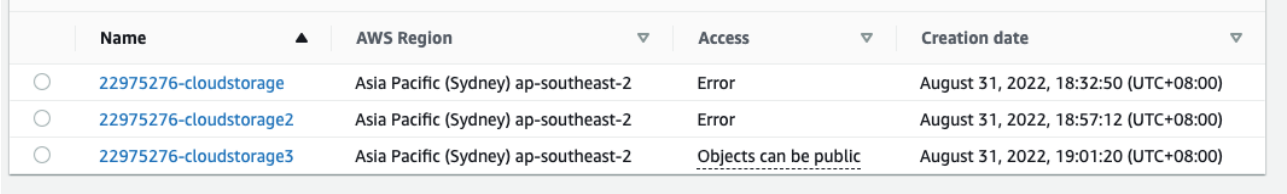
1. Create Policy



Name	AWS Region	Access	Creation date
22975276-cloudstorage	Asia Pacific (Sydney) ap-southeast-2	Error	August 31, 2022, 18:32:50 (UTC+08:00)
22975276-cloudstorage2	Asia Pacific (Sydney) ap-southeast-2	Error	August 31, 2022, 18:57:12 (UTC+08:00)
22975276-cloudstorage3	Asia Pacific (Sydney) ap-southeast-2	Objects can be public	August 31, 2022, 19:01:20 (UTC+08:00)

To test if my policy was working I applied the restrictions to my own account by using **string like** in the condition which lead to me being locked out. Somehow I accidentally made the same mistake again with my second attempt meaning I had to create a third bucket with the policy applied. Because it had locked me out in the past I knew it worked.
The final policy I used can be seen below and the code I used to apply it is also below, however the code exists within the file cloudstorageencrypt.py - the entire python file exists later in the document.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::22975276-cloudstorage3",
        "arn:aws:s3:::22975276-cloudstorage3/*"
      ],
      "Condition": {
        "StringNotLike": {
          "aws:username": "22975276@student.uwa.edu.au"
        }
      }
    }
  ]
```

```
65      # Create bucket
66      bucket_policy = {
67      "Version": "2012-10-17",
68      "Statement": [
69          {
70              "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
71              "Effect": "Deny",
72              "Principal": "*",
73              "Action": "s3:*",
74              "Resource": [
75                  "arn:aws:s3:::22975276-cloudstorage3",
76                  "arn:aws:s3:::22975276-cloudstorage3/*"
77              ],
78              "Condition": {
79                  "StringNotLike": {
80                      "aws:username": "22975276@student.uwa.edu.au"
81                  }
82              }
83          }
84      ]
85      }
86      bucket_policy = json.dumps(bucket_policy)
87      try:
88              s3_client = boto3.client('s3', region_name=region)
89              location = {'LocationConstraint': region}
90              s3_client.create_bucket(Bucket=bucket_name, CreateBucketConfiguration=location)
91              s3_client.put_bucket_policy(Bucket=bucket_name, Policy = bucket_policy)
92      except ClientError as e:
93              logging.error(e)
94              return False
95      return True
```

2. AES encryption using kms

First I created my kms key using the following python code.

```python
import json
import logging
from datetime import date, datetime

import boto3
from botocore.exceptions import ClientError

AWS_REGION = 'ap-southeast-2'

# logger config
logger = logging.getLogger()
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s: %(levelname)s: %(message)s')

kms_client = boto3.client("kms", region_name=AWS_REGION)


def json_datetime_serializer(obj):
    """
    Helper method to serialize datetime fields
    """
    if isinstance(obj, (datetime, date)):
        return obj.isoformat()
    raise TypeError("Type %s not serializable" % type(obj))


def create_kms_key():
    """
    Creates a unique customer managed KMS key.
    """
    try:
        response = kms_client.create_key(Description='22975276-key2',
                                         Tags=[{
                                             'TagKey': 'Name',
                                             'TagValue': '22975276-key2'
                                         }])

    except ClientError:
        logger.exception('Could not create a CMK key.')
        raise
    else:
        return response


if __name__ == '__main__':
    # Constants
    logger.info('Creating a symetric CMK...')
    kms = create_kms_key()
    logger.info(
        f'Symetric CMK is created with details: {json.dumps(kms, indent=4, default=json_datetime_serializer)}'
    )
```

After running this file the output was the following.

