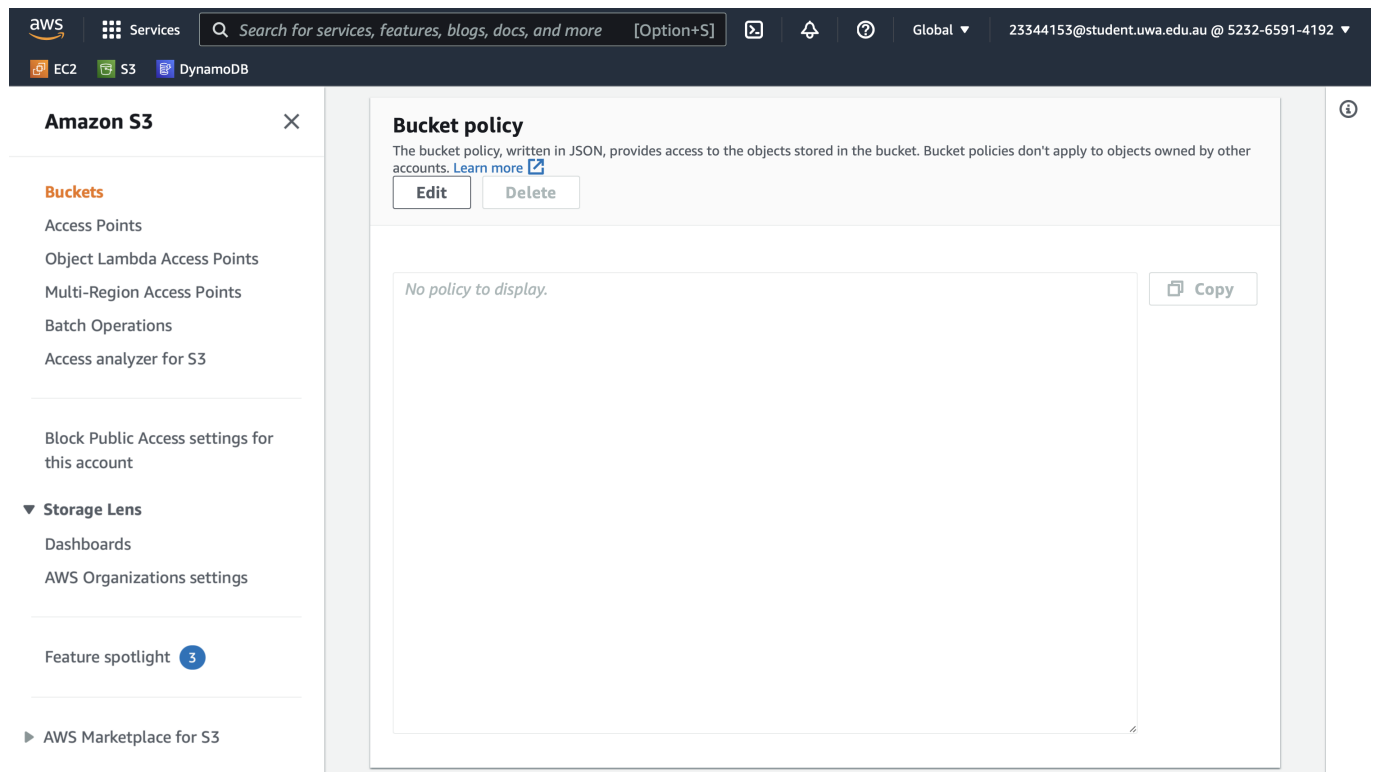


Practical Worksheet 4

Apply policy to restrict permissions on bucket



Before the bucket has no policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
    "Effect": "DENY",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::23344153-cloudstorage",
    "Condition": {
      "StringNotLike": {
        "aws:username": "23344153@student.uwa.edu.au"
      }
    }
  }
}
```

```
'''apply_policy.py'''
import boto3
from lab4 import ROOT_S3_DIR
client = boto3.client('s3')
```

```
with open('policy.json', 'r') as policy:
    client.put_bucket_policy(
        Bucket=R00T_S3_DIR,
        Policy=policy.read(),
    )
print(client.get_bucket_policy(Bucket=R00T_S3_DIR) ['Policy'])
```

Run `apply_policy.py` to apply the policy in the `policy.json` file to the bucket.

