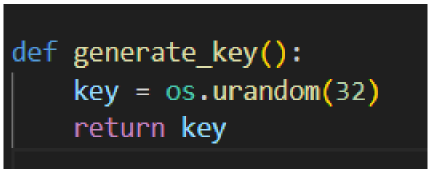
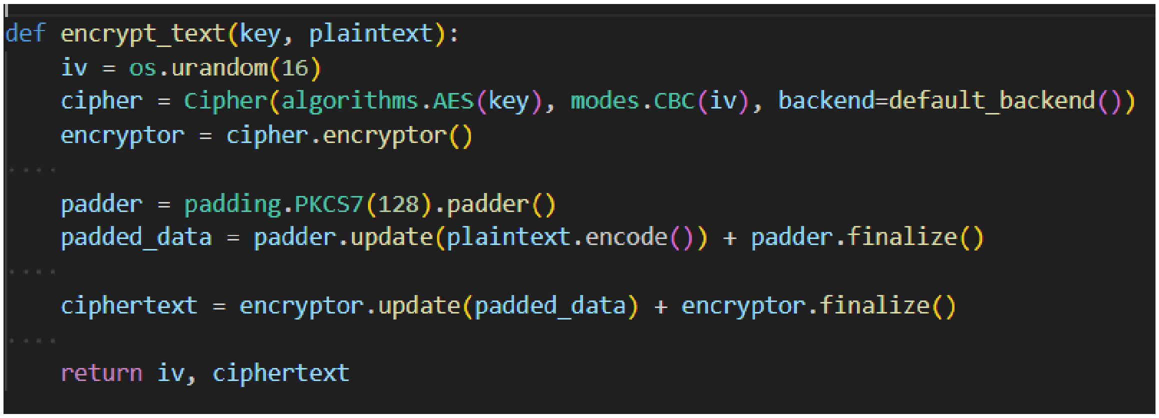
GIT REPO - git@github.com:Cabba4/aliceAndBob.git (private but can add access if requested)

Exercise 1:

- 

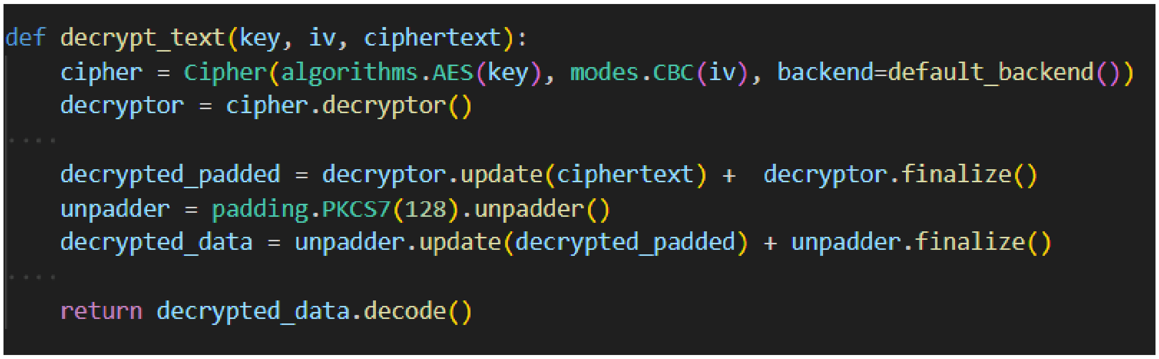
  ```python
  def generate_key():
      key = os.urandom(32)
      return key
  ```

