



Configuring Amazon S3 security settings and access controls



Getting started at an AWS hosted workshop

▼ S3 Security Best Practices

▼ Prepare Your Lab

Attach IAM Role to EC2 Instance

Connect to the EC2 Instance

Bucket Name

Bucket Name

▼ Lab 1 - S3 Security Exercises

Require HTTPS

Require SSE-KMS Encryption

Restrict Access to an S3 VPC Endpoint

Use AWS Config Rules to Detect a Public Bucket

Use Amazon Access Analyzer for S3

► Lab 2 - S3 Access Grants

► Lab 3 - Enabling Malware Protection for S3 by using GuardDuty

► Lab 4 - S3 Access Control Lists

Lab Summary

[Configuring Amazon S3 security settings and access controls](#) > [S3 Security Best Practices](#) > [Lab 1 - S3 Security Exercises](#)

Require SSE-KMS Encryption

In this exercise, we will configure default bucket encryption to SSE-KMS with customer managed keys to encrypt your data at rest and enforce encryption with a bucket policy.

In following with best practices, we will leverage both SSE-KMS and create a customer managed key. This is generally a best practice because it allows key usage auditing, ability to define key policies for access, and enables cross account sharing as needed in the organization.

We will start by creating a KMS key in the account and assigning it the proper permissions for encryption and decryption.

From the AWS console in the top search bar, search and select kms for Key Management Service

- Click the Create a Key button to get started
- We will keep the defaults and click Next as shown below:

Configure 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.



Generate and verify MAC

Use the key only to generate and verify hash-based message authentication codes (HMAC).

► Advanced options

Cancel

Next

In the next section, under Alias we will name it "s3lab" as shown.



Add labels

Alias

You can change the alias at any time. [Learn more](#)

Alias

s3lab

Description - optional

You can change the description at any time.

Description

Description of the key

Tags - optional

You can use tags to categorize and identify your KMS keys and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags.

Cancel

Previous

Next

Click Next

The next page is labeled Define key administrative permissions this is where we will enable an administrator who can perform different actions on the key, like modifying attributes or even deleting the key. Notice in the screenshot below we are logged in as the WSParticipantRole as noted in the top right and we typed "wsp" in the search box. Click on the checkbox next to the WSParticipantRole and click Next

The screenshot shows the 'Define key administrative permissions' page in the AWS IAM console. At the top, the user is logged in as 'WSParticipantRole/Participant @'. The page title is 'Define key administrative permissions'. Below the title, there is a section 'Key administrators (1/33)' with a search bar containing 'wsp' and a table listing the results. The table has columns for 'Name', 'Path', and 'Type'. The first row is 'WSParticipantRole' with a path of '/' and a type of 'Role'. The checkbox next to 'WSParticipantRole' is checked. Below the table, there is a section 'Key deletion' with a checkbox 'Allow key administrators to delete this key.' which is also checked. At the bottom right, there are buttons for 'Cancel', 'Previous', and 'Next'.

In the next section labeled Define key usage permissions we will create the IAM User or Role that is able to use the key. This defines who can perform decrypt and encrypt actions using the key. It is important to note, this allows customers to choose who has administrative access to the key and able to do cryptographic operations on the key, in our case this is two different users/roles. This is a key aspect to least privilege access and distinguishing different roles and responsibilities from a human perspective.

Type in ssw-base in the search bar and check the box next to ssw-base-template-SidS3AccessRole-xxxx where xxxx may be an alpha-numeric string. Click Next

Define key usage permissions

Key users (1/33)

Select the IAM users and roles that can use the KMS key in cryptographic operations. [Learn more](#)

Q ssw-base X 2 matches < 1 >

<input type="checkbox"/>	Name	Path
<input checked="" type="checkbox"/>	ssw-base-template-SidS3AccessRole-XeGjXBcFjImf	/
<input type="checkbox"/>	ssw-base-template-SidSecurityStack--SidS3AccessRole-j97ooF5j7tWp	/

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

Add another AWS account

Cancel

Previous

Next

At the last screen keep everything as default and scroll down to the bottom and click Finish.

You should now see a key created with an alias of `s3lab`. We will use this key later in the section.

Open an SSH session to the SID-security-instance using EC2 Instance Connect if it is not already open. Run the following commands to create a test file.

```
cd ~
echo "123456789abcdefg" > textfile
```



Run the following command in your SSH session to put a text01 object to your bucket.

```
aws s3api put-object --key text01 --body textfile --bucket ${bucket}
```



The request should succeed, notice how the encryption is AES256 (SSE-S3)

```
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text01 --body textfile --bucket ${bucket}
{
  "ETag": "\"3ca451faac980583cffaadf8b63e6820\"",
  "ServerSideEncryption": "AES256"
}
[ec2-user@storage-workshop ~]$
```

This is because the current bucket default encryption is configured as SSE-S3.

