9

Google Cloud Key Management Service

In this chapter, we will look at Google Cloud **Key Management Service** (**KMS**). Cloud KMS is a foundational service for all cryptographic operations in Google Cloud. Every workload that you deploy on Google Cloud is going to need the ability to encrypt data and use it for authorized purposes. There are various options presented by Cloud KMS, and it's essential to understand them and make an informed choice to help with regulatory and audit requirements.

In this chapter, we will cover the following topics:

- Overview of Cloud KMS
- Encryption and key management in Cloud KMS
- Key management options
- Customer-supplied encryption key
- Symmetric and asymmetric key encryption
- Bringing your own key to the cloud
- Key lifecycle management
- Key IAM permissions
- Cloud HSM
- Cloud External Key Manager
- Cloud KMS best practices
- Cloud KMS APIs and logging

Let us start by learning about the capabilities of Cloud KMS.

Overview of Cloud KMS

With Cloud KMS, Google's focus is to provide a scalable, reliable, and performant solution with a wide spectrum of options that you can control on a platform that is straightforward to use. Let us start with a quick overview of the Cloud KMS architecture.

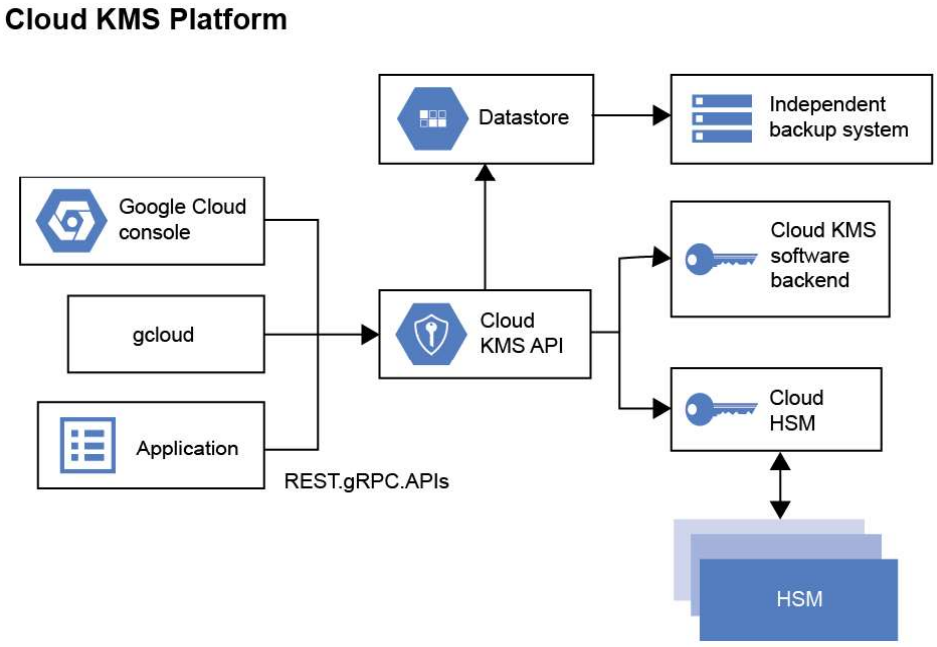

Figure 9.1 – The Cloud KMS architecture

The key components of the Cloud KMS platform are depicted in *Figure 9.1*. Administrators can access key management services through the Google Cloud console or CLI, as well as through the REST or gRPC APIs. A REST API or gRPC is used by applications to access key management services.

When creating a key on the Cloud KMS platform, you can select a protection level to define which key backend the key should use. The Cloud KMS platform has two backends (excluding Cloud EKM): the **software** and **HSM** protection levels. The software protection level is for keys that are protected by the **software security module**. HSM refers to keys that are protected by **hardware security modules** (**HSMs**).

Cloud KMS cryptographic operations are performed by FIPS 140-2-validated modules. Keys with the software protection level, and the cryptographic operations performed with them, comply with FIPS 140-2 Level 1. Keys with the HSM protection level, and the cryptographic operations performed with them, comply with FIPS 140-2 Level 3.

Let us now see the current offerings of Cloud KMS.

Current Cloud KMS encryption offerings

In this section, we will look at various KMS offerings. The offerings are based on the source of the key material, starting from Google-managed encryption and going up to where the key material is managed externally.