  Key is generated using urandom() function with 32 bytes = 256 bits for AES-256

- 

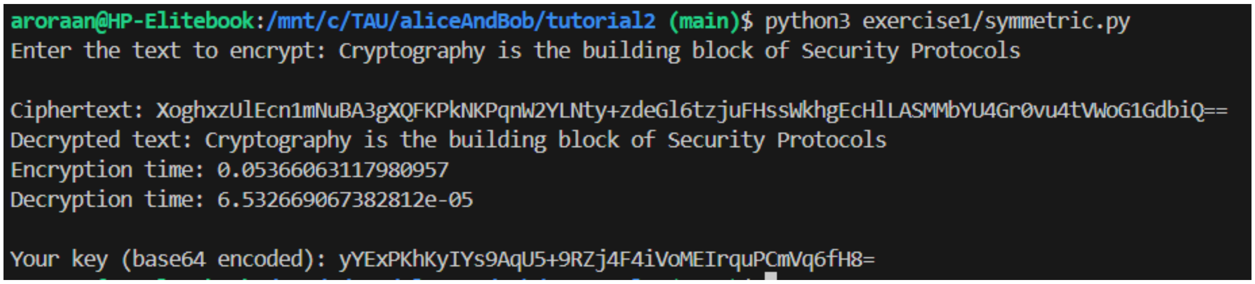
  ```python
  def encrypt_text(key, plaintext):
      iv = os.urandom(16)
      cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
      encryptor = cipher.encryptor()

      padder = padding.PKCS7(128).padder()
      padded_data = padder.update(plaintext.encode()) + padder.finalize()

      ciphertext = encryptor.update(padded_data) + encryptor.finalize()

      return iv, ciphertext
  ```

  User input text is in plaintext and key is generated from generate_key function.

- 

  ```python
  def decrypt_text(key, iv, ciphertext):
      cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
      decryptor = cipher.decryptor()

      decrypted_padded = decryptor.update(ciphertext) +  decryptor.finalize()
      unpadder = padding.PKCS7(128).unpadder()
      decrypted_data = unpadder.update(decrypted_padded) + unpadder.finalize()

      return decrypted_data.decode()
  ```

  Similarly decrypt function uses the same padding, mode and algorithm to decrypt.

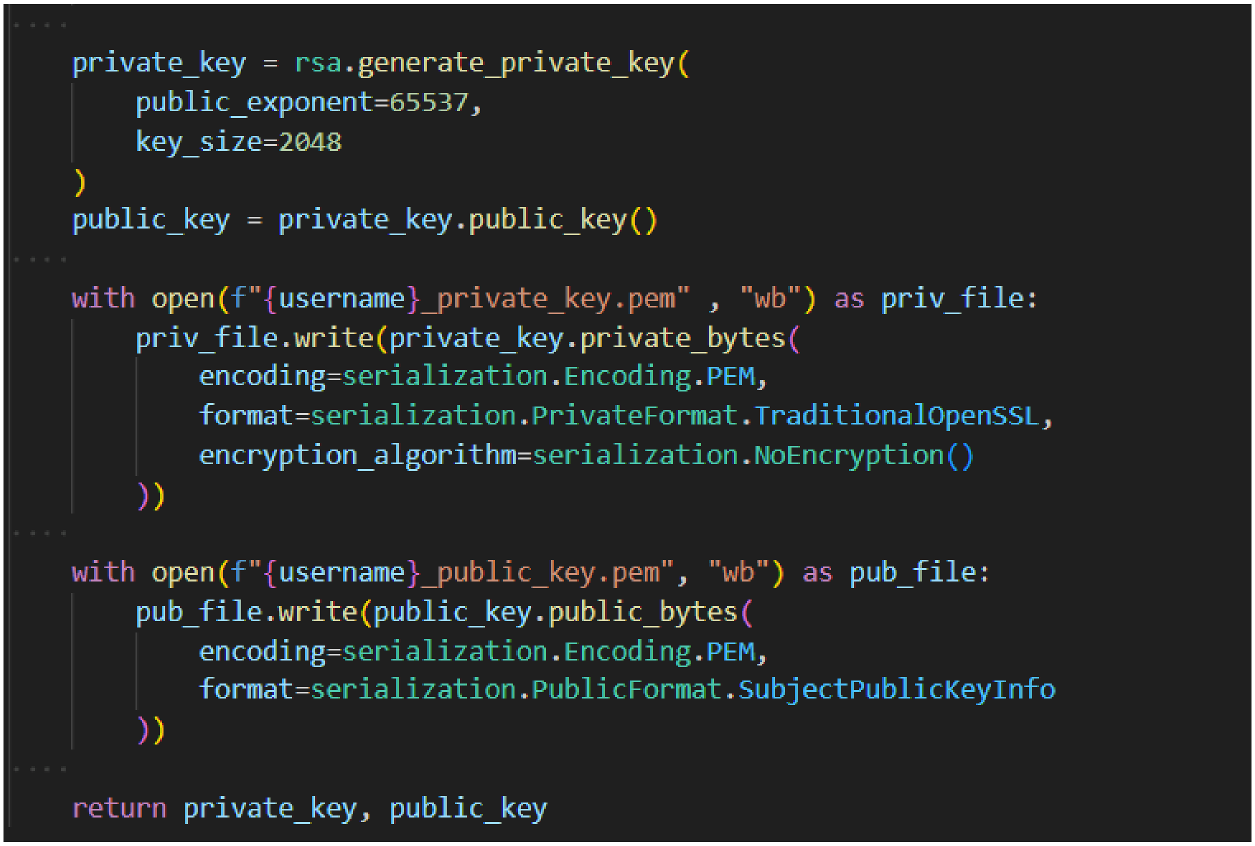
- 

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2 (main)$ python3 exercise1/symmetric.py
Enter the text to encrypt: Cryptography is the building block of Security Protocols

