Cryptography in the Cloud

...

By Saketh Mahesh

What is the Cloud?

- On demand access to a system of virtualized resources and services that have extensive capabilities
    - Computing power, storage, software capabilities, network management, etc
- Cloud providers include AWS, Azure, GCP
- Companies rely on cloud computing more than ever
    - But does it cryptographic services for security?
    - Let’s analyze AWS to get an answer

aws

Google Cloud Platform

Cryptography Use in AWS (Amazon Web Services) Cloud

- Data Encryption:
  - As companies store vast amounts of data on the cloud (many of which is sensitive info), the need to encrypt and secure data is important
  - Services: AWS KMS (Key Management Service), AWS S3 (Simple Storage Service), AWS CloudHSM, AWS WAF, and many more
- Secure Communication
  - Communicating securely between cloud services and users is imperative
  - Services: TLS, SSH, VPN, SFTP (Secure File Transfer Protocol)
- Identity and Access Management (IAM)
  - Verifying identities and detecting individuals responsible for malicious activities
  - Managing access to services, resources, and private data
  - Services: SAML, MFA, Digital Signature, PKI

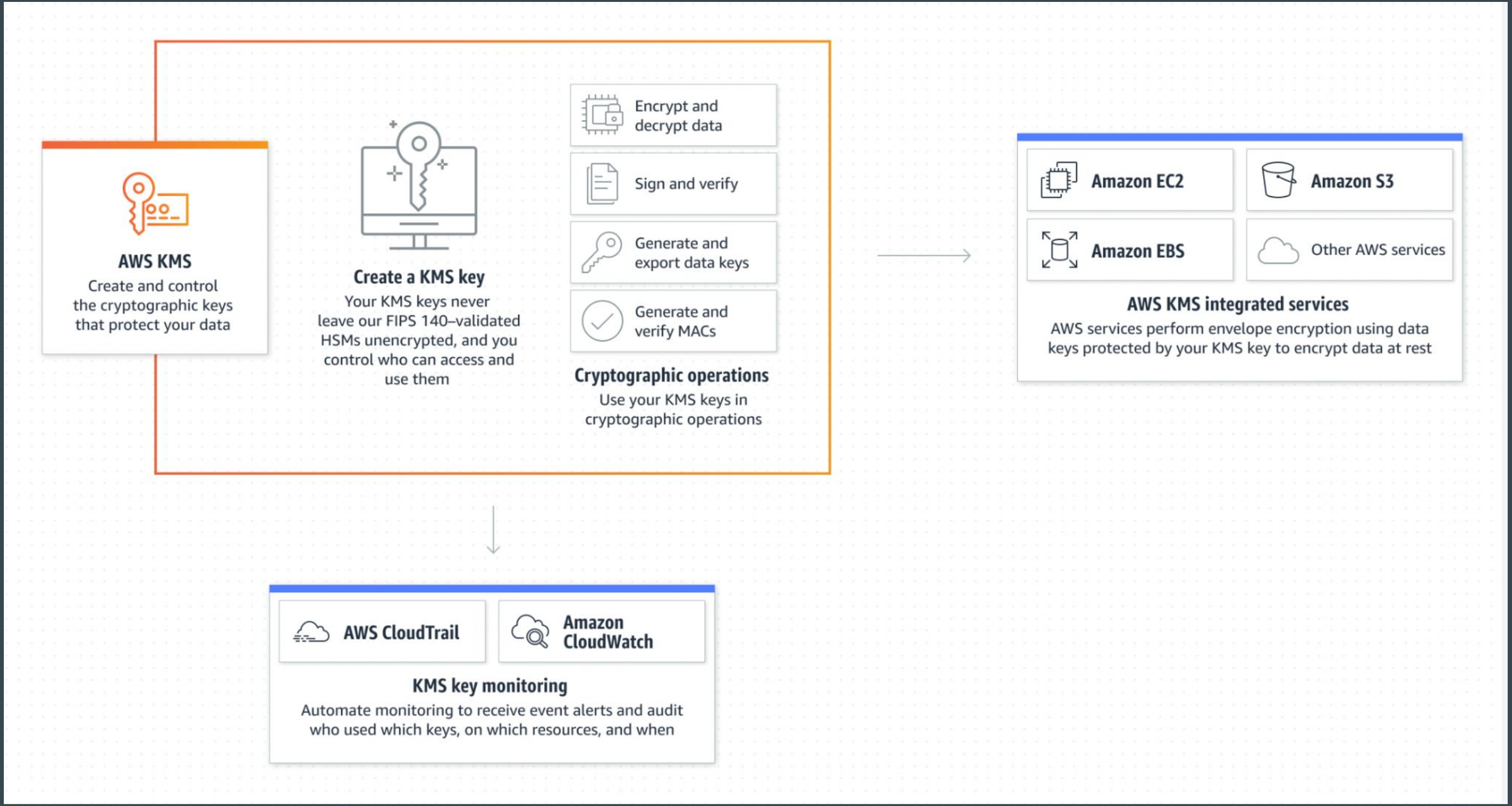
Example of Data Encryption in AWS

- Scenario: I am a manager at a company and I have a text file containing very sensitive information (usernames and passwords of my employees). I want store and encrypt this text file as securely as possible using AWS and its services

