# Using Customer-Managed Encryption Keys with Cloud Storage and Cloud KMS

1 hour 30 minutes No cost

## Overview

Cloud Key Management Service (Cloud KMS) allows you to manage encryption keys on Google Cloud. Encryption keys are created by Cloud KMS and managed by you in the same manner you would manage them on-premises.

Using Cloud KMS you can generate, use, rotate and destroy AES256 symmetric encryption keys for direct use by all of your cloud services.

In this lab, you will use Cloud KMS to create KeyRings and CryptoKeys and then use those keys with Cloud Storage to set default keys on buckets, and encrypt individual objects with a Cloud KMS key.

Additionally, you will manually perform server-side encryption with your Cloud KMS keys, and upload encrypted data to Cloud Storage.

Cloud KMS permissions will be managed with IAM, and Cloud Audit Logs will be used to view all activity for CryptoKeys and KeyRings.

## Objectives

In this lab, you will learn how to do the following:

- Manage keys and encrypted data using Cloud KMS.
- Create KeyRings and CryptoKeys.
- Set a default encryption key for a storage bucket.
- Encrypt an object with a Cloud KMS key.
- Rotate encryption keys.
- Perform server-side encryption manually with Cloud KMS keys.

# Setup and requirements

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.

2. Note the lab's access time (for example, `1:15:00`), and make sure you can finish within that time.
There is no pause feature. You can restart if needed, but you have to start at the beginning.

3. When ready, click **Start lab**.

4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.

5. Click **Open Google Console**.

6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
If you use other credentials, you'll receive errors or **incur charges**.

7. Accept the terms and skip the recovery resource page.

**Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

## Task 1: Configure required resources

In this task, you configure the required resources for this lab.

## Create a Cloud Storage bucket

**Note:** A bucket must have a globally unique name. For this lab, you will use your Google Cloud project ID as part of the bucket name to help ensure it will be unique. Your Google Cloud project ID is automatically stored in a Cloud Shell environment variable named `DEVSHELL_PROJECT_ID`.

1. On the Google Cloud Console title bar, click **Activate Cloud Shell** (). If prompted, click **Continue**.

2. Run the following command to create the bucket:

```
gsutil mb -l us gs://$DEVSHELL_PROJECT_ID-kms
```
content_c

3. Run the following commands to create a few sample files that will be uploaded to the bucket:

```
echo "This is sample file 1" > file1.txt
echo "This is sample file 2" > file2.txt
echo "This is sample file 3" > file3.txt
```
content_c

4. Run the following command to copy **file1.txt** to the bucket:

```
gsutil cp file1.txt gs://$DEVSHELL_PROJECT_ID-kms
```
content_c

## Enable Cloud KMS

**Note:** Before using Cloud KMS you need to enable it in your project. In the Qwiklab Google Cloud Project you have been provisioned, Cloud KMS should already have been enabled. Just to be safe, you will issue the command to enable it anyway.

- Run the following command in Cloud shell to enable Cloud KMS:

```
gcloud services enable cloudkms.googleapis.com
```

content_c

Click *Check my progress* to verify the objective.

Create a Cloud Storage bucket

Check my progress

## Task 2. Use Cloud KMS

In this task, you use Cloud KMS to create a keyring and a cryptokey.

### Create a KeyRing and CryptoKey

**Note:** In order to encrypt data, you need to create a KeyRing and a CryptoKey. KeyRings are useful for grouping keys. Keys can be grouped by environment (like test, staging, and prod) or by some other conceptual grouping. For this lab, your KeyRing will be called **test** and your CryptoKey will be called **labkey**.

1. In Cloud Shell, run the following commands to create variables to hold the KeyRing name and CryptoKey name:

```
KEYRING_NAME=lab-keyring
CRYPTOKEY_1_NAME=labkey-1
CRYPTOKEY_2_NAME=labkey-2
```

content_c

2. Execute the following command to create the KeyRing.

For this lab, the location will be set to `us`, but it could also be a specific region.

```
gcloud kms keyrings create $KEYRING_NAME --location us
```

content_c

3. Next, using the new KeyRing, create a CryptoKey named **labkey-1**:

```
gcloud kms keys create $CRYPTOKEY_1_NAME --location us \
--keyring $KEYRING_NAME --purpose encryption
```

