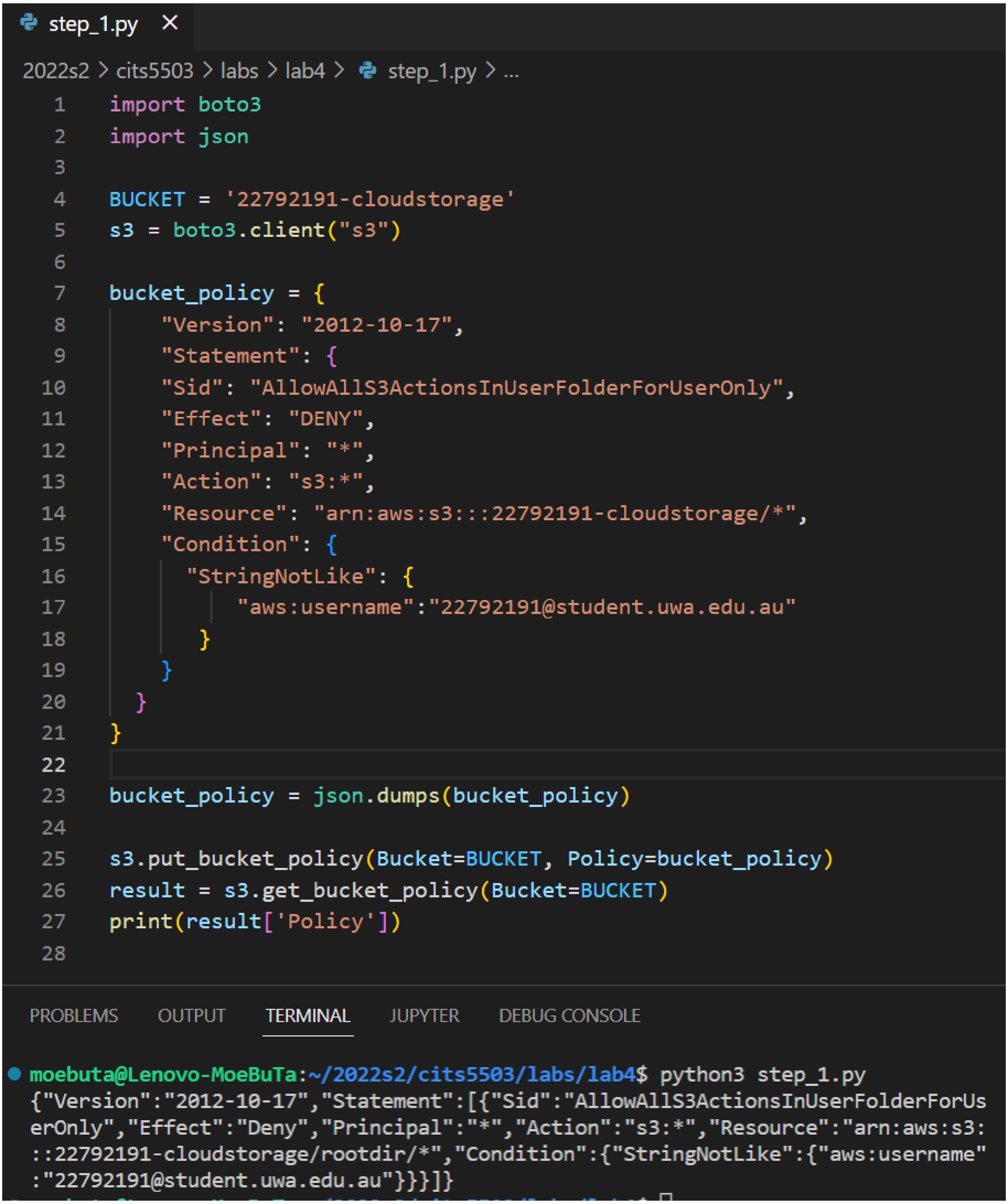
CITS5503 Lab4

Wenxiao Zhang 22792191

[Step 1] Apply policy to restrict permissions on bucket

We use `put_bucket_policy` to set the bucket policy and use `get_bucket_policy` to retrieve the bucket policy. The python script and the output are shown below:

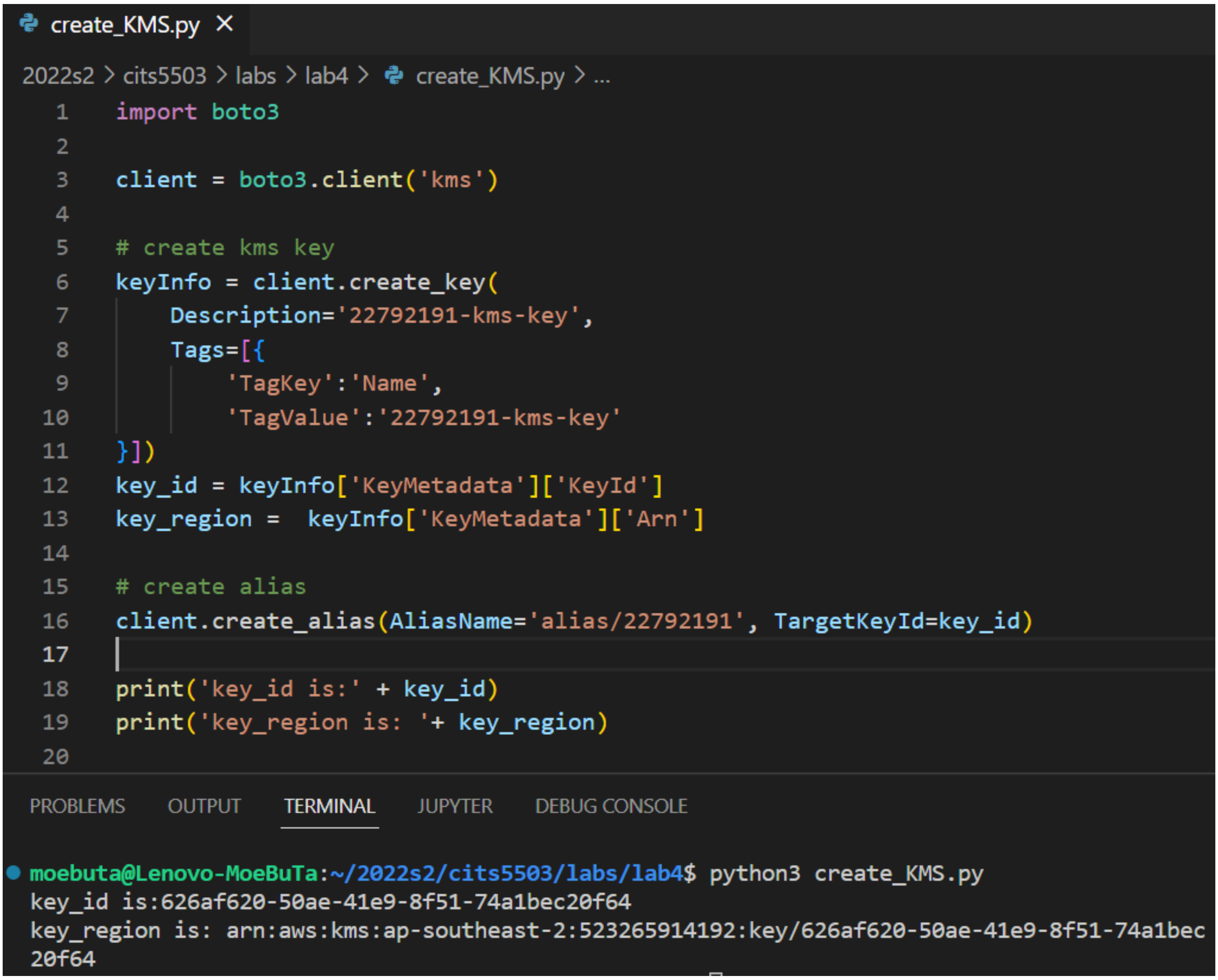


```python
import boto3
import json

BUCKET = '22792191-cloudstorage'
s3 = boto3.client("s3")

bucket_policy = {
    "Version": "2012-10-17",
    "Statement": {
    "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
    "Effect": "DENY",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::22792191-cloudstorage/*",
    "Condition": {
      "StringNotLike": {
          "aws:username":"22792191@student.uwa.edu.au"
       }
    }
  }
}

bucket_policy = json.dumps(bucket_policy)

s3.put_bucket_policy(Bucket=BUCKET, Policy=bucket_policy)
result = s3.get_bucket_policy(Bucket=BUCKET)
print(result['Policy'])