Ciphertext: XoghxzUlEcn1mNuBA3gXQFKPkNKPqnW2YLNty+zdeGl6tzjuFHssWkhgEcHlLASMMbYU4Gr0vu4tVWoG1GdbiQ==
Decrypted text: Cryptography is the building block of Security Protocols
Encryption time: 0.05366063117980957
Decryption time: 6.532669067382812e-05

Your key (base64 encoded): yYExPKhKyIYs9AqU5+9RZj4F4iVoMEIrquPCmVq6fH8=
```

- There is a significant time difference between encryption and decryption functions which was persistent even after running the script multiple times. In theory both operations should take similar times, the variations can be because of improper time measurements from python time module and inner workings of the processor.

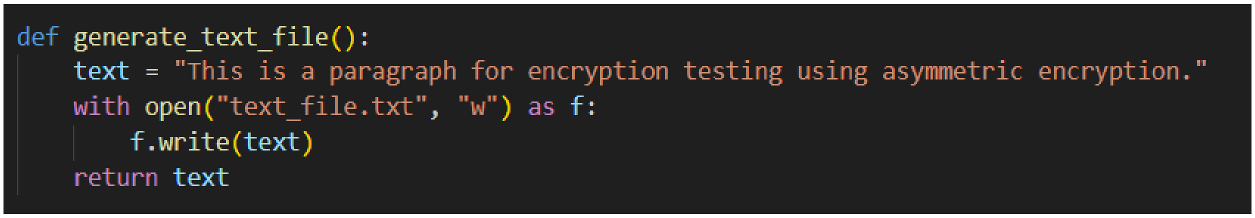
- EMAILED - Ciphertext : w/JXRjpurylpN5BDd09x2PwSrU/o8ilf+m4hX7XIxubTL7gOxdRiZLaoWybTQg4noHr+9m5jE0/N5tj5CHI2BA==
  EMAILED key : YoLVv0PJ7RDLKMrkvIqS0aVBdrFjaFmHuifPV+eFjZ0=

Exercise 2

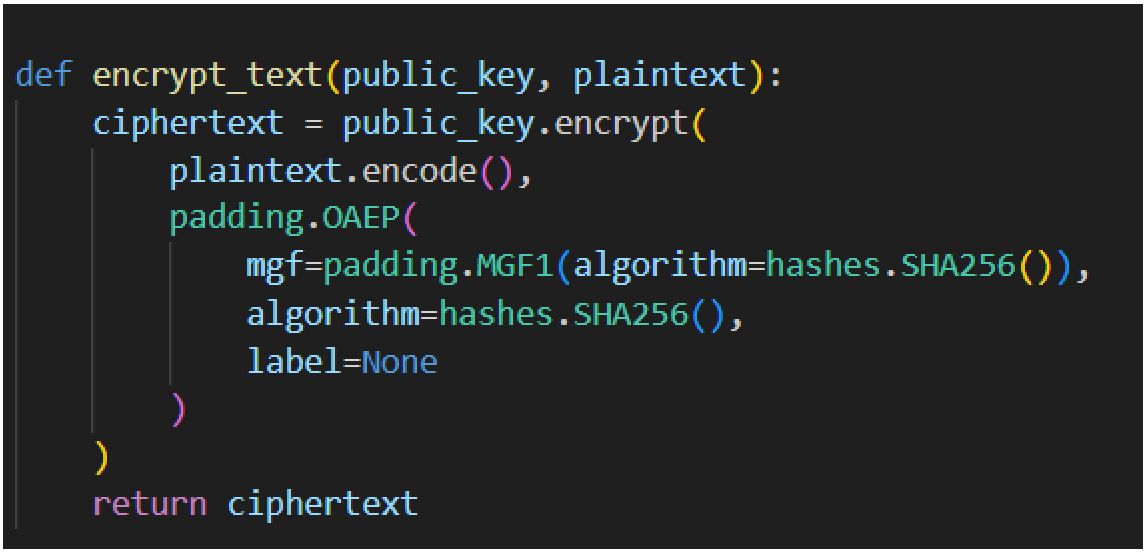


