Lab 4: Encryption

Section 1: Applying Policy to Restrict Permissions

1. The following code was used to apply a policy to allow only my username (22489437@student.uwa.edu.au) to access to the S3 bucket identified by `arn:aws:s3:::22489437-cloudstorage` as well as the objects inside the bucket.

```
ROOT_DIR = '.'
ROOT_S3_DIR = '22489437-cloudstorage'
REGION = 'ap-southeast-2'
student_number = '22489437'

s3 = boto3.client("s3", region_name=REGION)
bucket_config = {'LocationConstraint': 'ap-southeast-2'}

policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
            "Effect": "DENY",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": [
        "arn:aws:s3:::22489437-cloudstorage/*",
        "arn:aws:s3:::22489437-cloudstorage"
        ],
            "Condition": {
                "StringNotLike": {
                    "aws:username": f"{student_number}@student.uwa.edu.au"
                }
            }
        }
    ]
}

def main(argv):
    policyJson = json.dumps(policy)
    s3.put_bucket_policy(Bucket=ROOT_S3_DIR, Policy=policyJson)
    print("Updated bucket policy")
    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```

```
jookai@jookai:~/Desktop/cits5503/lab3$ python3 applypolicy.py
Updated bucket policy
```

2. Inspecting the bucket from the AWS console reveals the updated policy that was added to the bucket.

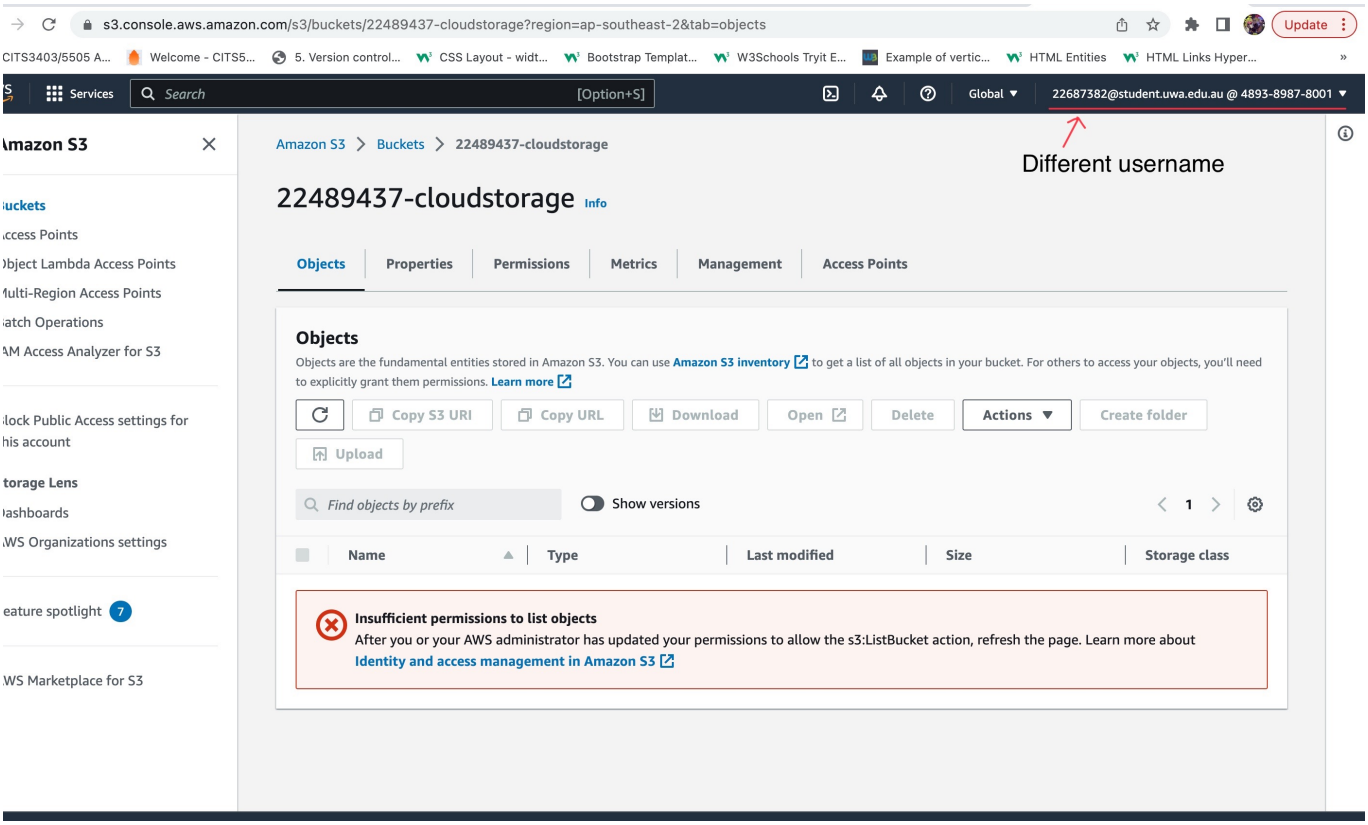
Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Public access is blocked because Block Public Access settings are turned on for this bucket
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about using Amazon S3 Block Public Access


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::22489437-cloudstorage/*",
        "arn:aws:s3:::22489437-cloudstorage"
      ],
      "Condition": {
        "StringNotLike": {
          "aws:username": "22489437@student.uwa.edu.au"
        }
      }
    }
  ]
}
```