```

```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ python3 step_1.py
{"Version":"2012-10-17","Statement":[{"Sid":"AllowAllS3ActionsInUserFolderForUs
erOnly","Effect":"Deny","Principal":"*","Action":"s3:*","Resource":"arn:aws:s3:
::22792191-cloudstorage/rootdir/*","Condition":{"StringNotLike":{"aws:username"
:"22792191@student.uwa.edu.au"}}}]}
```

[Step 2] AES Encryption using KMS

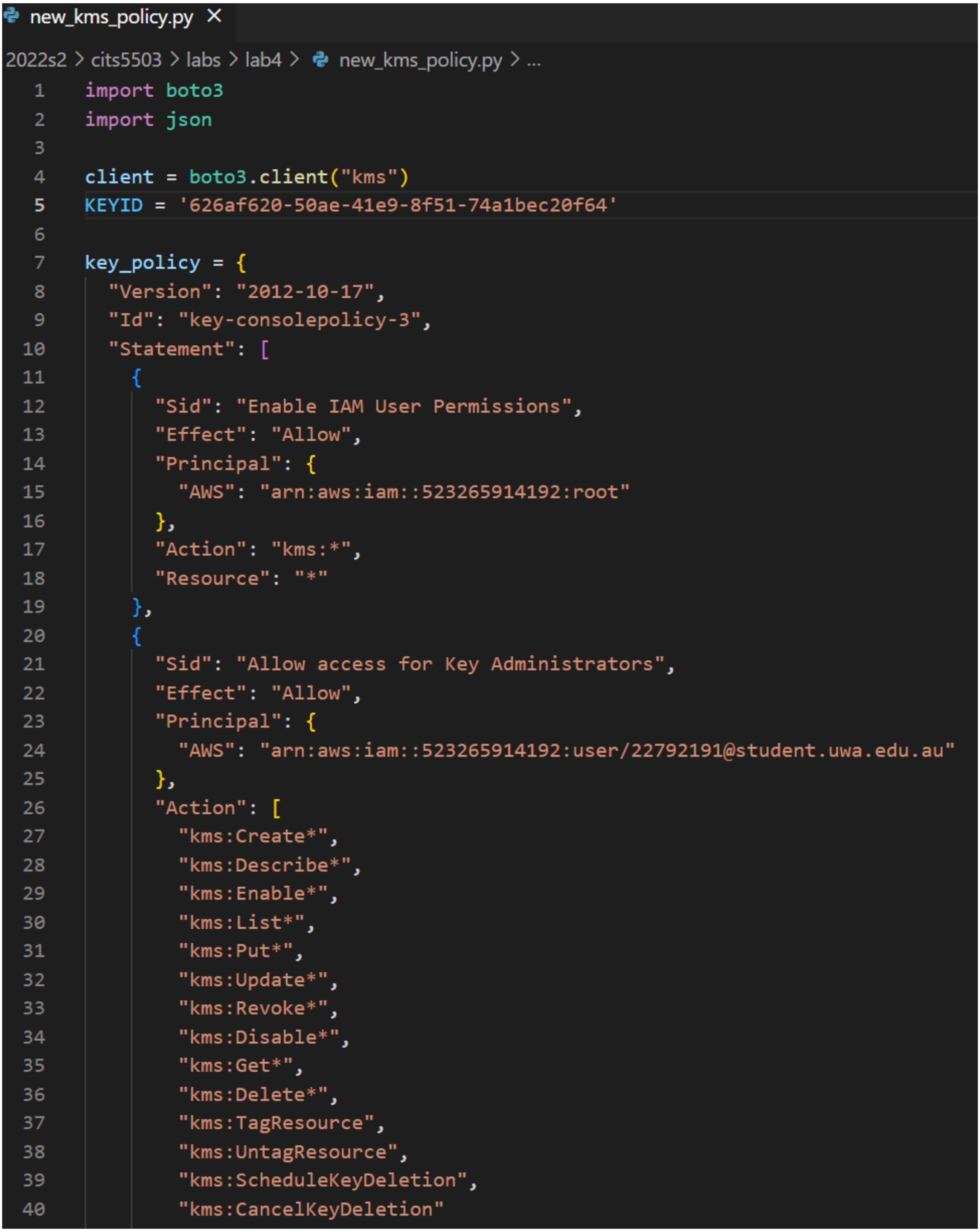
We use `create_key()` and `create_alias()` to create a key and add the alias. The python script and the output are shown below



