

Reduce encryption costs by using Amazon S3 Bucket Keys on existing objects

by Mathieu Seguin and Aditya Badhwar | on 30 MAR 2022 | in [Amazon Athena](#), [Amazon Simple Storage Service \(S3\)](#), [AWS Cloud Financial Management](#), [AWS CloudTrail](#), [AWS Key Management Service](#), [AWS Lambda](#), [Customer Solutions](#), [Security, Identity, & Compliance](#), [Storage](#), [Technical How-To](#) | [Permalink](#) | [Comments](#) | [Share](#)

As more organizations look to operate faster and at scale, they need ways to meet critical compliance requirements and improve data security. Encryption is a critical component of a defense in depth strategy, and when used correctly, can provide an additional layer of protection above basic access control. However, workloads that access millions or billions of encrypted objects can generate large request volumes, which can lead to high encryption/decryption cost. Given this, many organizations are looking for ways to ensure data security at scale while optimizing for cost.

[Simon Data](#) is a customer data platform that enables driving marketing results faster with a solution purpose built for growth. Simon Data was incurring high [AWS Key Management Service \(AWS KMS\)](#) costs because of API requests to AWS KMS for decryption of billions of objects in [Amazon S3](#). In December of 2020, AWS introduced the [Amazon S3 Bucket Keys](#) feature to help customers reduce the costs of server-side encryption with AWS Key Management Service (SSE-KMS) by decreasing the request traffic from S3 to KMS. S3 Bucket Keys reduce request traffic through use of bucket-level keys generated by KMS, instead of individual KMS keys for KMS encrypted objects, with each bucket-level key creating a unique data key for each *new* object added to a bucket, thereby avoiding the need for additional KMS requests to complete encryption operations. To minimize migration cost and processing time for billions of S3 objects, Simon Data opted to enable S3 Bucket Keys on the objects that triggered AWS KMS API calls within the last three months.

If all your *existing* objects in a bucket use the same KMS key, [enabling S3 Bucket Keys for them is relatively straightforward](#). However, *if your objects are encrypted using different KMS keys*, enabling the feature requires a few extra steps. As Simon Data was using multiple [customer managed keys](#) for objects within same Amazon S3 buckets, it was not straightforward for them to benefit from enabling Amazon S3 Bucket Keys. In addition, accessing objects created before the feature got enabled will continue to trigger requests to AWS KMS.

In this blog post, we cover how Simon Data reduced its AWS KMS monthly spending by over 80 percent by applying S3 Bucket Keys to *existing* objects encrypted using different KMS keys. With this solution you can greatly reduce request traffic between Amazon S3 and AWS KMS, allowing you to use your encrypted data with less cost concerns while keeping your data secure.

Enabling S3 Bucket Keys on mixed SSE-KMS keys buckets

How can you enable S3 Bucket Keys for existing objects encrypted with different SSE-KMS keys within the same bucket?

In order to achieve this with maximum results while keeping migration costs under control, you want to enable the feature solely on objects that:

- Are SSE-KMS encrypted
- Have the S3 Bucket Keys feature disabled
- are accessed frequently and generating requests to SSE-KMS on a regular basis

To identify objects that are using S3 Bucket Keys, you can use one of the following methods:

1. List the buckets objects (and their S3 Bucket Key status) using [Amazon S3 Inventory](#).
2. Use [Amazon Athena](#) to extract the list of S3 objects that generated an AWS KMS decrypt request from the [AWS CloudTrail](#) logs.

Given Simon Data's S3 object scale, they preferred the second option as they wanted to only enable bucket keys (re-encrypt) for frequently accessed objects, and information about object last access is available in AWS CloudTrail.