Attempting to view this bucket from another user with username `22687382@student.uwa.edu.au` shows that they have insufficient permission to list objects in this bucket. This shows that the policy applied has it's intended effect.



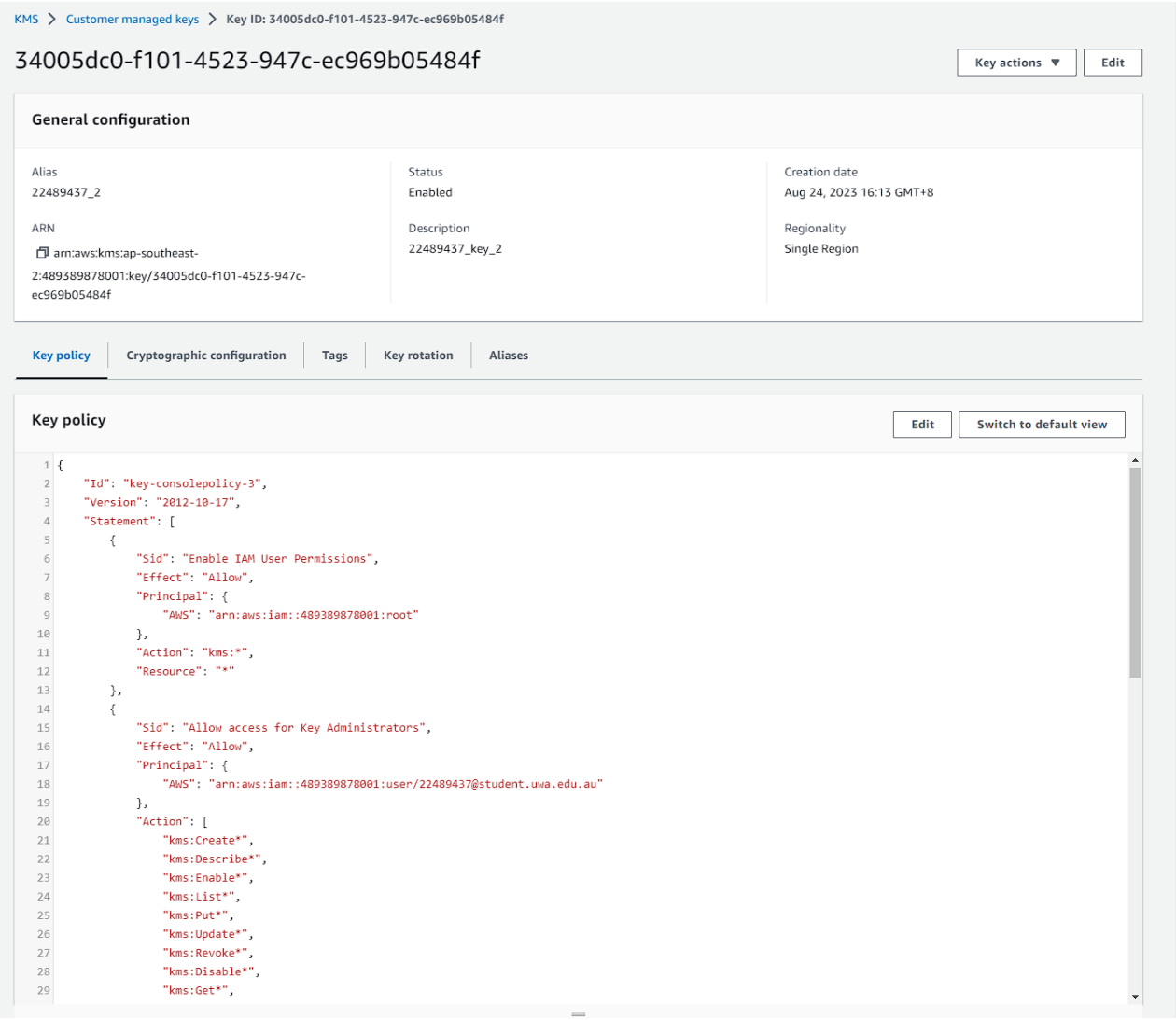
Section 2: AES Encryption using KMS

1. The following code creates a KMS key and attaches an alias (22489437_2) to it. The code also atatches a policy to the key which makes 22489437@student.uwa.edu.au the user and administrator. The policy, represented by the `key_policy` variable which can be found in Lab Sheet 4 is not added to this report.

```
def main(argv):
    kms_client = boto3.client('kms', region_name=REGION)
    # Create Key
    response = kms_client.create_key(
        Description='22489437_key_2',
        KeyUsage='ENCRYPT_DECRYPT',
        Origin='AWS_KMS'
    )
    key_id = response['KeyMetadata']['KeyId']
    print("KMS key id:", key_id)

    # Attack key policy from labsheet
    kms_client.put_key_policy(
        KeyId=key_id,
        PolicyName='default',
        Policy=json.dumps(key_policy)
    )

    # Create alias for key
    kms_client.create_alias(
        AliasName='alias/22489437_2',
        TargetKeyId=key_id
    )
    print(f"Created alias '22489437' for KMS key with ID: {key_id}")

    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```

```
jookai@jookai:~/Desktop/cits5503/lab3$ python3 createkey.py
KMS key id: 34005dc0-f101-4523-947c-ec969b05484f