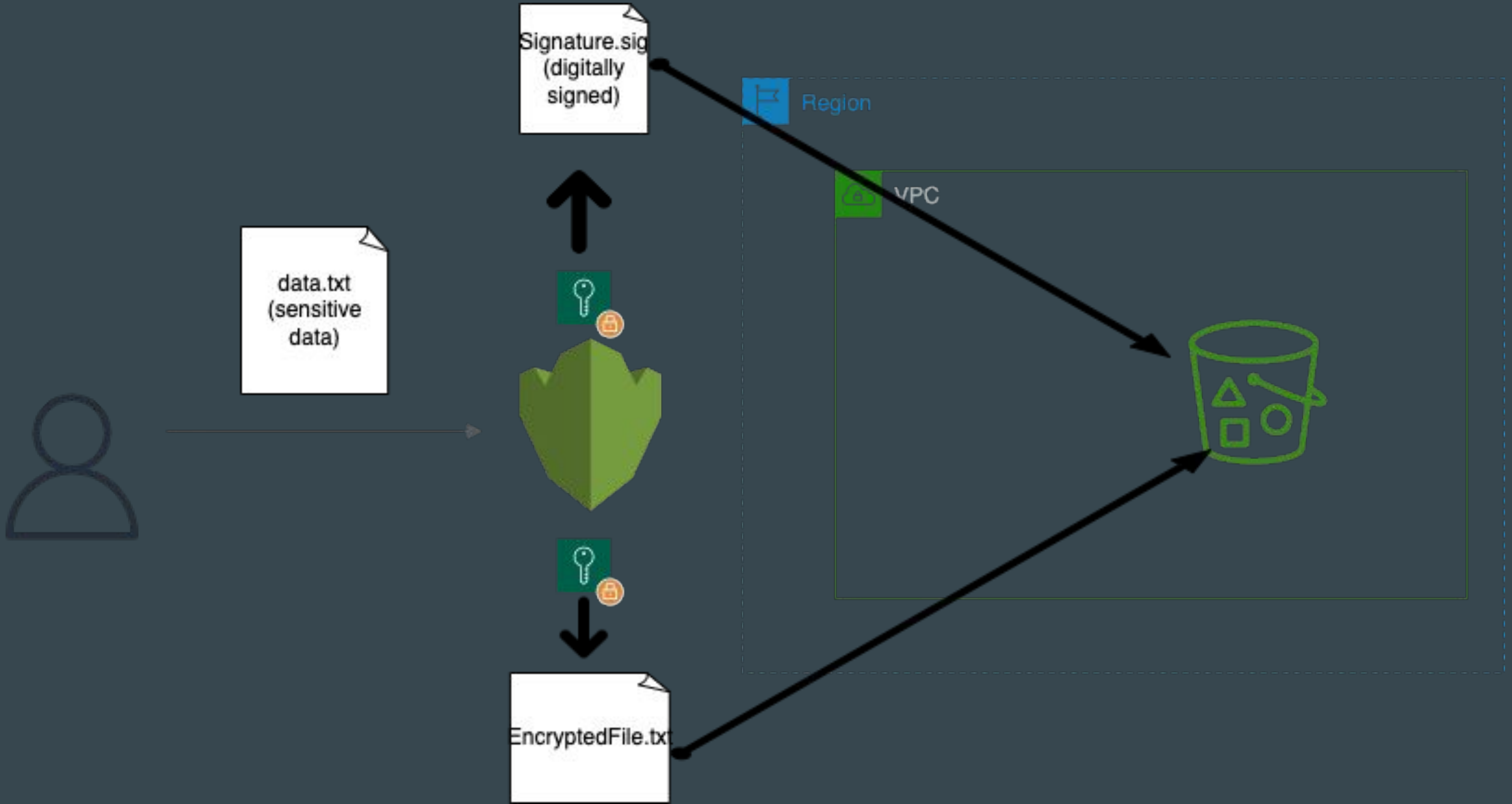
- What is the Plan?:
  - I will use AWS KMS to encrypt and sign the file
  - I will use AWS S3 to store to encrypted file

- 3 Main Ways To Work With AWS:
  - AWS GUI
  - AWS SDK
  - AWS CLI

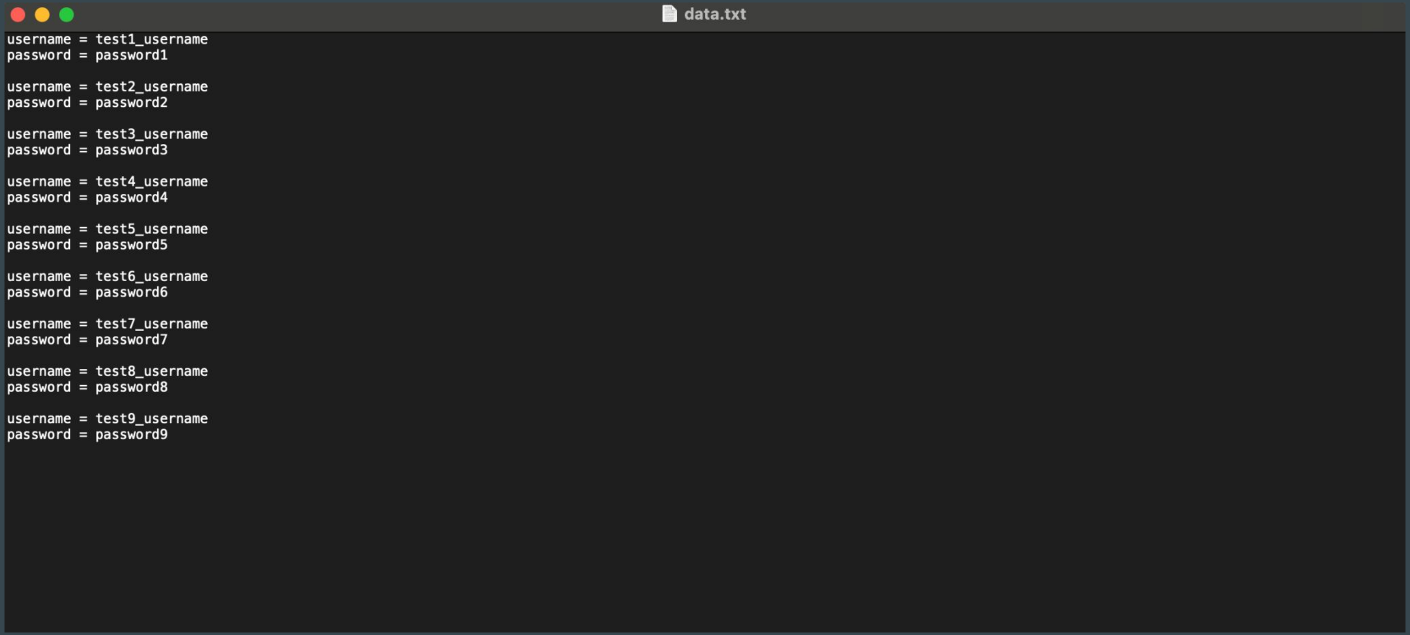
AWS KMS (Key Management Service)