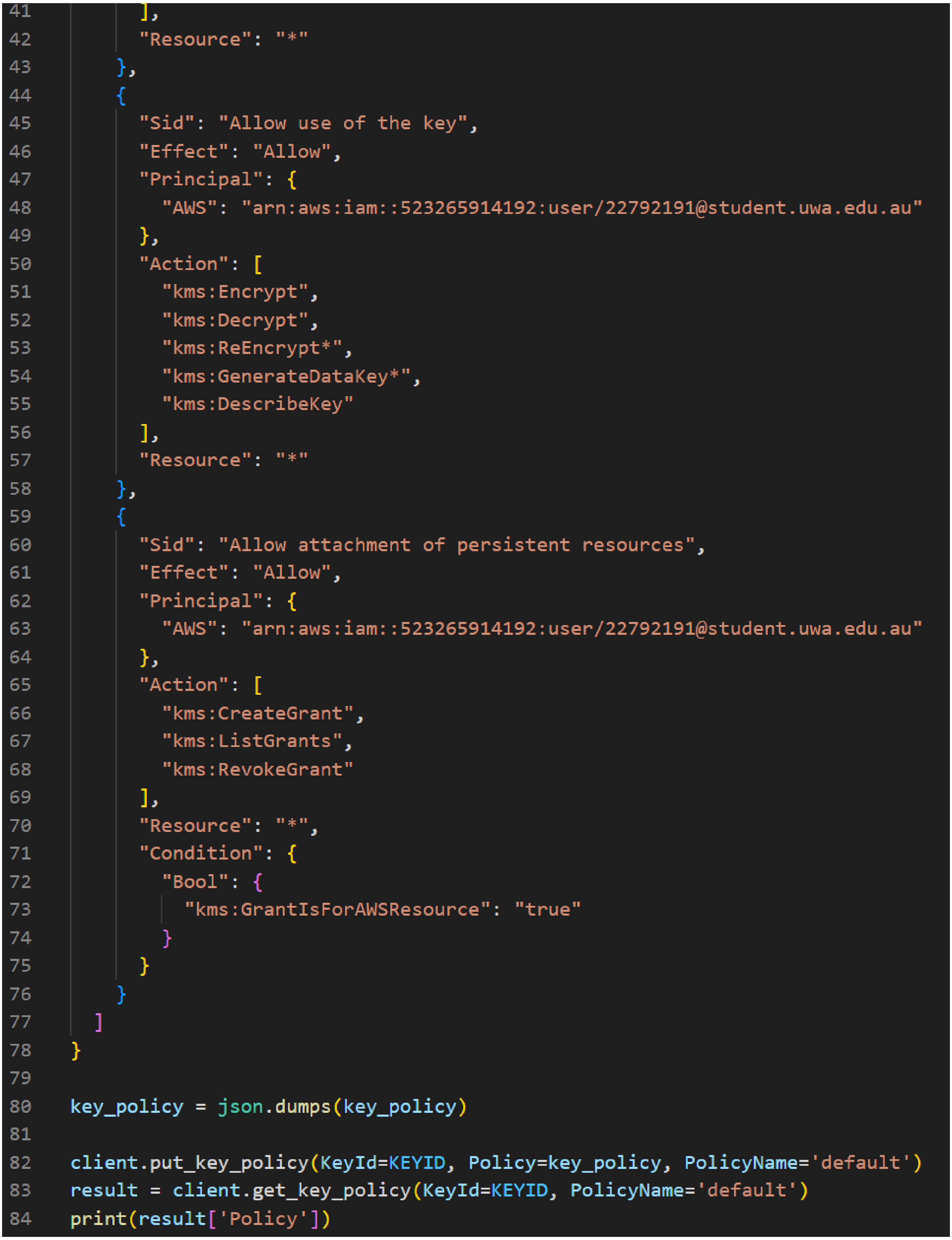
```python
import boto3

client = boto3.client('kms')

# create kms key
keyInfo = client.create_key(
    Description='22792191-kms-key',
    Tags=[{
        'TagKey':'Name',
        'TagValue':'22792191-kms-key'
}])
key_id = keyInfo['KeyMetadata']['KeyId']
key_region =  keyInfo['KeyMetadata']['Arn']

# create alias
client.create_alias(AliasName='alias/22792191', TargetKeyId=key_id)

print('key_id is:' + key_id)
print('key_region is: '+ key_region)

```

```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ python3 create_KMS.py
key_id is:626af620-50ae-41e9-8f51-74a1bec20f64
key_region is: arn:aws:kms:ap-southeast-2:523265914192:key/626af620-50ae-41e9-8f51-74a1bec
20f64
```

We use put_key_policy() set the key policy, and get_key_policy() to retrieve it. The python script is shown below:

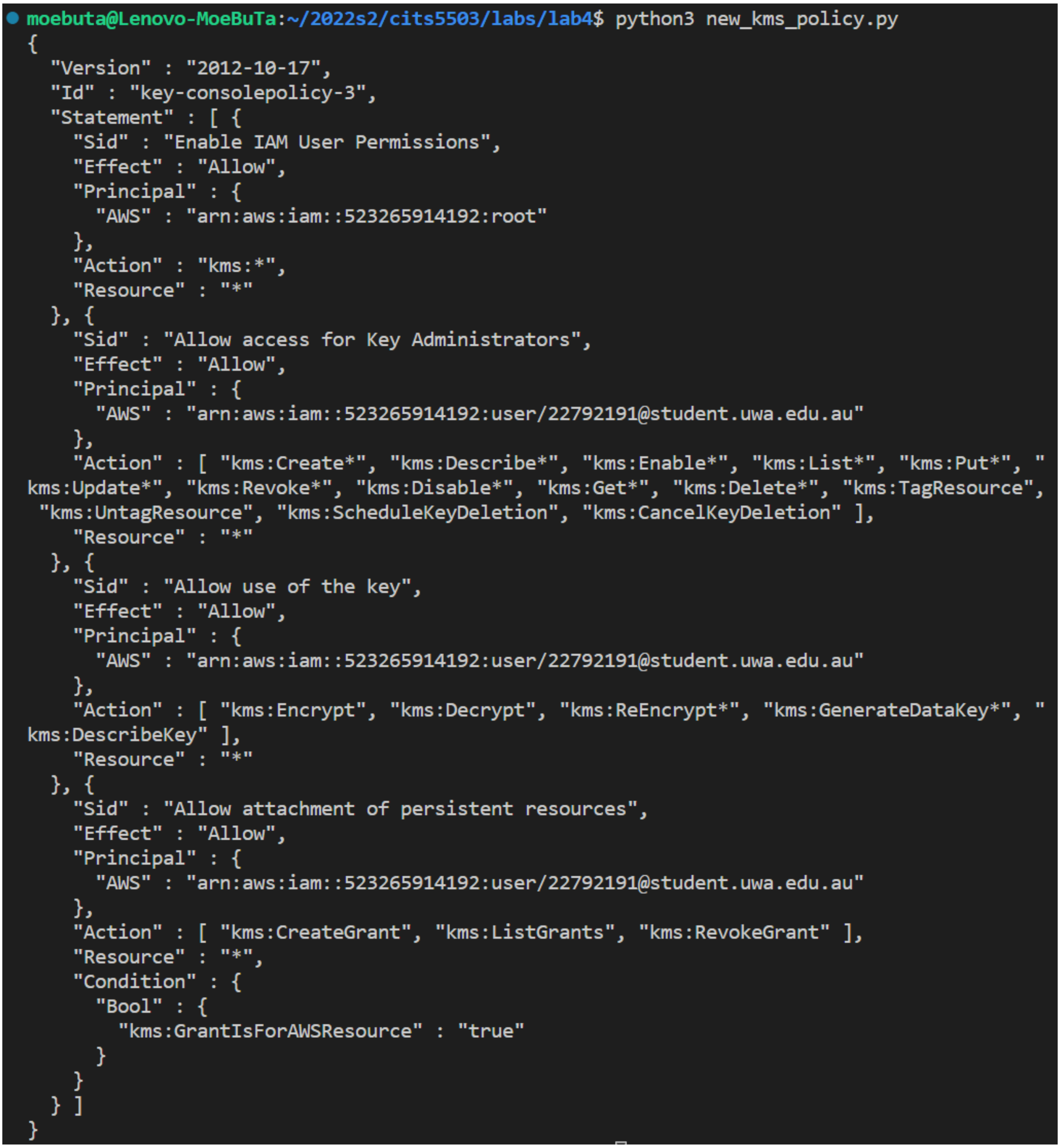


```
import boto3
import json

client = boto3.client("kms")
KEYID = '626af620-50ae-41e9-8f51-74a1bec20f64'

key_policy = {
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-3",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::523265914192:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow access for Key Administrators",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
      },
      "Action": [
        "kms:Create*",
        "kms:Describe*",
        "kms:Enable*",
        "kms:List*",
        "kms:Put*",
        "kms:Update*",
        "kms:Revoke*",
        "kms:Disable*",
        "kms:Get*",
        "kms:Delete*",
        "kms:TagResource",
        "kms:UntagResource",
        "kms:ScheduleKeyDeletion",
        "kms:CancelKeyDeletion"
```



```
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
      },
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    }
  ]
}

key_policy = json.dumps(key_policy)

client.put_key_policy(KeyId=KEYID, Policy=key_policy, PolicyName='default')
result = client.get_key_policy(KeyId=KEYID, PolicyName='default')
print(result['Policy'])
```