Simon Data opted for AWS CloudTrail, which works as follows:

1. Identify the objects that have triggered an SSE-KMS request recently by querying the CloudTrail logs using [Amazon Athena](#) and then exporting it as a CSV file.
2. Feed the CSV file to [S3 Batch Operations](#). Further, we have to trigger an [AWS Lambda](#) function instead of using the built-in copy operation since objects that are encrypted have different KMS keys.
3. For each object in the list, the Lambda will:
 - Retrieve the current object properties, including the current SSE-KMS key ID.
 - Double-check whether S3 Bucket Keys can be enabled, and if they need to be enabled.
 - Enable S3 Bucket Keys by copying the object in place.

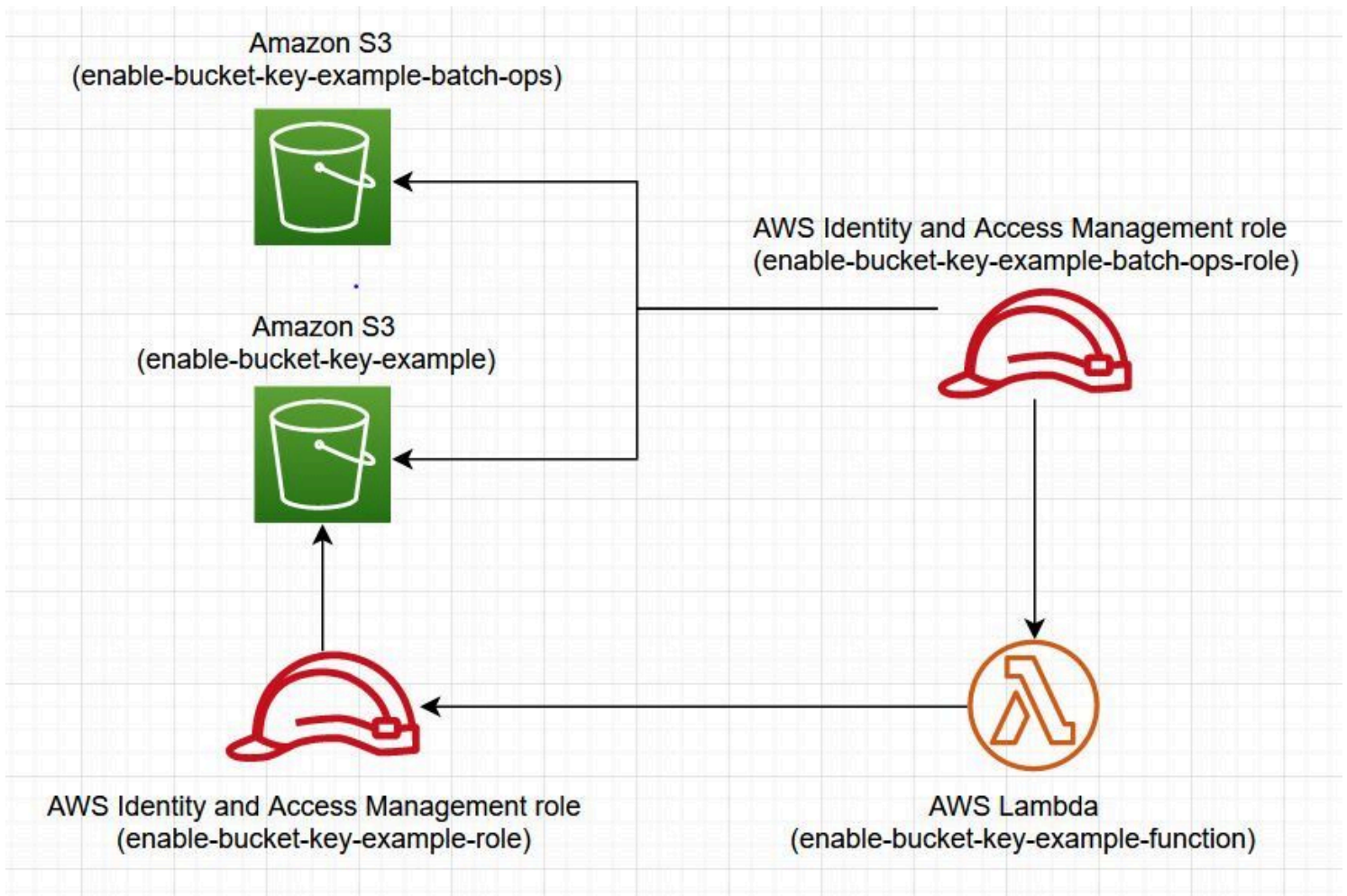
Implementation

To begin with, let's have a look at how to set up the required infrastructure.