```
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
    )
    public_key = private_key.public_key()

    with open(f"{username}_private_key.pem" , "wb") as priv_file:
        priv_file.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))

    with open(f"{username}_public_key.pem", "wb") as pub_file:
        pub_file.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    return private_key, public_key
```

Function to create rsa keys for given username – omitted check for existing possible keys

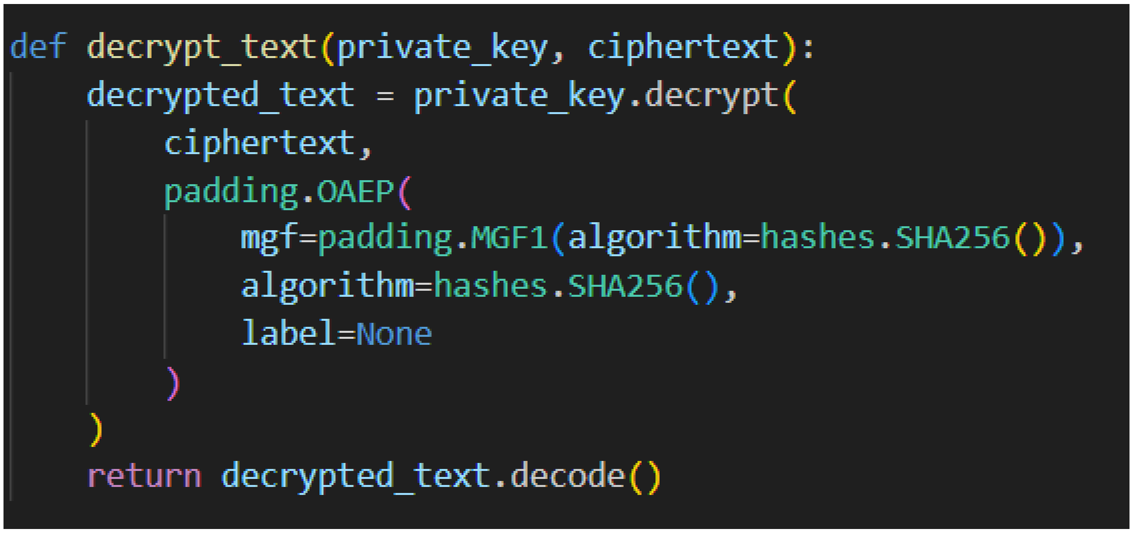