AWS KMS
Create and control the cryptographic keys that protect your data
Create a KMS key
Your KMS keys never leave our FIPS 140–validated HSMs unencrypted, and you control who can access and use them
Encrypt and decrypt data
Sign and verify
Generate and export data keys
Generate and verify MACs
Cryptographic operations
Use your KMS keys in cryptographic operations
Amazon EC2
Amazon S3
Amazon EBS
Other AWS services
AWS KMS integrated services
AWS services perform envelope encryption using data keys protected by your KMS key to encrypt data at rest
AWS CloudTrail
Amazon CloudWatch
KMS key monitoring
Automate monitoring to receive event alerts and audit who used which keys, on which resources, and when

Diagram

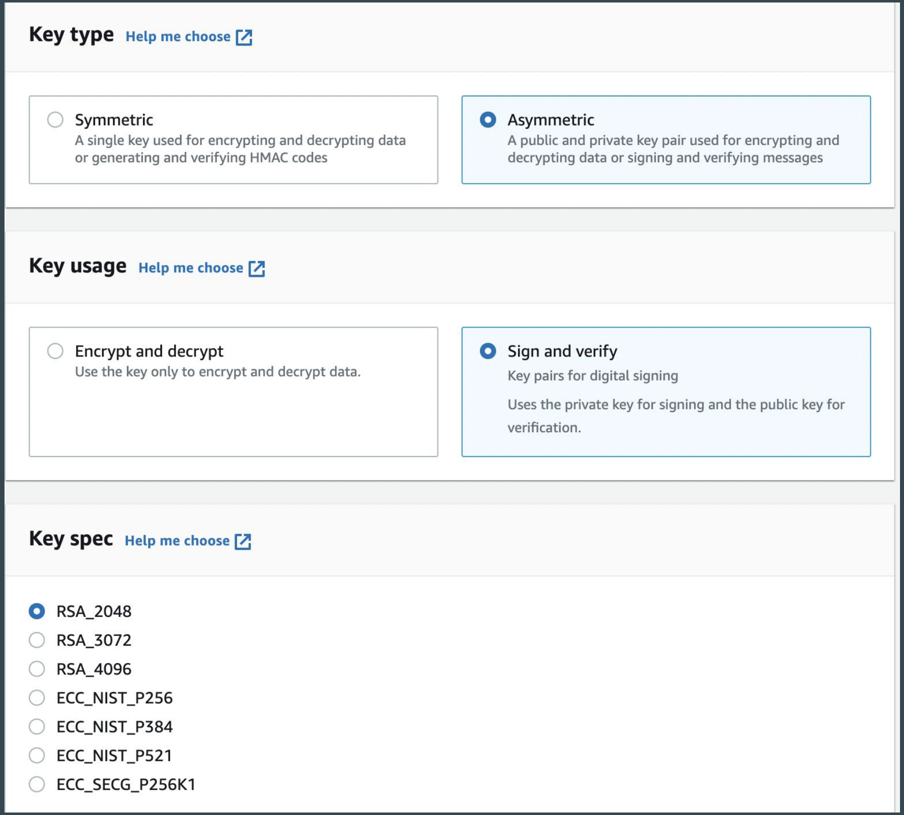

Signature.sig (digitally signed)
data.txt (sensitive data)
Region
VPC
EncryptedFile.txt

Original Data File



```
username = test1_username
password = password1

username = test2_username
password = password2

username = test3_username
password = password3

username = test4_username
password = password4

username = test5_username
password = password5

username = test6_username
password = password6

username = test7_username
password = password7

username = test8_username
password = password8

username = test9_username
password = password9
```

Generate Signing Key



Key type Help me choose

- [ ] Symmetric
  A single key used for encrypting and decrypting data or generating and verifying HMAC codes
- [✓ Asymmetric]
  A public and private key pair used for encrypting and decrypting data or signing and verifying messages

Key usage Help me choose

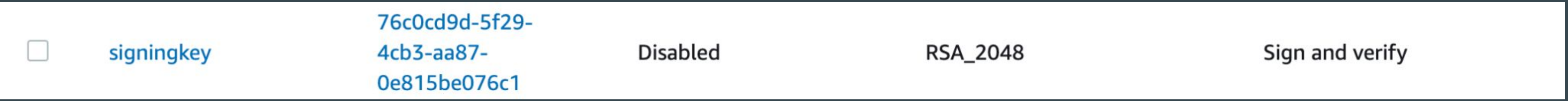
- [ ] Encrypt and decrypt
  Use the key only to encrypt and decrypt data.
- [✓ Sign and verify]
  Key pairs for digital signing

  Uses the private key for signing and the public key for verification.

