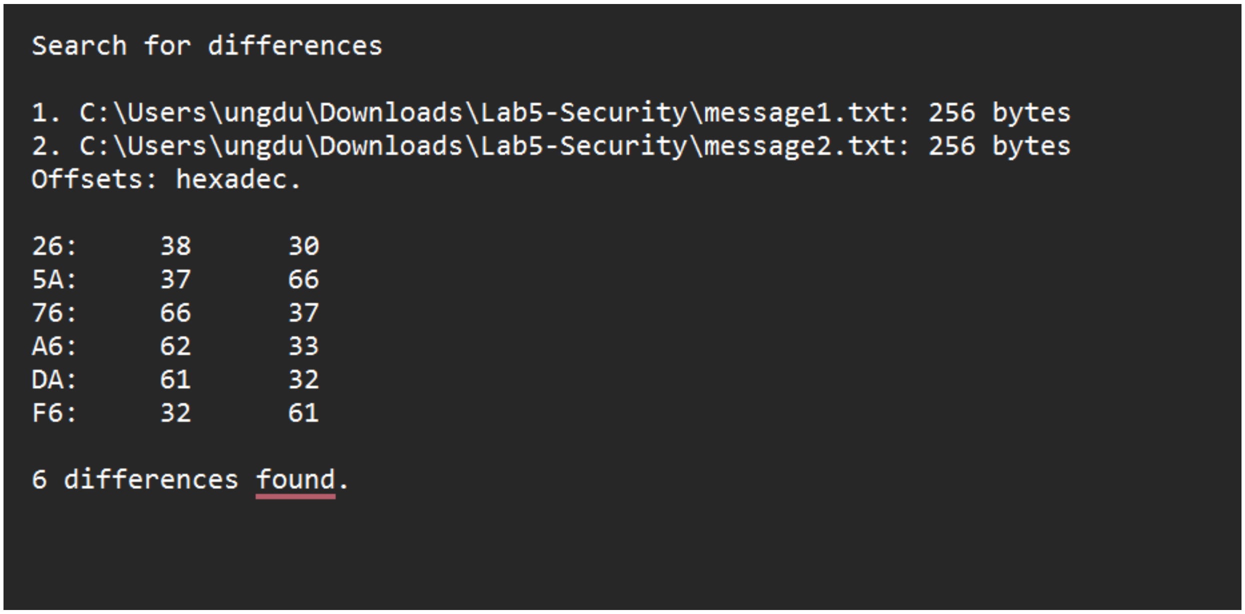
● Output as C:\Users\ungdu\Downloads\Lab5-Sec ...

● Search for differences

○ Search for identical data

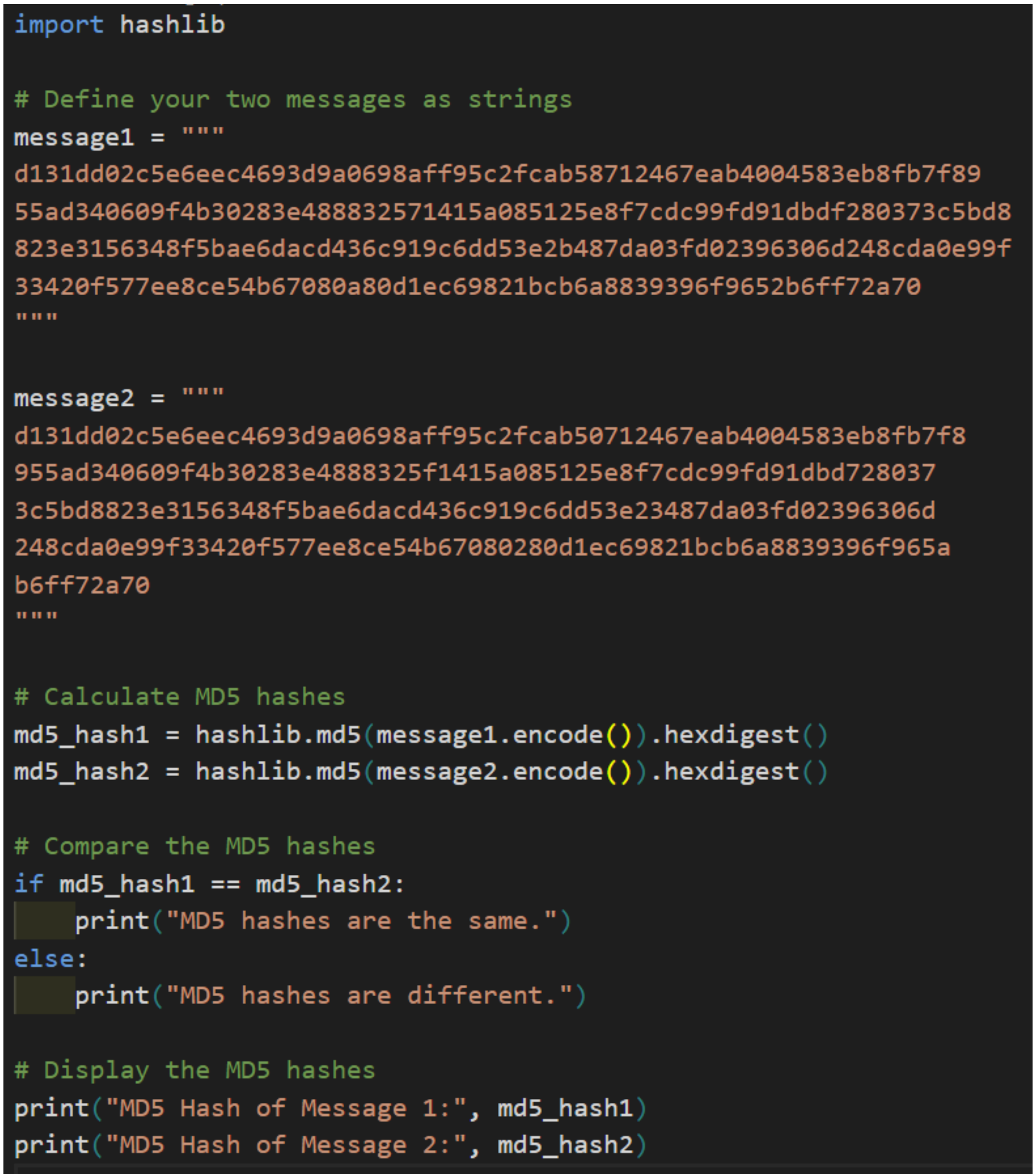
□ Discontinue comparison after 1 differences

Here is result of the output



```
Search for differences

1. C:\Users\ungdu\Downloads\Lab5-Security\message1.txt: 256 bytes
2. C:\Users\ungdu\Downloads\Lab5-Security\message2.txt: 256 bytes
Offsets: hexadec.

26:     38      30
5A:     37      66
76:     66      37
A6:     62      33
DA:     61      32
F6:     32      61

6 differences found.
```

So, we have 6 different byte can be found.



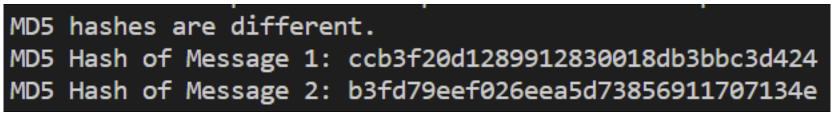
```
import hashlib

# Define your two messages as strings
message1 = """
d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8
823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f
33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
"""

message2 = """
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8
955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd728037
3c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d
248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965a
b6ff72a70
"""

# Calculate MD5 hashes
md5_hash1 = hashlib.md5(message1.encode()).hexdigest()
md5_hash2 = hashlib.md5(message2.encode()).hexdigest()

# Compare the MD5 hashes
if md5_hash1 == md5_hash2:
    print("MD5 hashes are the same.")
else:
    print("MD5 hashes are different.")

# Display the MD5 hashes
print("MD5 Hash of Message 1:", md5_hash1)
print("MD5 Hash of Message 2:", md5_hash2)
```

Let's generate MD5 hash values for each message. Please observe whether

these MD5 are similar or not and describe your observations in the lab report

Here is my code to do it

When I run the code, I get the result:



```
MD5 hashes are different.
MD5 Hash of Message 1: ccb3f20d1289912830018db3bbc3d424
MD5 Hash of Message 2: b3fd79eef026eea5d73856911707134e
```

With the result, we can see that these MD5 hashes are not similar; they are different. MD5 hashes are designed to change significantly even with small changes in the input data. Therefore, two slightly different messages will result in vastly different MD5 hashes.

In the context of cryptographic hash functions, MD5 is considered to be broken in terms of collision resistance, meaning that it's relatively easy to find two different inputs that produce the same MD5 hash (a collision). As a result, it's not recommended for security-critical applications, and more secure hash functions like SHA-256 are preferred.

2. Consider two executable programs named hello and erase:

• If you are using Windows, you can download these .exe files here.

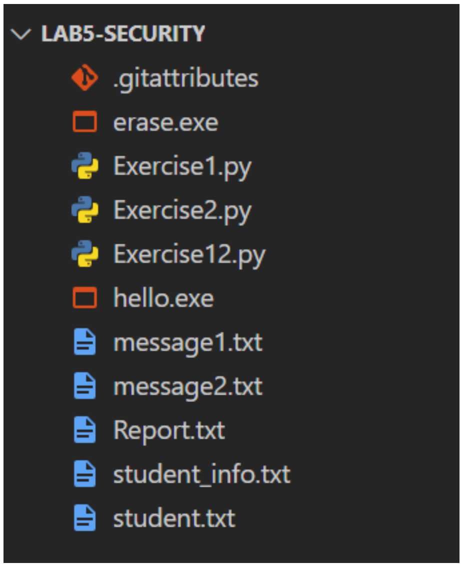
• If you are using Linux, you can download the similar: hello and erase.