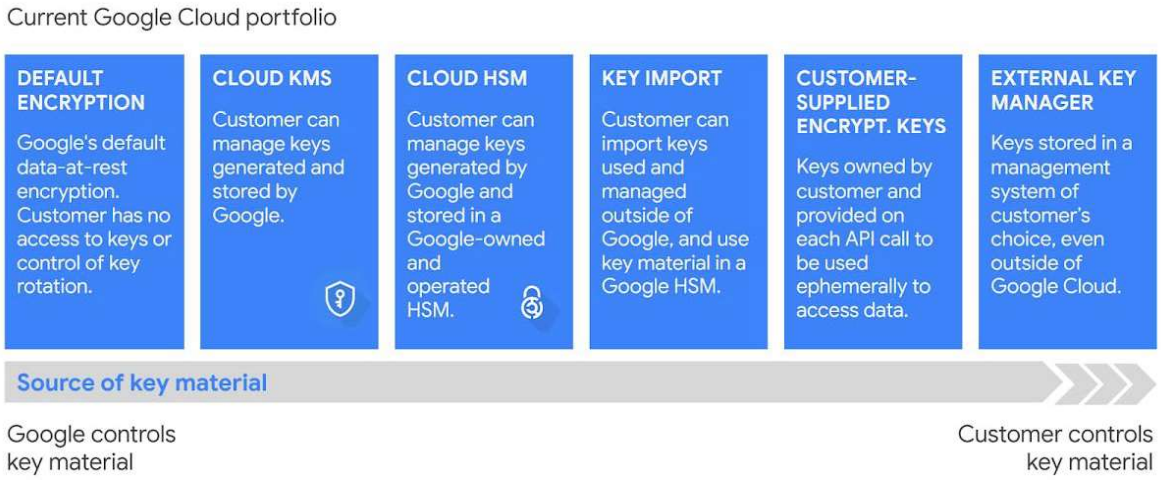

Figure 9.2 – Google Cloud encryption offerings

Figure 9.2 shows different offerings of encryption on Google Cloud.

Cloud KMS is supported by five design pillars:

1. **Customer control**: You can use Cloud KMS to handle software and hardware encryption keys, or you can provide your own via the key import functionality.
2. **Control and monitoring of access**: You can manage permissions on individual keys and see how they are used.
3. **Regionality**: Regionalization is built into Cloud KMS. Only the Google Cloud location you choose is used to produce, store, and process software keys in the service.
4. **Durability**: Cloud KMS meets the most stringent durability requirements. Cloud KMS checks and backs up all key material and metadata on a regular basis to protect against data corruption and ensure that data can be correctly decrypted.
5. **Security**: Cloud KMS is fully linked with **Identity and Access Management** (**IAM**) and **Cloud Audit Logs** controls, providing robust protections against unwanted access to keys. Cloud Audit Logs helps you understand who used a key and when.

Now that you understand the underlying principles of the Cloud KMS offerings, let us learn the basics of key management.

Encryption and key management in Cloud KMS

In this section, we will learn the basics of key management as it relates to Cloud KMS.

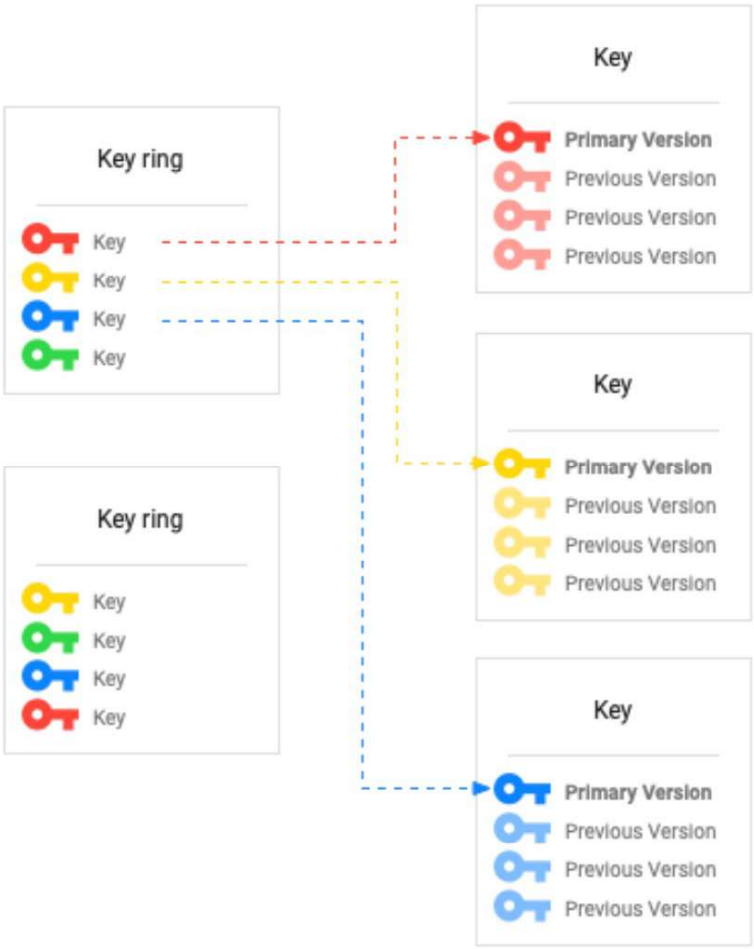

Figure 9.3 – Key structure

Figure 9.3 shows the structure of keys in Cloud KMS. Let us learn about the terms of key management:

- **Key**: A named object representing a cryptographic key that is used for a specific purpose. The key material—the actual bits used for cryptographic operations—can change over time as you create new key versions. Key purpose and other attributes of the key are connected with and managed using the key. Thus, the key is the most important object for understanding Cloud KMS usage. Cloud KMS supports both asymmetric keys and symmetric keys. A symmetric key is used for symmetric encryption to protect some corpus of data—for example, using AES-256 in GCM mode to encrypt a block of plaintext. An asymmetric key can be used for asymmetric encryption, or for creating digital signatures.