```python
def generate_text_file():
    text = "This is a paragraph for encryption testing using asymmetric encryption."
    with open("text_file.txt", "w") as f:
        f.write(text)
    return text
```



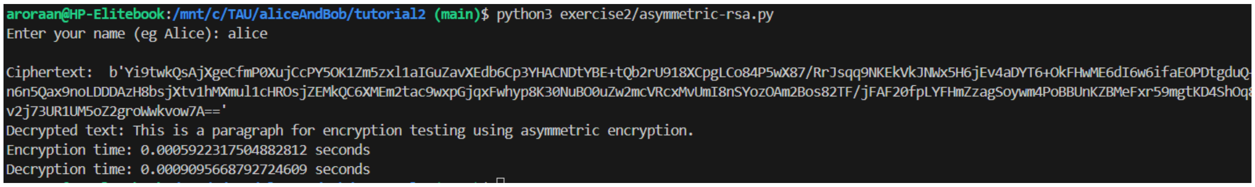
```python
def encrypt_text(public_key, plaintext):
    ciphertext = public_key.encrypt(
        plaintext.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return ciphertext
```

Encryption function using public_key and plaintext



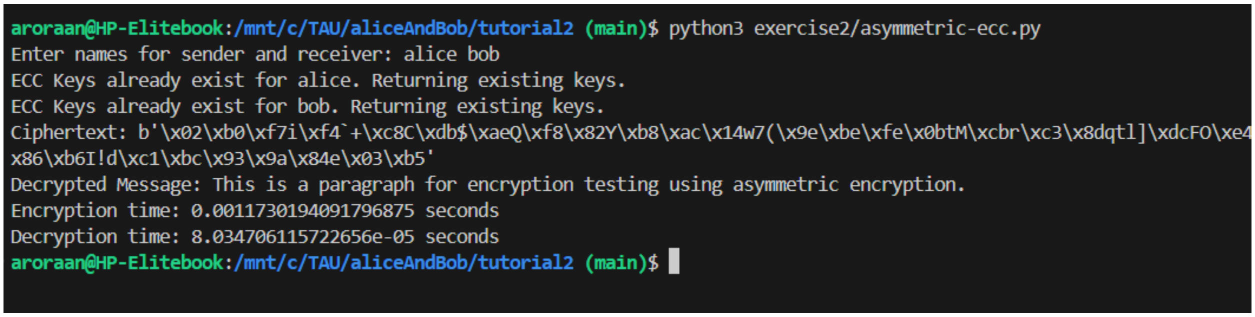
```python
def decrypt_text(private_key, ciphertext):
    decrypted_text = private_key.decrypt(
        ciphertext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_text.decode()
```

Decryption function using private_key

- 

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2 (main)$ python3 exercise2/asymmetric-rsa.py
Enter your name (eg Alice): alice

Ciphertext:  b'Yi9twkQsAjXgeCfmP0XujCcPY5OK1Zm5zxl1aIGuZavXEdb6Cp3YHACNDtYBE+tQb2rU918XCpgLCo84P5wX87/RrJsqq9NKEkVkJNWx5H6jEv4aDYT6+OkFHwME6dI6w6ifaEOPDtgduQ
n6n5Qax9noLDDDAzH8bsjXtv1hMXmul1cHROsjZEMkQC6XMEm2tac9wxpGjqxFwhyp8K30NuBO0uZw2mcVRcxMvUmI8nSYozOAm2Bos82TF/jFAF20fpLYFHmZzagSoywm4PoBBUnKZBMeFxr59mgtKD4ShOq
v2j73UR1UM5oZ2groWwkvow7A=='
Decrypted text: This is a paragraph for encryption testing using asymmetric encryption.
Encryption time: 0.0005922317504882812 seconds
Decryption time: 0.0009095668792724609 seconds
```

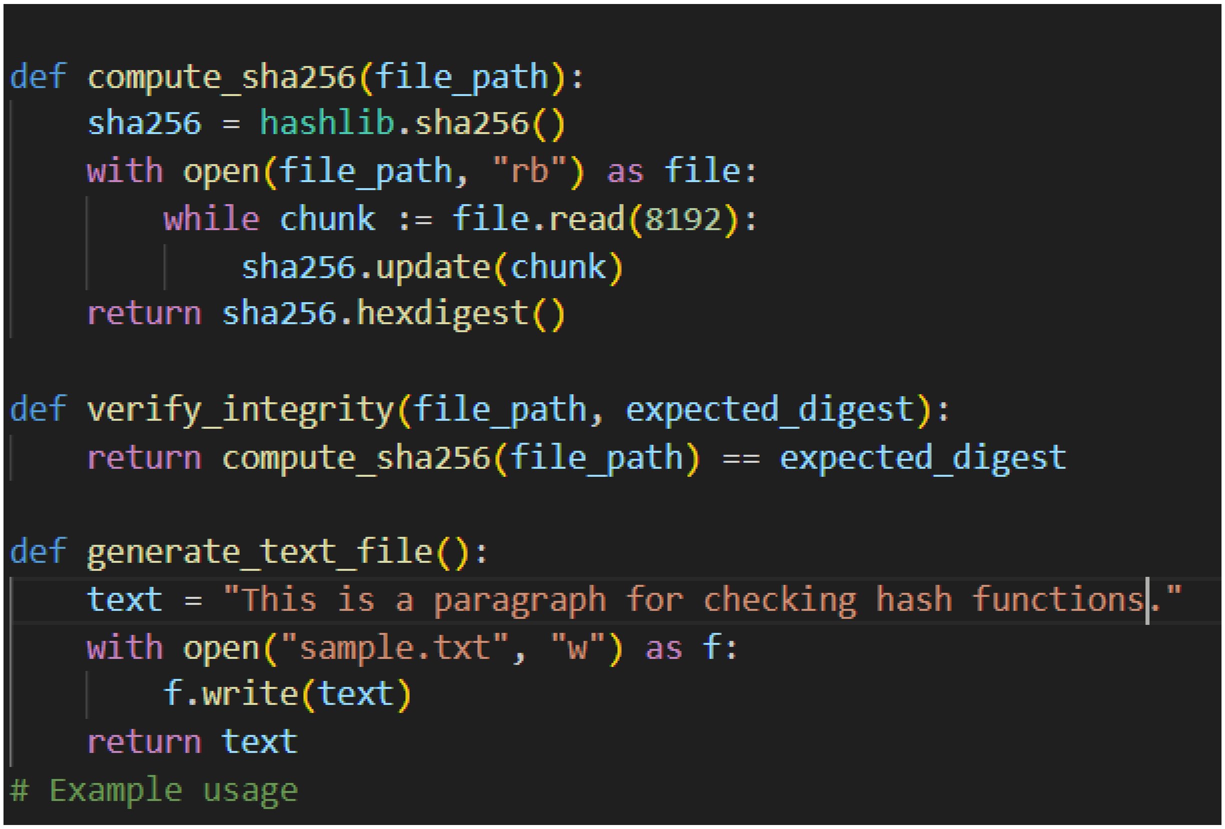
  Encryption is a bit faster than decryption in rsa because encryption involves fewer computations. Decryption involves modular exponentiation with private key and can be slower due to larger key size. Private key > public key.