```
(venv) > Labs/lab4 $ python apply_policy.py
{"Version": "2012-10-17", "Statement": [{"Sid": "AllowAllS3ActionsInUserFolderForUserOnly", "Effect": "Deny", "Principal": "*", "Action": "s3:*", "Resource": "arn:aws:s3:::23344153-cloudstorage", "Condition": {"StringNotLike": {"aws:username": "23344153@student.uwa.edu.au"}}}]}
```

The screenshot shows the AWS Management Console interface. On the left, the 'Amazon S3' sidebar is visible with options like Buckets, Access Points, and Storage Lens. The main panel displays the 'Bucket policy' for a bucket named 'R00T_S3_DIR'. The policy is shown in a JSON editor with a 'Copy' button. The policy text is as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::23344153-cloudstorage",
      "Condition": {
        "StringNotLike": {
          "aws:username": "23344153@student.uwa.edu.au"
        }
      }
    }
  ]
}
```

The policy is now attached to the bucket and any other account that does not have the username `23344153@student.uwa.edu.au` has no permissions to view the bucket.

Services

[Option+S]

Global
23294652@student.uwa.edu.au @ 5232-6591-4192

Amazon S3

Buckets

Access Points

Object Lambda Access Points

Multi-Region Access Points

Batch Operations

Access analyzer for S3

Block Public Access settings for this account

Storage Lens

Dashboards

AWS Organizations settings

Feature spotlight 3

AWS Marketplace for S3

Amazon S3 > Buckets > 23344153-cloudstorage

23344153-cloudstorage

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Refresh

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Show versions

< 1 >

Settings

	Name	Type	Last modified	Size	Storage class
<div> <div> Insufficient permissions to list objects <p>After you or your AWS administrator have updated your permissions to allow the <code>s3:ListBucket</code> action, refresh the page. Learn more about Identity and access management in Amazon S3</p> </div> </div>					

AES Encryption using KMS

```

{
  "Id": "key-consolepolicy-3",
  "Version": "2012-10-17",
  "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/23344153@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/23344153@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/23344153@student.uwa.edu.au"
    },
    "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
]
}

```

```

'''kms.py'''
import boto3
client = boto3.client('kms')

with open('key_policy.json', 'r') as policy:

```

```

# create a key with a KMS policy
response = client.create_key(
    Policy=policy.read(),
)
key_id = response['KeyMetadata']['KeyId']
key_region = response['KeyMetadata']['Arn']
print(f'{key_id = }\n{key_region = }')

# add an alias
try:
    client.create_alias(
        AliasName='alias/23344153',
        TargetKeyId=key_id
    )
except:
    pass

# output the policy
print(client.get_key_policy(
    KeyId=key_id,
    PolicyName='default'
)['Policy'])

# generate a data key for encryption
data_key = client.generate_data_key(
    KeyId=key_id,
    KeySpec='AES_256'
)
data_key_encrypted = data_key['CiphertextBlob']
data_key = data_key['Plaintext']
print(f'{data_key_encrypted = }\n{data_key = }')