Run these programs and observe what happens. Note these programs must

be run from the console. Let’s generate MD5 hash values for these programs

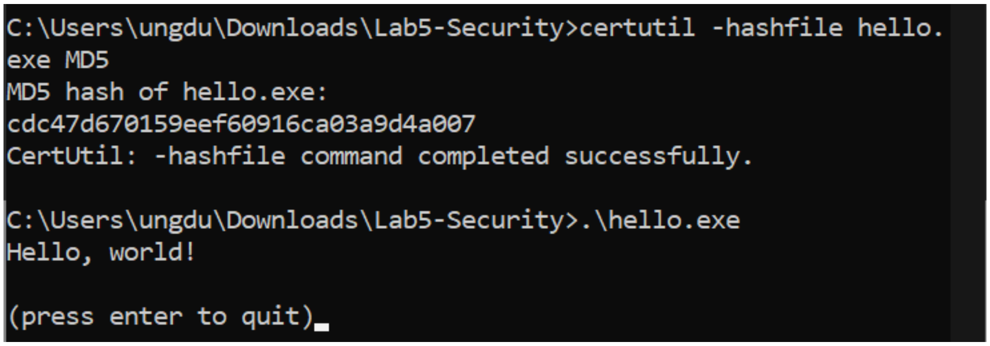
and report your observations

I downloaded the file in https://www.mscs.dal.ca/~selinger/md5collision/



LAB5-SECURITY
- .gitattributes
- erase.exe
- Exercise1.py
- Exercise2.py
- Exercise12.py
- hello.exe
- message1.txt
- message2.txt
- Report.txt
- student_info.txt
- student.txt

Here is my result:

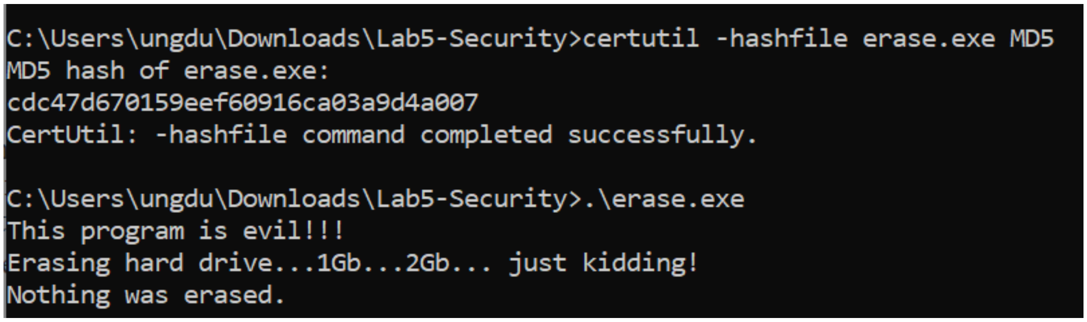


```
C:\Users\ungdu\Downloads\Lab5-Security>certutil -hashfile hello.
exe MD5
MD5 hash of hello.exe:
cdc47d670159eef60916ca03a9d4a007
CertUtil: -hashfile command completed successfully.

C:\Users\ungdu\Downloads\Lab5-Security>.\hello.exe
Hello, world!

(press enter to quit)
```

Report:

The first command calculates the MD5 hash value of the "hello.exe" file, and the resulting MD5 hash is cdc47d670159eef60916ca03a9d4a007.

The second command runs the "hello.exe" executable, which prints the message "Hello, world! (press enter to quit)" to the console.



```
C:\Users\ungdu\Downloads\Lab5-Security>certutil -hashfile erase.exe MD5
MD5 hash of erase.exe:
cdc47d670159eef60916ca03a9d4a007
CertUtil: -hashfile command completed successfully.

C:\Users\ungdu\Downloads\Lab5-Security>.\erase.exe
This program is evil!!!
Erasing hard drive...1Gb...2Gb... just kidding!
Nothing was erased.
```

Report:

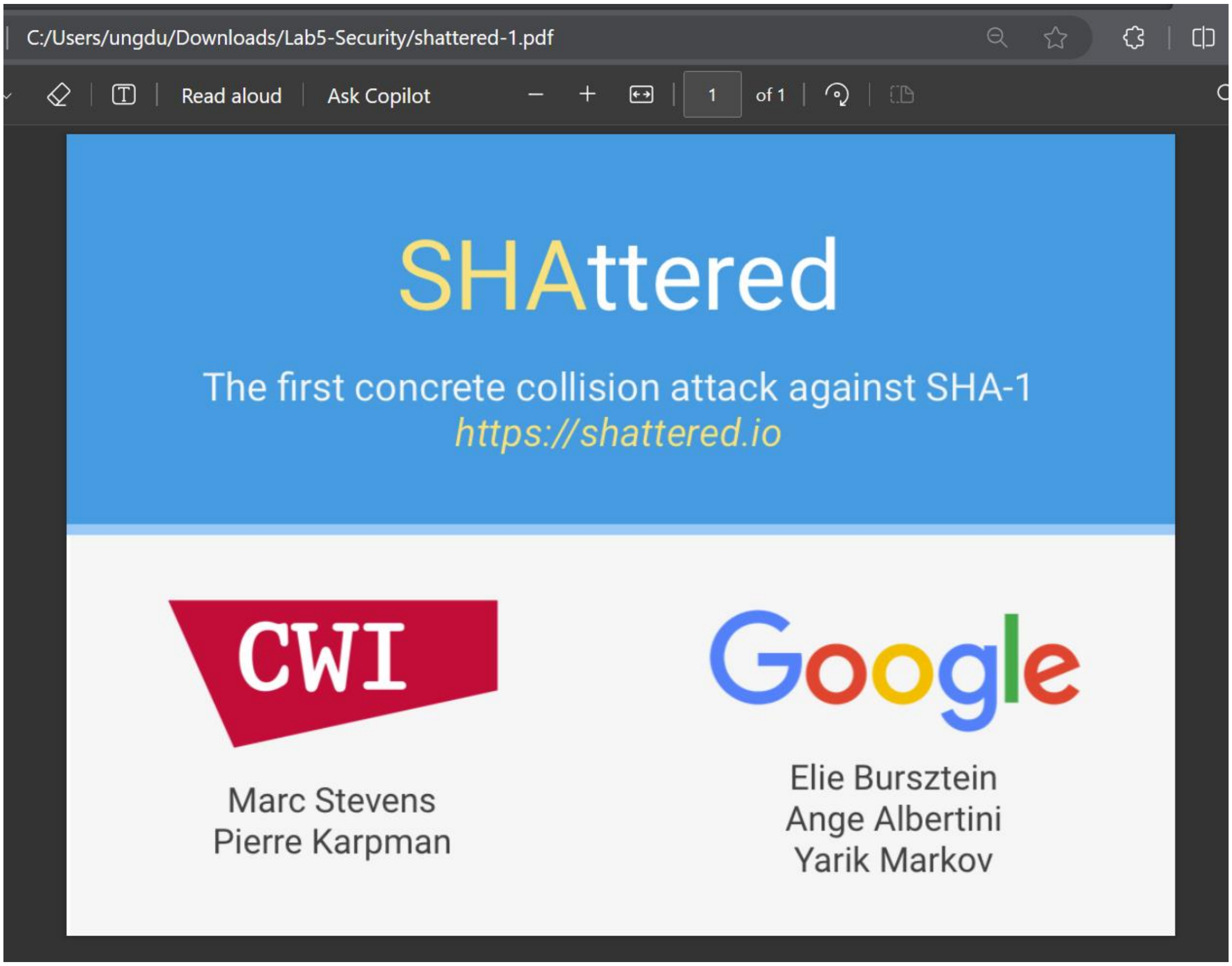
The first command calculates the MD5 hash value of the "erase.exe" file, and the resulting MD5 hash is cdc47d670159eef60916ca03a9d4a007. This MD5 hash can be used to verify the integrity of the "erase.exe" file.

The second command runs the "erase.exe" executable, which displays a humorous message indicating that it is an "evil" program that pretends to erase the hard drive but does not actually perform any data erasure.

In this case, both "hello.exe" and "erase.exe" have the same MD5 hash value (cdc47d670159eef60916ca03a9d4a007), which indicates that the content of these two files is the same. This is why they have the same MD5 hash.

The difference in behavior when executing these files is due to the instructions and logic contained within the programs themselves. The content of the files is the same, but the code inside each executable file determines how they behave when run. In the case of "hello.exe," it simply prints "Hello, world!" to the console, while "erase.exe" displays a humorous message about erasing the hard drive but does nothing harmful. The difference in behavior is determined by the code within the executables, not their MD5 hash values.

3. Download two PDF files: shattered-1.pdf and shattered-2.pdf. Open these files to check the difference. Then generate SHA-1 hash for them, and observe the result.