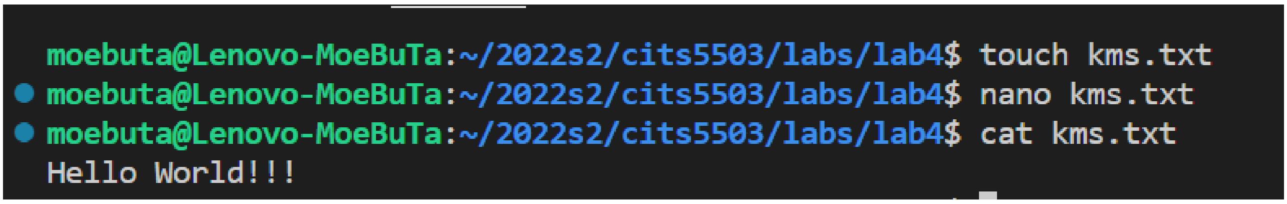
The output is shown below:



```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ python3 new_kms_policy.py
{
  "Version" : "2012-10-17",
  "Id" : "key-consolepolicy-3",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::523265914192:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  }, {
    "Sid" : "Allow access for Key Administrators",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
    },
    "Action" : [ "kms:Create*", "kms:Describe*", "kms:Enable*", "kms:List*", "kms:Put*", "
kms:Update*", "kms:Revoke*", "kms:Disable*", "kms:Get*", "kms:Delete*", "kms:TagResource",
 "kms:UntagResource", "kms:ScheduleKeyDeletion", "kms:CancelKeyDeletion" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow use of the key",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*", "kms:GenerateDataKey*", "
kms:DescribeKey" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow attachment of persistent resources",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::523265914192:user/22792191@student.uwa.edu.au"
    },
    "Action" : [ "kms:CreateGrant", "kms:ListGrants", "kms:RevokeGrant" ],
    "Resource" : "*",
    "Condition" : {
      "Bool" : {
        "kms:GrantIsForAWSResource" : "true"
      }
    }
  } ]
}
```

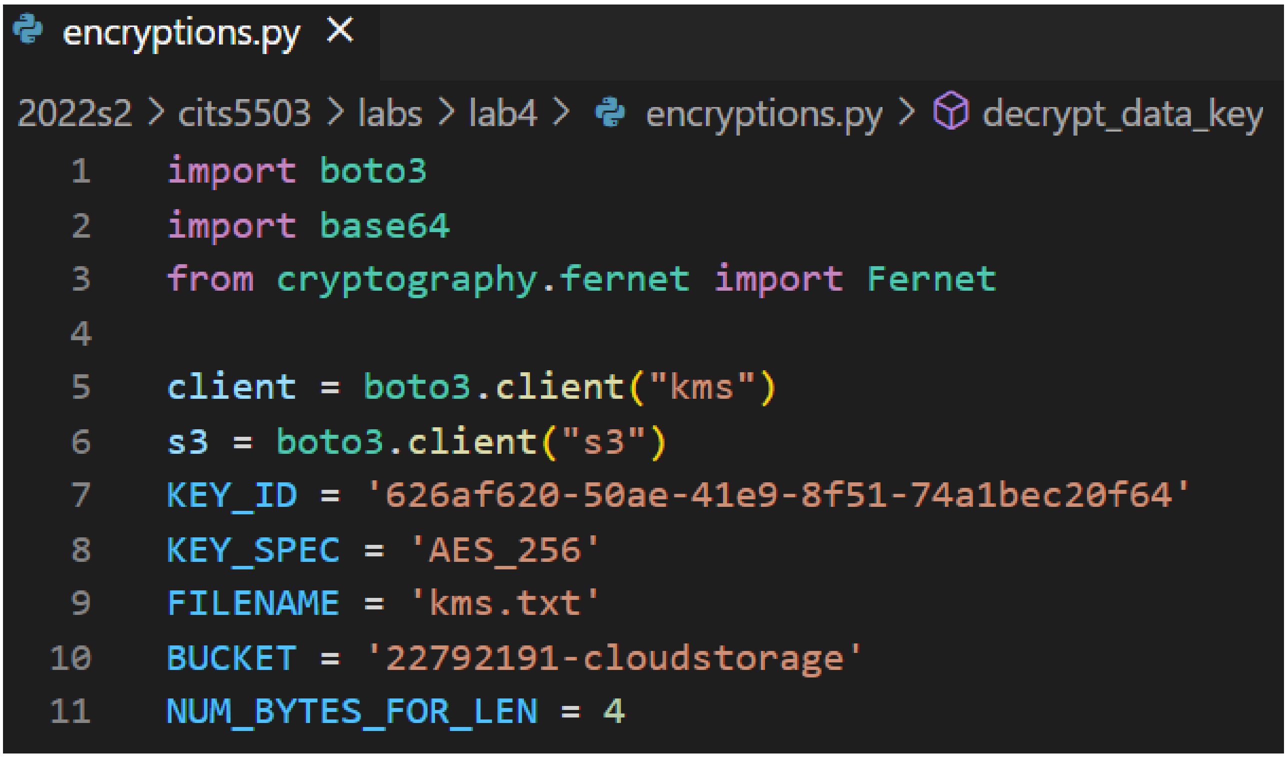
The following works for encryptions and decryptions are written in a python file called `encryptions.py`.

1. before we executing the script, we create a text file `kms.txt` and write `Hello World!!!` into it.



```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ touch kms.txt
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ nano kms.txt
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ cat kms.txt
Hello World!!!
```