- 

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2 (main)$ python3 exercise2/asymmetric-ecc.py
Enter names for sender and receiver: alice bob
ECC Keys already exist for alice. Returning existing keys.
ECC Keys already exist for bob. Returning existing keys.
Ciphertext: b'\x02\xb0\xf7i\xf4`+\xc8C\xdb$\xaeQ\xf8\x82Y\xb8\xac\x14w7(\x9e\xbe\xfe\x0btM\xcbr\xc3\x8dqtl]\xdcFO\xe4
x86\xb6I!d\xc1\xbc\x93\x9a\x84e\x03\xb5'
Decrypted Message: This is a paragraph for encryption testing using asymmetric encryption.
Encryption time: 0.0011730194091796875 seconds
Decryption time: 8.034706115722656e-05 seconds
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2 (main)$
```

  Difference between encryption and decryption is minimal. However now we need to generate keys for both alice and bob and then create extra functions to derive shared sending and receiving keys.

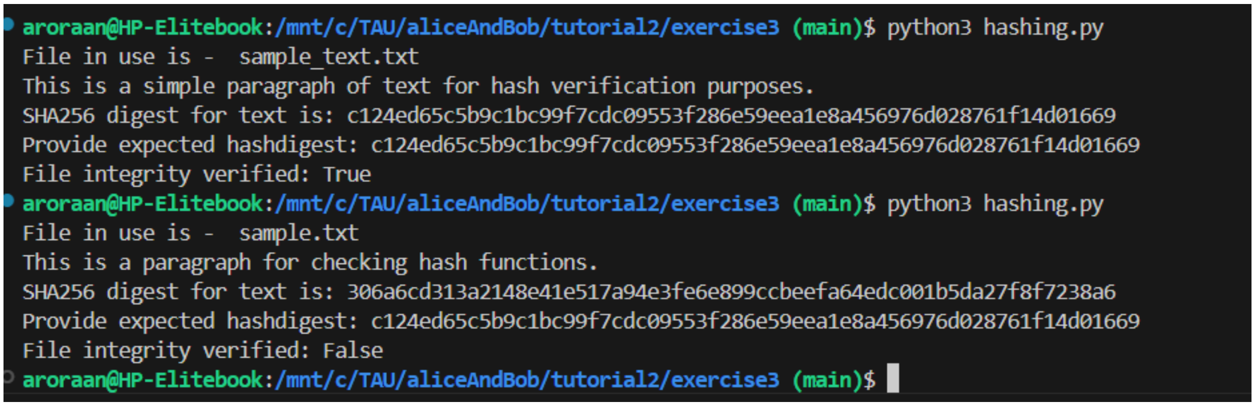
Exercise 3