- **Key ring**: A grouping of keys for organizational purposes. A key ring belongs to a Google Cloud project and resides in a specific location. Keys inherit IAM policies from the key ring that contains them. Grouping keys with related permissions in a key ring lets you grant, revoke, or modify permissions to those keys at the key ring level without needing to act on each key individually. Key rings provide convenience and categorization, but if the grouping of key rings is not useful to you, you can manage permissions directly on keys. To prevent resource name collisions, a key ring cannot be deleted. Key rings and keys do not have billable costs or quota limitations, so their continued existence does not affect costs or production limits.
- **Key metadata**: Resource names, properties of Cloud KMS resources such as IAM policies, key type, key size, key state, and any data derived from them. Key metadata can be managed differently than the key material.
- **Key version**: Represents the key material associated with a key at some point in time. The key version is the resource that contains the actual key material. Versions are numbered sequentially, beginning with version 1. When a key is rotated, a new key version is created with new key material. The same logical key can have multiple versions over time, thus limiting the use of any single version. Symmetric keys will always have a primary version. This version is used for encrypting by default. When Cloud KMS performs decryption using symmetric keys, it automatically identifies which key version is needed to perform the decryption. A key version can only be used when it is enabled. Key versions in any state other than destroyed incur costs.
- **Purpose**: A key's purpose determines whether the key can be used for encryption or for signing. You choose the purpose when creating the key, and all versions have the same purpose. The purpose of a symmetric key is always symmetric encrypt/decrypt. The purpose of an asymmetric key is either asymmetric encrypt/decrypt or asymmetric signing. You create a key with a specific purpose in mind and it cannot be changed after the key is created.
- **State**: A key version's state is always one of the following:
  - Enabled
  - Disabled
  - Scheduled for destruction
  - Destroyed

Let us now move on to understand the key hierarchy in Cloud KMS.

Key hierarchy

The following diagram shows the key hierarchy of Cloud KMS. Cloud KMS uses Google's internal KMS in that Cloud-KMS-encrypted keys are wrapped by Google's KMS. Cloud KMS uses the same root of trust as Google's Cloud KMS.

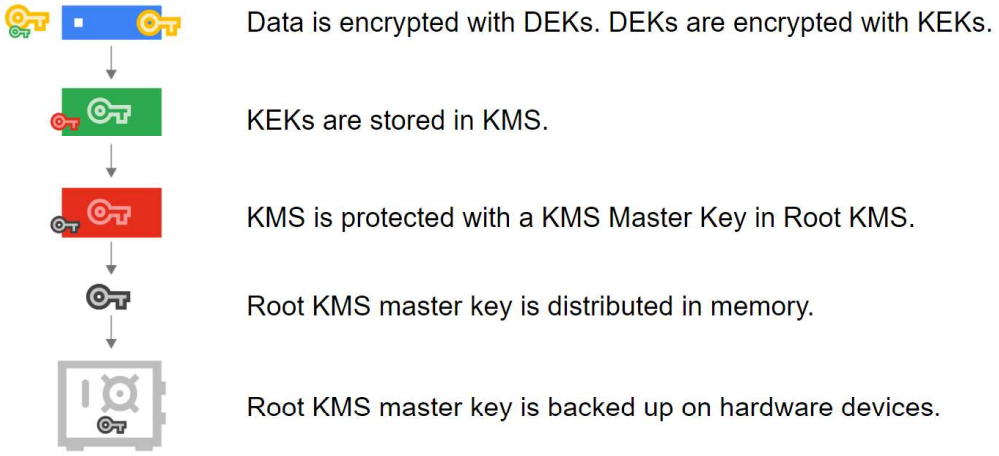

Figure 9.4 – Key hierarchy

Figure 9.4 shows the key hierarchy in Cloud KMS. There are several layers in the hierarchy:

- **Data encryption key** (**DEK**): A key used to encrypt actual data. DEKs are managed by a Google Cloud service on your behalf.
- **Key encryption key** (**KEK**): A key used to encrypt, or wrap, a data encryption key. All Cloud KMS platform options (software, hardware, and external backends) let you control the KEK.
- **KMS master key**: The key used to encrypt the KEK. This key is distributed in memory. The KMS master key is backed up on hardware devices. This key is responsible for encrypting KEKs.
- **Root KMS master key**: Google's internal KMS key.

Now that you understand how key hierarchy works, let us go over the basics of encryption.

Envelope encryption

As depicted in *Figure 9.5*, the process of encrypting data is to generate a DEK locally, encrypt data with the DEK, use a KEK to wrap the DEK, and then store the encrypted data and the wrapped DEK together. The KEK never leaves Cloud KMS. This is called **envelope encryption**. This section is more about understanding how Google handles envelope encryption, which is not something Google Cloud customers need to manage.

Figure 9.5 – Envelope encryption

Here are the steps that are followed to encrypt data using envelope encryption:

1. A DEK is generated locally, specifying a cipher type and a key material from which to generate the key.
2. This DEK is now used to encrypt the data.
3. A new KEK is either generated or an existing KEK from Cloud KMS is used—based on user preference. This KEK is used to encrypt (wrap) the DEK.
4. The wrapped DEK is stored with the encrypted data.