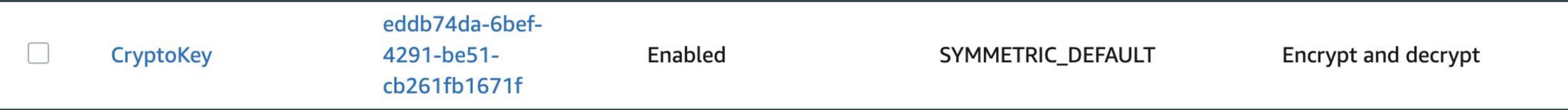
Key spec Help me choose

- [✓ RSA_2048]
- RSA_3072
- RSA_4096
- ECC_NIST_P256
- ECC_NIST_P384
- ECC_NIST_P521
- ECC_SECG_P256K1

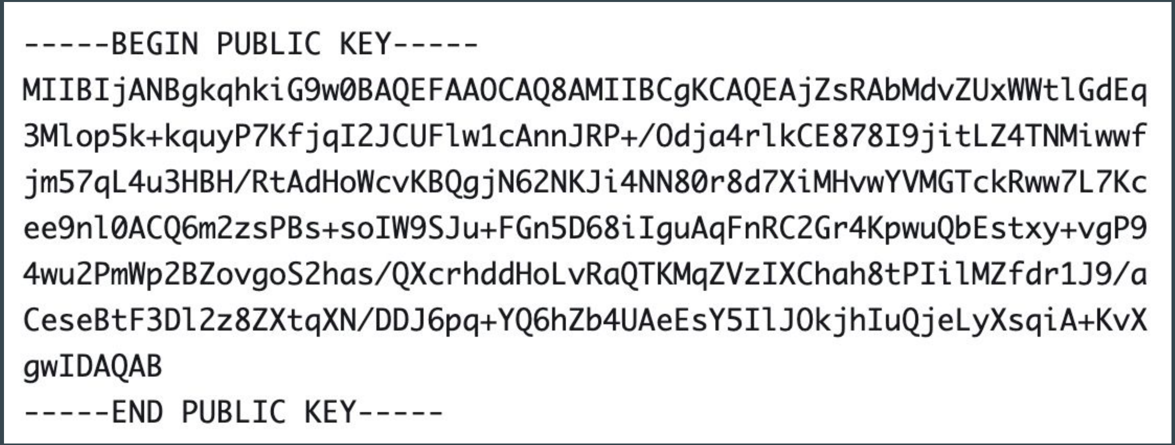
Keys



☐	signingkey	76c0cd9d-5f29-4cb3-aa87-0e815be076c1	Disabled	RSA_2048	Sign and verify

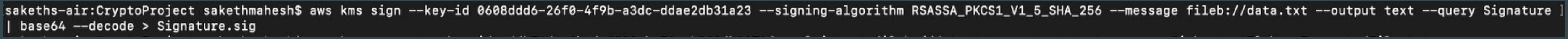


☐	CryptoKey	eddb74da-6bef-4291-be51-cb261fb1671f	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt

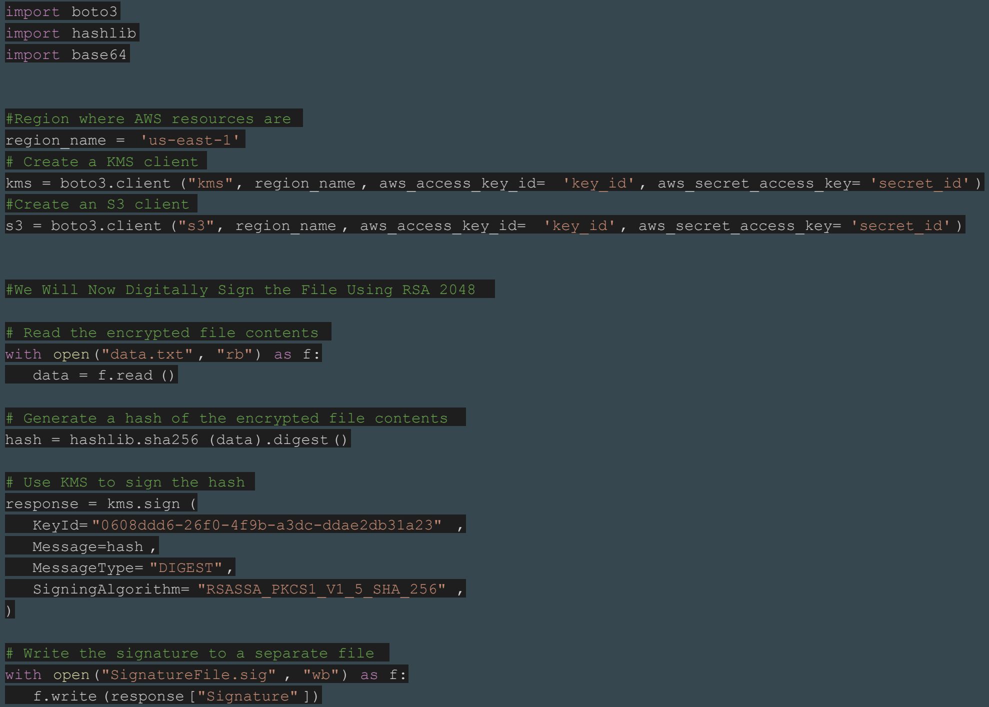


```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAjZsRAbMdvZUxWWtlGdEq
3Mlop5k+kquyP7KfjqI2JCUFlw1cAnnJRP+/Odja4rlkCE878I9jitLZ4TNMiwwf
jm57qL4u3HBH/RtAdHoWcvKBQgjN62NKJi4NN80r8d7XiMHvwYVMGTckRww7L7Kc
ee9nl0ACQ6m2zsPBs+soIW9SJu+FGn5D68iIguAqFnRC2Gr4KpwuQbEstxy+vgP9
4wu2PmWp2BZovgoS2has/QXcrhddHoLvRaQTKMqZVzIXChah8tPIilMZfdr1J9/a
CeseBtF3Dl2z8ZXtqXN/DDJ6pq+YQ6hZb4UAeEsY5IlJOkjhIuQjeLyXsqiA+KvX
gwIDAQAB
-----END PUBLIC KEY-----
```

Sign the Text File

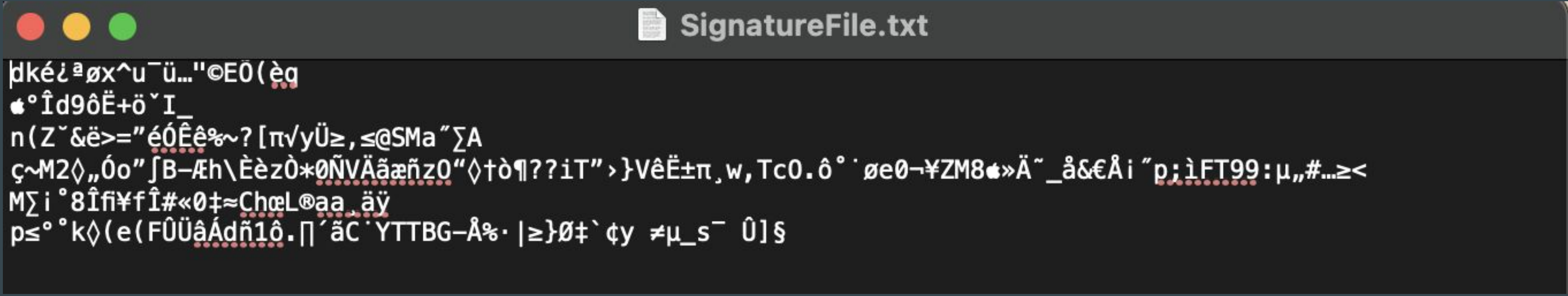


```
saketh-air:CryptoProject sakethmahesh$ aws kms sign --key-id 0608ddd6-26f0-4f9b-a3dc-ddae2db31a23 --signing-algorithm RSASSA_PKCS1_V1_5_SHA_256 --message fileb://data.txt --output text --query Signature 
| base64 --decode > Signature.sig
```



```python
import boto3
import hashlib
import base64


#Region where AWS resources are
region_name =  'us-east-1'
# Create a KMS client
kms = boto3.client ("kms", region_name , aws_access_key_id=  'key_id', aws_secret_access_key= 'secret_id')
#Create an S3 client
s3 = boto3.client ("s3", region_name , aws_access_key_id=  'key_id', aws_secret_access_key= 'secret_id')


#We Will Now Digitally Sign the File Using RSA 2048

# Read the encrypted file contents
with open("data.txt" , "rb") as f:
   data = f.read ()

# Generate a hash of the encrypted file contents
hash = hashlib.sha256 (data).digest()

# Use KMS to sign the hash
response = kms.sign (
   KeyId="0608ddd6-26f0-4f9b-a3dc-ddae2db31a23" ,
   Message=hash ,
   MessageType= "DIGEST",
   SigningAlgorithm= "RSASSA_PKCS1_V1_5_SHA_256" ,
)

# Write the signature to a separate file
with open("SignatureFile.sig" , "wb") as f:
   f.write (response ["Signature" ])
```