content_c

4. Create another CryptoKey named **labkey-2**:

```
gcloud kms keys create $CRYPTOKEY_2_NAME --location us \
```

content_c

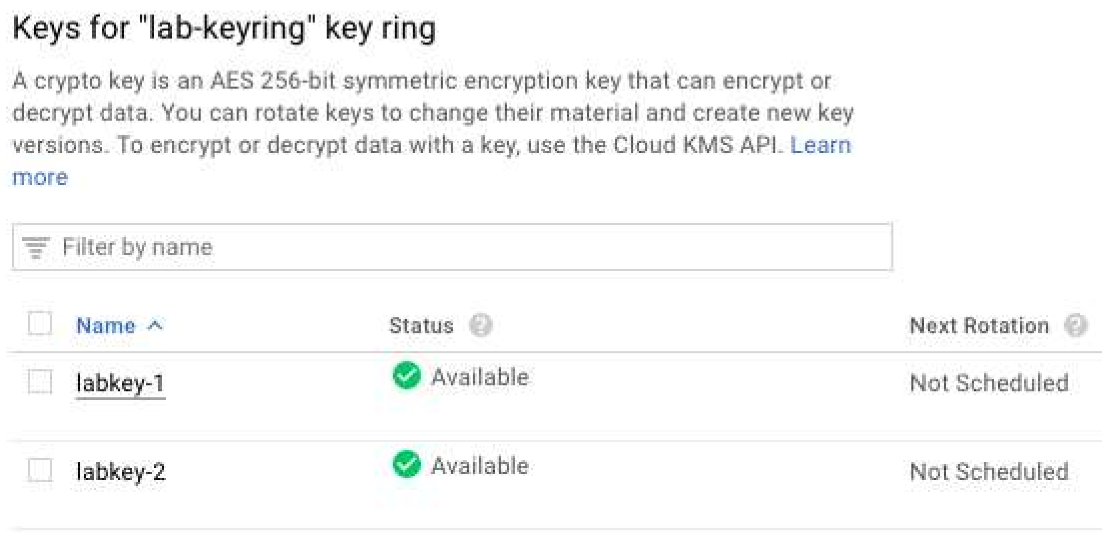
```
--keyring $KEYRING_NAME --purpose encryption
```

You can view the KeyRing and keys in the Cloud Console.

5. In the Cloud Console, go to **Navigation menu > Security > Key Management**.

You will see the KeyRing named **lab-keyring**.

6. Click on the KeyRing named **lab-keyring** to view the encryption keys named **labkey-1** and **labkey-2**.


Keys for "lab-keyring" key ring

A crypto key is an AES 256-bit symmetric encryption key that can encrypt or decrypt data. You can rotate keys to change their material and create new key versions. To encrypt or decrypt data with a key, use the Cloud KMS API. Learn more

Filter by name

| Name | Status | Next Rotation |
|---|---|---|
| labkey-1 | ✓ Available | Not Scheduled |
| labkey-2 | ✓ Available | Not Scheduled |

Click *Check my progress* to verify the objective.

Create a Keyring and Cryptokey

Check my progress

## Task 3. Add a default key for a bucket

In this task, you add a default key for your bucket.

### View the current default key for a bucket

- Run the following command to view the default encryption key for your bucket:

```
gsutil kms encryption gs://$DEVSHELL_PROJECT_ID-kms
```

content_c

**Note:** The bucket should not currently have a default encryption key. This means all data in the bucket will be encrypted by Google-managed encryption keys.

### Assign Cloud KMS keys to a service account

- Run the following commands to give your Cloud Storage service account permission to use both of your Cloud KMS keys:

```
gsutil kms authorize -p $DEVSHELL_PROJECT_ID -k \
projects/$DEVSHELL_PROJECT_ID/locations/us/keyRings\
/$KEYRING_NAME/cryptoKeys/$CRYPTOKEY_1_NAME
gsutil kms authorize -p $DEVSHELL_PROJECT_ID -k \
```

content_c

```
projects/$DEVSHELL_PROJECT_ID/locations/us/keyRings\
/$KEYRING_NAME/cryptoKeys/$CRYPTOKEY_2_NAME
```

The general syntax of the above commands is given below:

```
gsutil kms authorize -p [PROJECT_STORING_OBJECTS] -k [KEY_RESOURCE]
```

Where *[KEY_RESOURCE]* is in the format:

```
projects/[PROJECT_STORING_KEYS]/locations/[LOCATION]/keyRings/[KEY_RING_N
```

content_c

## Set the default key for a bucket

A Cloud KMS key can be set as the default key when objects are written to a bucket.