Now let us see the steps that are followed to decrypt data using envelope encryption.

As depicted in *Figure 9.6*, the process of decrypting data is to retrieve the encrypted data and the wrapped DEK, retrieve the KEK that wrapped the DEK, use the KEK to unwrap the DEK, and then use the unwrapped DEK to decrypt the data. The KEK never leaves Cloud KMS.

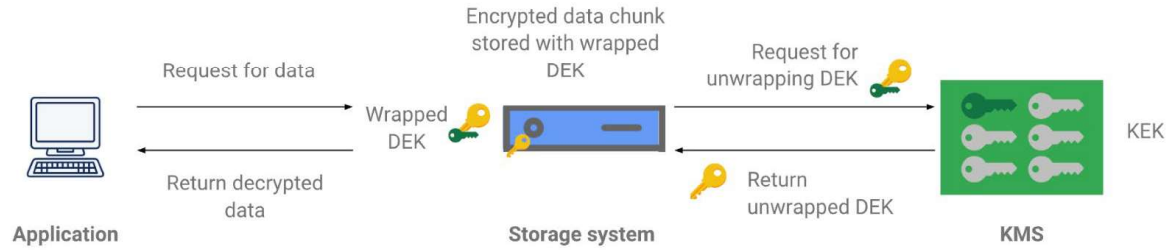

Figure 9.6 – Envelope decryption

To decrypt data using envelope encryption, you do the following:

1. Retrieve the encrypted data and the wrapped DEK.
2. Use the KEK stored in Cloud KMS to unwrap the encrypted DEK.
3. Use the plaintext DEK to decrypt the encrypted data.

Now that we understand the basics of keys and envelope encryption, let us look at key management options within Cloud KMS.

Key management options

In this section, we will cover several aspects of key management that are either Google-managed or customer-managed. It is important to know which option is best suited to which scenario so that you can make the right decision for a given use case.

Google Cloud's default encryption

Google Cloud stores all data encrypted at rest using a Google-managed default encryption key. The key is the AES-256 symmetric encryption key. There is no setup of keys or configuration required to turn on this option; all data by default uses this type of encryption. Google manages the keys and the rotation period of those keys. Google Cloud's default encryption is best suited for those customers who do not have specific requirements related to compliance or regional requirements for cryptographic key material. It is simple to use and does not require additional configuration to create keys, hence there is no cost to use it.

Customer-managed encryption keys (CMEKs)

Some customers of Google Cloud have stringent regulatory and compliance requirements to have control over the keys used to encrypt data at rest. To help with these requirements, Google Cloud offers the ability to encrypt data related to those services using an encryption key managed by the customer within Cloud KMS. CMEKs just mean that the customer decides the key type (software versus hardware), algorithm strength, and key rotation period when creating a key in Cloud KMS.

CMEKs give you control over aspects of the key lifecycle, including (but not limited to) the following:

- Customers can disable the keys used to safeguard data at rest, limiting Google's ability to decode it
- Customers can protect their data by utilizing a key that is personal to their location
- Customers can rotate the keys used to protect the data either automatically or manually
- Customers can use a Cloud HSM key, a Cloud **External Key Manager** (**EKM**) key, or an existing key that they import into Cloud KMS to protect their data

Customer-Managed Encryption Keys (**CMEKs**) are a supported feature in certain Google Cloud services. Let's explore what it entails for services to have CMEK support.

CMEK integrations

When a service supports CMEKs, it is said to have a CMEK integration. Some services, such as GKE, have multiple CMEK integrations for protecting several types of data related to the service.

For the exact steps to enable CMEKs, see the documentation for the relevant Google Cloud service. You can expect to follow steps like these:

1. You create or import a Cloud KMS key, selecting a location as geographically near as possible to the location of the service's resources. The service and the key can be in the same project or different projects. This is the CMEK.
2. You grant the CryptoKey Encrypter/Decrypter IAM role (`roles/cloudkms.cryptoKeyEncrypterDecrypter`) on the CMEK to the service account for that service.
3. You configure the service to use the CMEK to protect its data. For example, you can configure a GKE cluster to use a CMEK to protect data at rest on the boot disks of the nodes.

- If a service account has a `cryptoKeyEncrypterDecrypter` role, the service can encrypt and decrypt its data. If you revoke the role, or if you disable or destroy the CMEK, that data cannot be accessed.

> **Note**
> The rotation of the CMEK will not impact data availability since the service keeps track of which CMEK version was used to encrypt the data and will use that version to decrypt it.

CMEK compliance

Some services do not directly store data, or store data for only a brief period as an intermediate step in a long-running operation. For this type of workload, it is not practical to encrypt each write separately. These services do not offer CMEK integrations but can offer CMEK compliance, often with no configuration on your part.

A CMEK-compliant service encrypts temporary data by using an ephemeral key that only exists in memory and is never written to disk. When the temporary data is no longer needed, the ephemeral key is flushed from memory, and the encrypted data cannot be accessed, even if the storage resource still exists.

A CMEK-compliant service might offer the ability to send its output to a service with a CMEK integration, such as Cloud Storage.