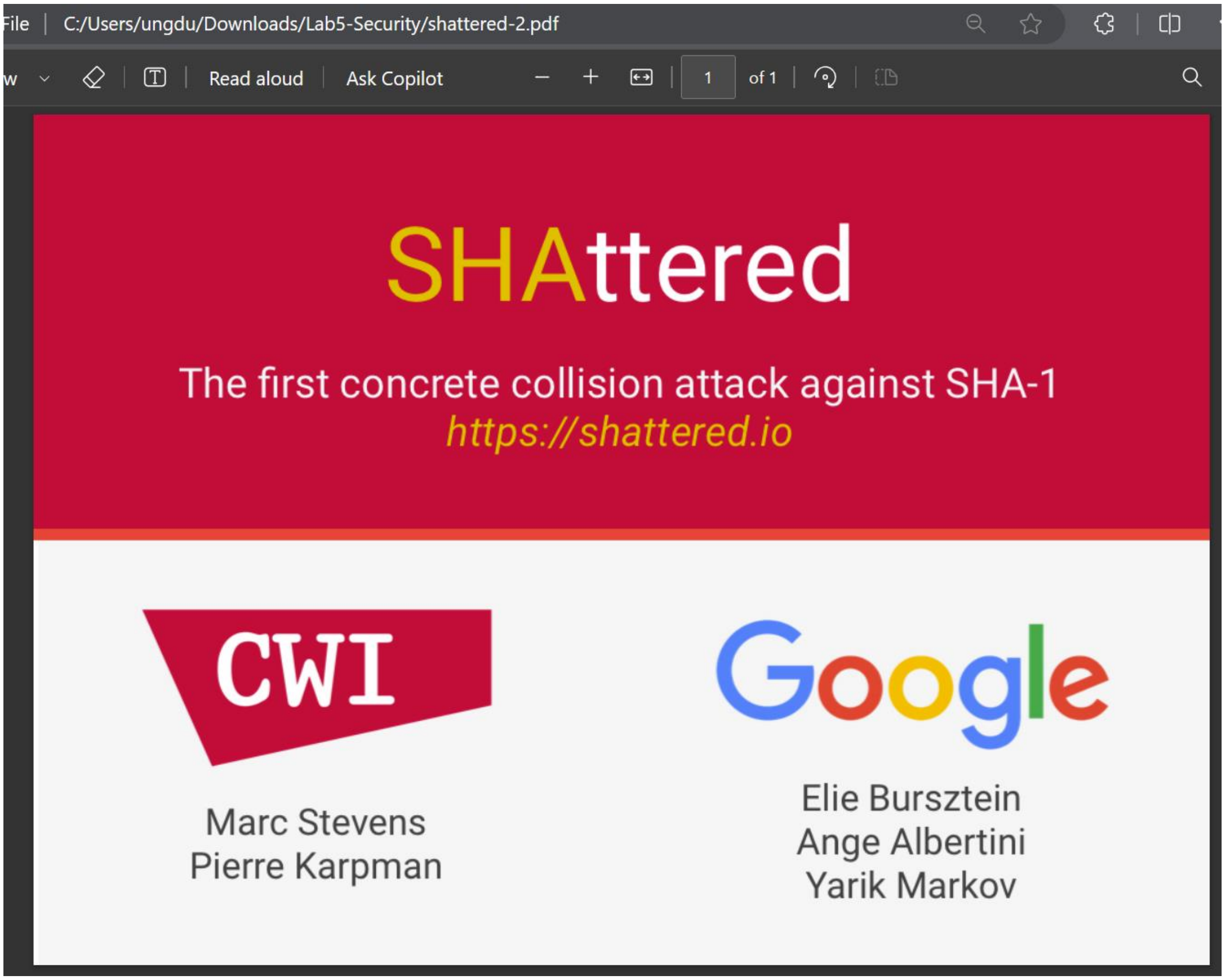
C:/Users/ungdu/Downloads/Lab5-Security/shattered-1.pdf

SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

Marc Stevens
Pierre Karpman

Elie Bursztein
Ange Albertini
Yarik Markov



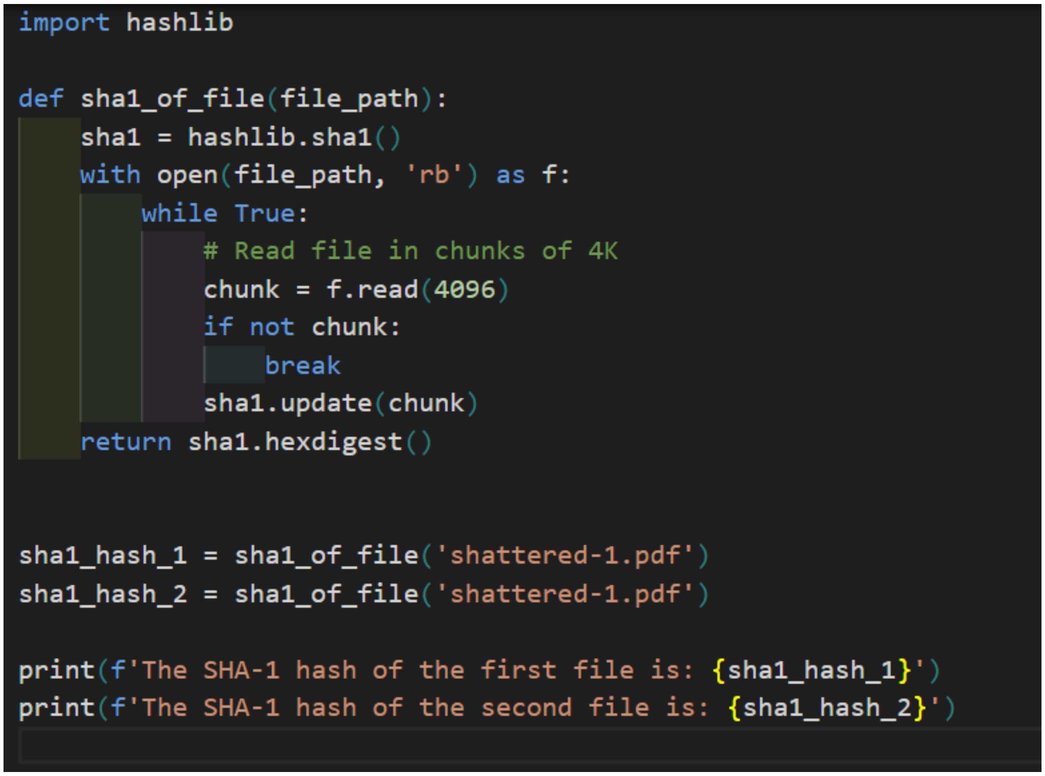
SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

Marc Stevens
Pierre Karpman

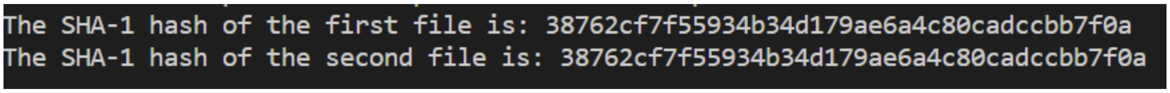
Elie Bursztein
Ange Albertini
Yarik Markov

Here is my code to check the SHA-1 file:



```
import hashlib

def sha1_of_file(file_path):
    sha1 = hashlib.sha1()
    with open(file_path, 'rb') as f:
        while True:
            # Read file in chunks of 4K
            chunk = f.read(4096)
            if not chunk:
                break
            sha1.update(chunk)
    return sha1.hexdigest()


sha1_hash_1 = sha1_of_file('shattered-1.pdf')
sha1_hash_2 = sha1_of_file('shattered-1.pdf')

print(f'The SHA-1 hash of the first file is: {sha1_hash_1}')
print(f'The SHA-1 hash of the second file is: {sha1_hash_2}')
```

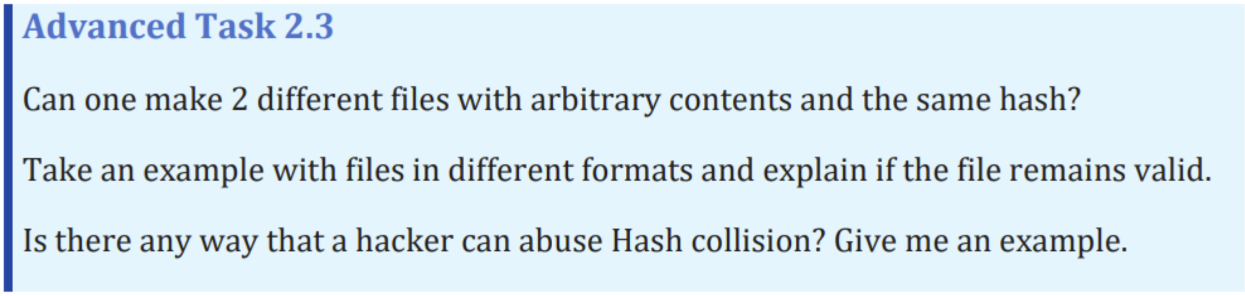
Here is my result:



```
The SHA-1 hash of the first file is: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
The SHA-1 hash of the second file is: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
```

You can see the output with the same characteristics, so it demonstrates:

This result is a demonstration of a collision attack against the SHA-1 cryptographic hash function, where two different inputs produce the same output hash. This is a significant vulnerability because hash functions like SHA-1 are supposed to generate a unique hash for each unique input; collisions undermine this principle and can compromise the security of systems relying on the hash function for integrity verification or other security guarantees.



Advanced Task 2.3

Can one make 2 different files with arbitrary contents and the same hash?

Take an example with files in different formats and explain if the file remains valid.

Is there any way that a hacker can abuse Hash collision? Give me an example.

Can one make 2 different files with arbitrary contents and the same hash?

Creating two different files with the same hash, especially with arbitrary contents, is a significant challenge. This is known as a hash collision. While it is theoretically possible, especially with weaker hash functions like MD5 and SHA-1, it is computationally intensive and requires a deep understanding of the hash function's structure. The process typically involves crafting a specific section of the file that is designed to produce the same hash output despite changes in other parts of the file. The SHAttered attack demonstrated that this is possible with SHA-1.

For files in different formats to have the same hash and remain valid, one would need to exploit specific vulnerabilities in the file format's structure to insert the collision-producing data in a way that doesn't invalidate the file. This is much more complex and is not typically feasible with arbitrary content since file formats have specific requirements and structures that must be adhered to.