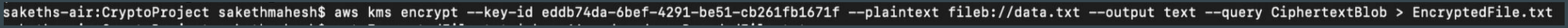
Signature File



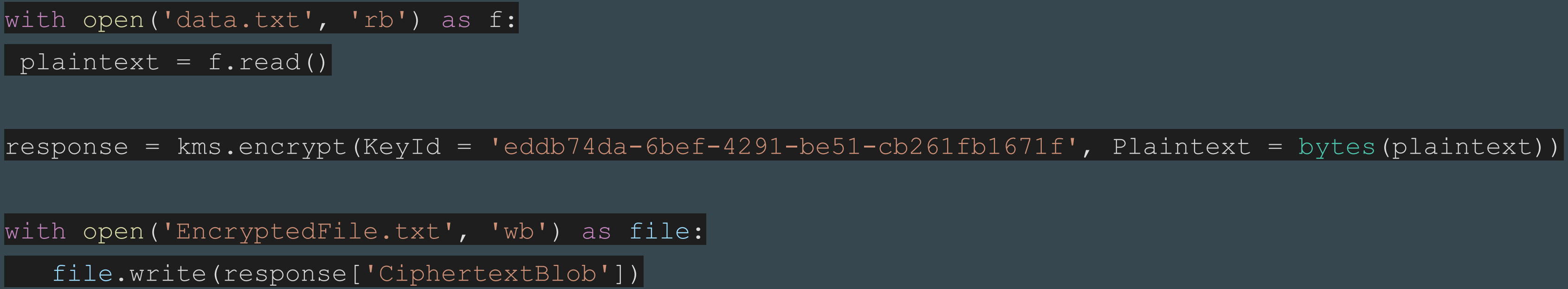
SignatureFile.txt

```
dké¿ªøx^u¯ü…"©EÖ(èq
°Îd9ôË+öˇI_
n(Z˘&ë>=”éÓÊê%~?[π√yÜ≥,≤@SMa˝∑A
ç~M2◊„Óo”∫B–Æh\ÈèzÒ*0ÑVÄãæñzO“◊†ò¶??iT”›}VêË±π¸w,TcO.ô˚˙øe0¬¥ZM8»Ä˜_å&€Åi˝p;ìFT99:µ„#…≥<
M∑i˚8Îfi¥fÎ#«0‡≈ChœL®aa¸äÿ
p≤°˚k◊(e(FÛÜâÁdñ1ô.∏´ãC˙YTTBG–Å%·|≥}Ø‡`¢y ≠µ_s¯ Û]§
```

Encrypt the Text File



```
saketh-air:CryptoProject sakethmahesh$ aws kms encrypt --key-id eddb74da-6bef-4291-be51-cb261fb1671f --plaintext fileb://data.txt --output text --query CiphertextBlob > EncryptedFile.txt
```



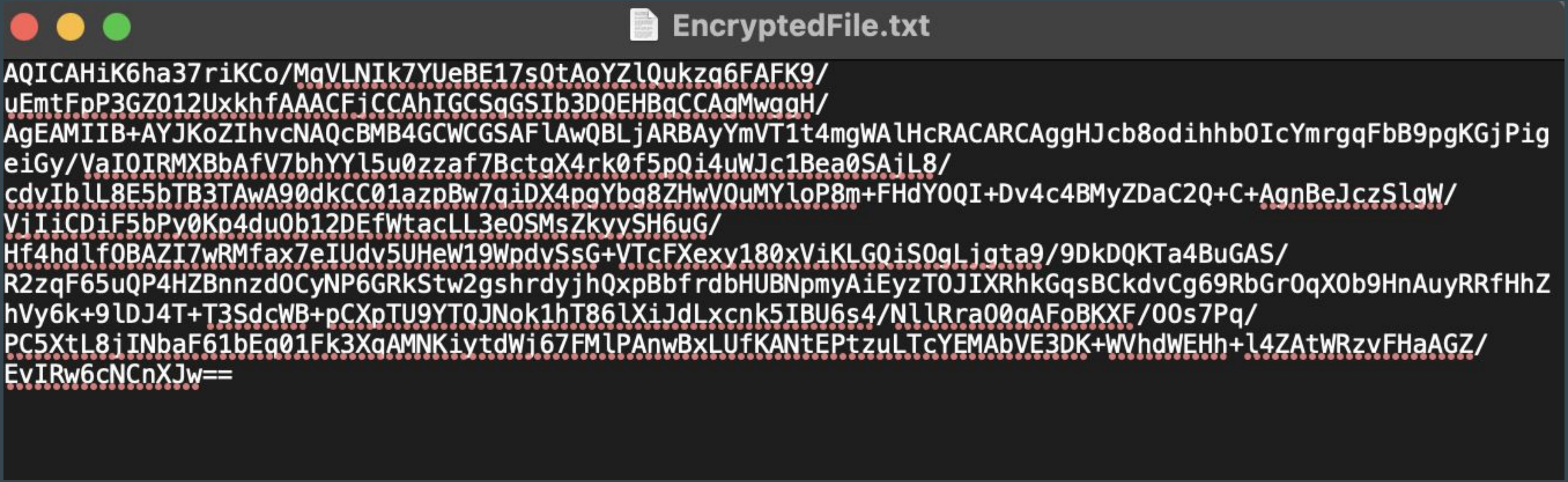
```python
with open('data.txt', 'rb') as f:
 plaintext = f.read()

response = kms.encrypt(KeyId = 'eddb74da-6bef-4291-be51-cb261fb1671f', Plaintext = bytes(plaintext))

with open('EncryptedFile.txt', 'wb') as file:
    file.write(response['CiphertextBlob'])
```