You will need:

- Two S3 buckets
 - One with the list of files to process
 - One for the S3 Batch Operations output files
- One AWS Lambda function with its IAM role
 - Enables S3 Bucket Keys on our S3 objects one-by-one
- One IAM role for S3 Batch Operations
 - Runs the AWS Lambda function on a list of files

These parts are depicted in the following architecture diagram:



The following steps along with this [CloudFormation template](#) sets up all of the necessary resources.

1. Clone the repository to local disk

```
$ git clone git@github.com:simon-data/enable-bucket-key-example.git && cd enable-bucket-key-example
```

2. Create the stack using CloudFormation

```
$ aws cloudformation create-stack \
  --stack-name enable-bucket-key-example \
  --template-body file://enable-bucket-key-example-template.cloudformation.yaml \
  --capabilities CAPABILITY_NAMED_IAM \
  --no-cli-pager \
  --output json
```

You can also customize the bucket names by configuring the prefix value as such:

```
--parameters ParameterKey=Prefix,ParameterValue=enable-bucket-key-example2
```

3. Set the (bucket) stage

Next, recreate your bucket state.

Before enabling S3 Bucket Keys at the bucket level, create and upload a test file.

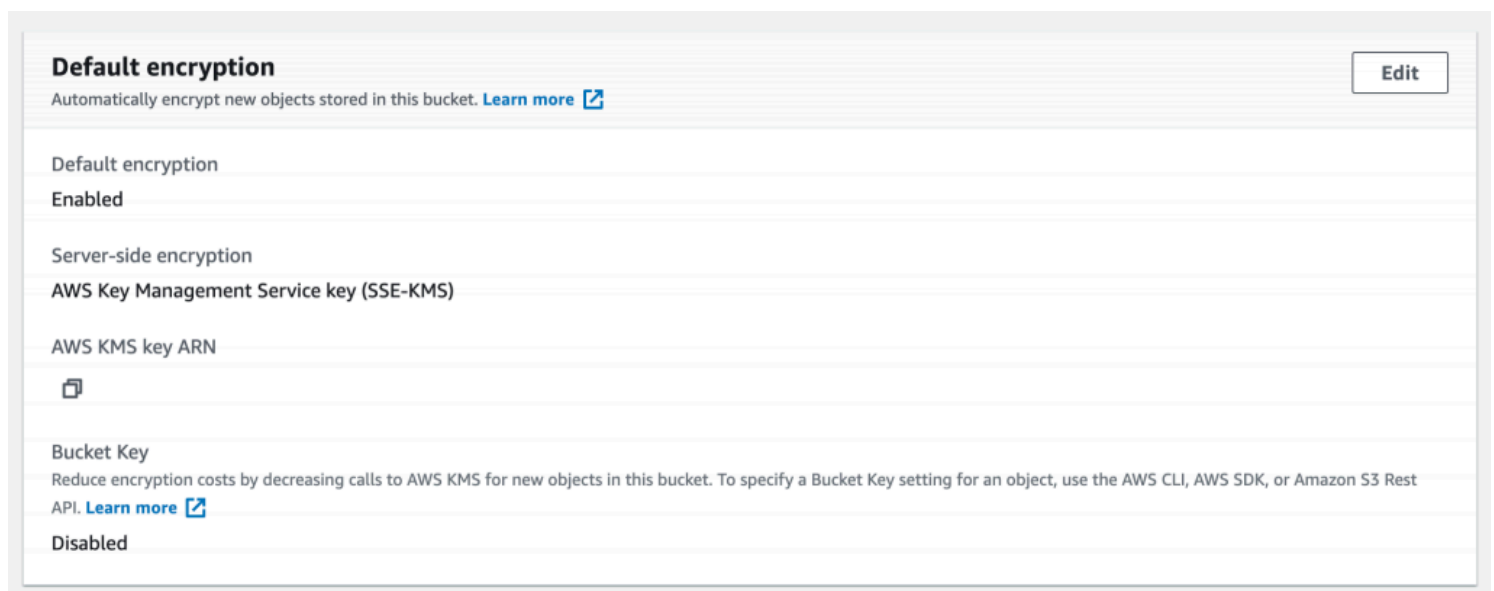
```
$ echo "Before bucket-level Bucket Key enabled" > before.txt

$ aws s3api put-object \
  --bucket enable-bucket-key-example \
  --key before.txt \
  --body before.txt \
  --no-cli-pager

{
  "ETag": "\"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\"",
  "ServerSideEncryption": "aws:SSE-KMS",
  "SSE-SSE-KMSKeyId": "arn:aws:SSE-KMS:us-east-1:XXXXXXXXXXXX:key/XXXXXXXX-XXXX-XXXX-XXXX"
}
```