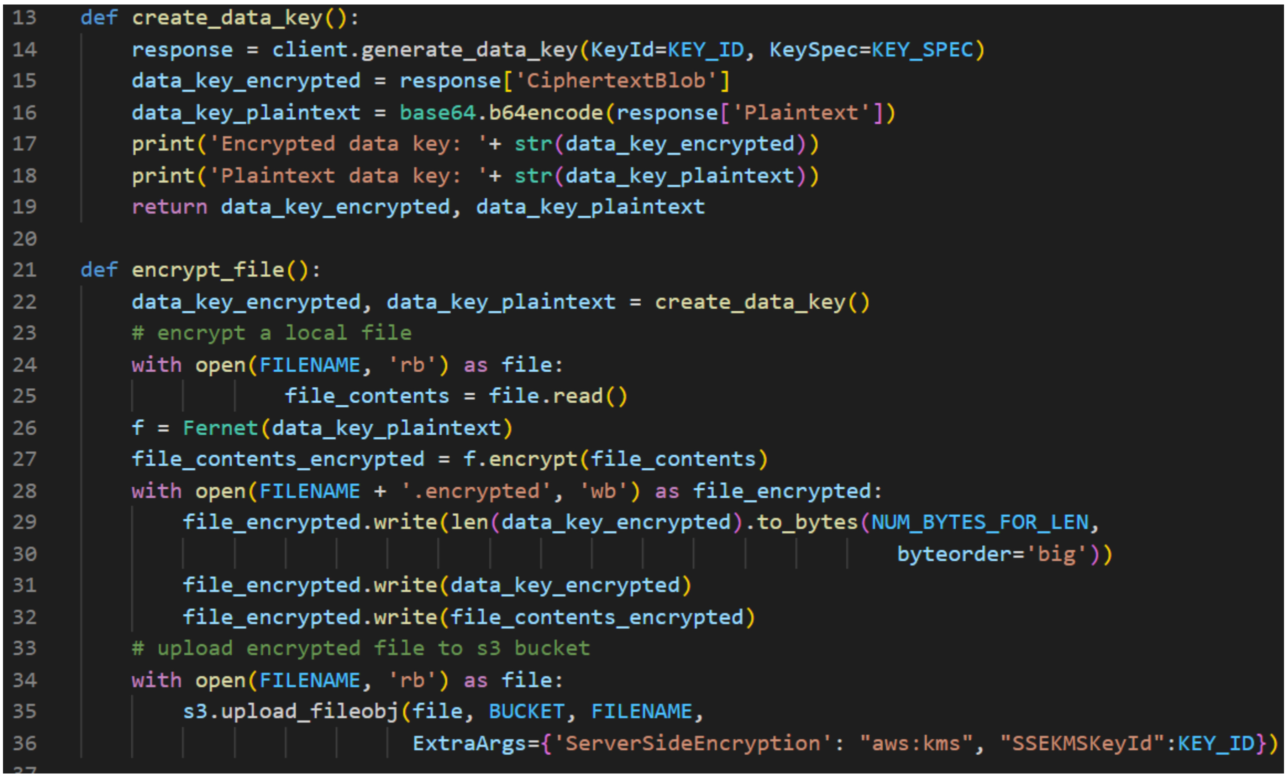
2. import libraries and define necessary variables.



encryptions.py ✕
2022s2 > cits5503 > labs > lab4 > encryptions.py > decrypt_data_key

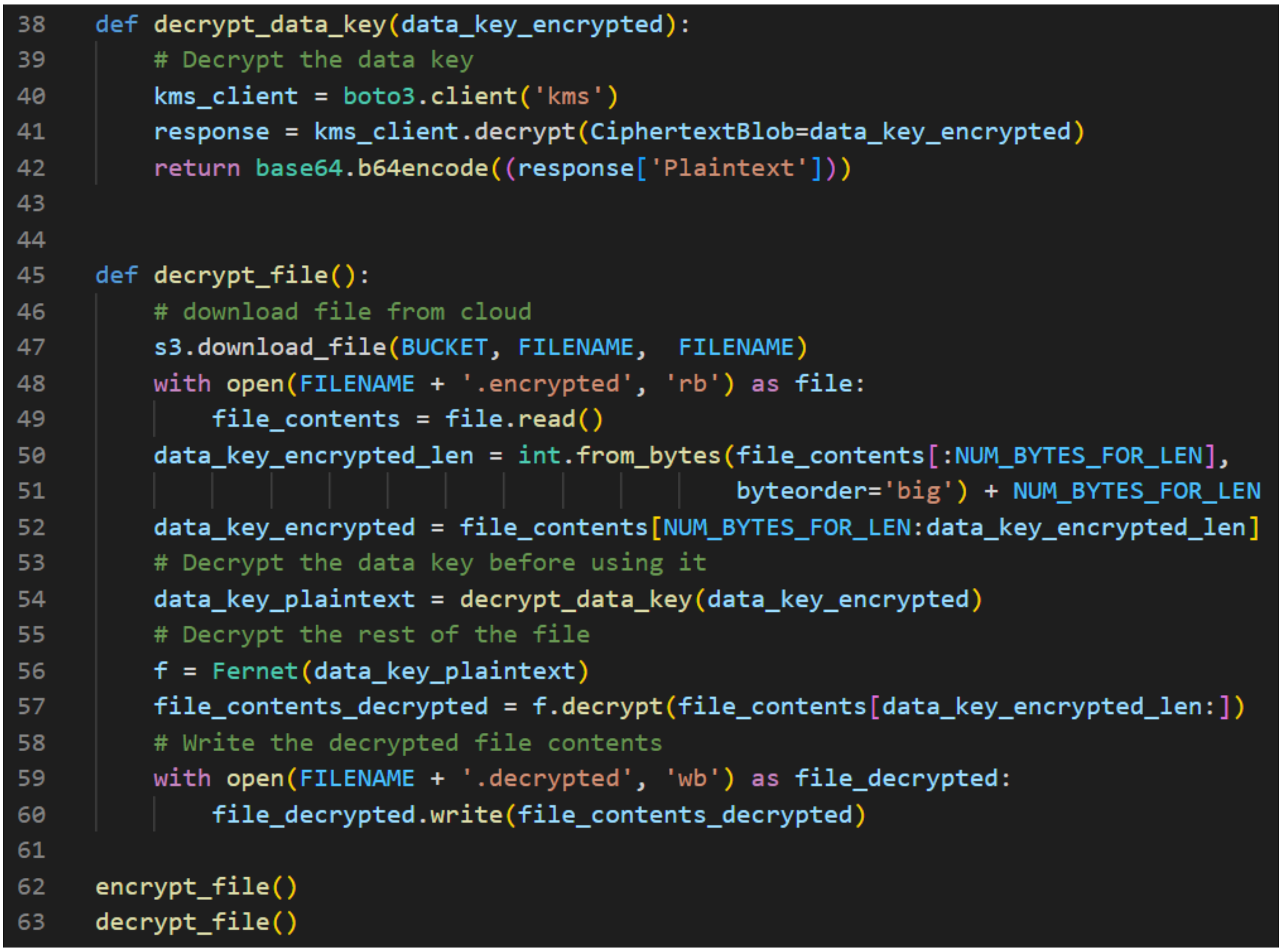
```python
import boto3
import base64
from cryptography.fernet import Fernet

client = boto3.client("kms")
s3 = boto3.client("s3")
KEY_ID = '626af620-50ae-41e9-8f51-74a1bec20f64'
KEY_SPEC = 'AES_256'
FILENAME = 'kms.txt'
BUCKET = '22792191-cloudstorage'
NUM_BYTES_FOR_LEN = 4
```

3. create_data_key() is for generating the data key from the previous generated kms key. encrypt_file() is for encrypting kms.txt and uploading it to the bucket.