Created alias '22489437' for KMS key with ID: 34005dc0-f101-4523-947c-ec969b0548
4f
```

2. Inspecting the created key in the KMS console reveals the same key ID as step 1 as well as the attached key policy.



3. The following code will encrypt the file and upload it to S3. The code uses the `kms.generate_data_key()` function in combination with the keyID from step 1 to generate a data key that will be used to encrypt the file. This will return a data key in plaintext and ciphertext. The plaintext data key is used to encrypt the file and the encrypted data key is written into the file where it can be retreived for decryption later. The encrypted file is then uploaded to S3.

```
ROOT_DIR = '.'
ROOT_S3_DIR = '22489437-cloudstorage'
REGION = 'ap-southeast-2'
KEY_ID = '34005dc0-f101-4523-947c-ec969b05484f'
FILE_NAME = 'enc_test.txt'
NUM_BYTES_FOR_LEN = 4
s3 = boto3.client("s3", region_name=REGION)
kms = boto3.client("kms", region_name=REGION)

def encrypt_file():
    # Create Data Key
    try:
        response = kms.generate_data_key(KeyId=KEY_ID, KeySpec='AES_256')
    except ClientError as e:
        logging.error(e)
```

```
    data_key_encrypted, data_key_plaintext = response['CiphertextBlob'],
base64.b64encode(response['Plaintext'])
    print("Data key encrypted:", data_key_encrypted)
    print("Data key plaintext:", data_key_plaintext)

    # Read File
    try:
        with open(FILE_NAME, 'rb') as file:
            file_contents = file.read()
    except IOError as e:
        logging.error(e)
        return False