Then, enable the S3 Bucket Key, by default, to see the difference.

Go to your bucket's page, and then in the **Properties** tab, scroll down to the **Default encryption** section and select **Edit**.




Default encryption Edit

Automatically encrypt new objects stored in this bucket. [Learn more](#)

Default encryption
Enabled

Server-side encryption
AWS Key Management Service key (SSE-KMS)

AWS KMS key ARN


Bucket Key
Reduce encryption costs by decreasing calls to AWS KMS for new objects in this bucket. To specify a Bucket Key setting for an object, use the AWS CLI, AWS SDK, or Amazon S3 Rest API. [Learn more](#)
Disabled

Enable the S3 Bucket Key and select **Save changes**.

Bucket Key

Reduce encryption costs by decreasing calls to AWS KMS for new objects in this bucket. To specify a Bucket Key setting for an object, use the AWS CLI, AWS SDK, or Amazon S3 Rest API. [Learn more](#)

☐ Disable
 ☒ **Enable**

Cancel

Save changes

Once that's done, upload a second file to validate the bucket key feature enabled in previous step

```
$ echo "After bucket-level S3 Bucket Key enabled" > after.txt

$ aws s3api put-object \
  --bucket enable-bucket-key-example \
  --key after.txt \
  --body after.txt \
  --no-cli-pager

{
  "ETag": "\"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\"",
  "ServerSideEncryption": "aws:SSE-KMS",
  "SSEKMSKeyId": "arn:aws:SSE-KMS:us-east-1:XXXXXXXXXXXX:key/XXXXXXXX-XXXX-XXXX-XXXX",
  "BucketKeyEnabled": true
}
```

As you can see, this one has the BucketKeyEnabled property set to true.

If we double check our other file, we would notice that the S3 Bucket Key still hasn't been enabled (the BucketKeyEnabled attribute is missing):

```
$ aws s3api head-object --bucket enable-bucket-key-example --key before.txt --no-cli-pager

{
  "AcceptRanges": "bytes",
  "LastModified": "2022-03-29T20:01:18+00:00",
  "ContentLength": 39,
  "ETag": "\"fdfad7141cf16b2982d0c826ed9a2f\"",
  "ContentType": "binary/octet-stream",
```

```
"ServerSideEncryption": "aws:kms",
"Metadata": {},
"SSEKMSKeyId": "arn:aws:kms:us-east-1:XXXXXXXXXXXX:key/XXXXXXXX-XXXX-XXXX-XXXX-XXXX"
}
```

4. Prepare the AWS Lambda function

So far, we have one bucket with two files. One has the S3 Bucket Key enabled, and the other does not. Let's enable the S3 Bucket Key for our "before.txt" file!