```

Run `kms.py`.

```
(venv) > _ Labs/lab4 $ python kms.py
key_id = '1d493282-5509-48db-9b6a-f3094b55e0cf'
key_region = 'arn:aws:kms:ap-southeast-2:523265914192:key/1d493282-5509-48db-9b6a-f3094b55e0cf'
{
  "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/23344153@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/23344153@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/23344153@student.uwa.edu.au"
    },
    "Action" : [ "kms:CreateGrant", "kms:ListGrants", "kms:RevokeGrant" ],
    "Resource" : "*",
    "Condition" : {
      "Bool" : {
        "kms:GrantIsForAWSResource" : "true"
      }
    }
  } ]
}
data_key_encrypted = b'\x01\x02\x03\x00\x94\r\x19 C\xbe\xde\xb0u|\xdb-\x1c\x8b\xec\xfe \x07F\x88mE\xda?\x87\x94\x86<\x87Y1c\x01 (\xf4\xafW\x89Q\xd9\x19\xc2t\x95\x0cZ\xe1B\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\x0c\xfeB#\xe23\x117\xcd\xfd2\x7frg\x02\x01\x10\x80;V\xed\x8f\xb5\xd5==W.\xb0\xe1\xb1\x9ek\x8fq\xd8h\xdcqp\xb9r\xbd\xc0w\x9fr\xa5"B\xe4B\xe6\xf0b(+Z\xe30@E\xfb\x1e\xab\xec\xbd\x82;Go\xc3\x86\x92\x8an?A'
data_key = b'n\x87\x95\xb1k\xc3/\xdb\xa2\xc2z\x10\xe3\x11\xc8|N\xd8+\x80\x11\xb1\xef)1\xa4\xba\xc2\xf8]\x04'
```

```
...
```

```
lab4.py
```

```
Sample application to encrypt and decrypt files using AES
```

```
...
```

```
import os, struct
from Crypto.Cipher import AES
from Crypto import Random
import hashlib
```

```
ROOT_S3_DIR = '23344153-cloudstorage'
CHUNK_SIZE = 64 * 1024
```

```

def encrypt_file(key, file):

    iv = Random.new().read(AES.block_size)
    encryptor = AES.new(key, AES.MODE_CBC, iv)
    filesize = os.path.getsize(file)

    with open(file, 'rb') as infile:
        with open(f'{file}.enc', '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(key, file):

    with open(file, '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(f'{file}.dec', 'wb') as outfile:
            while True:
                chunk = infile.read(CHUNK_SIZE)
                if len(chunk) == 0:
                    break
                outfile.write(decryptor.decrypt(chunk))

            outfile.truncate(origsize)

password = 'password'
key = hashlib.sha256(password.encode("utf-8")).digest()
if __name__ == '__main__':
    encrypt_file(key, "test.txt")
    decrypt_file(key, "test.txt.enc")

```

```
'''
```

```
cloudstorage.py
```

Skeleton application to copy local files to S3

Given a root local directory, will return files in each level and copy to same path on S3

```
'''
```

```

import os
import sys
import boto3
import base64
from lab4 import ROOT_S3_DIR, key, encrypt_file

client = boto3.client('s3')
s3 = boto3.resource('s3')

# create bucket if not there
argv = sys.argv[1:]
if len(argv) > 0 and '-i' in argv:
    try:
        response = client.create_bucket(
            Bucket=ROOT_S3_DIR,
            CreateBucketConfiguration={
                'LocationConstraint': 'ap-southeast-2'
            },
        )
        print(response)
    except Exception as error:
        print(error)

def upload_file(file):
    out_filename = f'{file}.enc'
    if '-k' in argv:
        # KMS
        from kms import data_key, data_key_encrypted
        encrypt_file(data_key, file)
        client.put_object(
            Body=open(out_filename, 'rb'),
            Bucket=ROOT_S3_DIR,
            # enable SSE-KMS
            ServerSideEncryption='aws:kms',
            Key=file,
            # put encrypted data key in the metadata so the data key can
            # be obtained by decrypting the encrypted data key
            Metadata={
                'encryption-key':
base64.b64encode(data_key_encrypted).decode()
            }
        )
        print("Uploading %s" % out_filename)
    elif '-c' in argv:
        # using a hash key to encrypt
        encrypt_file(key, file)
        s3.meta.client.upload_file(out_filename, ROOT_S3_DIR, file)
        print("Uploading %s" % out_filename)
    else:
        # upload files unencrypted
        s3.meta.client.upload_file(file, ROOT_S3_DIR, file)
        print("Uploading %s" % file)

# parse the current directory and upload all the text files

```



```

for dir_name, subdir_list, file_list in os.walk('.', topdown=True):
    for fname in file_list:
        # only upload files with a txt extension
        if fname.endswith('.txt'):
            upload_file(f'{dir_name}/{fname}')

print("done")