    # Encrypt file
    f = Fernet(data_key_plaintext)
    file_contents_encrypted = f.encrypt(file_contents)

    # Write the encrypted data key and encrypted file contents together
    try:
        with open(FILE_NAME + '.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)
    except IOError as e:
        logging.error(e)
        return False

    # Upload the file to S3
    file_name = FILE_NAME + '.encrypted'
    try:
        s3.upload_file(file_name, ROOT_S3_DIR, file_name, ExtraArgs=
{'ServerSideEncryption': "aws:kms", "SSEKMSKeyId": KEY_ID})
        print(f"Uploaded {file_name} to S3")
    except Exception as e:
        print(f"Error uploading {file_name}: {e}")
```

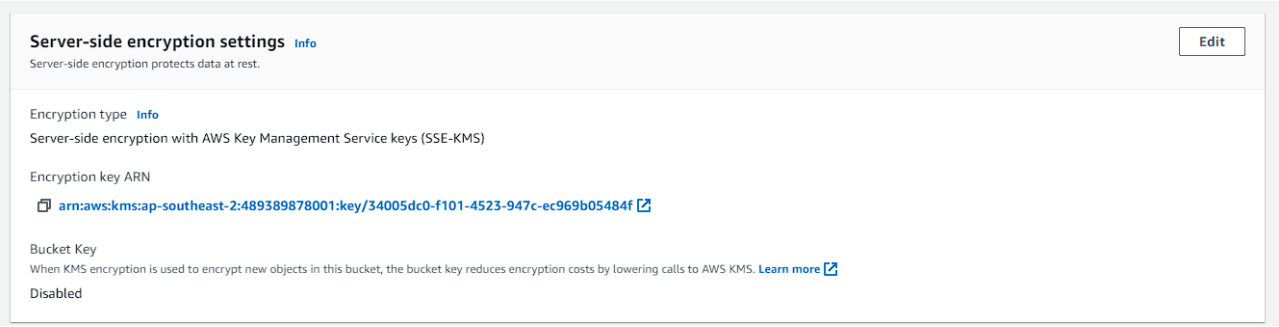
4. The following file named `enc_test.txt` will be the subject of the encryption and decryption for the next step

```
jookai@jookai:~/Desktop/cits5503/lab3$ cat enc_test.txt
This text will be encrypted as part of lab 4
```

The output of the encryption:

```
jookai@jookai:~/Desktop/cits5503/lab3$ cat enc_test.txt.encrypted
0o0m0hK◆`◆He.0◆!3Z◆Ko◆◆f◆◆◆◆+◆◆◆H◆|◆!HUC◆K◆s◆Q~0|      *◆H◆◆
             ◆J◆J◆◆8◆j◆)◆;
                        ◆◆4j◆◆◆,z(1^◆;}◆u◆◆h◆3◆
                                        ◆2R◆阖
                                                    ◆s◆◆◆◆◆@◆◆*◆◆
NLgAAAAABk5ybfdX-1COrhPwnwAU6GhTy7L199QYE4R7KtyLq_hI2-iVSKiqAvrSXH39sodf7iZMqbZD
ofBCZkLkypblT6o_yiEYHf5_j229xgl5fKayFoHftaOuO5FzfjBJkANHXaTZnUjookai@jookai:~/De
sktop/cits5503/lab3$
```

The file can be viewed in the AWS console. Note the server-side encryption setting matching the key generated in step 1.



5. The following code will download the file from S3 and decrypt it. The file is downloaded from S3. The encrypted data key which was placed in the file in step 4 will be read from the file and decrypted. The decrypted data key will be used to decrypt the rest of the file.