a. Install the dependencies

Our AWS Lambda uses boto3, which we can install in a local "package" directory using the following command:

```
$ python -m pip install --target ./package boto3 botocore
```

We also need [s3transfer](#), which automatically uses multipart uploads when required. Unfortunately, at the time of this writing `s3transfer` does not support the `BucketKey` argument. In the meantime, we can use our own fork:

```
$ rm -rf package/s3transfer && python -m pip install --target ./package git+https://gi
```

b. Update the AWS Lambda function

Next, create a .zip file and update your AWS Lambda function:

```
$ bash package_lambda.sh
$ aws lambda update-function-code \
  --function-name enable-bucket-key-example-function \
  --zip-file fileb://enable-bucket-key-example-function.zip \
  --no-cli-pager \
  --output json
```

5. Enable the S3 Bucket Key using S3 Batch Operations

With our Lambda function updated, we are now ready to start processing files!

a. Create an S3 Batch Operations manifest

All we need is a manifest file to feed to S3 Batch Operations, with the list of files we want to enable the S3 Bucket Key for.

S3 Batch Operations expects a comma-separated values (CSV) manifest file with following two columns:

bucket_name, object_key

Create the manifest for your before.txt file:

```
$ echo "enable-bucket-key-example,before.txt" > objects_to_enable_bucket_key_for.csv

$ aws s3api put-object \
  --bucket enable-bucket-key-example-batch-ops \
  --key objects_to_enable_bucket_key_for.csv \
  --body objects_to_enable_bucket_key_for.csv \
  --no-cli-pager \
  --output json
```

b. Run the S3 Batch Operations job

Finally, start your Batch Operations job!

```
$ export AWS_ACCOUNT_ID=XXXXXXXXXXXX # Replace with your account id
$ aws s3control create-job \
  --account-id ${AWS_ACCOUNT_ID} \
  --no-confirmation-required \
  --priority 10 \
  --role-arn "arn:aws:iam::${AWS_ACCOUNT_ID}:role/enable-bucket-key-example-batch-ops" \
  --operation '{
    "LambdaInvoke": {
      "FunctionArn": "arn:aws:lambda:us-east-1:${AWS_ACCOUNT_ID}:function:enable-bucket-key-example"
    }
  }' \
  --manifest '{
    "Spec": {
      "Format": "S3BatchOperations_CSV_20180820",
      "Fields": [
        "Bucket",
        "Key"
      ]
    },
    "Location": {
      "ObjectArn": "arn:aws:s3:::enable-bucket-key-example-batch-ops/objects_to_enable_bucket_key_for.csv"
```

```

    "ETag": '"$(aws s3api head-object --bucket enable-bucket-key-example-batch-ops --key before.txt --no-cli-pager)' \
  }' \
  --report '{"Enabled": false}' \
  --no-cli-pager \
  --output json

```

Once the job is complete, we can validate that our before.txt file was successfully processed and that the **BucketKeyEnabled** property is set to true:

```

$ aws s3api head-object --bucket enable-bucket-key-example --key before.txt --no-cli-pager

{
  "AcceptRanges": "bytes",
  "LastModified": "2022-03-29T20:03:36+00:00",
  "ContentLength": 39,
  "ETag": "\"fdfad27141cf16b2982d0c826ed9a2f\"",
  "ContentType": "binary/octet-stream",
  "ServerSideEncryption": "aws:kms",
  "Metadata": {},
  "SSEKMSKeyId": "arn:aws:kms:us-east-1:XXXXXXXXXXXX:key/XXXXXXXX-XXXX-XXXX-XXXX-XXXX",
  "BucketKeyEnabled": true
}

```

Applying at scale

The following assumes that you have an [existing trail configured](#) as well as an [Athena table](#) to query your logs.

The architecture created in the example is already scalable. It is the same one we used internally to apply this change to hundreds of millions of files.

As mentioned before, while unique S3 requests and Lambda costs are insignificant, at scale, they can quickly add up. Therefore, we want to focus our migration efforts on the files that have triggered SSE-KMS decrypt requests in the last few months.