When setting the default key, the key resource must be specified in the same format as the previous command: `projects/[PROJECT_STORING_KEYS]/locations/[LOCATION]/keyRings/ [KEY_RING_NAME]/cryptoKeys/[KEY_NAME]` .

1. Run the following command to set the default key for your bucket to the first key you generated:

```
gsutil kms encryption -k \
projects/$DEVSHELL_PROJECT_ID/locations/us/keyRings\
/$KEYRING_NAME/cryptoKeys/$CRYPTOKEY_1_NAME \
gs://$DEVSHELL_PROJECT_ID-kms
```

content_c

2. Run the following command to view the default key for the bucket to verify the last command was successful:

```
gsutil kms encryption gs://$DEVSHELL_PROJECT_ID-kms
```
content_c

The default encryption key for the bucket should now be your first encryption key.

**Note:** Objects that were written to a bucket prior to adding or changing the default key remain encrypted with the previous encryption method.

3. Run the following command to copy **file2.txt** to the bucket:

```
gsutil cp file2.txt gs://$DEVSHELL_PROJECT_ID-kms
```
content_c

## View the files in the bucket

- In the Cloud Console, go to **Navigation menu** > **Cloud Storage** > **Browser** and click on your bucket for this lab.

You will see **file 1** was encrypted with a Google-managed key and **file 2** was encrypted with a customer-managed key.

Click *Check my progress* to verify the objective.

Add a default key for the bucket

Check my progress

## Task 4. Encrypt individual objects with a Cloud KMS key

In this task, you encrypt an individual object with a Cloud KMS key. This is useful if you want to use a different key from the default key set on the bucket, or if you don't have a default key set on the bucket. This can be done by passing the key to use in each gsutil command by using the `-o` flag: `-o &#34;GSUtil:encryption_key=[KEY_RESOURCE]&#34;`

1. Run the following command to copy **file3.txt** to the bucket, encrypting it with your second encryption key:

```
gsutil -o \
"GSUtil:encryption_key=projects/$DEVSHELL_PROJECT_ID/locations/us/keyRing
/$KEYRING_NAME/cryptoKeys/$CRYPTOKEY_2_NAME" \
cp file3.txt gs://$DEVSHELL_PROJECT_ID-kms
```

content_c

2. In the Cloud Console, refresh the **Bucket details** screen and you will see **file3.txt** is also encrypted with a customer-managed key.

### Identify the key used to encrypt an object

In the Cloud Console, look at the encryption column. It shows you what kind of key is used: a Google-managed key or a customer-managed key. If you hover over the key, you can get details about the key. You can also get key details using gsutil, which you will do in the next step.

1. Run the following command to display details about an object (the **-L** option causes **gsutil ls** to display all file details):

```
gsutil ls -L gs://$DEVSHELL_PROJECT_ID-kms/file3.txt
```

content_c

2. In the information returned, locate the KMS key line.

This displays the encryption key being used by that file.

3. Run the previous command again for **file1.txt** and **file2.txt**.

Click *Check my progress* to verify the objective.

Encrypt individual objects with a Cloud KMS key

Check my progress

## Task 5. Perform key rotation

In this task, you perform automatic and manual key rotation.

**Note:** In Cloud KMS, a key rotation is triggered by generating a new version of a key, and marking that version as the primary version. Each key has a designated primary version at any point in time, which Cloud KMS uses to encrypt data. After rotating a key, its previous key versions (which no longer are primary) are neither disabled or destroyed, and remain available for decrypting data.

## Automatically rotate keys

**Note:** By providing a rotation schedule, Cloud KMS will automatically rotate your keys for you. A key's rotation schedule can be set using the gcloud command-line tool or via the Cloud Console.

1. In the Cloud Console, go to **Navigation menu > Security > Key Management** and click on the KeyRing named **lab-keyring** to view your encryption keys named **labkey-1** and **labkey-2**.

2. Click on the key named **labkey-1** to view all versions.

Currently you only have one version.

3. Click the **EDIT ROTATION PERIOD** button.

4. Set the **rotation period** dropdown to **30 days**.

Notice that the rotation period can also be set to a Custom period that allows you to specify any desired period.

5. Click **SAVE**.

The Cloud Console now displays the next rotation date for this key.

## Manually rotate keys

**Note:** Manually rotating keys can also be done with the gcloud command-line tool or via the Cloud Console.

1. In the Cloud Console, go back to the KeyRing named **lab-keyring** and click on the key named **labkey-2** to view all versions.

2. Click the **Rotate Key** button and then click **Rotate**.

You now have two versions of this key, `version 2` is the primary one.