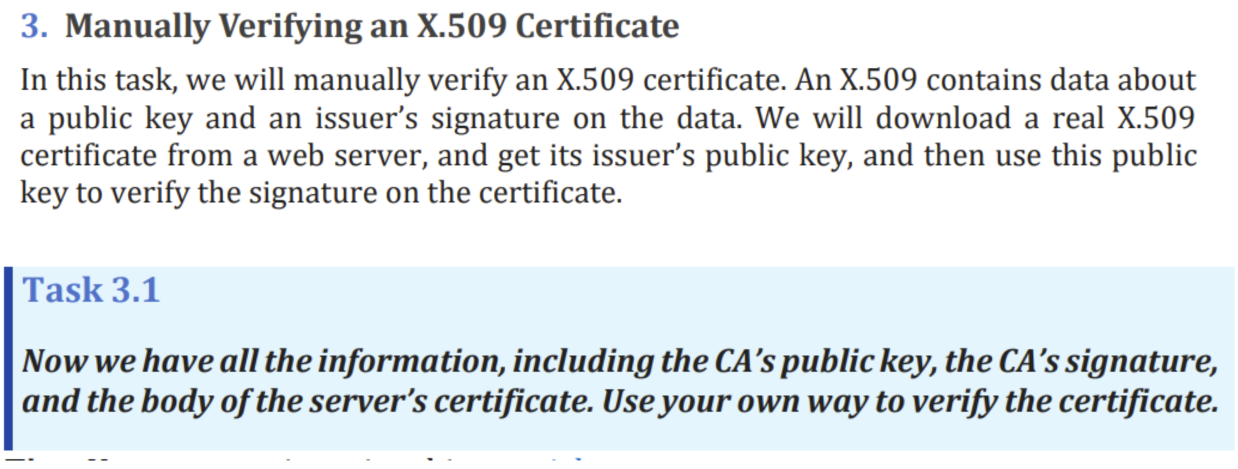
Can a hacker abuse Hash collision?

Yes, hash collisions can be abused by hackers in several ways. Here’s an example:

Example of Hash Collision Abuse: Digital Certificates

Consider digital certificates that use a hash function to ensure the integrity of the certificate contents. If a hacker can create two different certificates with the same hash value (a collision), they could get a legitimate certificate authority (CA) to sign a valid certificate and then replace it with a fraudulent one that has the same hash. The fraudulent certificate would appear valid because it has a valid signature from the CA. This could enable a wide range of security breaches, such as man-in-the-middle attacks, where the hacker could impersonate websites or services.

In real-world scenarios, a hash collision attack requires significant computational resources and expertise. Security algorithms are constantly being updated to address vulnerabilities, and the use of stronger hash functions like SHA-256 and SHA-3 is becoming more widespread precisely because they are currently resistant to known collision attacks.



3. Manually Verifying an X.509 Certificate

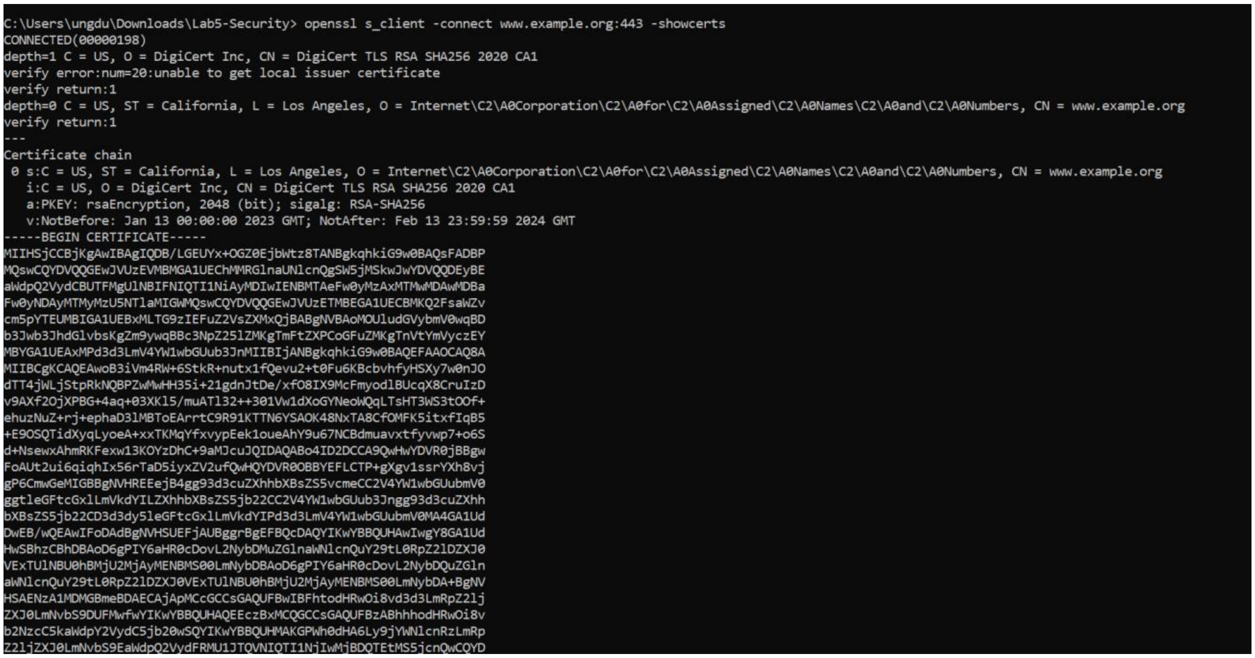
In this task, we will manually verify an X.509 certificate. An X.509 contains data about a public key and an issuer's signature on the data. We will download a real X.509 certificate from a web server, and get its issuer's public key, and then use this public key to verify the signature on the certificate.

Task 3.1

Now we have all the information, including the CA's public key, the CA's signature, and the body of the server's certificate. Use your own way to verify the certificate.

Answer:

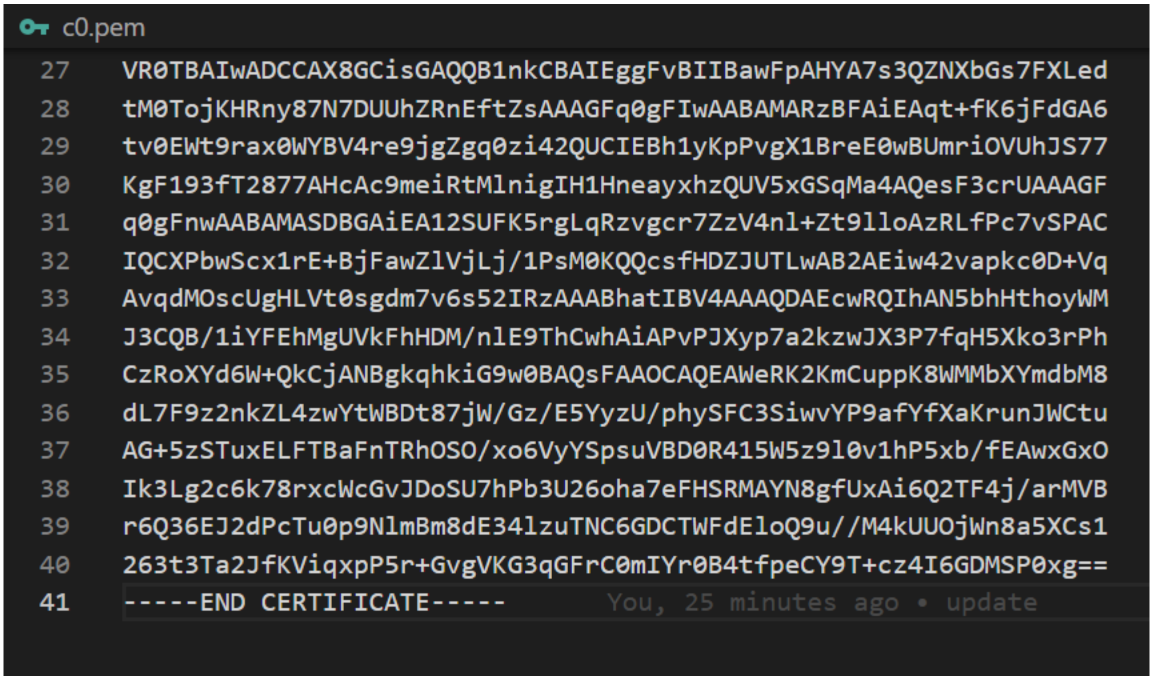
Step 1: Download a certificate from a real webserver



```
C:\Users\ungdu\Downloads\Lab5-Security> openssl s_client -connect www.example.org:443 -showcerts
CONNECTED(00000198)
depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = www.example.org
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Los Angeles, O = Internet\C2\A0Corporation\C2\A0for\C2\A0Assigned\C2\A0Names\C2\A0and\C2\A0Numbers, CN = www.example.org
   i:C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
   a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jan 13 00:00:00 2023 GMT; NotAfter: Feb 13 23:59:59 2024 GMT
-----BEGIN CERTIFICATE-----
MIIHSjCCBjKgAwIBAgIQDB/LGEUYx+OGZ0EjbWtz8TANBgkqhkiG9w0BAQsFADBP
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSkwJwYDVQQDEyBE
aWdpQ2VydCBUTFMgUlNBIFNIQTI1NiAyMDIwIENBMTAeFw0yMzAxMTMwMDAwMDBa
Fw0yNDAyMTMyMzU5NTlaMIGWMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZv
cm5pYTEUMBIGA1UEBxMLTG9zIEFuZ2VsZXMxQjBABgNVBAoMOUludGVybmV0wqBD
b3Jwb3JhdGlvbsKgZm9ywqBBc3NpZ25lZMKgTmFtZXPCoGFuZMKgTnVtYmVyczEY
MBYGA1UEAxMPd3d3LmV4YW1wbGUub3JnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEAwoB3iVm4RW+6StkR+nutx1fQevu2+t0Fu6KBcbvhfyHSXy7w0nJO
dTT4jWLjStpRkNQBPZwMwHH35i+21gdnJtDe/xfO8IX9McFmyodlBUcqX8CruIzD
v9AXf2OjXPBG+4aq+03XKl5/muATl32++301Vw1dXoGYNeoWQqLTsHT3WS3tOOf+
ehuzNuZ+rj+ephaD3lMBToEArrtC9R91KTTN6YSAOK48NxTA8CfOMFK5itxfIqB5
+E9OSQTidXyqLyoeA+xxTKMqYfxvypEek1oueAhY9u67NCBdmuavxtfyvwp7+o6S
d+NsewxAhmRKFexw13KOYzDhC+9aMJcuJQIDAQABo4ID2DCCA9QwHwYDVR0jBBgw
FoAUt2ui6qiqhIx56rTaD5iyxZV2ufQwHQYDVR0OBBYEFLCTP+gXgv1ssrYXh8vj
gP6CmwGeMIGBBgNVHREEejB4gg93d3cuZXhhbXBsZS5vcmeCC2V4YW1wbGUubmV0
ggtleGFtcGxlLmVkdYILZXhhbXBsZS5jb22CC2V4YW1wbGUub3Jngg93d3cuZXhh
bXBsZS5jb22CD3d3dy5leGFtcGxlLmVkdYIPd3d3LmV4YW1wbGUubmV0MA4GA1Ud
DwEB/wQEAwIFoDAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwgY8GA1Ud
HwSBhzCBhDBAoD6gPIY6aHR0cDovL2NybDMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0
VExTUlNBU0hBMjU2MjAyMENBMS00LmNybDBAoD6gPIY6aHR0cDovL2NybDQuZGln
aWNlcnQuY29tL0RpZ2lDZXJ0VExTUlNBU0hBMjU2MjAyMENBMS00LmNybDA+BgNV
HSAENzA1MDMGBmeBDAECAjApMCcGCCsGAQUFBwIBFhtodHRwOi8vd3d3LmRpZ2lj
ZXJ0LmNvbS9DUFMwfwYIKwYBBQUHAQEEczBxMCQGCCsGAQUFBzABhhhodHRwOi8v
b2NzcC5kaWdpY2VydC5jb20wSQYIKwYBBQUHMAKGPWh0dHA6Ly9jYWNlcnRzLmRp
Z2ljZXJ0LmNvbS9EaWdpQ2VydFRMU1JTQVNIQTI1NjIwMjBDQTEtMS5jcnQwCQYD
```