We can retrieve a list of objects that triggered SSE-KMS decrypt requests using the following Athena query (the query is taking the previous day context – you can modify the query to look at any window that fits your use case).


```
SELECT
  REGEXP_EXTRACT(obj_arn, '::::([^\./]+)', 1) as bucket,
  REGEXP_EXTRACT(obj_arn, '^[^\./]+\/(.*)', 1) as path
FROM (
  SELECT
    DISTINCT CAST(json_extract(requestparameters, '$.encryptionContext.aws:s3:arn')
  FROM cloudtrail_logs_part as logs
  CROSS JOIN UNNEST(logs.resources) unnested (resources_entry)
  WHERE eventsource='SSE-KMS.amazonaws.com'
    AND year = CAST(EXTRACT(year FROM current_date - interval '1' day) as VARCHAR(4))
    AND month = CAST(EXTRACT(month FROM current_date - interval '1' day) as VARCHAR(2))
    AND day = CAST(EXTRACT(day FROM current_date - interval '1' day) as VARCHAR(2))
)
WHERE obj_arn LIKE '%/%'
```

You can then select **Download Results** to get a CSV file, feed the file to S3 Batch Operations, and let it handle it from there!

S3 Bucket Keys considerations

Before enabling S3 Bucket Keys, there are a few things to consider:

Migration costs

- S3 Batch Operations, SSE-KMS, and Lambda will incur costs relative to the number of objects that are processed.
- S3 Versioning: If S3 Versioning is enabled, you will be creating a new version for each object you are enabling the S3 Bucket Key for, effectively doubling your hosting costs for the processed objects if you do not remove the older versions.

S3 Inventory configurations

S3 Inventory lists are a great way to extract the list of objects from a bucket. This is the preferred approach if you want to process all the existing objects on your bucket or within a given prefix.

Only favor the CloudTrail solution if your goal is to solely process the objects that have made an AWS KMS decrypt request recently. Note that this option requires having CloudTrail set up beforehand.

Safe rollout plan and rollback mechanism

To help ensure that your migrations are successful, assume they *will* fail.

The most straightforward rollback implementation is to enable versioning on your bucket and have a script ready to delete the latest version should you need to restore the previous version.

Cleaning up

Note that the steps described in this blog post will deploy resources to your AWS account that may be chargeable. You can easily decommission the infrastructure created by emptying the buckets and deleting the CloudFormation stack.

Conclusion

In this blog, we've walked through the steps that Simon Data followed to implement Amazon S3 Bucket Keys for objects with different AWS KMS keys within same bucket. By doing so, Simon Data was able to significantly reduce request traffic from Amazon S3 to AWS KMS, decreasing AWS KMS costs by 80 percent.

With the solution demonstrated, you can effectively implement Amazon S3 Bucket Keys to reduce AWS KMS cost by lessening charges associated with requests and encryption operations, even when existing objects are originally managed with different KMS keys. Using S3 Bucket Keys, you can continue encrypting your objects while allowing more cost-optimized access to your data, giving you more flexibility and incentive to make the most of your data, all while maintaining security.

Thanks for reading this blog post! If you have any questions or suggestions, feel free to leave your feedback in the comments section.

TAGS: [Amazon Athena](#), [Amazon S3 Batch Operations](#), [Amazon S3 Bucket Keys](#), [Amazon S3 Inventory](#), [Amazon Simple Storage Service \(Amazon S3\)](#), [AWS Cloud Storage](#), [AWS CloudTrail](#), [AWS Key Management Service \(AWS KMS\)](#), [AWS Lambda](#)

Comments

What do you think?

3 Responses



2

Upvote



0

Funny



1

Love



0

Surprised



0

Angry



0

Sad

Comments and reactions for this thread are now closed.



o Comments

4 Prashanth ▼



Share

Best Newest Oldest

This discussion has been closed.

Subscribe

Privacy

Do Not Sell My Data