Exercise 2: Adding Multi-CMK Support to the Document Bucket

In this section, you will configure the AWS Encryption SDK to use multiple CMKs that reside in different regions.

Background

Now your Document Bucket will encrypt files when you `store` them, and will decrypt the files for you when you `retrieve` them.

You're using one of your CMKs, Faythe. But what if you want to use multiple CMKs?

You might want to use a partner team's CMK, so that they can access documents relevant to them.

Perhaps you want the Document Bucket to have two independent regions to access the contents, for high availability, or to put the contents closer to the recipients.

Configuring multiple CMKs this way does not require re-encryption of the document data. That's because the data is still encrypted with a single data key, used exclusively for that document. Configuring multiple CMKs causes the AWS Encryption SDK to encrypt that data key again using the additional CMKs, and store that additional version of the data key on the encrypted message format. As long as there is one available CMK to decrypt any encrypted version of the data key, the document will be accessible.

(There are ways to configure the Encryption SDK to be more restrictive about which CMKs it will try -- but for now you'll start with the simple case.)

There's many reasons why using more than one CMK can be useful. And in this exercise, you're going to see how to set that up with KMS and the Encryption SDK.

You already have another CMK waiting to be used. When you deployed the Faythe CMK, you also deployed a second CMK, nicknamed Walter. In this exercise, we're going to configure Walter, and then use some scripts in the repository to add and remove permission to use each of Faythe and Walter. Doing so will change how your document is encrypted -- if you remove permission to both Faythe and Walter, you won't be able to encrypt or decrypt anymore! -- and let you observe how the system behavior changes when keys are accessible or inaccessible.

Each attempt to use a CMK is checked against that CMK's permissions. An audit trail entry is also written to CloudTrail.

Decryption attempts will continue for each version of the encrypted data key until the Encryption SDK either succeds at decrypting an encrypted data key with its associated CMK, or runs out of encrypted data keys to try.

For encryption, the Encryption SDK will attempt to use every CMK it is configured to attempt to produce another encryption of that data key.

You'll get to see all of this in action in just a minute, after a couple small code changes.

Make the Change

Starting Directory

If you just finished Adding the Encryption SDK [../add-the-encryption-sdk/], you are all set.

If you aren't sure, or want to catch up, jump into the `multi-cmk-start` directory for the language of your choice.

Java Typescript Node.JS JavaScript Node.JS Python

```
cd ~/environment/workshop/exercises/java/multi-cmk-start
```

Step 1: Configure Walter

Java JavaScript Node.JS Typescript Node.JS Python

```
// Edit ./src/main/java/sfw/example/esdkworkshop/App.java
    String faytheCMK = state.contents.FaytheCMK;
    // MULTI-CMK-START: Configure Walter
    String walterCMK = state.contents.WalterCMK;
```

What Just Happened

When you launched your workshop stacks in Getting Started [../getting-started/], along with the Faythe CMK, you also launched a CMK called Walter. Walter's ARN was also plumbed through to the configuration state file that is set up for you by the workshop. Now that ARN is being pulled into a variable to use in the Encryption SDK configuration.

Step 2: Add Walter to the CMKs to Use

Java JavaScript Node.JS Typescript Node.JS Python

```
// Edit ./src/main/java/sfw/example/esdkworkshop/App.java
    // MULTI-CMK-START: Add Walter to the CMKs to Use
    KmsMasterKeyProvider mkp =

KmsMasterKeyProvider.builder().withKeysForEncryption(faytheCMK,
walterCMK).build();
```

What Just Happened

In the previous exercise, you configured the Encryption SDK to use a list of CMKs that contained only Faythe. Configuring the Encryption SDK to also use Walter for encrypt, and to also try Walter for decrypt, required adding the ARN for Walter to the configuration list.

Checking Your Work

If you want to check your progress, or compare what you've done versus a finished example, check out the code in one of the `-complete` folders to compare.

There is a `-complete` folder for each language.

Java Typescript Node.JS JavaScript Node.JS Python

```
1 cd ~/environment/workshop/exercises/java/multi-cmk-complete
```

Try it Out

Adding the Walter CMK to the list of CMKs that the application will (attempt) to use was a couple of lines of code, but has powerful implications.

To help you explore the behavior of the system, there are some additional `make` targets to change the permissions configuration of Faythe and Walter.

Using these targets, you can add and remove permission for the application to use Faythe and Walter to generate data keys, encrypt, and decrypt, and observe how the application behavior changes -- as well as what is logged to CloudTrail.