```
def decrypt_file():
    # Download file from S3
    file_name = FILE_NAME + '.encrypted'
    s3.download_file(ROOT_S3_DIR, file_name, file_name)
    print(f"Downloaded {file_name} from S3")

    # Read the encrypted file into memory
    try:
        with open(file_name, 'rb') as file:
            file_contents = file.read()
    except IOError as e:
        logging.error(e)
        return False

    # Get encrypted data key from file
    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 data key
    response = kms.decrypt(CiphertextBlob=data_key_encrypted)
    data_key_plaintext = base64.b64encode((response['Plaintext']))

    # Decrypt file
    f = Fernet(data_key_plaintext)
    file_contents_decrypted = f.decrypt(file_contents[data_key_encrypted_len:])
```

```
    # Write the decrypted file contents
    try:
        with open(FILE_NAME + '.decrypted', 'wb') as file_decrypted:
            file_decrypted.write(file_contents_decrypted)
    except IOError as e:
        logging.error(e)
        return False

    print(f"Decrypted {FILE_NAME}")
```

The output of the decryption:

```
jookai@jookai:~/Desktop/cits5503/lab3$ cat enc_test.txt.decrypted
This text will be encrypted as part of lab 4
jookai@jookai:~/Desktop/cits5503/lab3$
```

Section 3: AES Encryption using local python library pycryptodome

1. The example code in `fileencrypt.py` was used for the local encryption and decryption process following the same steps as above. The file was encrypted, uploaded to S3 then downloaded and decrypted. The code below shows the main program of the program and does not include the `encrypt_file` and `decrypt-file` functions which have not been modified from the example code:

```
s3 = boto3.client("s3", region_name=REGION)
password = 'kitty and the kat'
encrypt_file(password,"enc_test.txt", out_filename="enc_test.txt.enc")
try:
    s3.upload_file("enc_test.txt.enc", ROOT_S3_DIR, "enc_test.txt.enc")
    print(f"Uploaded enc_test.txt.enc to S3")
except Exception as e:
    print(f"Error uploading enc_test.txt.enc: {e}")

s3.download_file(ROOT_S3_DIR, "enc_test.txt.enc", "enc_test.txt.enc")
print(f"Downloaded enc_test.txt.enc from S3")

decrypt_file(password, "enc_test.txt.enc", out_filename="enc_test_decrypted.txt")

print("--- %s seconds ---" % (time.time() - start_time))
```

```
jookai@jookai:~/Desktop/cits5503/lab3$ cat enc_test.txt.enc
-o^oooo$,oPoo_ox oYoIzooo.>ooZooV=oe,ooo/oWoooooZ_uL5oGjookai@jookai:~/Desktop/c
its5503/lab3$
```

```
jookai@jookai:~/Desktop/cits5503/lab3$ cat enc_test_decrypted.txt
This text will be encrypted as part of lab 4
jookai@jookai:~/Desktop/cits5503/lab3$
```

Section 4: Answer the question

1. Both programs were timed in their execution. The program using the AWS KMS encryption took 1 second to run while the local encryption using PyCryotoDome took 0.55 seconds to run.

```
jookai@jookai:~/Desktop/cits5503/lab3$ python3 encryptfile.py
Data key encrypted: b'\x01\x02\x03\x00x\x9e\xc4y\x88K\x9b\\\x97%W\xae\xc1\xea!3\
x03Z\xc5Ko\x96\xf2f\xa9\xd1\xc5\xe3\x96\xce+\x8b\x97\x01\xc4H\xc5|\xe6!\xd4\x8at
C\x93K\xc8s\xa2Q\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\xceJ\x95J\xbc\xd48\xdcj\x16\xb0)\x02\x01\x10\x80;\x0b\xda\xc54j\xd1\xce
\xf1,z(1^\x9f;}\xcfu\x86\xa6h\xbb3\xe5\x0c\xc02R\xc6\xe9\x98\x96\x0c\t\xda\x80\x
a3s\xcbl\x08\xb8\x0e\xf9\xc8\xce@\xee\xf9*\xb5\x8d\xeb\x96\xb1\x0fNL\x03'
Data key plaintext: b'zrQK2kzLSWgH4ozajwWUKWhLrwYO29k2uXFQWqucn3Y='
Uploaded enc_test.txt.encrypted to S3
Downloaded enc_test.txt.encrypted from S3
Decrypted enc_test.txt
--- 1.0099689960479736 seconds ---
```

```
jookai@jookai:~/Desktop/cits5503/lab3$ python3 fileencrypt.py
Uploaded enc_test.txt.enc to S3
Downloaded enc_test.txt.enc from S3
--- 0.5549328327178955 seconds ---
```