```

```

'''restorefromcloud.py'''
import os
import sys
import boto3
import base64
from lab4 import R00T_S3_DIR, key, decrypt_file

client = boto3.client('s3')
s3 = boto3.resource('s3')
kms = boto3.client('kms')

argv = sys.argv[1:]
for content in client.list_objects(Bucket=R00T_S3_DIR)['Contents']:
    paths = content['Key'].split('/')
    file = paths[-1]
    # create the directories listed in the path
    for dir in paths[:-1]:
        if not os.path.isdir(dir):
            os.mkdir(dir)
        os.chdir(dir)
    # download the file from S3 to the deepest directory
    s3.meta.client.download_file(R00T_S3_DIR, content['Key'], file)

    if len(argv) == 1:
        # using KMS data key to decrypt
        if argv[0] in ['-k', '--kms=True']:
            key = client.get_object(
                Bucket=R00T_S3_DIR,
                Key=content['Key']
            )['Metadata']['encryption-key']
            # decrypt the encrypted data key from the meta data first
            decrypt_file(
                kms.decrypt(CiphertextBlob=base64.b64decode(key))
                ['Plaintext'],
                file
            )
        # using a hash key to decrypt
        elif argv[0] in ['-c', '--crypto=True']:
            decrypt_file(key, file)

    # go back to the root directory
    for i in range(len(paths) - 1):
        os.chdir('..')

```

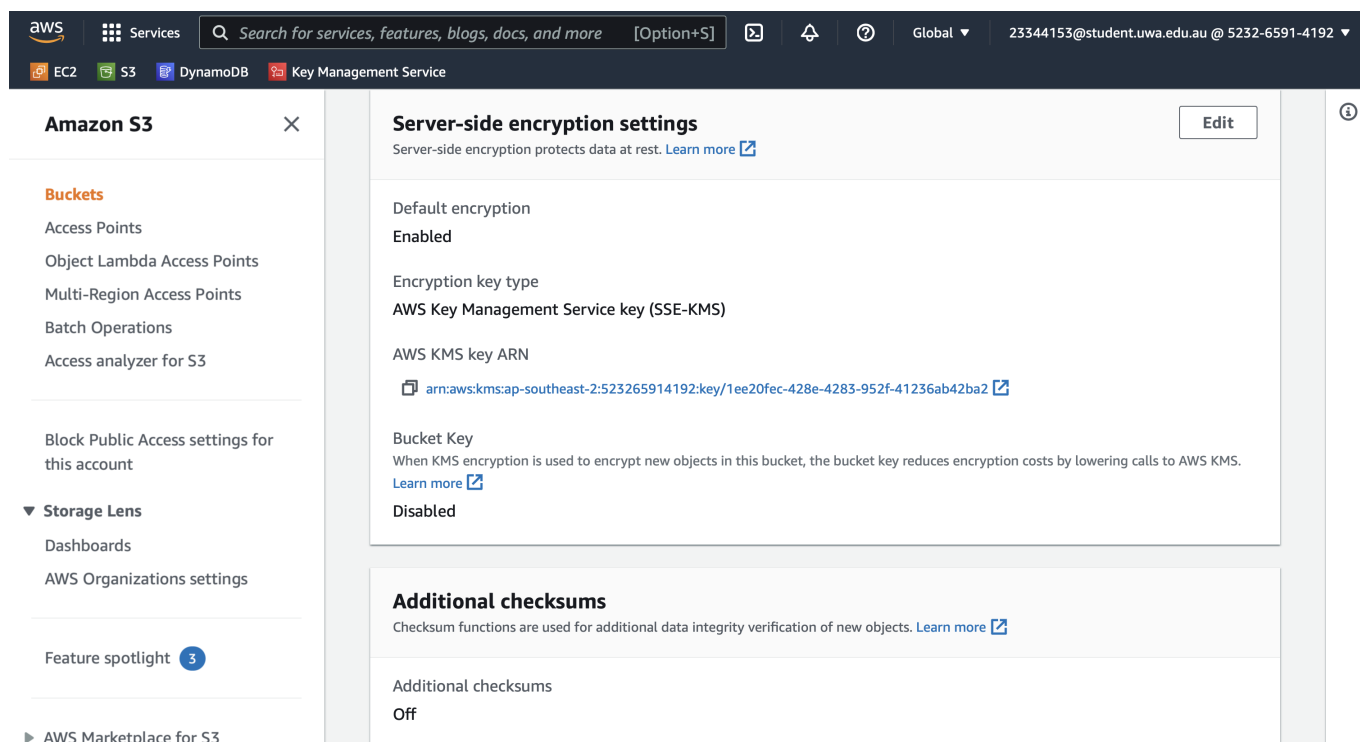
Run `cloudstorage.py` to upload the file that has been encrypted locally to **S3**.