```python
def compute_sha256(file_path):
    sha256 = hashlib.sha256()
    with open(file_path, "rb") as file:
        while chunk := file.read(8192):
            sha256.update(chunk)
    return sha256.hexdigest()

def verify_integrity(file_path, expected_digest):
    return compute_sha256(file_path) == expected_digest

def generate_text_file():
    text = "This is a paragraph for checking hash functions."
    with open("sample.txt", "w") as f:
        f.write(text)
    return text
# Example usage
```

- Function to compute sha256 hash of provided file
- Also included function to validate integrity of received file.
- Received file – sample_text.text with contents
  “This is a simple paragraph of text for hash verification purposes.”
- Received hash -
  c124ed65c5b9c1bc99f7cdc09553f286e59eea1e8a456976d028761f14d01669

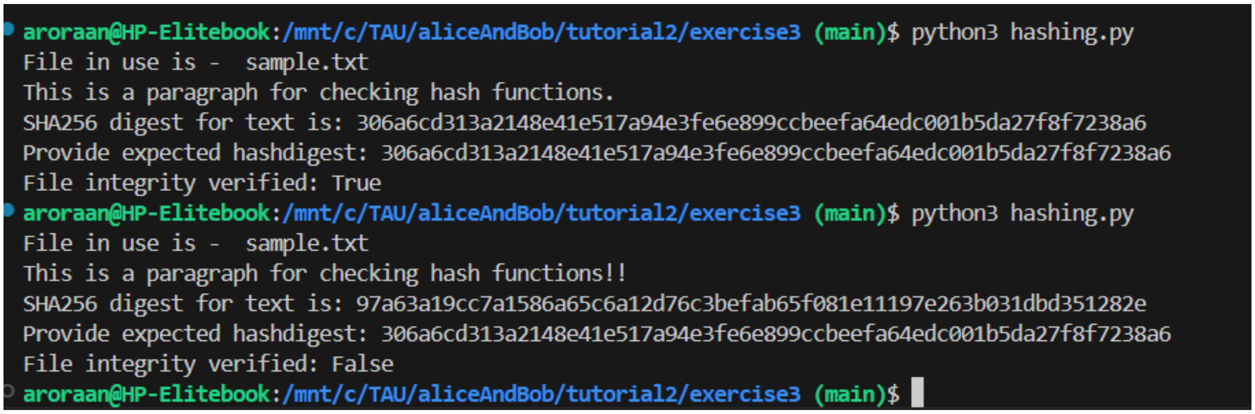
- Integrity check and hashing my text file -
  sample.txt – “This is a simple paragraph of text for hash verification purposes.”
  hash -
  306a6cd313a2148e41e517a94e3fe6e899ccbeefa64edc001b5da27f8f7238a6

- 

```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$ python3 hashing.py
File in use is -  sample_text.txt
This is a simple paragraph of text for hash verification purposes.
SHA256 digest for text is: c124ed65c5b9c1bc99f7cdc09553f286e59eea1e8a456976d028761f14d01669
Provide expected hashdigest: c124ed65c5b9c1bc99f7cdc09553f286e59eea1e8a456976d028761f14d01669
File integrity verified: True
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$ python3 hashing.py
File in use is -  sample.txt
This is a paragraph for checking hash functions.
SHA256 digest for text is: 306a6cd313a2148e41e517a94e3fe6e899ccbeefa64edc001b5da27f8f7238a6
Provide expected hashdigest: c124ed65c5b9c1bc99f7cdc09553f286e59eea1e8a456976d028761f14d01669
File integrity verified: False
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$
```

- First one with sample_text.txt and comparing with hash provided by course mate. Then running with sample.txt which is generated by the generate_text() function.

- Making small change in sample.txt



```
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$ python3 hashing.py
File in use is -  sample.txt
This is a paragraph for checking hash functions.
SHA256 digest for text is: 306a6cd313a2148e41e517a94e3fe6e899ccbeefa64edc001b5da27f8f7238a6
Provide expected hashdigest: 306a6cd313a2148e41e517a94e3fe6e899ccbeefa64edc001b5da27f8f7238a6
File integrity verified: True
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$ python3 hashing.py
File in use is -  sample.txt
This is a paragraph for checking hash functions!!
SHA256 digest for text is: 97a63a19cc7a1586a65c6a12d76c3befab65f081e11197e263b031dbd351282e
Provide expected hashdigest: 306a6cd313a2148e41e517a94e3fe6e899ccbeefa64edc001b5da27f8f7238a6
File integrity verified: False
aroraan@HP-Elitebook:/mnt/c/TAU/aliceAndBob/tutorial2/exercise3 (main)$
```

Only adding two “!!” entirely changed the sha256 digest.