Customer-supplied encryption key

You can provide your own AES-256 key, encoded in standard Base64, as an additional layer on top of Google-managed encryption keys. This key is known as a **customer-supplied encryption key** (**CSEK**).

The CSEK is currently supported by only two services: Cloud Storage and Compute Engine.

CSEKs with Cloud Storage

If you provide a CSEK, Cloud Storage does not permanently store your key on Google's servers or otherwise manage your key.

Instead, you provide your key for each Cloud Storage operation, and your key is purged from Google's servers after the operation is complete. Cloud Storage stores only a cryptographic hash of the key so that future requests can be validated against the hash. Your key cannot be recovered from this hash, and the hash cannot be used to decrypt your data. Here are the best practices when using CSEKs:

- You should back up each key to a secure location and take precautions to ensure that keys are not shared with untrusted parties
- If any file or machine containing the encryption key is compromised, you should immediately perform key rotation for all objects encrypted with the compromised key

When you apply a customer-supplied encryption key to an object, Cloud Storage uses the key when encrypting:

- The object's data
- The object's CRC32C checksum
- The object's MD5 hash

Cloud Storage uses standard server-side keys to encrypt the remaining metadata for the object, including the object's name. This allows you to read and update general metadata, as well as list and delete objects, without needing the CSEK. However, to perform any of these actions, you must have sufficient permission to do so.

For example, if an object is encrypted with a CSEK, the key must be used to perform operations on the object such as downloading or moving it. If you attempt to read the object's metadata without providing the key, you receive metadata, such as the object name and content type, but not the object's CRC32C checksum or MD5 hash. If you do supply your key with the request for the object metadata, the object's CRC32C checksum and MD5 hash are included with the metadata.

The following restrictions apply when using CSEKs:

- Cloud Storage Transfer Service and Cloud Dataflow do not currently support objects encrypted with CSEKs.
- You cannot use the Google Cloud console to download objects that are encrypted with a CSEK. Similarly, when you use the Google Cloud console to upload an object, you cannot encrypt the object with a CSEK.
- You can only set CSEKs on individual objects. You cannot set a default CSEK for a bucket.

- If you are performing a composition operation on objects encrypted by CSEKs, the component objects must be encrypted by the same key, and you need to provide the key with the compose request. The resulting composite object is encrypted by the same key.

Now that you understand the restrictions of using CSEKs with Google Cloud Storage, here is how you can use CSEKs with `gsutil`.

CSEKs with gsutil

To use a CSEK with `gsutil`, add the `encryption_key` parameter to the `[GSUtil]` section of your boto configuration file. `encryption_key` is an RFC 6848 Base64-encoded string of your AES-256 encryption key.

You can optionally specify one or more decryption keys, up to 100. While the `encryption_key` option is used by `gsutil` as both an encryption and decryption key, any `decryption_key` options you specify are used only to decrypt objects. Multiple decryption keys must be listed in the boto configuration file as follows:

```
decryption_key1 = ...
decryption_key2 = ...
decryption_key3 = ...
```

If you have encryption or decryption keys in your boto configuration file, they are used for all `gsutil` commands. When decrypting, `gsutil` calculates the SHA256 hash of the supplied encryption and decryption keys and selects the correct key to use for a particular object by matching the SHA256 hash in the object's metadata.

Objects encrypted with CSEKs require the matching decryption key during **encryption key rotation**. If an object is encrypted using a CSEK, you can rotate the object's key by rewriting the object. Rewrites are supported through the JSON API, but not the XML API.

Now that we have seen how CSEKs are used with Cloud Storage, let us see how they can be used with Compute Engine.

CSEKs with Compute Engine

Compute Engine encrypts all data at rest by default on the persistent disks by using Google default encryption keys. You can bring your own encryption key if you want to control and manage this encryption yourself.

If you provide your own encryption key, Compute Engine uses your key to protect the Google-generated keys used to encrypt and decrypt your data. Only users who can provide the correct key can use resources protected by a CSEK.

Google does not store your keys on its servers, and it cannot access your protected data unless you provide it to them. This also implies that if you forget or lose your key, Google will be unable to recover it, as well as any data encrypted with it.

Google discards the cryptographic keys when you destroy a persistent disk, making the data unrecoverable. This is an irreversible procedure.

There are some technical restrictions in using CSEKs with Compute Engine:

- You can only encrypt new persistent disks with your own key. You cannot encrypt existing persistent disks with your own key.
- You cannot use your own keys with local SSDs because local SSDs do not persist beyond the life of a VM. Local SSDs are already protected with an ephemeral encryption key that Google does not retain.
- Compute Engine does not store encryption keys with instance templates, so you need to store your own keys in Cloud KMS to encrypt disks in a managed instance group.

Now you understand the restrictions of using CSEKs with Compute Engine, please look at the Google Cloud documentation on how to use CSEKs with Compute Engine. Try out a few of these use cases on your own to get a fair idea of how a CSEK is used:

- Encrypt a new persistent disk with a CSEK
- Create a snapshot from a disk encrypted with a CSEK
- Create a new image from a disk or custom image encrypted with a CSEK
- Encrypt an imported image with a CSEK
- Create a persistent disk from a resource encrypted with a CSEK
- Attach a disk encrypted with a CSEK to a new VM
- Start or restart VMs that have disks encrypted with a CSEK

So far, we have seen different types of keys and where you can use them. Let us move on now to understand symmetric and asymmetric key operations.