```
lachlan@lachlan-VirtualBox:~/Cloud$ python3 create_kms.py
2022-09-06 22:45:46,507: INFO: Found credentials in shared cre
dentials file: ~/.aws/credentials
2022-09-06 22:45:46,613: INFO: Creating a symetric CMK...
2022-09-06 22:45:47,588: INFO: Symetric CMK is created with de
tails: {
    "KeyMetadata": {
        "AWSAccountId": "523265914192",
        "KeyId": "81fe7ff3-4821-47fe-a0e9-aef6065b97af",
        "Arn": "arn:aws:kms:ap-southeast-2:523265914192:key/81
fe7ff3-4821-47fe-a0e9-aef6065b97af",
```

Then I applied the policy using the following code

Open ▾ | new_kms_policy.py ~/Cloud

```
import boto3
import boto3
import argparse
parser = argparse.ArgumentParser(description = 'Initialise to True')
parser.add_argument('-i',action="store_true", help="-i to create bucket")
args = parser.parse_args()
# ----------------------------
# CITS5503
#
# cloudstorage.py
#
# skeleton application to copy local files to S3
#
kms_client.put_key_policy(KeyId="81fe7ff3-4821-47fe-a0e9-aef6065b97af",
                                                PolicyName='default',
                                                Policy=policy)
pol = kms_client.get_key_policy(KeyId="81fe7ff3-4821-47fe-a0e9-aef6065b97af",
                                                PolicyName='default')
print(pol)
        {
            "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/22975276@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": "*"
        }
    ]
}'''