Note: AWS KMS symmetric encryption uses AES 256 by default

AES 256 Alg

1. Key Expansion: 256 bit key is expanded into a large set of round keys
2. Round Key Addition: Plaintext combines with first round key using XOR operation
3. Substitution: Non linear substitution where each byte is replaced with another
4. Shift Rows: Rows are shifted
5. Mix Columns: Columns are mixed
6. Add Round Key: Same as step 2 (process repeats for multiple rounds)
7. Substitution and shift rows are performed again

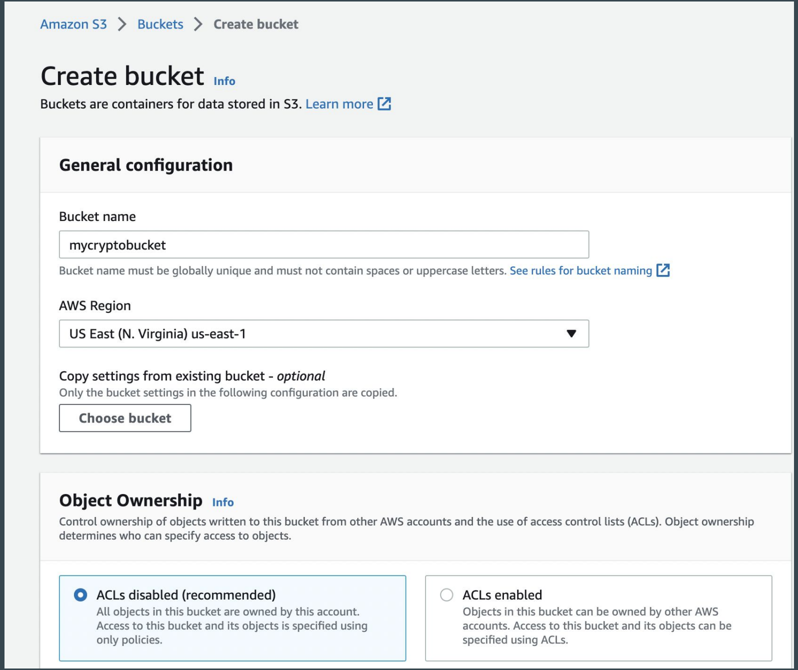
Encrypted File



EncryptedFile.txt

```
AQICAHiK6ha37riKCo/MqVLNIk7YUeBE17sOtAoYZlQukzq6FAFK9/
uEmtFpP3GZO12UxkhfAAACFjCCAhIGCSqGSIb3DQEHBqCCAgMwggH/
AgEAMIIB+AYJKoZIhvcNAQcBMB4GCWCGSAFlAwQBLjARBAyYmVT1t4mgWAlHcRACARCAggHJcb8odihhbOIcYmrgqFbB9pgKGjPig
eiGy/VaIOIRMXBbAfV7bhYYl5u0zzaf7BctgX4rk0f5pOi4uWJc1Bea0SAjL8/
cdvIblL8E5bTB3TAwA90dkCC01azpBw7qiDX4pgYbg8ZHwVOuMYloP8m+FHdYOQI+Dv4c4BMyZDaC2Q+C+AgnBeJczSlgW/
VjIiCDiF5bPy0Kp4duOb12DEfWtacLL3eOSMsZkyySH6uG/
Hf4hdlfOBAZI7wRMfax7eIUdv5UHeW19WpdvSsG+VTcFXexy180xViKLGQiSOgLjgta9/9DkDQKTa4BuGAS/
R2zqF65uQP4HZBnnzdOCyNP6GRkStw2gshrdyjhQxpBbfrdbHUBNpmyAiEyzTOJIXRhkGqsBCkdvCg69RbGrOqXOb9HnAuyRRfHhZ
hVy6k+9lDJ4T+T3SdcWB+pCXpTU9YTQJNok1hT86lXiJdLxcnk5IBU6s4/NllRraO0qAFoBKXF/OOs7Pq/
PC5XtL8jINbaF61bEq01Fk3XqAMNKiytdWj67FMlPAnwBxLUfKANtEPtzuLTcYEMAbVE3DK+WVhdWEHh+l4ZAtWRzvFHaAGZ/
EvIRw6cNCnXJw==
```

Configure My S3 Bucket



Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3. Learn more

General configuration

Bucket name: mycryptobucket

Bucket name must be globally unique and must not contain spaces or uppercase letters. See rules for bucket naming