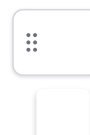
Let's update our bucket encryption default to use SSE-KMS so new objects will inherit SSE-KMS as it's default encryption.

From the AWS console in the top search bar, search and select S3.

- Click the bucket name starting with `sid-security-xxxxxxx`.
- Click on the Properties tab.
- Under Default encryption click Edit

Within the Edit Default Encryption, select the following:

- Encryption type, select **Server-side encryption with AWS Key Management Service keys (SSE-KMS)**
- AWS KMS key, select **Choose from your AWS KMS keys**
- Available AWS KMS keys, select **alias s3lab**



- Bucket Key, select **Enable**

Default encryption

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

Encryption type [Info](#)

☐ Server-side encryption with Amazon S3 managed keys (SSE-S3)
☒ Server-side encryption with AWS Key Management Service keys (SSE-KMS)
☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)
 Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the **Storage** tab of the [Amazon S3 pricing page](#).

AWS KMS key [Info](#)

☒ Choose from your AWS KMS keys
☐ Enter AWS KMS key ARN

Available AWS KMS keys

↻
Create a KMS key

Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

☐ Disable
☒ Enable

⚠ Changing the default encryption settings might cause in-progress replication and Batch Replication jobs to fail. These jobs might fail because of missing AWS KMS permissions on the IAM role that's specified in the replication configuration. If you change the default encryption settings, make sure that this IAM role has the necessary AWS KMS permissions. [Learn more](#)

Cancel
Save changes

Click **Save Changes**.

Run the following command in your EC2 SSH session to put a new text02 object to your bucket.

```
aws s3api put-object --key text02 --body textfile --bucket ${bucket}
```



The request should succeed, notice how the server side encryption is `aws:kms` instead of `AES256`

```
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text02 --body textfile --bucket ${bucket}
{"SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
 "ETag": "\"8c48255c9fa02c784dec50297dcdcf832\"",
 "ServerSideEncryption": "aws:kms"}
[ec2-user@storage-workshop ~]$
```

Changing the bucket default encryption will affect any new objects written to the bucket, but existing objects will retain their existing encryption.

Run the following commands in your EC2 SSH session to compare your text01 and text02 to your bucket.

```
aws s3api head-object --key text01 --bucket ${bucket}
aws s3api head-object --key text02 --bucket ${bucket}
```



text01 was written before we changed the bucket default, hence it is AES256 (SSE-S3)

text02 was written after we changed the bucket default, confirming it has inherited the `aws:kms` (KMS) encryption.



```
[ec2-user@storage-workshop ~]$ aws s3api head-object --key text01 --bucket ${bucket}
{
  "AcceptRanges": "bytes",
  "ContentType": "binary/octet-stream",
  "LastModified": "Mon, 09 Oct 2023 00:06:43 GMT",
  "ContentLength": 17,
  "ETag": "\"3ca451faac980583cffaadf8b63e6820\"",
  "ServerSideEncryption": "AES256",
  "Metadata": {}
}
[ec2-user@storage-workshop ~]$ aws s3api head-object --key text02 --bucket ${bucket}
{
  "AcceptRanges": "bytes",
  "ContentType": "binary/octet-stream",
  "LastModified": "Mon, 09 Oct 2023 00:15:07 GMT",
  "ContentLength": 17,
  "ETag": "\"8c48255c9fa02c784dec50297dcd832\"",
  "ServerSideEncryption": "aws:kms",
  "SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
  "Metadata": {}
}
[ec2-user@storage-workshop ~]$
```

For our scenario, let us assume we want certain prefixes to enforce SSE-KMS but still allow any encryption for other prefixes.

We will use a bucket policy to enforce that any new objects placed into our bucket under the `ssekms-only/` prefix must have SSE-KMS encryption.

From the AWS console in the top search bar, search and select S3.

- Click the bucket name starting with `sid-security-xxxxxxx`.
- Click on the Permissions tab.
- Under Bucket Policy click Edit

Remove the current policy and replace it with the bucket policy below and paste into the Bucket Policy Editor.

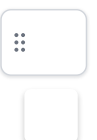
```
{
  "Id": "S3-Security-Deny-unless-SSE-KMS",
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3::BUCKET_NAME/ssekms-only/*",
    "Condition": {
      "StringNotEquals": {
        "s3:x-amz-server-side-encryption": "aws:kms"
      }
    }
  }]
}
```



Replace `BUCKET_NAME` with the bucket name you copied to your text editor and click Save changes.

Ensure your bucket policy keeps the prefix `ssekms-only/*` after the `BUCKET_NAME`

Your bucket policy should look similar to this example.




Edit bucket policy [Info](#)

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.