kms_client.put_key_policy(KeyId="81fe7ff3-4821-47fe-a0e9-aef6065b97af",
```

```
lachlan@lachlan-VirtualBox:~/Cloud$ python3 new_kms_policy.py
{'Policy': '{\n  "Version" : "2012-10-17",\n  "Id" : "key-consolepolicy-3",\n  "Statement" : [ {\n    "Sid" : "Enable IAM User Permissio
ns",\n    "Effect" : "Allow",\n    "Principal" : {\n      "AWS" : "arn:aws:iam::523265914192:root"\n    },\n    "Action" : "kms:*",\n
 "Resource" : "*"\n  }, {\n    "Sid" : "Allow access for Key Administrators",\n    "Effect" : "Allow",\n    "Principal" : {\n      "AWS"
 : "arn:aws:iam::523265914192:user/22975276@student.uwa.edu.au"\n    },\n    "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" ],\n    "Resource" : "*"\n  } ]\n}', 'ResponseMetadata': {'RequestId': '89aa88d7-f
7ad-4087-8c05-fa582525f15c', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '89aa88d7-f7ad-4087-8c05-fa582525f15c', 'cache-c
ontrol': 'no-cache, no-store, must-revalidate, private', 'expires': '0', 'pragma': 'no-cache', 'date': 'Tue, 06 Sep 2022 15:24:23 GMT',
'content-type': 'application/x-amz-json-1.1', 'content-length': '849'}, 'RetryAttempts': 0}}
```

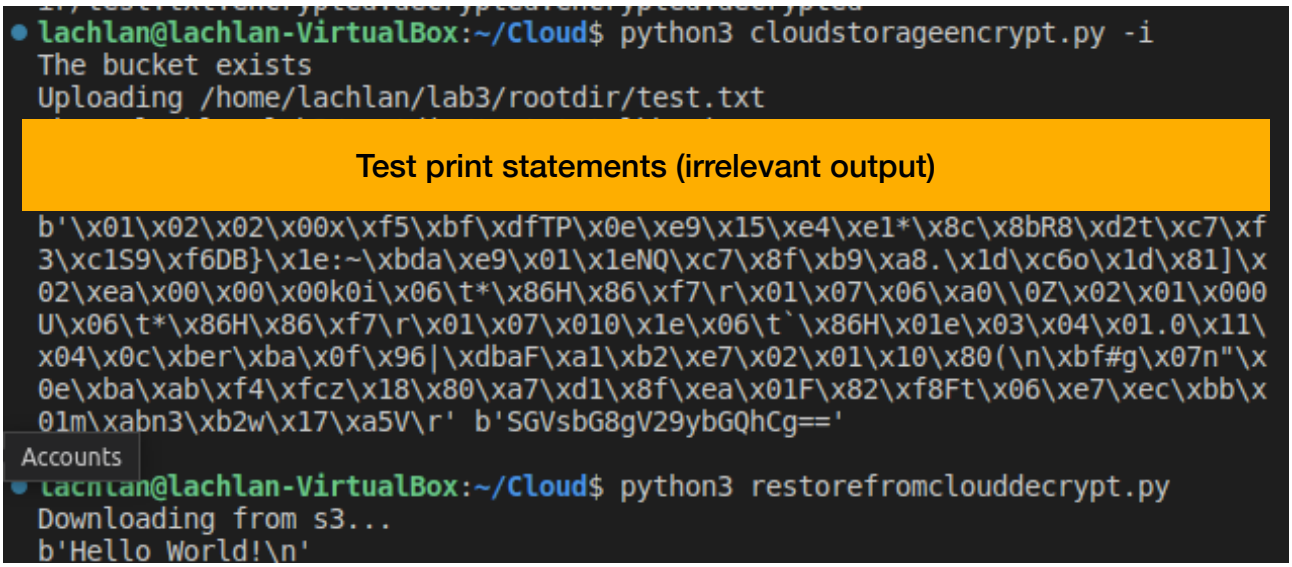
From here I generated a data key, encrypted my file and uploaded it to my s3 bucket. Then I downloaded it generated the data key, decrypted it and looked at the file contents. The code for both encrypting/uploading and decrypting/downloading are at the end. Also the code contains a lot of unknown library errors underlined in yellow, this is because I opened the files on my own machine which doesn't contain all the libraries, this was so I could take bigger screenshots.

The test text file for the following encryption and decryption.

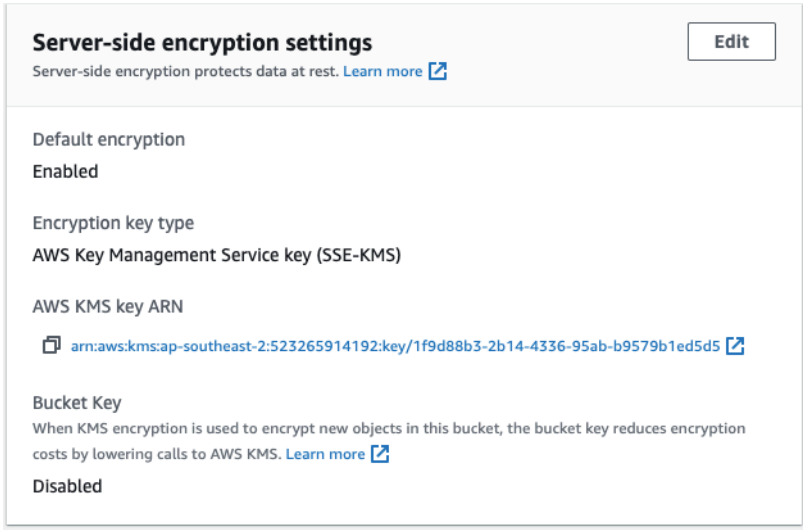
Open ▾ | test.txt ~/lab3/rootdir | Save