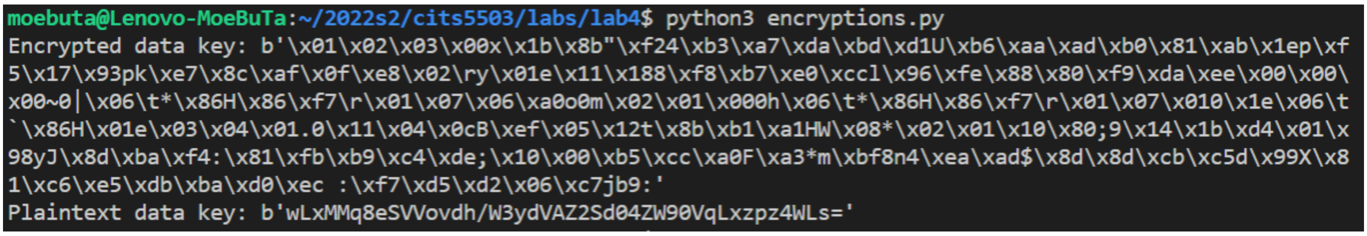
```
def create_data_key():
    response = client.generate_data_key(KeyId=KEY_ID, KeySpec=KEY_SPEC)
    data_key_encrypted = response['CiphertextBlob']
    data_key_plaintext = base64.b64encode(response['Plaintext'])
    print('Encrypted data key: '+ str(data_key_encrypted))
    print('Plaintext data key: '+ str(data_key_plaintext))
    return data_key_encrypted, data_key_plaintext

def encrypt_file():
    data_key_encrypted, data_key_plaintext = create_data_key()
    # encrypt a local file
    with open(FILENAME, 'rb') as file:
                file_contents = file.read()
    f = Fernet(data_key_plaintext)
    file_contents_encrypted = f.encrypt(file_contents)
    with open(FILENAME + '.encrypted', 'wb') as file_encrypted:
        file_encrypted.write(len(data_key_encrypted).to_bytes(NUM_BYTES_FOR_LEN,
                                                                byteorder='big'))
        file_encrypted.write(data_key_encrypted)
        file_encrypted.write(file_contents_encrypted)
    # upload encrypted file to s3 bucket
    with open(FILENAME, 'rb') as file:
        s3.upload_fileobj(file, BUCKET, FILENAME,
                          ExtraArgs={'ServerSideEncryption': "aws:kms", "SSEKMSKeyId":KEY_ID})
```

4. decrypt_data_key() is for decrypting the data key from the encrypted data key. decrypt_file() is for decrypting the file kms.txt downloaded from the bucket.



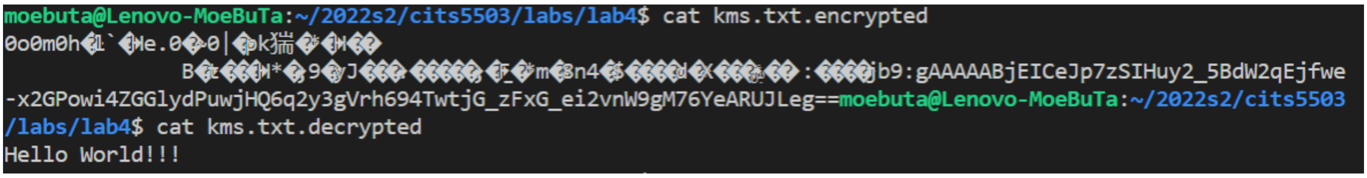
```
def decrypt_data_key(data_key_encrypted):
    # Decrypt the data key
    kms_client = boto3.client('kms')
    response = kms_client.decrypt(CiphertextBlob=data_key_encrypted)
    return base64.b64encode((response['Plaintext']))


def decrypt_file():
    # download file from cloud
    s3.download_file(BUCKET, FILENAME,  FILENAME)
    with open(FILENAME + '.encrypted', 'rb') as file:
        file_contents = file.read()
    data_key_encrypted_len = int.from_bytes(file_contents[:NUM_BYTES_FOR_LEN],
                                            byteorder='big') + NUM_BYTES_FOR_LEN
    data_key_encrypted = file_contents[NUM_BYTES_FOR_LEN:data_key_encrypted_len]
    # Decrypt the data key before using it
    data_key_plaintext = decrypt_data_key(data_key_encrypted)
    # Decrypt the rest of the file
    f = Fernet(data_key_plaintext)
    file_contents_decrypted = f.decrypt(file_contents[data_key_encrypted_len:])
    # Write the decrypted file contents
    with open(FILENAME + '.decrypted', 'wb') as file_decrypted:
        file_decrypted.write(file_contents_decrypted)

encrypt_file()
decrypt_file()
```

The output for generating data key:



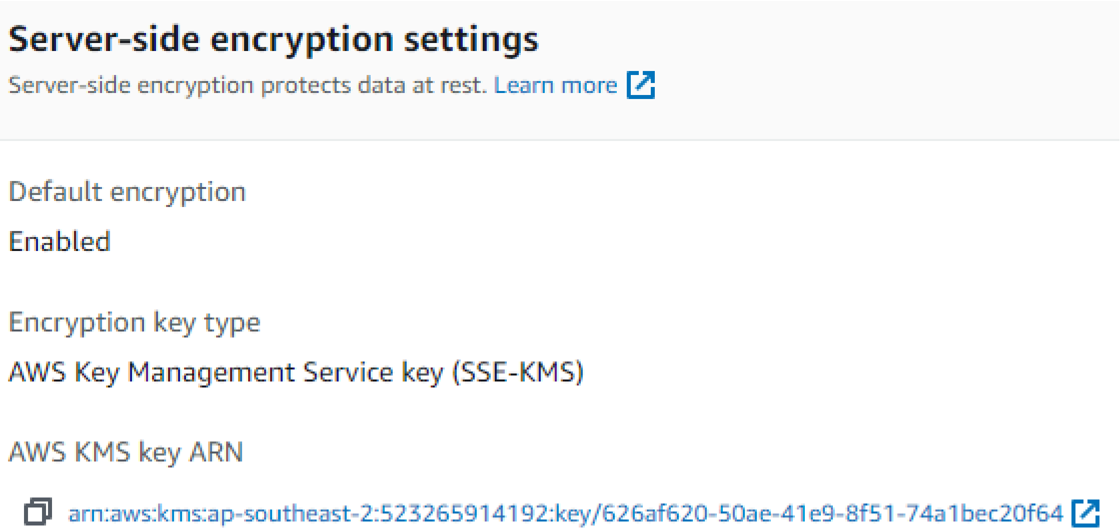
```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ python3 encryptions.py
Encrypted data key: b'\x01\x02\x03\x00x\x1b\x8b"\xf24\xb3\xa7\xda\xbd\xd1U\xb6\xaa\xad\xb0\x81\xab\x1ep\xf
5\x17\x93pk\xe7\x8c\xaf\x0f\xe8\x02\ry\x01e\x11\x188\xf8\xb7\xe0\xccl\x96\xfe\x88\x80\xf9\xda\xee\x00\x00\
x00~0|\x06\t*\x86H\x86\xf7\r\x01\x07\x06\xa0o0m\x02\x01\x000h\x06\t*\x86H\x86\xf7\r\x01\x07\x010\x1e\x06\t
`\x86H\x01e\x03\x04\x01.0\x11\x04\x0cB\xef\x05\x12t\x8b\xb1\xa1HW\x08*\x02\x01\x10\x80;9\x14\x1b\xd4\x01\x
98yJ\x8d\xba\xf4:\x81\xfb\xb9\xc4\xde;\x10\x00\xb5\xcc\xa0F\xa3*m\xbf8n4\xea\xad$\x8d\x8d\xcb\xc5d\x99X\x8
1\xc6\xe5\xdb\xba\xd0\xec :\xf7\xd5\xd2\x06\xc7jb9:'
Plaintext data key: b'wLxMMq8eSVVovdh/W3ydVAZ2Sd04ZW90VqLxzpz4WLs='
```