Run `restorefromcloud.py` to download the encrypted file and decrypt it locally to view the original content.

With the `-k` flag it uses the data key generated from the **KMS customer managed key** created earlier for encryption and also adds Server-side encryption.

```
(venv) >_ Labs/lab4 $ cat test.txt
hello world
(venv) >_ Labs/lab4 $ python cloudstorage.py -k
Uploading ./test.txt.enc
done
(venv) >_ Labs/lab4 $ cat test.txt.enc
>??T.???Bh??
    ?N??A#)??Y??ZR? E%
(venv) >_ Labs/lab4 $ rm test.txt*
(venv) >_ Labs/lab4 $ python restorefromcloud.py -k
(venv) >_ Labs/lab4 $ cat test.txt
>??T.???Bh??
    ?N??A#)??Y??ZR? E%
(venv) >_ Labs/lab4 $ cat test.txt.dec
hello world
```

Server-side encryption is set to **SSE-KMS**.



The screenshot shows the AWS Management Console interface. The top navigation bar includes the AWS logo, a search bar, and user information. The left sidebar shows the 'Amazon S3' service selected. The main content area displays the 'Server-side encryption settings' for a bucket. The settings are as follows:

- 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/1ee20fec-428e-4283-952f-41236ab42ba2`
- Bucket Key:** Disabled. A note states: 'When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. [Learn more](#)'
- Additional checksums:** Off. A note states: 'Checksum functions are used for additional data integrity verification of new objects. [Learn more](#)'

AES Encryption using local python library pycryptodome

Run `cloudstorage.py` and `restorefromcloud.py` with the `-c` flag to use a custom hash as the symmetric key for encryption and decryption.

```
(venv) >_ Labs/lab4 $ cat test.txt
hello world
(venv) >_ Labs/lab4 $ python cloudstorage.py -c
Uploading ./test.txt.enc
done
(venv) >_ Labs/lab4 $ cat test.txt.enc

???M"2[W????#mA?Eq6oS}x?)uZ"%
(venv) >_ Labs/lab4 $ rm test.txt*
(venv) >_ Labs/lab4 $ python restorefromcloud.py -c
(venv) >_ Labs/lab4 $ cat test.txt

???M"2[W????#mA?Eq6oS}x?)uZ"%
(venv) >_ Labs/lab4 $ cat test.txt.dec
hello world
```

Server-side encryption is disabled.

The screenshot shows the AWS Management Console interface for an Amazon S3 bucket. The left sidebar contains navigation links for Buckets, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, Access analyzer for S3, Block Public Access settings, Storage Lens, Dashboards, AWS Organizations settings, Feature spotlight, and AWS Marketplace for S3. The main content area is titled 'Server-side encryption settings' and includes an 'Edit' button. It displays the following settings:

- Default encryption:** Disabled
- Server-side encryption:** None
- Additional checksums:** Off
- Tags (0):** No tags associated with this resource.

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

```
(venv) >_ Labs/lab4 $ time python cloudstorage.py -k
Uploading ./rootdir/rootfile.txt.enc
Uploading ./rootdir/subdir/subfile.txt.enc
done
python3 cloudstorage.py -k 0.40s user 0.10s system 8% cpu 5.956 total
(venv) >_ Labs/lab4 $ time python cloudstorage.py -c
Uploading ./rootdir/rootfile.txt.enc
Uploading ./rootdir/subdir/subfile.txt.enc
done
python3 cloudstorage.py -c 0.42s user 0.11s system 9% cpu 5.694 total
```

I constructed a slightly larger file structure to compare the performance of the different approaches to encryption. The result is using **KMS** for encryption and **SSE-KMS** is slightly slower than the custom solution. However I would still use **KMS** because there are separate permissions for the use of a **KMS** key which provides added protection against unauthorised access to objects in **S3** and **SSE-KMS** also provides an audit trail that shows when the **KMS** key was used and by whom.