```
Hello World!
```

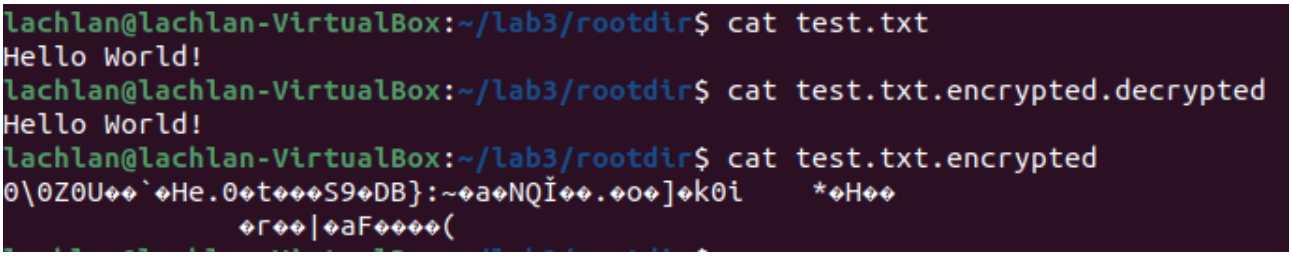
The output from encrypting and decrypting, the final text appears to show that the process was successful



Encryption settings in bucket set to the KMS.



Contents of files appear as expected.



Cloudstorageencrypt.py

```python
import os
import boto3
import json
import base64
import logging
import boto3
from botocore.exceptions import ClientError
import argparse
parser = argparse.ArgumentParser(description = 'Initialise to True')
parser.add_argument('-i',action="store_true", help="-i to create bucket")
args = parser.parse_args()
# ------------------------------
# CITS5503
#
# 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
#
# ------------------------------


ROOT_DIR = '/home/lachlan/lab3'
ROOT_S3_DIR = '22975276-cloudstorage3'


s3 = boto3.client("s3")

bucket_config = {'LocationConstraint': 'ap-southeast-2'}
kms_client = boto3.client("kms", region_name = 'ap-southeast-2')
data_key = kms_client.generate_data_key(KeyId = "81fe7ff3-4821-47fe-a0e9-aef6065b97af",
def upload_file(folder_name, file, file_name):
    print("Uploading %s" % file)
    s3_client = boto3.client('s3')
    bucket = "22975276-cloudstorage3"
    print(file, "lkbenis")
    with open(file,"rb") as f:
        file_contents = f.read()
        print(data_key['KeyId'])
        fileencrypt = kms_client.encrypt(KeyId = data_key['KeyId'], Plaintext = file_co
        with open(file,"wb") as f:
            f.write(fileencrypt['CiphertextBlob'])
            print(fileencrypt['CiphertextBlob'], base64.b64encode(file_contents))
    os.rename(file,file+".encrypted")
    file = file+".encrypted"
    try:
        response = s3_client.upload_file(file,bucket,file)
    except ClientError as e:
        logging.error(e)
        return False

def create_bucket(bucket_name, region=None):
    """Create an S3 bucket in a specified region

    If a region is not specified, the bucket is created in the S3 default
    region (us-east-1).

    :param bucket_name: Bucket to create
    :param region: String region to create bucket in, e.g., 'us-west-2'
    :return: True if bucket created, else False
    """

    # Create bucket
    bucket_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::22975276-cloudstorage3",
                "arn:aws:s3:::22975276-cloudstorage3/*"
            ],
            "Condition": {
                "StringNotLike": {
                    "aws:username": "22975276@student.uwa.edu.au"
                }
            }
        }
    ]
    }
    bucket_policy = json.dumps(bucket_policy)
    try:
            s3_client = boto3.client('s3', region_name=region)
            location = {'LocationConstraint': region}
            s3_client.create_bucket(Bucket=bucket_name, CreateBucketConfiguration=locat
            s3_client.put_bucket_policy(Bucket=bucket_name, Policy = bucket_policy)
    except ClientError as e:
            logging.error(e)
            return False
    return True
# Main program
# Insert code to create bucket if not there

s3 = boto3.resource('s3')
bucket = s3.Bucket('22975276-cloudstorage3')
if args.i:
    if bucket.creation_date:
        print("The bucket exists")
    else:
        print("The bucket does not exist")
        create_bucket('22975276-cloudstorage3', 'ap-southeast-2')

for dir_name, subdir_list, file_list in os.walk(ROOT_DIR, topdown=True):
    if dir_name != ROOT_DIR:
        for fname in file_list:
            upload_file("%s/" % dir_name[2:], "%s/%s" % (dir_name, fname), fname)


print("done")
```

restorefromclouddecrypt.py

```python
import boto3
from boto3.session import Session
import base64
import os
import fernet
ACCESS_KEY_ID ='AKIAXTVIUGVIMEGAMQ5A'
SECRET_KEY = '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'