AWS Region: US East (N. Virginia) us-east-1

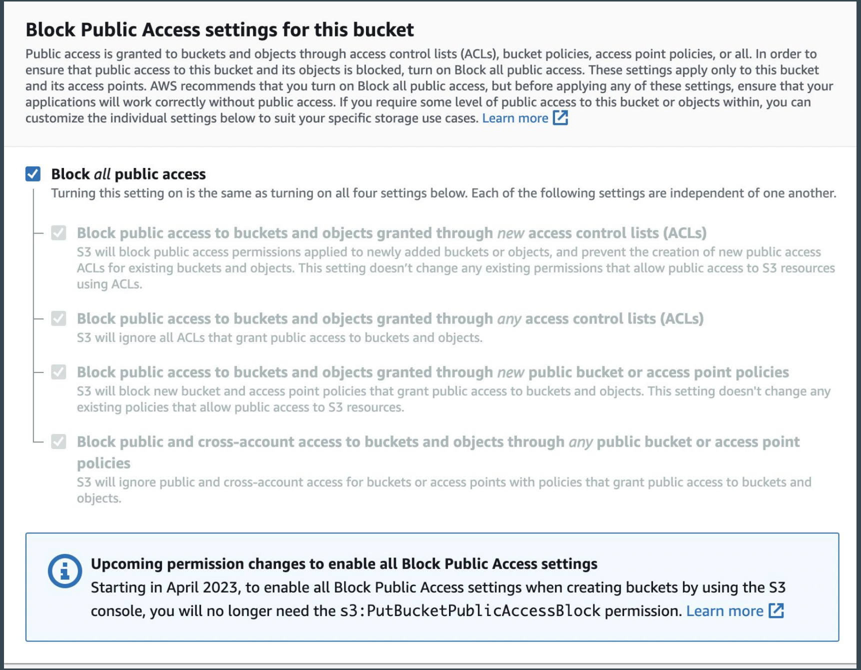
Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

- ● ACLs disabled (recommended) — All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
- ○ ACLs enabled — Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.



Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

☑ **Block *all* public access**

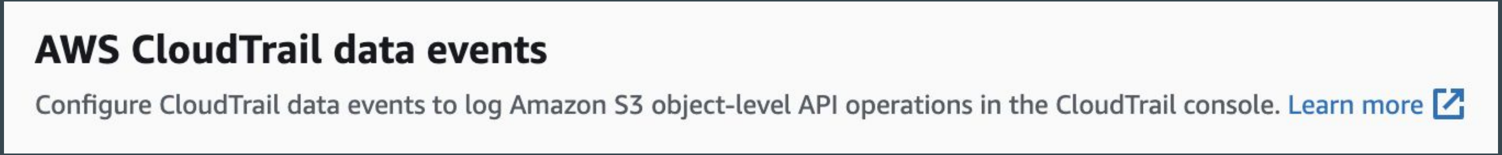
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☑ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
  S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☑ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
  S3 will ignore all ACLs that grant public access to buckets and objects.
- ☑ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
  S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☑ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
  S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Upcoming permission changes to enable all Block Public Access settings

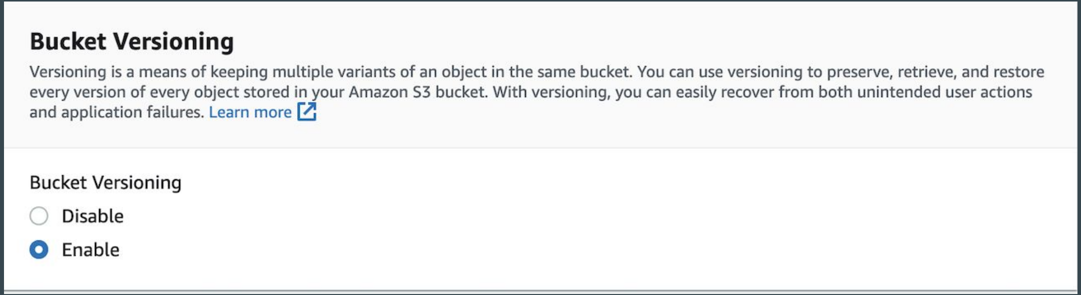
Starting in April 2023, to enable all Block Public Access settings when creating buckets by using the S3 console, you will no longer need the `s3:PutBucketPublicAccessBlock` permission. Learn more

Configure Bucket Continued



AWS CloudTrail data events

Configure CloudTrail data events to log Amazon S3 object-level API operations in the CloudTrail console. Learn more

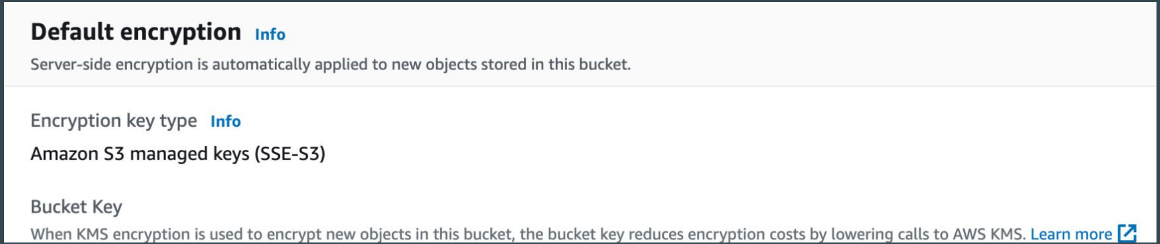


Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning

Disable [✓ Enable]



Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

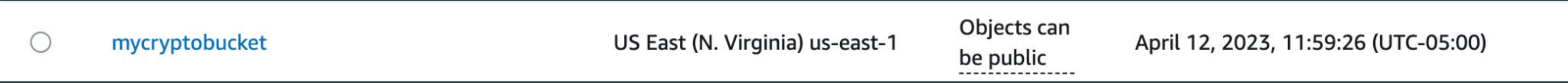
Encryption key type Info

Amazon S3 managed keys (SSE-S3)