In `~/environment/workshop/exercises`, you'll find a `Makefile` with several targets for you to experiment with:

- `make list_grants` will show you the current state of grants on your CMKs
- `make revoke_walter_grant` will remove the Grant providing permissions to use Walter in the application
- `make revoke_faythe_grant` will remove the Grant providing permissions to use Faythe in the application
- `make revoke_grants` will remove the Grants for both CMKs
- `make create_grants` will add Grants to use either or both CMK, as needed

The application behavior will change. **Important** when you revoke permissions to the first CMK in the list for a keyring (which is Faythe by default), you will need to change the keyring configuration to use Walter as your generator to resume operations. See documentation of Generator CMKs [https://docs.aws.amazon.com/encryption-sdk/latest/developer-guide/js-examples.html] for more.

You can also observe the impact of changing Granted permissions by monitoring CloudTrail. Note that log entries take a few minutes to propagate to CloudTrail.

- Faythe is in `us-east-2`, so check CloudTrail in that region with these links:
  - GenerateDataKey operations in CloudTrail in us-east-2 [https://console.aws.amazon.com/cloudtrail/home?region=us-east-2#/events?EventName=GenerateDataKey]
  - Encrypt operations in CloudTrail in us-east-2 [https://us-east-2.console.aws.amazon.com/cloudtrail/home?region=us-east-2#/events?EventName=Encrypt]
  - Decrypt operations in CloudTrail in us-east-2 [https://us-east-2.console.aws.amazon.com/cloudtrail/home?region=us-east-2#/events?EventName=Decrypt]
- Walter is in `us-west-2`, so check CloudTrail in that region with these links:
  - GenerateDataKey operations in CloudTrail in us-west-2 [https://console.aws.amazon.com/cloudtrail/home?region=us-west-2#/events?EventName=GenerateDataKey]
  - Encrypt operations in CloudTrail in us-west-2 [https://us-west-2.console.aws.amazon.com/cloudtrail/home?region=us-west-2#/events?EventName=Encrypt]
  - Decrypt operations in CloudTrail in us-west-2 [https://us-west-2.console.aws.amazon.com/cloudtrail/home?region=us-west-2#/events?EventName=Decrypt]

Try out combinations of Grant permissions for your application and watch how the behavior changes:

- Revoke permission to use Faythe, and watch calls move to Walter in CloudTrail and in your application
- With permission to use Faythe revoked, try retrieving an older document protected by Faythe
- Revoke permissions to both Faythe and Walter -- now operations fail
- Encrypt some data with both Faythe and Walter active, and revoke permission to either one -- notice that application operations continue to work
- Change the configuration order of Faythe and Walter, and watch how call patterns change to use the two CMKs
- Revoke permission to Walter, and encrypt some data with Faythe. Then, add permission back to Walter, revoke permission to use Faythe, and try to decrypt that data
- What other interesting access patterns can you imagine?

Java JavaScript Node.JS JavaScript Node.JS CLI Typescript Node.JS Typescript Node.JS CLI Python

```
// Compile your code
mvn compile

// To use the API programmatically, use this target to launch jshell
mvn jshell:run
/open startup.jsh
Api documentBucket = App.initializeDocumentBucket();
documentBucket.list();
documentBucket.store("Store me in the Document Bucket!".getBytes());
for (PointerItem item : documentBucket.list()) {
    DocumentBundle document = documentBucket.retrieve(item.partitionKey().getS());
    System.out.println(document.toString());
}
// Ctrl+D to exit jshell

// Use the make targets to change the Grants and see what
```

```
happens!
// To run logic that you write in App.java, use this target
after compile
mvn exec:java
```

Explore Further

Want to dive into more content related to this exercise? Try out these links.

- AWS KMS: Key Grants [https://docs.aws.amazon.com/kms/latest/developerguide/grants.html]
- AWS KMS: Key Policies [https://docs.aws.amazon.com/kms/latest/developerguide/key-policies.html]
- AWS KMS: Cross-account CMK Usage [https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-modifying-external-accounts.html]
- Blog Post: How to decrypt ciphertexts in multiple regions with the AWS Encryption SDK in C [https://aws.amazon.com/blogs/security/how-to-decrypt-ciphertexts-multiple-regions-aws-encryption-sdk-in-c/]

Next exercise

Ready for more? Next you will work with Encryption Context [../encryption-context/].