session = Session(aws_access_key_id=ACCESS_KEY_ID,
aws_secret_access_key=SECRET_KEY)

s3 = session.resource('s3')

bucket ='22975276-cloudstorage3'

my_bucket = s3.Bucket(bucket)
kms_client = boto3.client("kms", region_name = 'ap-southeast-2')
data_key = kms_client.generate_data_key(KeyId = "81fe7ff3-4821-47fe-a0e9-aef6065b97af", NumberOfBytes = 123)
print("Downloading from s3...")
filename = ""
for s3_files in my_bucket.objects.all():
    #print(s3_files.key)
    filename = s3_files.key
    os.rename(filename, filename + ".decrypted")
    filename = filename + (".decrypted")
    #print(filename)
    with open(filename,"rb") as f:
        file_contents = f.read()
        #print(data_key['KeyId'])
        filedecrypt = kms_client.decrypt(KeyId = data_key['KeyId'], CiphertextBlob = file_contents)
        #print(filedecrypt)
        with open(filename,"wb") as f:
                f.write(filedecrypt['Plaintext'])
                print(filedecrypt['Plaintext'])

#initiate s3 resource
s3 = boto3.resource('s3')
# download file into current directory
for s3_object in my_bucket.objects.all():
        filename = s3_object.key
        my_bucket.download_file(s3_object.key, filename)
```

3. Use pycryptodome

Since the code for encryption and decryption was provided, I simply altered the previous py files to use the provided encrypt and decrypt functions instead of the data key to encrypt the files. (Once again code at the end).

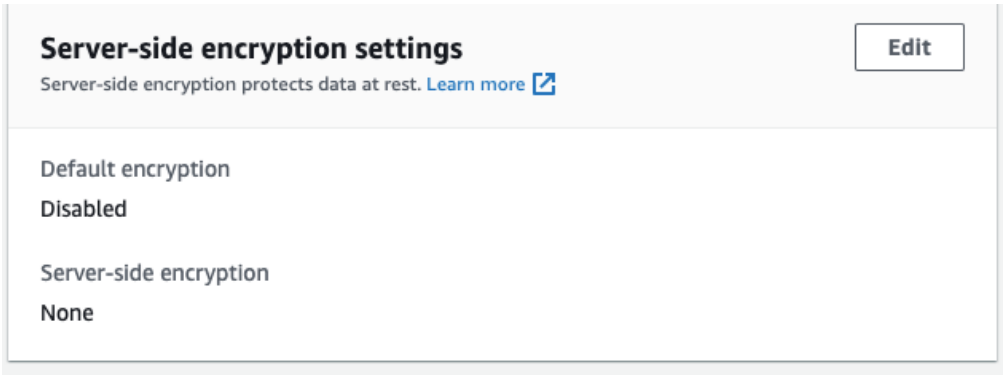
Following screenshot is of running both functions (uploading and downloading files)

```
lachlan@lachlan-VirtualBox:~/Cloud$ python3 cloudstorageencryptQ3.py -i
The bucket exists
Uploading /home/lachlan/lab3/rootdir/test2.txt.enc
done
lachlan@lachlan-VirtualBox:~/Cloud$ python3 restorefromclouddecryptQ3.py
Downloading from s3...
```

Exploring file contents, as you can see the encryption and decryption process worked successfully.

```
lachlan@lachlan-VirtualBox:~/lab3/rootdir$ ls
subdir  test2_dec.txt  test2.txt  test2.txt.enc
lachlan@lachlan-VirtualBox:~/lab3/rootdir$ cat test2.txt
Hello World!
lachlan@lachlan-VirtualBox:~/lab3/rootdir$ cat test2.txt.enc
r◆cr*)x◆;◆◆◆◆ÿ}E◆`◆◆f◆◆O◆L◆◆◆lachlan@lachlan-VirtualBox:~/lab3/rootdir$ cat test
2_dec.txt
Hello World!
```

The encryption settings of the file in the bucket.



CloudstorageencryptQ3.py

Users > lachlanbassi > Downloads > cloudstorageencryptQ3.py > ...