Symmetric key encryption

Recall that symmetric keys are used for encryption to protect some data—for example, using AES-256 in GCM mode to encrypt a block of plaintext.

Creating a symmetric key

To create a symmetric key, you will first need to create a key ring. The key ring determines the location of the key. Let us start with creating that.

Step 1: Creating a key ring

Here is a `gcloud` command to create a key ring:

```
gcloud kms keyrings create key-ring-name \
    --location location
```

Replace `key-ring-name` with a name for the key ring to hold the key. Replace `location` with the Cloud KMS location for the key ring and its keys.

Step 2: Creating a key

Use the following command to create a key in an existing key ring:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "encryption"
```

Replace `key` with the name of the key you would like to use and `key-ring-name` with the name of the key ring you created in the previous step. Replace `location` with the Cloud KMS location of the key ring. Note the *purpose* of this key.

Step 3: Setting a key rotation period and starting time

Once the key is created, you can set a rotation period with a starting time. Here is the command that will allow you to do so:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "encryption" \
    --rotation-period rotation-period \
    --next-rotation-time next-rotation-time
```

Replace `key` with a name for the key. Replace `key-ring-name` with the name of the existing key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring. Replace `rotation-period` with an interval, such as `30d` to rotate the key every 30 days. Replace `next-rotation-time` with a timestamp at which to begin the first rotation, such as `1970-01-01T01:02:03`.

Now that you have created a key for symmetric encryption, let us see how to use it to encrypt content.

Encrypting content with a symmetric key

Here is a command to use an existing key for encryption:

```
gcloud kms encrypt \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --plaintext-file file-with-data-to-encrypt \
    --ciphertext-file file-to-store-encrypted-data
```

Replace `key` with the name of the key to use for encryption. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-with-data-to-encrypt` and `file-to-store-encrypted-data` with the local file paths (include the actual filename as well) for reading the plaintext data and saving the encrypted output.

This command will produce a file containing encrypted data. The name of the file is the one you provided for the `ciphertext-file` argument.

Now that you have seen how to encrypt the content, let us see how to decrypt it.

Decrypting content with a symmetric key

Here is a command to use an existing key for decryption:

```
gcloud kms decrypt \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --ciphertext-file file-path-with-encrypted-data \
    --plaintext-file file-path-to-store-plaintext
```

Replace `key` with the name of the key to use for encryption. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-path-with-encrypted-data` and `file-path-to-store-plaintext` with the local file paths (include the actual filename as well) for reading the plaintext data and saving the encrypted output.

This command will produce a file containing decrypted data. The name of the file is the one you have provided for `plaintext-file`.

Now that you have seen symmetric key encryption and decryption using Cloud KMS, let us move on to see how to use asymmetric key encryption.

Asymmetric key encryption

The following section describes the flow for using an asymmetric key to encrypt and decrypt data. Asymmetric key encryptions involve a key pair (public and private key pair). As the name suggests, the private key is not shared while the public key is shared. There are two participants in this workflow—a sender and a recipient. The sender creates a ciphertext using the recipient's public key, and then the recipient decrypts the ciphertext using the private key it holds. Only someone with knowledge of the private key can decrypt the ciphertext.

Cloud KMS provides the following functionality as it relates to asymmetric encryption:

- The ability to create an asymmetric key with the key purpose of `ASYMMETRIC_DECRYPT`. For information about which algorithms Cloud KMS supports, see asymmetric encryption algorithms in the Google Cloud documentation.
- CloudKMS asymmetric keys also support `ASYMMETRIC_SIGN` (ECC and RSA).
- The ability to retrieve the public key for an asymmetric key. You use the public key to encrypt data. Cloud KMS does not directly provide a method to asymmetrically encrypt data. Instead, you encrypt data using openly available SDKs and tools, such as `OpenSSL`. These SDKs and tools require the public key that you retrieve from Cloud KMS.
- The ability to decrypt data with an asymmetric key.

Now let us look at the steps to be able to do asymmetric encryption and signing. Similar to symmetric key creation, in order to create an asymmetric key, you will first need to create a key ring. The key ring determines the location of the key. Let us start.

Step 1: Creating a key ring

Use the following command to create a key ring for your asymmetric key:

```
gcloud kms keyrings create key-ring-name \
    --location location
```

Replace `key-ring-name` with a name for the key ring. Replace `location` with the Cloud KMS location for the key ring and its keys.

Step 2: Creating an asymmetric decryption key

Follow these steps to create an asymmetric decryption key on the specified key ring and location. These examples use the software protection level and an `rsa-decrypt-oaep-2048-sha256` algorithm. When you first create the key, the key's initial version has a state of pending generation. When the state changes to enabled, you can use the key:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "asymmetric-encryption" \
    --default-algorithm "rsa-decrypt-oaep-2048-sha256"
```

Replace `key` with a name for the new key. Replace `key-ring-name` with the name of the existing key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring.

Step 3: (Optional) Creating an asymmetric signing key

Follow these steps to create an asymmetric signing key on the specified key ring and location. The following command uses the software protection level and the `rsa-sign-pkcs1-2048-sha256` algorithm. When you first create the key, the key's initial version has a state of pending generation. When the state changes to enabled, you can use the key:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "asymmetric-signing" \
    --default-algorithm "rsa-sign-pkcs1-2048-sha256"
```

Now let us move on to understand how to use an asymmetric key.

