

# Lab 4

---

Saif Ali Athyaab - 23810756

## Background

The aim of this lab is to write a program that will:





1. Apply a policy to your bucket to allow only you as a user to access it
2. Create a key in KMS and use it to encrypt files on the client before uploading to S3 and decrypt them after downloading from S3
3. Implement AES using python and test the difference in performance between the KMS solution and the local one.

### [Step 1] Apply policy to restrict permissions on bucket

We are required to apply the following policy to the S3 bucket created in the last lab to allow only our username to access the bucket. I have ensured to make the appropriate changes (folders, username, etc) to the policy as necessary.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllowAllS3ActionsInUserFolderForUserOnly",
    "Effect": "DENY",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::23810756-cloudstorage/rootdir/subdir/*",
    "Condition": {
      "StringNotLike": {
        "aws:username": "23810756@student.uwa.edu.au"
      }
    }
  }
}
```

## Output

 policy op We can test it by applying the policy to a single folder and using a username that is not your own. In my case, I have created another Test user on the IAM console.  New user Logged in to other user  sub file Confirm that you no longer have access to that folder's contents.  Access denied

### [Step 2] AES Encryption using KMS

## Creation of KMS key

We start of by defining the local **create\_key()** function. It has 2 arguments

1. kms containing the **client** object concerning KMS
2. key description - string

The standard methodology of try and except used to raise error if required.

```
def create_key(kms,key_desc):
    if not key_desc:
        key_desc = "Key management demo key"
    try:
        key = kms.create_key(Description=key_desc)
    except:
        print("Couldn't create your key.")
        raise
    else:
        return (key['KeyMetadata']['KeyId'])
```

## Creation of alias

**create\_alias()** has 3 arguments

1. kms client object
2. key id of created key
3. alias - string

We make use of the **kms.create\_alias()** function and pass in alias and key

```
def create_alias(kms,key_id,alias):
    if not alias:
        alias = "Key management demo alias"
    try:
        kms.create_alias(AliasName=alias, TargetKeyId=key_id)
    except:
        print("Couldn't create alias.")
        raise
    else:
        return alias
```

## Applying policy to the key

**set\_policy() local** takes 3 arguments

1. kms client object
2. key id
3. policy dictionary

```
def set_policy(kms, key_id, policy):
    import json
    try:
        kms.put_key_policy(
            KeyId=key_id, PolicyName='default', Policy=json.dumps(policy))
    except:
        print("Couldn't set policy for key.")
        raise
    else:
        print(f"Set policy for key {key_id}.")
```

In the **main()** function we call the functions and store the policy to be set up to the key

```
def main():
    import boto3
    kms = boto3.client("kms")
    bucket = '23810756-cloudstorage'
```

Policy is stored in *given\_policy* variable. I have modified the Principal attributes plugging in my relevant

- account ID and
- IAM user ID
- 

```
given_policy={
    "Version": "2012-10-17",
    "Id": "key-consolepolicy-3",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::489389878001:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow access for Key Administrators",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::489389878001:user/23810756@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::489389878001:user/23810756@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::489389878001:user/23810756@student.uwa.edu.au"
    },
    "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
]
}

```

Calling the create function and printing response - **KeyID**

```
keyID=create_key(kms,'testKey7')
print("Key created with KeyID: %s"%keyID)
```

Creating alias using AWS specified **naming convention**

'alias/string'


```
alias=create_alias(kms,keyID,'alias/23810756v7')
print(f"Created alias {alias} for key {keyID}.")
```

Setting the policy and main functions

```
set_policy(kms,keyID,given_policy)

if __name__ == "__main__":
    main()
```

## OUTPUT

 Key, alias and policy

Key from the Console  key console


In my CloudStorage application, I have added the ability to encrypt and decrypt the files my program lists using the KMS Client apis of boto3.

## OUTPUT

nsword@nsword-HP-Pavilion-x360-Convertible:~/CC/Labs Mds\$ /bin/python3  
/home/nsword/CC/cloudstorage.py

Below you can see the 2 files encrypted and decrypted.  All 4 files

## [Step 3] AES Encryption using local python library pycryptodome

We also create another version of the CloudStorage program that uses the python library pycryptodome to encrypt and decrypt your files.  pycryptodome

We start off by importing the required libraries and setting the constants

```
import os, struct
from Cryptodome.Cipher import AES
from Cryptodome import Random
import boto3
import hashlib
```

```

ROOT_S3_DIR = '23810756-cloudstorage'
s3 = boto3.client("s3")

BLOCK_SIZE = 16
CHUNK_SIZE = 64 * 1024

```

Our main functionality is stored with this function

1. Get the list of Objects
2. Iterate through the list
3. Get the Key and Display filename
4. Create a local copy of the file
5. Call our helper functions - `encrypt_file()` and `decrypt_file()`

```


def encrypt_decrypt(s3, ROOT_S3_DIR):
    bucket=ROOT_S3_DIR
    kms=boto3.client("kms")
    result = s3.list_objects(Bucket = bucket)
    for o in result.get('Contents'):
        print("File Name: %s"%o.get('Key'))
        data = s3.get_object(Bucket=bucket, Key=o.get('Key'))
        contents = data['Body'].read()
        orig_file=os.path.basename(o.get('Key'))
        f=open(orig_file,"wb")
        f.write(contents)
        f.close()
        try:
            encrypt_file(password,orig_file,(orig_file+'.enc'))
            decrypt_file(password,(orig_file+'.enc'),('orig_'+orig_file))
        except:
            print("Couldn't encrypt/decrypt text.")
            raise

```

I have used the given encrypt and decrypt functions from example code(`filencrpyt.py`) as is so I have not added them in my report.

## Files

As seen below, we have the

1. Files from S3
2. Encrypted Files
3. Decrypted files (prefixed by *orig\_*)  Alt text

## Rootfile comparision

 rootfile comp

## Subfile comparision

subfile comp

### QUESTION

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

Getting the runtime of the local AES method aes runtime

Getting the runtime of the S3 KMS method kms runtime

### ANSWER

We can see that KMS is slightly slower for all 3 metrics.

This is also taking into consideration that in local method, we are downloading and then encrypting.

While in the KMS file we are also printing on the CLI.

Having said that, I would recommend the KMS method due to its secure nature of working independently on the cloud.