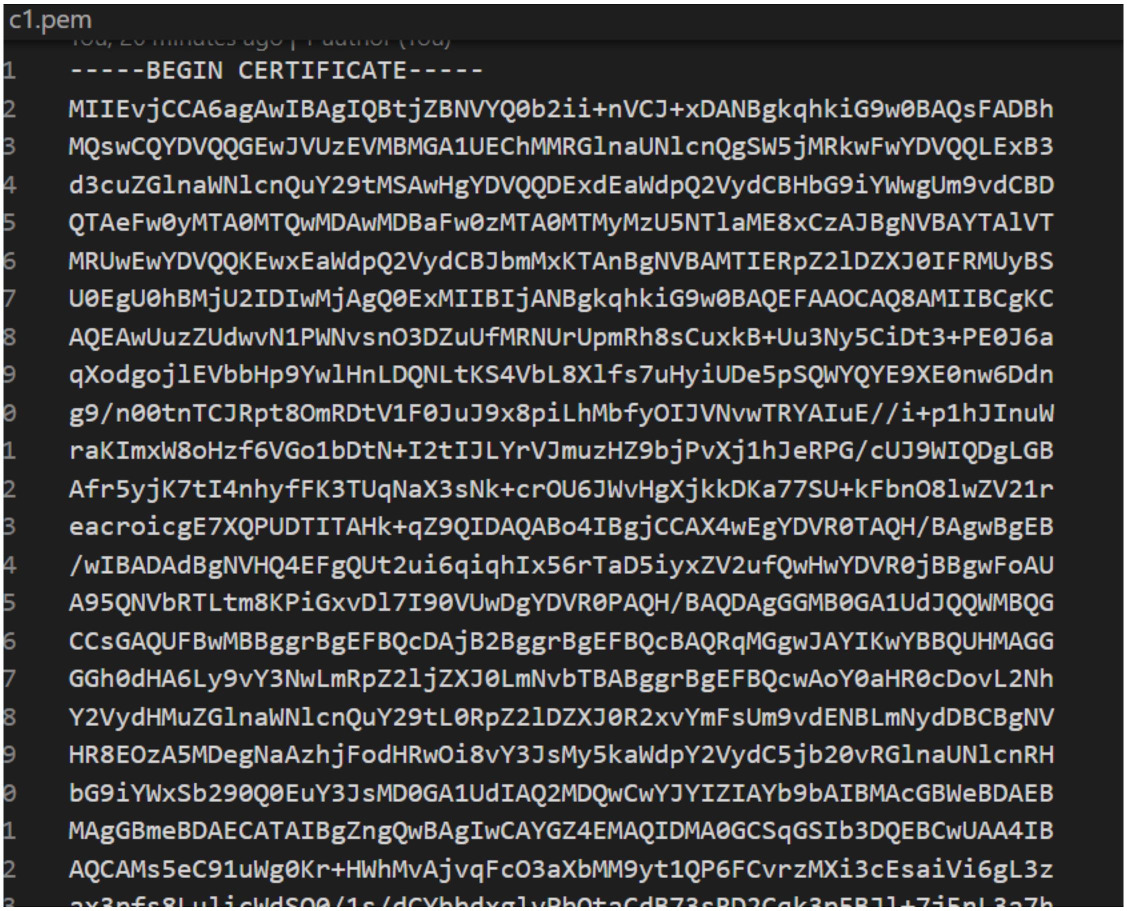
Next I copy 2 certificates into 2 file

c0.pem:



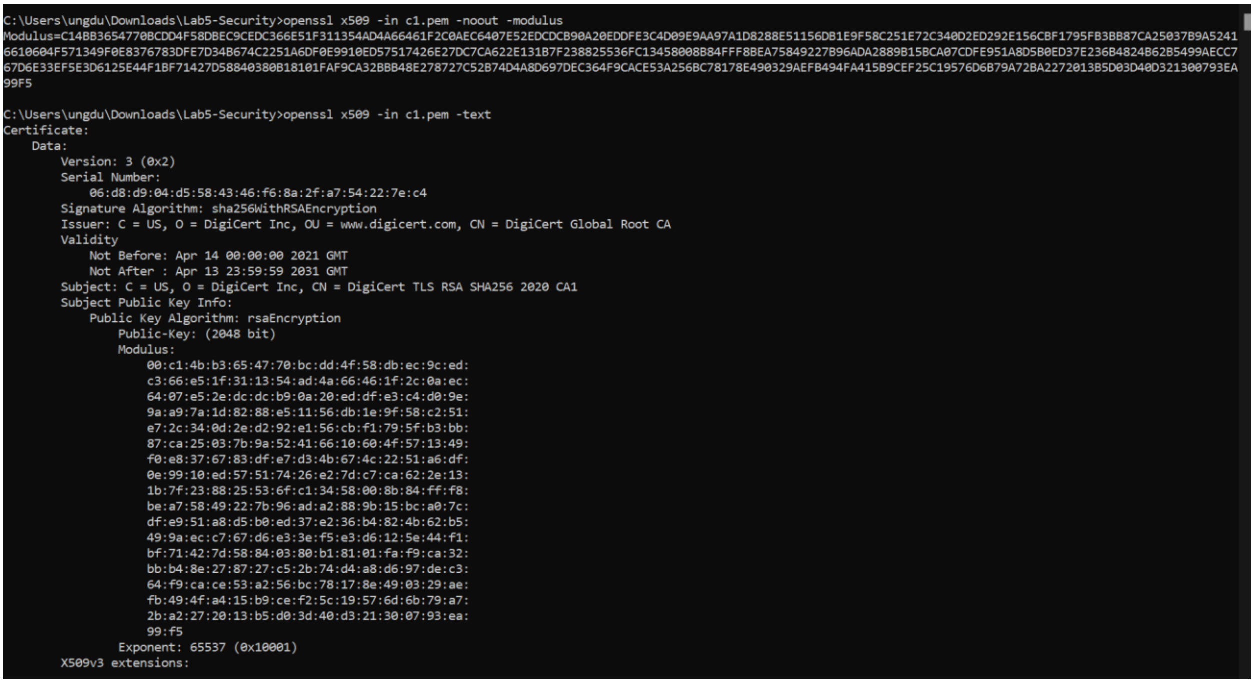
```
VR0TBAIwADCCAX8GCisGAQQB1nkCBAIEggFvBIIBawFpAHYA7s3QZNXbGs7FXLed
tM0TojKHRny87N7DUUhZRnEftZsAAAGFq0gFIwAABAMARzBFAiEAqt+fK6jFdGA6
tv0EWt9rax0WYBV4re9jgZgq0zi42QUCIEBh1yKpPvgX1BreE0wBUmriOVUhJS77
KgF193fT2877AHcAc9meiRtMlnigIH1HneayxhzQUV5xGSqMa4AQesF3crUAAAGF
q0gFnwAABAMASDBGAiEA12SUFK5rgLqRzvgcr7ZzV4nl+Zt9lloAzRLfPc7vSPAC
IQCXPbwScx1rE+BjFawZlVjLj/1PsM0KQQcsfHDZJUTLwAB2AEiw42vapkc0D+Vq
AvqdMOscUgHLVt0sgdm7v6s52IRzAAABhatIBV4AAAQDAEcwRQIhAN5bhHthoyWM
J3CQB/1iYFEhMgUVkFhHDM/nlE9ThCwhAiAPvPJXyp7a2kzwJX3P7fqH5Xko3rPh
CzRoXYd6W+QkCjANBgkqhkiG9w0BAQsFAAOCAQEAWeRK2KmCuppK8WMMbXYmdbM8
dL7F9z2nkZL4zwYtWBDt87jW/Gz/E5YyzU/phySFC3SiwvYP9afYfXaKrunJWCtu
AG+5zSTuxELFTBaFnTRhOSO/xo6VyYSpsuVBD0R415W5z9l0v1hP5xb/fEAwxGxO
Ik3Lg2c6k78rxcWcGvJDoSU7hPb3U26oha7eFHSRMAYN8gfUxAi6Q2TF4j/arMVB
r6Q36EJ2dPcTu0p9NlmBm8dE34lzuTNC6GDCTWFdEloQ9u//M4kUUOjWn8a5XCs1
263t3Ta2JfKViqxpP5r+GvgVKG3qGFrC0mIYr0B4tfpeCY9T+cz4I6GDMSP0xg==
-----END CERTIFICATE-----        You, 25 minutes ago • update
```