```python
import os
import boto3
import json
import base64
import logging
from botocore.exceptions import ClientError
import argparse
parser = argparse.ArgumentParser(description = 'Initialise to True')
parser.add_argument('-i',action="store_true", help="-i to create bucket")
args = parser.parse_args()
import os, random, struct
from Crypto.Cipher import AES
from Crypto import Random
import hashlib
password = 'kitty and the kat'

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))


ROOT_DIR = '/home/lachlan/lab3'
ROOT_S3_DIR = '22975276-cloudstorage3'


s3 = boto3.client("s3")

bucket_config = {'LocationConstraint': 'ap-southeast-2'}
kms_client = boto3.client("kms", region_name = 'ap-southeast-2')
data_key = kms_client.generate_data_key(KeyId = "81fe7ff3-4821-47fe-a0e9-aef6065b97af", NumberOfBytes = 123)
def upload_file(folder_name, file, file_name):
    s3_client = boto3.client('s3')
    bucket = "22975276-cloudstorage3"
    encrypt_file(password, file, file+".enc")
    file = file+".enc"
    print("Uploading %s" % file)
    try:
        response = s3_client.upload_file(file,bucket,file)
    except ClientError as e:
        logging.error(e)
        return False

def create_bucket(bucket_name, region=None):
    """Create an S3 bucket in a specified region

    If a region is not specified, the bucket is created in the S3 default
    region (us-east-1).

    :param bucket_name: Bucket to create
    :param region: String region to create bucket in, e.g., 'us-west-2'
    :return: True if bucket created, else False
    """

    # Create bucket
    bucket_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::22975276-cloudstorage3",
                "arn:aws:s3:::22975276-cloudstorage3/*"
            ],
            "Condition": {
                "StringNotLike": {
                    "aws:username": "22975276@student.uwa.edu.au"
                }
            }
        }
    ]
    }
    bucket_policy = json.dumps(bucket_policy)
    try:
            s3_client = boto3.client('s3', region_name=region)
            location = {'LocationConstraint': region}
            s3_client.create_bucket(Bucket=bucket_name, CreateBucketConfiguration=location)
            s3_client.put_bucket_policy(Bucket=bucket_name, Policy = bucket_policy)
    except ClientError as e:
            logging.error(e)
            return False
    return True

s3 = boto3.resource('s3')
bucket = s3.Bucket('22975276-cloudstorage3')
if args.i:
    if bucket.creation_date:
        print("The bucket exists")
    else:
        print("The bucket does not exist")
        create_bucket('22975276-cloudstorage3', 'ap-southeast-2')
for dir_name, subdir_list, file_list in os.walk(ROOT_DIR, topdown=True):
    if dir_name != ROOT_DIR:
        for fname in file_list:
            upload_file("%s/" % dir_name[2:], "%s/%s" % (dir_name, fname), fname)
print("done")
```

restorefromclouddecryptQ3.py

```python
import boto3
from boto3.session import Session
import base64
import os
import fernet
import os, random, struct
from Crypto.Cipher import AES
from Crypto import Random
import boto3
import base64
import hashlib
BLOCK_SIZE = 16
CHUNK_SIZE = 64 * 1024
ACCESS_KEY_ID ='AKIAXTVIUGVIMEGAMQ5A'
SECRET_KEY = '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'
password = 'kitty and the kat'
session = Session(aws_access_key_id=ACCESS_KEY_ID,
aws_secret_access_key=SECRET_KEY)

s3 = session.resource('s3')

bucket ='22975276-cloudstorage3'

my_bucket = s3.Bucket(bucket)
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)


print("Downloading from s3...")
filename = ""
for s3_files in my_bucket.objects.all():
    #print(s3_files.key)
    filename = s3_files.key
    decrypt_file(password, filename, "/home/lachlan/lab3/rootdir/test2_dec.txt")

#initiate s3 resource
s3 = boto3.resource('s3')
# download file into current directory
for s3_object in my_bucket.objects.all():
    filename = s3_object.key
    my_bucket.download_file(s3_object.key, filename)
```