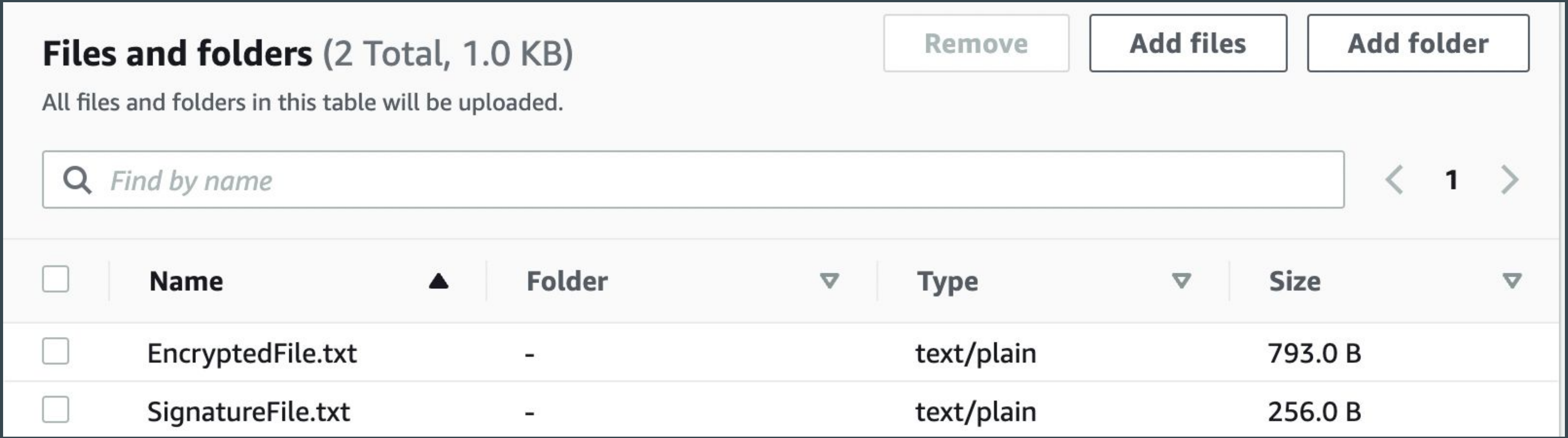
Bucket Key

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

Store Files (Objects) in Bucket



○	mycryptobucket	US East (N. Virginia) us-east-1	Objects can be public	April 12, 2023, 11:59:26 (UTC-05:00)



Files and folders (2 Total, 1.0 KB)

All files and folders in this table will be uploaded.

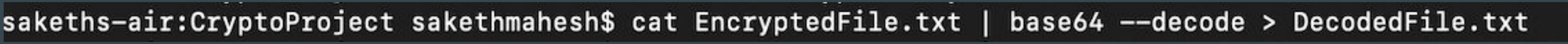
Name	Folder	Type	Size
EncryptedFile.txt	-	text/plain	793.0 B
SignatureFile.txt	-	text/plain	256.0 B

How do We Retrieve Files?

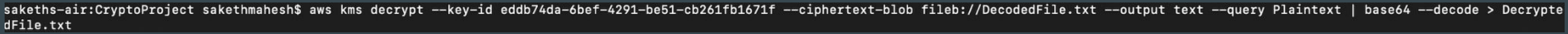
- Scenario: Let's say I deleted those files from my machine and left them in the S3 bucket for a year. Now, I want to access those files again. What do I do?

- Solution:
  - Download files to machine
  - Decode/decrypt encrypted file
  - Compare the hash of decrypted file to the digital signature
    - If they match: Signature is valid (files have not been tampered with)
    - If they don't match: Signature is invalid (Files have been tampered/corrupted)

Decode/Decrypt the Encrypted File



```
sakeths-air:CryptoProject sakethmahesh$ cat EncryptedFile.txt | base64 --decode > DecodedFile.txt
```

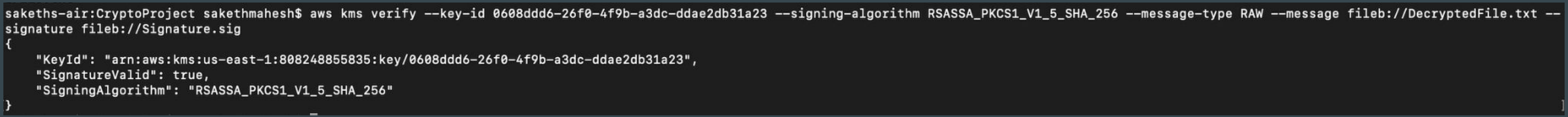


```
sakeths-air:CryptoProject sakethmahesh$ aws kms decrypt --key-id eddb74da-6bef-4291-be51-cb261fb1671f --ciphertext-blob fileb://DecodedFile.txt --output text --query Plaintext | base64 --decode > Decrypte
dFile.txt
```

```
#Open the Encrypted File again and decode base 64
with open('EncryptedFile.txt' , 'rb') as file:
   encrypted_data =  file.read()
   decoded_data = base64.b64decode (encrypted_data)

# Write the decoded data to a new file
with open('DecodedFile.txt' , 'wb') as file:
   file.write(decoded_data)

# Read the decoded data from the file into a variable
with open('DecodedFile.txt' , 'rb') as file:
   cipher_text =  file.read()

#Decrypt the decoded data
response = kms.decrypt (KeyId = 'eddb74da-6bef-4291-be51-cb261fb1671f' , CiphertextBlob=cipher_text )

# Write the decrypted data to a new file
with open('DecryptedFile.txt' , 'wb') as file:
   file.write(response['Plaintext'])

#Open the decrypted file and read contents
with open('DecryptedFile.txt' , 'r') as file:
 decrypted_data =  file.read()
#Create a hash of the decrypted data
hash = hashlib.sha256 (decrypted_data).digest()
```

Note: decoding into base64 means to decode the binary data into an ASCII format(64 characters)

Verify Signature



```
saketths-air:CryptoProject sakethmahesh$ aws kms verify --key-id 0608ddd6-26f0-4f9b-a3dc-ddae2db31a23 --signing-algorithm RSASSA_PKCS1_V1_5_SHA_256 --message-type RAW --message fileb://DecryptedFile.txt --
signature fileb://Signature.sig
{
    "KeyId": "arn:aws:kms:us-east-1:808248855835:key/0608ddd6-26f0-4f9b-a3dc-ddae2db31a23",
    "SignatureValid": true,
    "SigningAlgorithm": "RSASSA_PKCS1_V1_5_SHA_256"
}
```

```python
#Open the signature text file and read contents
with open('SignatureFile.txt', 'rb') as f:
    signature = f.read()

#We are checking that the hash of the decrypted signature file matches the
#hash of the decrypted text file
response = kms.verify(
    KeyId='0608ddd6-26f0-4f9b-a3dc-ddae2db31a23',
    Message=hash,
    Signature=signature,
    MessageType='DIGEST',
    SigningAlgorithm='RSASSA_PKCS1_V1_5_SHA_256'
)

#Simple print statement for clarity
if response['SignatureValid']:
    print('Signature is valid')
else:
    print('Signature is not valid')
```

Final Remarks

- Cryptography has extensive use cases in cloud computing
- This small project shows how powerful cryptographic concepts can really be when applied in real life scenarios