c1.pem



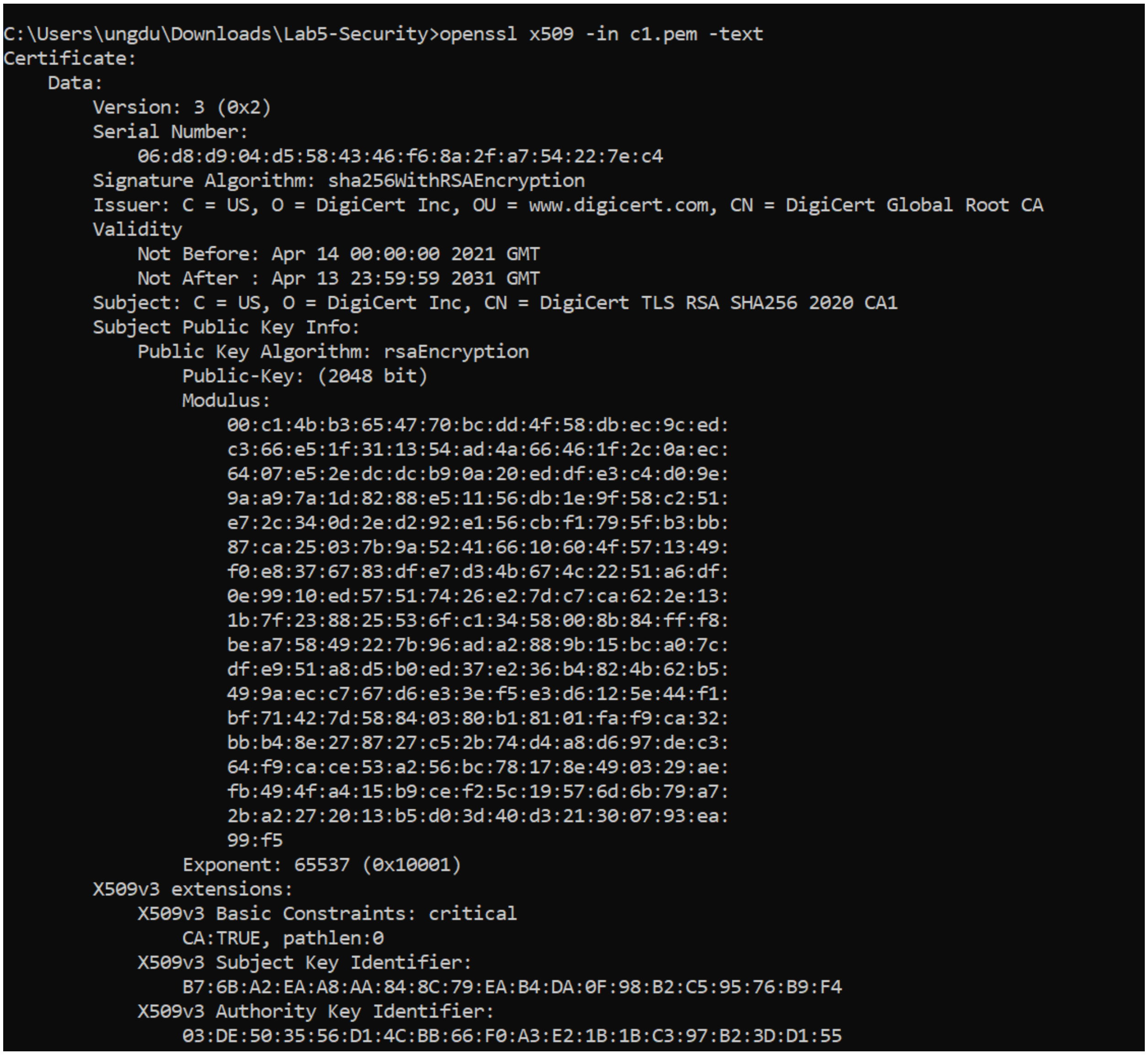
```
You, 20 minutes ago | 1 author (You)
-----BEGIN CERTIFICATE-----
MIIEvjCCA6agAwIBAgIQBtjZBNVYQ0b2ii+nVCJ+xDANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBD
QTAeFw0yMTA0MTQwMDAwMDBaFw0zMTA0MTMyMzU5NTlaME8xCzAJBgNVBAYTAlVT
MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxKTAnBgNVBAMTIERpZ2lDZXJ0IFRMUyBS
U0EgU0hBMjU2IDIwMjAgQ0ExMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKC
AQEAwUuzZUdwvN1PWNvsnO3DZuUfMRNUrUpmRh8sCuxkB+Uu3Ny5CiDt3+PE0J6a
qXodgojlEVbbHp9YwlHnLDQNLtKS4VbL8Xlfs7uHyiUDe5pSQWYQYE9XE0nw6Ddn
g9/n00tnTCJRpt8OmRDtV1F0JuJ9x8piLhMbfyOIJVNvwTRYAIuE//i+p1hJInuW
raKImxW8oHzf6VGo1bDtN+I2tIJLYrVJmuzHZ9bjPvXj1hJeRPG/cUJ9WIQDgLGB
Afr5yjK7tI4nhyfFK3TUqNaX3sNk+crOU6JWvHgXjkkDKa77SU+kFbnO8lwZV21r
eacroicgE7XQPUDTITAHk+qZ9QIDAQABo4IBgjCCAX4wEgYDVR0TAQH/BAgwBgEB
/wIBADAdBgNVHQ4EFgQUt2ui6qiqhIx56rTaD5iyxZV2ufQwHwYDVR0jBBgwFoAU
A95QNVbRTLtm8KPiGxvDl7I90VUwDgYDVR0PAQH/BAQDAgGGMB0GA1UdJQQWMBQG
CCsGAQUFBwMBBggrBgEFBQcDAjB2BggrBgEFBQcBAQRqMGgwJAYIKwYBBQUHMAGG
GGh0dHA6Ly9vY3NwLmRpZ2ljZXJ0LmNvbTBABggrBgEFBQcwAoY0aHR0cDovL2Nh
Y2VydHMuZGlnaWNlcnQuY29tL0RpZ2lDZXJ0R2xvYmFsUm9vdENBLmNydDBCBgNV
HR8EOzA5MDegNaAzhjFodHRwOi8vY3JsMy5kaWdpY2VydC5jb20vRGlnaUNlcnRH
bG9iYWxSb290Q0EuY3JsMD0GA1UdIAQ2MDQwCwYJYIZIAYb9bAIBMAcGBWeBDAEB
MAgGBmeBDAECATAIBgZngQwBAgIwCAYGZ4EMAQIDMA0GCSqGSIb3DQEBCwUAA4IB
AQCAMs5eC91uWg0Kr+HWhMvAjvqFcO3aXbMM9yt1QP6FCvrzMXi3cEsaiVi6gL3z
ax3pfs8LulicWdSQ0/1s/dCYbbdxglyPhOtaCdB73sRD2Cqk3p5BJl+7j5nL3a7b
```