[Policy examples](#)
[Policy generator](#)

Bucket ARN

 arn:aws:s3:::sid-security-900ed8e0-6a30-11ed-961c-0a7f8723e72d

Policy

```

1 {
2   "Version": "2012-10-17",
3   "Id": "S3-Security-Deny-unless-SSE-KMS",
4   "Statement": [
5     {
6       "Effect": "Deny",
7       "Principal": "*",
8       "Action": "s3:PutObject",
9       "Resource": "arn:aws:s3:::sid-security-900ed8e0-6a30-11ed-961c-0a7f8723e72d/ssekms-only/*",
10      "Condition": {
11        "StringNotEquals": {
12          "s3:x-amz-server-side-encryption": "aws:kms"
13        }
14      }
15    }
16  ]
17 }
```

Run the following commands in your EC2 SSH session to put a new text03, text04, text05 object to your bucket.

```

aws s3api put-object --key text03 --body textfile --bucket ${bucket}
aws s3api put-object --key text04 --body textfile --server-side-encryption AES256 --bucket ${bucket}
aws s3api put-object --key text05 --body textfile --server-side-encryption aws:kms --bucket ${bucket}
```

Behavior is as follows:

- Put object for text03 will succeed, and is aws:kms because it inherits the default encryption SSE-KMS as no encryption method is specified.
- Put object for text04 will succeed, and is AES256 because we specify it to use SSE-S3. Since server-side-encryption is explicitly configured, it will take precedence over the default bucket encryption SSE-KMS.
- Put object for text05 will succeed, and is aws:kms because we specify it to use SSE-KMS.

```

[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text03 --body textfile --bucket ${bucket}
{"SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
"ETag": "\"fd3416dd85d49cbb2af7a6311268fc\"",
"ServerSideEncryption": "aws:kms"}
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text04 --body textfile --server-side-encryption AES256 --bucket ${bucket}
{"ETag": "\"3ca451faac9805830ffaadf8b63e6820\"",
"ServerSideEncryption": "AES256"}
[ec2-user@storage-workshop ~]$ aws s3api put-object --key text05 --body textfile --server-side-encryption aws:kms --bucket ${bucket}
{"SSEKMSKeyId": "arn:aws:kms:us-west-2:329328435893:key/fa8ca065-5d27-4d56-a685-dfa9781dd7c3",
"ETag": "\"b2ff160f750328185e97461fc2c7fb74\"",
"ServerSideEncryption": "aws:kms"}
[ec2-user@storage-workshop ~]$
```

Let's create new objects under the prefix ssekms-only/* to see how the bucket policy changes the behavior.

```

aws s3api put-object --key ssekms-only/text06 --body textfile --bucket ${bucket}
aws s3api put-object --key ssekms-only/text07 --body textfile --server-side-encryption AES256
aws s3api put-object --key ssekms-only/text08 --body textfile --server-side-encryption aws:kms
```

Behavior in the ssekms-only/* prefix is as follows:

- Put object for text06 will fail, although we have default encryption as SSE-KMS the bucket policy requires explicit definition for SSE-KMS.

- Put object for text07 will fail, since AES256 will use SSE-S3 and we only permit SSE-KMS encryption in this prefix per the bucket policy.
- Put object for text08 will succeed, since SSE-KMS encryption is explicitly defined and allowed by the bucket policy.

```
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text06 --body textfile --bucket ${bucket}
An error occurred (AccessDenied) when calling the PutObject operation: User: arn:aws:sts::092526071050:assumed-role/ssw-base-template-SidS3AccessRole-xgqCgtNjPrb8/i-06ee8c015c1d6e501 is not authorized to perform: s3:PutObject on resource: "arn:aws:s3:::sid-security-ca51ea00-acec-11ef-ab19-02dc4668901f/ssekms-only/text06" with an explicit deny in a resource-based policy
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text07 --body textfile --server-side-encryption AES256 --bucket ${bucket}
An error occurred (AccessDenied) when calling the PutObject operation: User: arn:aws:sts::092526071050:assumed-role/ssw-base-template-SidS3AccessRole-xgqCgtNjPrb8/i-06ee8c015c1d6e501 is not authorized to perform: s3:PutObject on resource: "arn:aws:s3:::sid-security-ca51ea00-acec-11ef-ab19-02dc4668901f/ssekms-only/text07" with an explicit deny in a resource-based policy
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$
[ec2-user@storage-workshop ~]$ aws s3api put-object --key ssekms-only/text08 --body textfile --server-side-encryption aws:kms --bucket ${bucket}
{
  "SSEKMSKeyId": "arn:aws:kms:us-west-2:092526071050:key/7e293131-c1df-4d65-9e76-eb76d3521750",
  "ETag": "\"1a385d1dbe41b2d50fac538422d750a5\"",
  "ServerSideEncryption": "aws:kms"
}
[ec2-user@storage-workshop ~]$
```

The goal of this exercise was to demonstrate how to use either default encryption and bucket policies to ensure you can apply and enforce encryption to your buckets.

In summary, you have successfully:

- Configured default encryption SSE-KMS on an entire bucket.
- Configured a bucket policy to enforce SSE-KMS on a specific prefix within that bucket.

[Previous](#)
[Next](#)