The output for encrypting and decrypting kms.txt file:



```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ cat kms.txt.encrypted
0o0m0h�1`�He.0�0|�pk猫�*�H��
              B�t���H*�9�J��� ������F_�m�8n4�$�����d�X�������:����jb9:gAAAAABjEICeJp7zSIHuy2_5BdW2qEjfwe
-x2GPowi4ZGGlydPuwjHQ6q2y3gVrh694TwtjG_zFxG_ei2vnW9gM76YeARUJLeg==moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503
/labs/lab4$ cat kms.txt.decrypted
Hello World!!!
```

Server-side encryption settings of kms.txt.encrypted in AWS console



Server-side encryption settings

Server-side encryption protects data at rest. Learn more

Default encryption

Enabled

Encryption key type

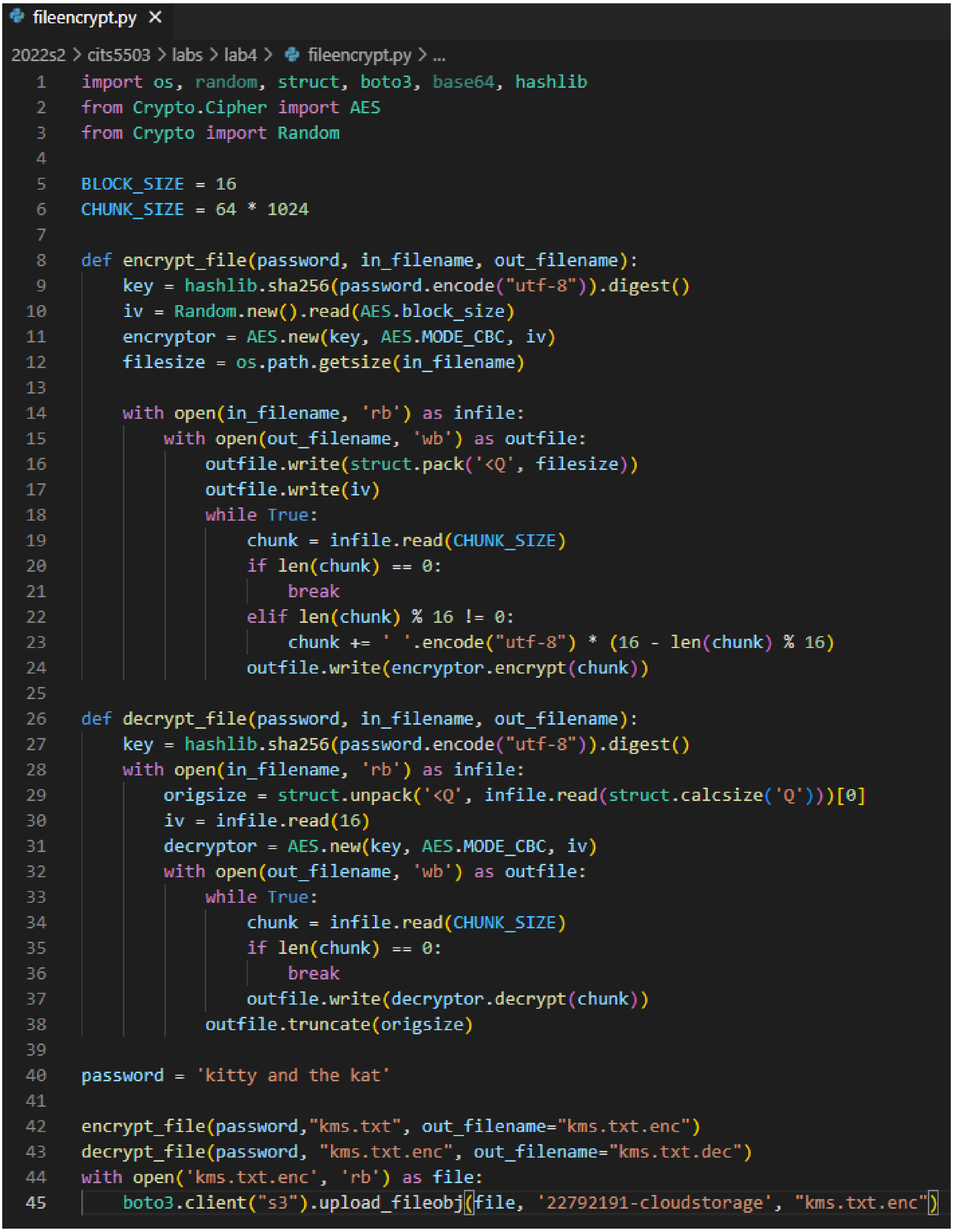
AWS Key Management Service key (SSE-KMS)

AWS KMS key ARN

arn:aws:kms:ap-southeast-2:523265914192:key/626af620-50ae-41e9-8f51-74a1bec20f64

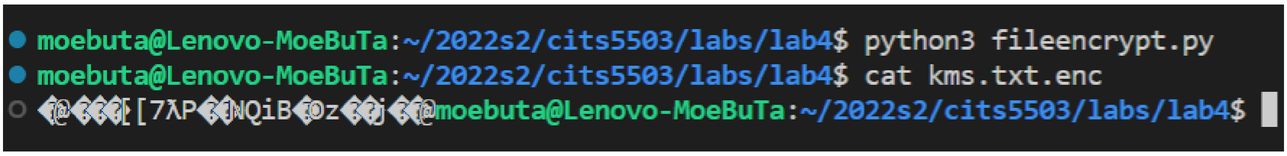
[Step 3] AES Encryption using local python library pycryptodome

1. we modify the file name into `kms.txt`, and upload the encrypted file `kms.txt.enc` into the bucket. The python code is shown below:



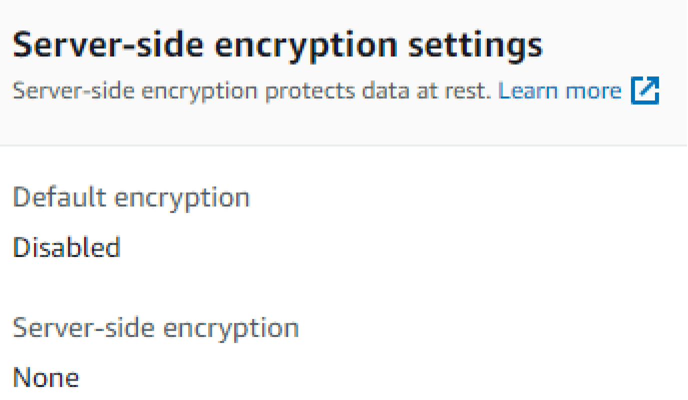
```python
import os, random, struct, boto3, base64, hashlib
from Crypto.Cipher import AES
from Crypto import Random

BLOCK_SIZE = 16
CHUNK_SIZE = 64 * 1024

def encrypt_file(password, in_filename, out_filename):
    key = hashlib.sha256(password.encode("utf-8")).digest()
    iv = Random.new().read(AES.block_size)
    encryptor = AES.new(key, AES.MODE_CBC, iv)
    filesize = os.path.getsize(in_filename)

    with open(in_filename, 'rb') as infile:
        with open(out_filename, 'wb') as outfile:
            outfile.write(struct.pack('<Q', filesize))
            outfile.write(iv)
            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += ' '.encode("utf-8") * (16 - len(chunk) % 16)
                outfile.write(encryptor.encrypt(chunk))

def decrypt_file(password, in_filename, out_filename):
    key = hashlib.sha256(password.encode("utf-8")).digest()
    with open(in_filename, 'rb') as infile:
        origsize = struct.unpack('<Q', infile.read(struct.calcsize('Q')))[0]
        iv = infile.read(16)
        decryptor = AES.new(key, AES.MODE_CBC, iv)
        with open(out_filename, 'wb') as outfile:
            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                outfile.write(decryptor.decrypt(chunk))
            outfile.truncate(origsize)

password = 'kitty and the kat'

encrypt_file(password,"kms.txt", out_filename="kms.txt.enc")
decrypt_file(password, "kms.txt.enc", out_filename="kms.txt.dec")
with open('kms.txt.enc', 'rb') as file:
    boto3.client("s3").upload_fileobj(file, '22792191-cloudstorage', "kms.txt.enc")
```