Encrypting data with an asymmetric key

This section provides examples that run at the command line. We use OpenSSL to encrypt data. Cloud Shell is pre-installed with OpenSSL, so go ahead and try this in Cloud Shell.

> **Note**
>
> You cannot follow this procedure with a key that has a purpose of `ASYMMETRIC_SIGN`.
>
> The version of OpenSSL installed on macOS does not support the flags used to decrypt data in this topic. To follow these steps on macOS, install OpenSSL from Homebrew.

Step 1: Downloading the public key

Run the following command to download the public key part of the asymmetric key that you created in *Step 2* in the previous section:

```
gcloud kms keys versions get-public-key key-version \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --output-file public-key-path
```

Replace `key-version` with the key version that has the public key. Replace `key` with the name of the key. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `public-key-path` with the location to save the public key on the local system.

Now encrypt the data with the public key you downloaded from Cloud KMS:

```
openssl pkeyutl -in cleartext-data-input-file \
    -encrypt \
    -pubin \
    -inkey public-key-path \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha256 \
    -pkeyopt rsa_mgf1_md:sha256 \
    > encrypted-data-output-file
```

Replace `cleartext-data-input-file` with the path and filename to encrypt. Replace `public-key-path` with the path and filename where you downloaded the public key. Replace `encrypted-data-output-file` with the path and filename to save the encrypted data.

Now let us see how to decrypt the data using an asymmetric key.

Decrypting data with an asymmetric key

We will use the private key part of the asymmetric key to decrypt the encrypted data. Run the `gcloud` command for decryption:

```
gcloud kms asymmetric-decrypt \
    --version key-version \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --ciphertext-file file-path-with-encrypted-data \
    --plaintext-file file-path-to-store-plaintext
```

Replace `key-version` with the key version or omit the `--version` flag to detect the version automatically. Replace `key` with the name of the key to use for decryption. Replace `key-ring-name` with the name of the key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-path-with-encrypted-data` and `file-path-to-store-plaintext` with the local file paths for reading the encrypted data and saving the decrypted output, respectively.

The decrypted text will be in `plaintext-file`.

We looked at some common operations with a key generated by KMS. Now let us look at an advanced option to bring your own key to the cloud, which some organizations may prefer to achieve compliance.

Importing a key (BYOK)

Google allows you to bring your own cryptographic key material. You can import that using the Software or Cloud HSM protection level. We will see step-by-step instructions on how to do this. But before we do that, let us understand the reasons you want to import a key:

- You may be using existing cryptographic keys that were created on your premises or in an external KMS.
- If you migrate an application to Google Cloud or if you add cryptographic support to an existing Google Cloud application, you can import the relevant keys into Cloud KMS.
- As part of key import, Cloud KMS generates a wrapping key, which is a public/private key pair, using one of the supported import methods. Encrypting your key material with this wrapping key protects the key material in transit.
- This Cloud KMS public wrapping key is used to *encrypt*, on the client, the key to be imported. The private key matching this public key is stored within Google Cloud and is used to unwrap the key after it reaches the Google Cloud project. The import method you choose determines the algorithm used to create the wrapping key.

Before we begin to go over instructions on how to bring your key, make sure of the following prerequisites:

- Prepare the local system by choosing *one* of the following options. Automatic key wrapping is recommended for most users.
    - **If you want to allow gcloud to wrap your keys automatically** before transmitting them to Google Cloud, you must install the Pyca cryptography library on your local system. The Pyca library is used by the import job that wraps and protects the key locally before sending it to Google Cloud.
    - **If you want to wrap your keys manually**, you must configure OpenSSL for manual key wrapping.

- In addition to these options, make sure you follow these guidelines as well:
  - Verify that your key's algorithm and length are supported by Google Cloud KMS. Allowable algorithms for a key depend on whether the key is used for symmetric encryption, asymmetric encryption, or asymmetric signing, and whether it is stored in software or an HSM. You will specify the key's algorithm as part of the import request.
  - Separately, you must also verify how the key is encoded, and adjust if necessary.

Now let us start with the instructions on importing keys to Google Cloud. For simplicity, we will be using `gcloud`.

Step 1: Creating a blank key

1. Create a project in the Google Cloud console, and enable Cloud KMS.
2. If you do not have a key to import but want to test importing keys, you can create a symmetric key on the local system, using the following command:

```
openssl rand 32 > ${HOME}/test.bin
```

> **Note**
> Use this key for testing only. A key created this way might not be appropriate for production use.