Step 2: Extract the public key (e, n) from the issuer’s certificate



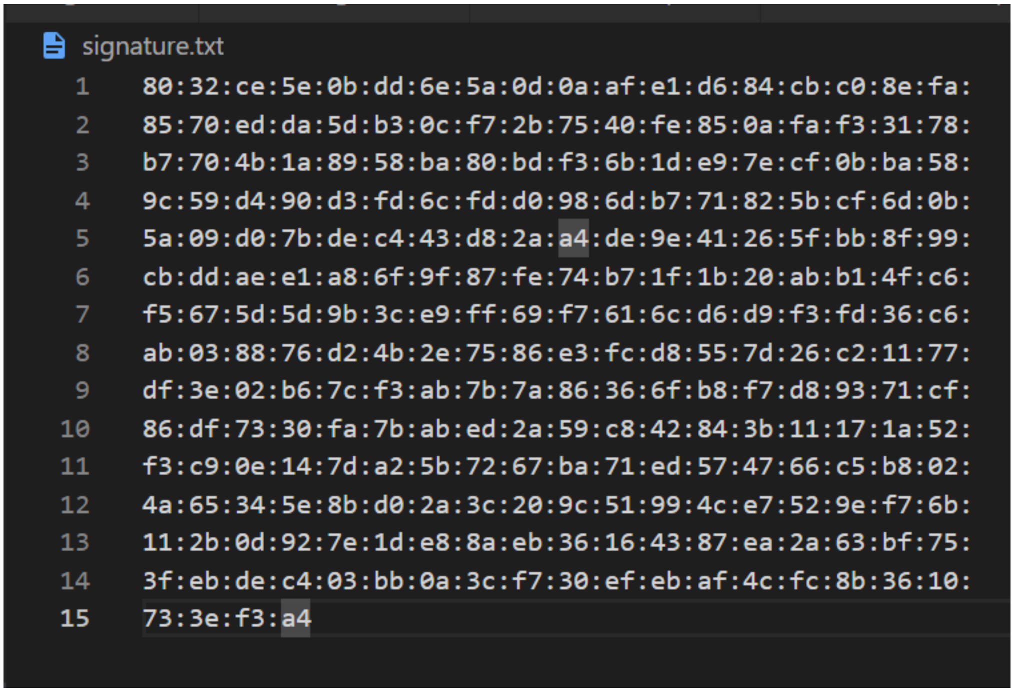
```
C:\Users\ungdu\Downloads\Lab5-Security>openssl x509 -in c1.pem -noout -modulus
Modulus=C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDCB90A20EDDFE3C4D09E9AA97A1D8288E51156DB1E9F58C251E72C340D2ED292E156CBF1795FB3BB87CA25037B9A5241
6610604F571349F0E8376783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA622E131B7F238825536FC13458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8D5B0ED37E236B4824B62B5499AECC7
67D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E278727C52B74D4A8D697DEC364F9CACE53A256BC78178E490329AEFB494FA415B9CEF25C19576D6B79A72BA2272013B5D03D40D321300793EA
99F5

C:\Users\ungdu\Downloads\Lab5-Security>openssl x509 -in c1.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:d8:d9:04:d5:58:43:46:f6:8a:2f:a7:54:22:7e:c4
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
        Validity
            Not Before: Apr 14 00:00:00 2021 GMT
            Not After : Apr 13 23:59:59 2031 GMT
        Subject: C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c1:4b:b3:65:47:70:bc:dd:4f:58:db:ec:9c:ed:
                    c3:66:e5:1f:31:13:54:ad:4a:66:46:1f:2c:0a:ec:
                    64:07:e5:2e:dc:dc:b9:0a:20:ed:df:e3:c4:d0:9e:
                    9a:a9:7a:1d:82:88:e5:11:56:db:1e:9f:58:c2:51:
                    e7:2c:34:0d:2e:d2:92:e1:56:cb:f1:79:5f:b3:bb:
                    87:ca:25:03:7b:9a:52:41:66:10:60:4f:57:13:49:
                    f0:e8:37:67:83:df:e7:d3:4b:67:4c:22:51:a6:df:
                    0e:99:10:ed:57:51:74:26:e2:7d:c7:ca:62:2e:13:
                    1b:7f:23:88:25:53:6f:c1:34:58:00:8b:84:ff:f8:
                    be:a7:58:49:22:7b:96:ad:a2:88:9b:15:bc:a0:7c:
                    df:e9:51:a8:d5:b0:ed:37:e2:36:b4:82:4b:62:b5:
                    49:9a:ec:c7:67:d6:e3:3e:f5:e3:d6:12:5e:44:f1:
                    bf:71:42:7d:58:84:03:80:b1:81:01:fa:f9:ca:32:
                    bb:b4:8e:27:87:27:c5:2b:74:d4:a8:d6:97:de:c3:
                    64:f9:ca:ce:53:a2:56:bc:78:17:8e:49:03:29:ae:
                    fb:49:4f:a4:15:b9:ce:f2:5c:19:57:6d:6b:79:a7:
                    2b:a2:27:20:13:b5:d0:3d:40:d3:21:30:07:93:ea:
                    99:f5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
```

Step 3: Extract the signature from the server’s certificate



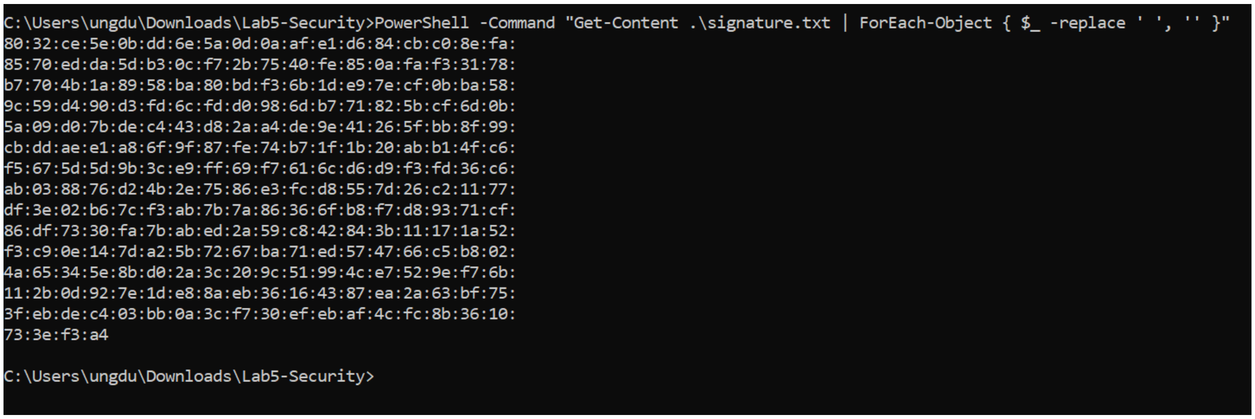
```
C:\Users\ungdu\Downloads\Lab5-Security>openssl x509 -in c1.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:d8:d9:04:d5:58:43:46:f6:8a:2f:a7:54:22:7e:c4
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
        Validity
            Not Before: Apr 14 00:00:00 2021 GMT
            Not After : Apr 13 23:59:59 2031 GMT
        Subject: C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c1:4b:b3:65:47:70:bc:dd:4f:58:db:ec:9c:ed:
                    c3:66:e5:1f:31:13:54:ad:4a:66:46:1f:2c:0a:ec:
                    64:07:e5:2e:dc:dc:b9:0a:20:ed:df:e3:c4:d0:9e:
                    9a:a9:7a:1d:82:88:e5:11:56:db:1e:9f:58:c2:51:
                    e7:2c:34:0d:2e:d2:92:e1:56:cb:f1:79:5f:b3:bb:
                    87:ca:25:03:7b:9a:52:41:66:10:60:4f:57:13:49:
                    f0:e8:37:67:83:df:e7:d3:4b:67:4c:22:51:a6:df:
                    0e:99:10:ed:57:51:74:26:e2:7d:c7:ca:62:2e:13:
                    1b:7f:23:88:25:53:6f:c1:34:58:00:8b:84:ff:f8:
                    be:a7:58:49:22:7b:96:ad:a2:88:9b:15:bc:a0:7c:
                    df:e9:51:a8:d5:b0:ed:37:e2:36:b4:82:4b:62:b5:
                    49:9a:ec:c7:67:d6:e3:3e:f5:e3:d6:12:5e:44:f1:
                    bf:71:42:7d:58:84:03:80:b1:81:01:fa:f9:ca:32:
                    bb:b4:8e:27:87:27:c5:2b:74:d4:a8:d6:97:de:c3:
                    64:f9:ca:ce:53:a2:56:bc:78:17:8e:49:03:29:ae:
                    fb:49:4f:a4:15:b9:ce:f2:5c:19:57:6d:6b:79:a7:
                    2b:a2:27:20:13:b5:d0:3d:40:d3:21:30:07:93:ea:
                    99:f5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Subject Key Identifier:
                B7:6B:A2:EA:A8:AA:84:8C:79:EA:B4:DA:0F:98:B2:C5:95:76:B9:F4
            X509v3 Authority Key Identifier:
                03:DE:50:35:56:D1:4C:BB:66:F0:A3:E2:1B:1B:C3:97:B2:3D:D1:55
```

Here is my signature file based the result of previous comment:



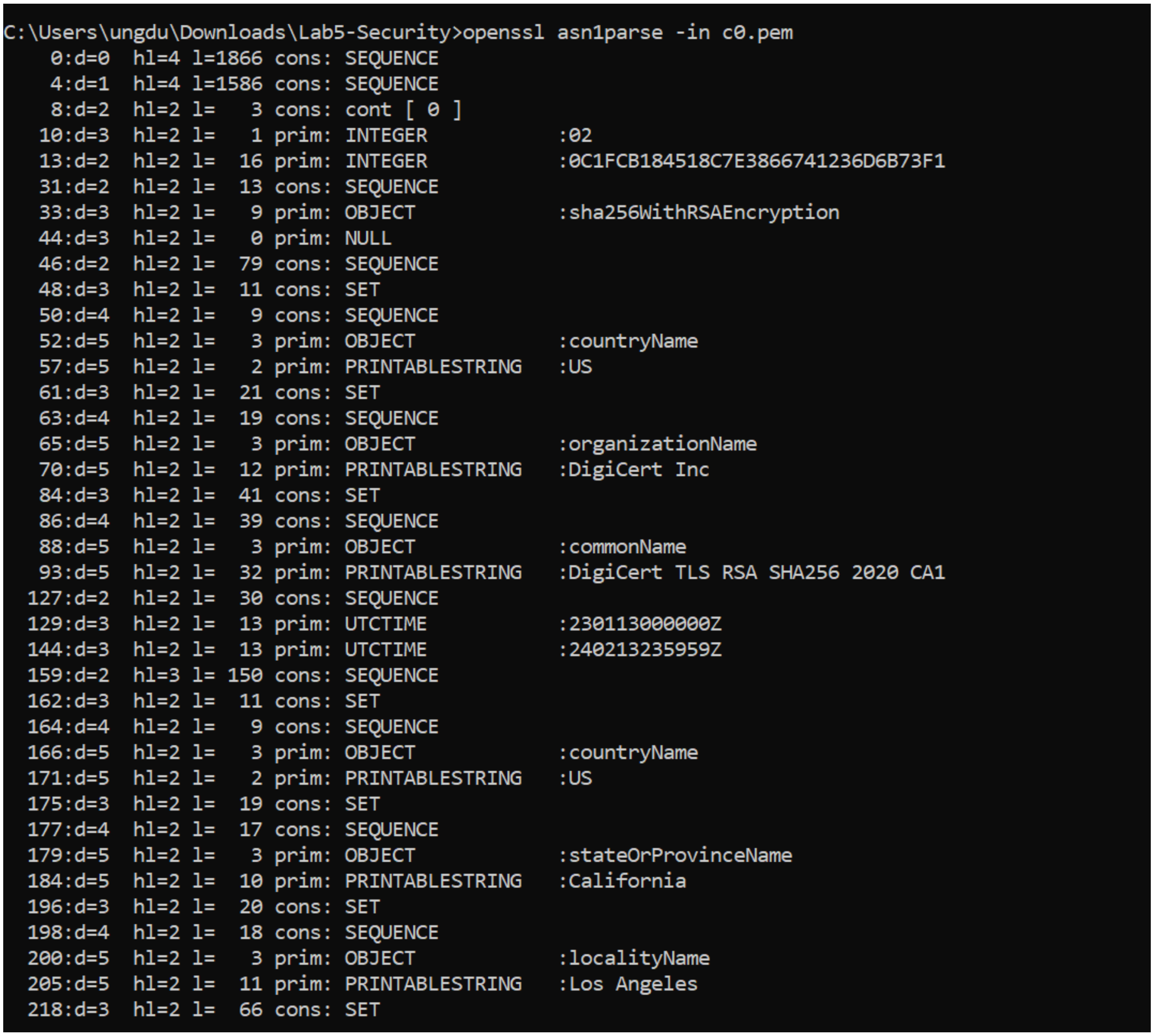
```
80:32:ce:5e:0b:dd:6e:5a:0d:0a:af:e1:d6:84:cb:c0:8e:fa:
85:70:ed:da:5d:b3:0c:f7:2b:75:40:fe:85:0a:fa:f3:31:78:
b7:70:4b:1a:89:58:ba:80:bd:f3:6b:1d:e9:7e:cf:0b:ba:58:
9c:59:d4:90:d3:fd:6c:fd:d0:98:6d:b7:71:82:5b:cf:6d:0b:
5a:09:d0:7b:de:c4:43:d8:2a:a4:de:9e:41:26:5f:bb:8f:99:
cb:dd:ae:e1:a8:6f:9f:87:fe:74:b7:1f:1b:20:ab:b1:4f:c6:
f5:67:5d:5d:9b:3c:e9:ff:69:f7:61:6c:d6:d9:f3:fd:36:c6:
ab:03:88:76:d2:4b:2e:75:86:e3:fc:d8:55:7d:26:c2:11:77:
df:3e:02:b6:7c:f3:ab:7b:7a:86:36:6f:b8:f7:d8:93:71:cf:
86:df:73:30:fa:7b:ab:ed:2a:59:c8:42:84:3b:11:17:1a:52:
f3:c9:0e:14:7d:a2:5b:72:67:ba:71:ed:57:47:66:c5:b8:02:
4a:65:34:5e:8b:d0:2a:3c:20:9c:51:99:4c:e7:52:9e:f7:6b:
11:2b:0d:92:7e:1d:e8:8a:eb:36:16:43:87:ea:2a:63:bf:75:
3f:eb:de:c4:03:bb:0a:3c:f7:30:ef:eb:af:4c:fc:8b:36:10:
73:3e:f3:a4
```

Then I run the code to get this file

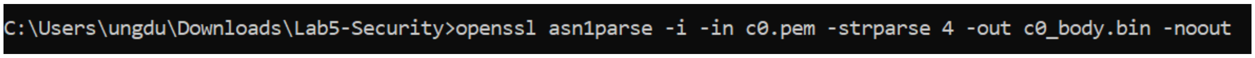


```
C:\Users\ungdu\Downloads\Lab5-Security>PowerShell -Command "Get-Content .\signature.txt | ForEach-Object { $_ -replace ' ', '' }"
80:32:ce:5e:0b:dd:6e:5a:0d:0a:af:e1:d6:84:cb:c0:8e:fa:
85:70:ed:da:5d:b3:0c:f7:2b:75:40:fe:85:0a:fa:f3:31:78:
b7:70:4b:1a:89:58:ba:80:bd:f3:6b:1d:e9:7e:cf:0b:ba:58:
9c:59:d4:90:d3:fd:6c:fd:d0:98:6d:b7:71:82:5b:cf:6d:0b:
5a:09:d0:7b:de:c4:43:d8:2a:a4:de:9e:41:26:5f:bb:8f:99:
cb:dd:ae:e1:a8:6f:9f:87:fe:74:b7:1f:1b:20:ab:b1:4f:c6:
f5:67:5d:5d:9b:3c:e9:ff:69:f7:61:6c:d6:d9:f3:fd:36:c6:
ab:03:88:76:d2:4b:2e:75:86:e3:fc:d8:55:7d:26:c2:11:77:
df:3e:02:b6:7c:f3:ab:7b:7a:86:36:6f:b8:f7:d8:93:71:cf:
86:df:73:30:fa:7b:ab:ed:2a:59:c8:42:84:3b:11:17:1a:52:
f3:c9:0e:14:7d:a2:5b:72:67:ba:71:ed:57:47:66:c5:b8:02:
4a:65:34:5e:8b:d0:2a:3c:20:9c:51:99:4c:e7:52:9e:f7:6b:
11:2b:0d:92:7e:1d:e8:8a:eb:36:16:43:87:ea:2a:63:bf:75:
3f:eb:de:c4:03:bb:0a:3c:f7:30:ef:eb:af:4c:fc:8b:36:10:
73:3e:f3:a4

C:\Users\ungdu\Downloads\Lab5-Security>
```

Step 4: Extract the body of the server’s certificate

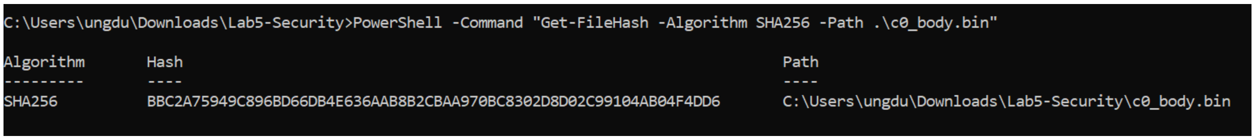


```
C:\Users\ungdu\Downloads\Lab5-Security>openssl asn1parse -in c0.pem
    0:d=0  hl=4 l=1866 cons: SEQUENCE
    4:d=1  hl=4 l=1586 cons: SEQUENCE
    8:d=2  hl=2 l=   3 cons: cont [ 0 ]
   10:d=3  hl=2 l=   1 prim: INTEGER           :02
   13:d=2  hl=2 l=  16 prim: INTEGER           :0C1FCB184518C7E3866741236D6B73F1
   31:d=2  hl=2 l=  13 cons: SEQUENCE
   33:d=3  hl=2 l=   9 prim: OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim: NULL
   46:d=2  hl=2 l=  79 cons: SEQUENCE
   48:d=3  hl=2 l=  11 cons: SET
   50:d=4  hl=2 l=   9 cons: SEQUENCE
   52:d=5  hl=2 l=   3 prim: OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :US
   61:d=3  hl=2 l=  21 cons: SET
   63:d=4  hl=2 l=  19 cons: SEQUENCE
   65:d=5  hl=2 l=   3 prim: OBJECT            :organizationName
   70:d=5  hl=2 l=  12 prim: PRINTABLESTRING   :DigiCert Inc
   84:d=3  hl=2 l=  41 cons: SET
   86:d=4  hl=2 l=  39 cons: SEQUENCE
   88:d=5  hl=2 l=   3 prim: OBJECT            :commonName
   93:d=5  hl=2 l=  32 prim: PRINTABLESTRING   :DigiCert TLS RSA SHA256 2020 CA1
  127:d=2  hl=2 l=  30 cons: SEQUENCE
  129:d=3  hl=2 l=  13 prim: UTCTIME           :230113000000Z
  144:d=3  hl=2 l=  13 prim: UTCTIME           :240213235959Z
  159:d=2  hl=3 l= 150 cons: SEQUENCE
  162:d=3  hl=2 l=  11 cons: SET
  164:d=4  hl=2 l=   9 cons: SEQUENCE
  166:d=5  hl=2 l=   3 prim: OBJECT            :countryName
  171:d=5  hl=2 l=   2 prim: PRINTABLESTRING   :US
  175:d=3  hl=2 l=  19 cons: SET
  177:d=4  hl=2 l=  17 cons: SEQUENCE
  179:d=5  hl=2 l=   3 prim: OBJECT            :stateOrProvinceName
  184:d=5  hl=2 l=  10 prim: PRINTABLESTRING   :California
  196:d=3  hl=2 l=  20 cons: SET
  198:d=4  hl=2 l=  18 cons: SEQUENCE
  200:d=5  hl=2 l=   3 prim: OBJECT            :localityName
  205:d=5  hl=2 l=  11 prim: PRINTABLESTRING   :Los Angeles
  218:d=3  hl=2 l=  66 cons: SET
```



```
C:\Users\ungdu\Downloads\Lab5-Security>openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

So, I get the output of algorithm SHA256 with has value and the path of hash values



```
C:\Users\ungdu\Downloads\Lab5-Security>PowerShell -Command "Get-FileHash -Algorithm SHA256 -Path .\c0_body.bin"

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          BBC2A75949C896BD66DB4E636AAB8B2CBAA970BC8302D8D02C99104AB04F4DD6       C:\Users\ungdu\Downloads\Lab5-Security\c0_body.bin
```

Here is my link on GitHub: https://github.com/amenosakura/Lab5-Security