**Note:** Using the key rotation commands above, key rotation does NOT re-encrypt already encrypted data with the newly generated key version. If you suspect unauthorized use of a key, you should re-encrypt the data protected by that key and then disable or schedule destruction of the prior key version.

## Destroy old keys

**Note:** If you destroy a key that encrypts existing objects, you will be unable to recover that data, but you will continue to be charged for storage of your objects until you delete them.

In this part, you will not actually destroy a key, but you will investigate the process for doing so.

1. From the **labkey-2** versions screen, click the three vertical dots on the far right of the line for `version 1` of the key and select **Destroy**.

2. Read the message in the **Schedule key version 1 for destruction** and click **Cancel** when done.

You have successfully used Cloud KMS keys to encrypt data in Cloud Storage.

Click *Check my progress* to verify the objective.

Key Rotation

Check my progress

## Bonus task. Encrypt data with the REST API

The Cloud KMS service also provides a REST API to perform encryption and decryption. The content to be encrypted is specified as part of a JSON document in the REST request, and this content must be encoded using Base64 encoding. This JSON document has the following form:

`{"plaintext":"Base64 encoded data to encrypt"}`.

In this bonus section to the lab, you will manually invoke the REST api using curl commands to demonstrate the capability of the API.

1. This section assumes you still have the Cloud Shell session open and the following environment variables are defined:

`KEYRING_NAME`

`CRYPTOKEY_1_NAME`

`CRYPTOKEY_2_NAME`.

If these variables are no longer defined, go back to earlier in the lab and run the commands to create these variables.

2. Run the following command to encode some sample text as base64 and store it in a variable named **PLAIN_TEXT**:

```
PLAIN_TEXT=$(echo -n "Some text to be encrypted" | base64)
```

content_c

3. Echo the PLAIN_TEXT variable to verify the text was encoded:

```
echo $PLAIN_TEXT
```

content_c

You should see the base64-encoded text.

4. Use the REST API to encrypt the encoded text by calling the `encrypt` method of your key.

5. Supply the base64-encoded content in the plaintext field of the JSON for your request:

```
curl \
"https://cloudkms.googleapis.com/v1/projects/$DEVSHELL_PROJECT_ID/locatio
\
-d "{\"plaintext\":\"$PLAIN_TEXT\"}" \
-H "Authorization:Bearer $(gcloud auth application-default \
```

content_c

```
print-access-token)" \
-H "Content-Type: application/json"
```

The response will be a JSON payload containing the encrypted text in the ciphertext field.

**Note:** The encrypted text can easily be extracted from the JSON response, and saved to a file by using the command-line utility **jq**. The response from the previous call can be piped into **jq**, which can parse out the **ciphertext** property and save to **data1.encrypted**.

6. Run the following command that repeats the encryption, but this time parses out the **ciphertext** property and saves it to the **data1.encrypted** file:

```
curl \
"https://cloudkms.googleapis.com/v1/projects/$DEVSHELL_PROJECT_ID/locatio
\
-d "{\"plaintext\":\"$PLAIN_TEXT\"}" \
-H "Authorization:Bearer $(gcloud auth application-default \
print-access-token)" \
-H "Content-Type: application/json" \
| jq .ciphertext -r > data1.encrypted
```

content_c

7. View the contents of the **data1.encrypted** file with the following command:

```
more data1.encrypted
```

content_c

**Note:** The encrypted text can be decrypted by calling the decrypt method of your key. You must use the same key that was used to encrypt the content.

8. Run the following command to decrypt the contents in the **data1.encrypted** file and save it into the file named **data1.decrypted**:

```
curl -v \
"https://cloudkms.googleapis.com/v1/projects/$DEVSHELL_PROJECT_ID/locatio
\
-d "{\"ciphertext\":\"$(cat data1.encrypted)\"}" \
-H "Authorization:Bearer $(gcloud auth application-default \
print-access-token)" \
-H "Content-Type:application/json" \
| jq .plaintext -r | base64 -d > data1.decrypted
```

content_c

9. View the contents of the **data1.decrypted** file with the following command:

```
more data1.decrypted
```

content_c

You have successfully used Cloud KMS keys.

## Congratulations!

In this lab, you have done the following:

1. Managed keys and encrypted data using Cloud KMS.

2. Created KeyRings and CryptoKeys.

3. Set a default encryption key for a storage bucket.

4. Encrypted an object with a Cloud KMS key.

5. Rotated encryption keys.

6. Manually performed server-side encryption with Cloud KMS keys.