2. Using cat `kms.txt.enc` to look at the encrypted file.



```
● moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ python3 fileencrypt.py
● moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ cat kms.txt.enc
○ �@���[7λP��NQiB�Oz��j��@moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ █
```

3. Server-side encryption settings of `kms.txt.enc` in AWS console:



Server-side encryption settings

Server-side encryption protects data at rest. Learn more

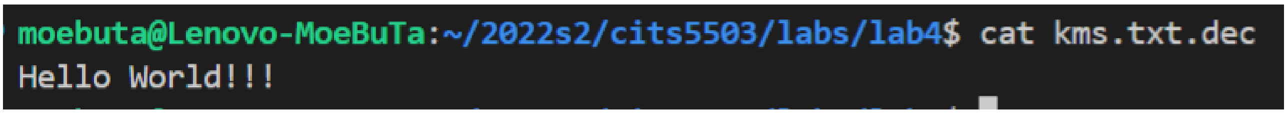
Default encryption

Disabled

Server-side encryption

None

4. Decrypt your encrypted file, present the content. Using cat `kms.txt.dec`.

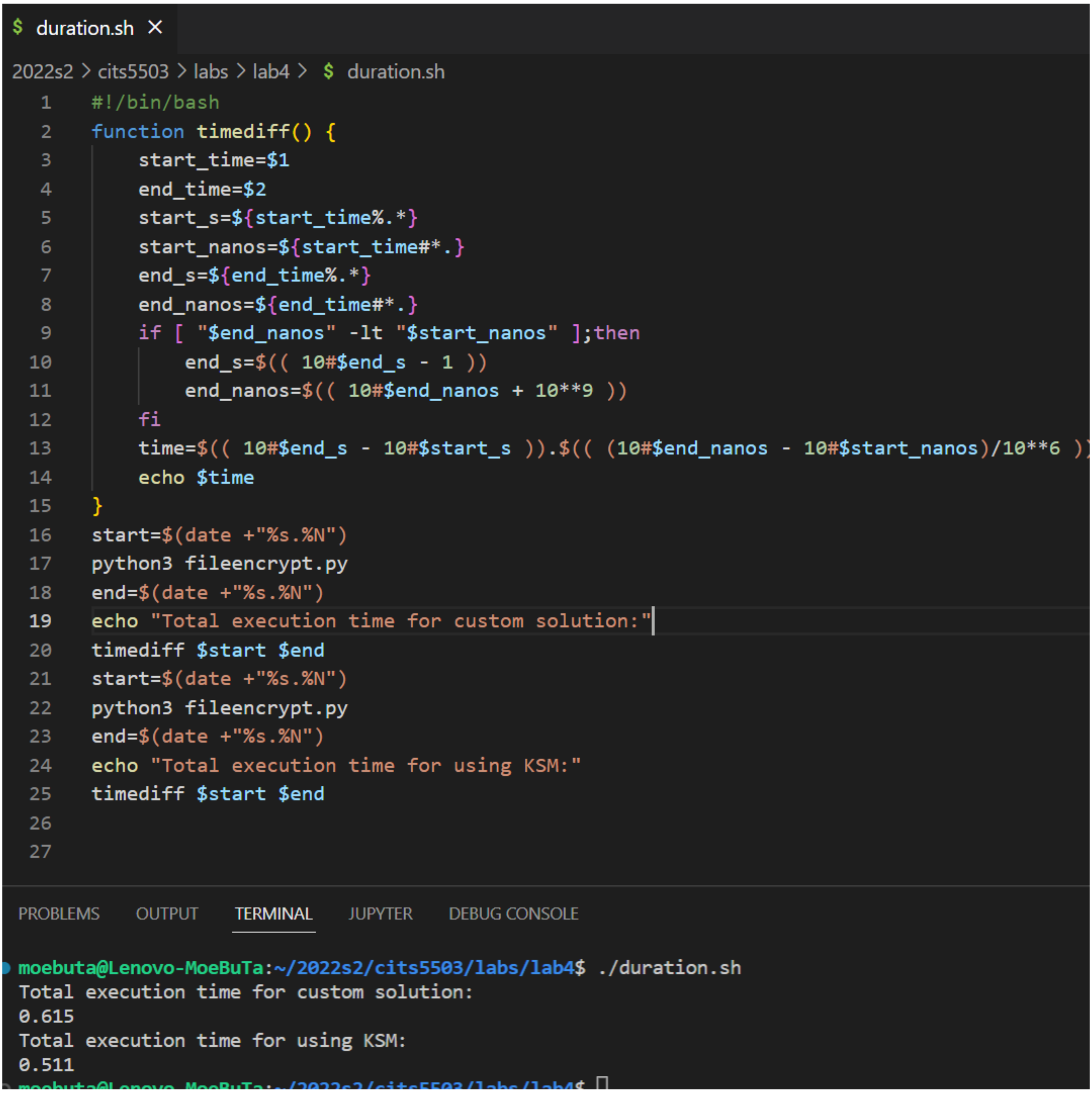


```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ cat kms.txt.dec
Hello World!!!
```

Q: What is the performance difference between using KMS and using the custom solution?

A: In terms of the performance of encryption, both of them can encrypt the message inside the file and make it unreadable. However, the custom solution takes longer to be executed compare to using KMS to encrypt and decrypt files, which means using KMS has a better performance. It may be because the custom solution uses sha256 function in the hashlib of Python as the encryption method to encrypt and decrypt file which may need a certain amount of computation resources to be executed.

The shell script for time calculation and output are shown below:



```
#!/bin/bash
function timediff() {
    start_time=$1
    end_time=$2
    start_s=${start_time%.*}
    start_nanos=${start_time#*.}
    end_s=${end_time%.*}
    end_nanos=${end_time#*.}
    if [ "$end_nanos" -lt "$start_nanos" ];then
        end_s=$(( 10#$end_s - 1 ))
        end_nanos=$(( 10#$end_nanos + 10**9 ))
    fi
    time=$(( 10#$end_s - 10#$start_s )).$(( (10#$end_nanos - 10#$start_nanos)/10**6 )
    echo $time
}
start=$(date +"%s.%N")
python3 fileencrypt.py
end=$(date +"%s.%N")
echo "Total execution time for custom solution:"
timediff $start $end
start=$(date +"%s.%N")
python3 fileencrypt.py
end=$(date +"%s.%N")
echo "Total execution time for using KSM:"
timediff $start $end


```

```
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ ./duration.sh
Total execution time for custom solution:
0.615
Total execution time for using KSM:
0.511
moebuta@Lenovo-MoeBuTa:~/2022s2/cits5503/labs/lab4$ 
```