3. Create a new key ring in a region of your choice.
4. Create a new key of type imported key by using the following parameters:
   I. **Name**: `ext-key-3`
   II. **Protection Level**: `Software`
   III. **Purpose**: `Symmetric encrypt/decrypt`
   IV. **Rotation Period**: `Manual`
5. An empty key is created with no version; you will be redirected to create a key version.
6. Use the following `gcloud` command to create the key version. We will be creating a symmetric key:

```
gcloud kms keys create ext-key-4 --location us-central1
--keyring external-central-key-ring-name --purpose encryption
--skip-initial-version-creation
```

7. Create a new import job.

8. Download the wrapping key.

   This key will not be available for use yet since it is an empty shell to hold the key material that you will import in the following steps. Now, you have two options:

   - Upload your pre-wrapped encryption key.
   - Use the `gcloud` command to provide your key to the job, which will automatically wrap it. You will use different flags to make the import request, depending on whether you want the `gcloud` command-line tool to wrap your key automatically or you have wrapped your key manually. We will use the `gcloud` option here as it is simple to understand.

Step 2: Importing the key using an import job

1. Set `export CLOUDSDK_PYTHON_SITEPACKAGES=1`, otherwise, you will see the following error:

   ```
   Cannot load the Pyca cryptography library. Either the library is
   not installed, or site packages are not enabled for the Google
   Cloud SDK. Please consult https://cloud.google.com/kms/docs/
   crypto for further instructions
   ```

2. Make sure the preceding environment variable is set by using this command:

   ```
   echo $CLOUDSDK_PYTHON_SITEPACKAGES
   ```

3. Execute the following command to upload your key to Cloud KMS:

   ```
   user@cloudshell:~ $ gcloud kms keys versions import   --import-
   job ext-key-import-job   --location us-central1   --keyring
   external-central-key-ring-name   --key ext-key-4   --algorithm
   google-symmetric-encryption   --target-key-file ${HOME}/test.bin
   --public-key-file /home/user/ext-key-import-job.pub --verbosity
   info
   ```

   You will see an output similar to what is shown here:

   ```
   INFO: Display format: "default"
   algorithm: GOOGLE_SYMMETRIC_ENCRYPTION
   createTime: '2020-04-08T22:25:46.597631370Z'
   importJob: projects/PROJECT_ID/locations/us-central1/keyRings/
   external-central-key-ring-name/importJobs/ext-key-import-job
   name: projects/PROJECT_ID/locations/us-central1/keyRings/
   external-central-key-ring-name/cryptoKeys/ext-key-1/
   cryptoKeyVersions/1
   protectionLevel: SOFTWARE
   state: PENDING_IMPORT
   ```

4. Wait until the key version is in the **ENABLED** state, then set it to primary using the following `gcloud` command:

```
gloud kms keys set-primary-version ext-key-1 --version=1
--keyring=external-central-key-ring-name --location=us-central1
```

Cloud KMS will only use the primary version of the key. Now that you have set it, let us move on to using it.

Step 3: Verifying key encryption and decryption

1. Create a `secrets.txt` file with some plain text in it. Encrypt that file using `gcloud`:

```
 gcloud kms encrypt --ciphertext-file=secrets.txt.enc
--plaintext-file=secrets.txt --key=ext-key-4 --keyring=external-
central-key-ring-name --location=us-central1
```

2. Now use the following command to decrypt it. Decrypt the file using the same key:

```
gcloud kms decrypt --ciphertext-file=secrets.txt.
enc --plaintext-file=secrets.txt.dec --key=ext-key-1
--keyring=external-central-key-ring-name --location=us-central1
```

   Verify the text you have decrypted is the same as the plain text. This tutorial demonstrated that you can bring your own key material and use it for encryption and decryption.

Now that we understand several types of keys you can use in Cloud KMS, let us understand the lifecycle of keys.

Key lifecycle management

While operating your workloads, you need to manage the lifecycle of your keys. The US **National Information Standards Institute** (**NIST**) **special publication** (**SP**) *800-57, Part 1* describes a key management lifecycle that is divided into four phases: pre-operational, operational, post-operational, and destroyed.

The following section provides a mapping of the NIST lifecycle functions from the publication to Google Cloud KMS lifecycle functions.

1. The **Pre-operational** lifecycle phase is mapped to the following:

   I. **NIST section 8.1.4 Keying-Material Installation Function**: The equivalent Cloud KMS operation is **Key import**.

   II. **NIST section 8.1.5 Key Establishment Function**: The equivalent Cloud KMS operation is **Key creation (symmetric, asymmetric)**.

2. The **Operational** lifecycle phase is mapped to the following:

    I. **NIST section 8.2.1 Normal Operational Storage Function**: The equivalent Cloud KMS operation is **Key creation in SOFTWARE, HSM, or EXTERNAL protection levels (symmetric, asymmetric)**.

    II. **NIST section 8.2.3 Key Change Function**: The equivalent Cloud KMS operation is **Key rotation**.

3. The **Post-operational** lifecycle phase is mapped to the following:

    I. **NIST section 8.3.4 Key Destruction Function**: The equivalent Cloud KMS operation is **Key destruction (and recovery)**.

In addition, Cloud KMS offers the capability to disable a key. Typically, you disable the old key after rotating to a new key. This gives you a period to confirm that no additional data needs to be re-encrypted before the key is destroyed. You can re-enable a disabled key if you discover data that needs to be re-encrypted. When you are sure that there is no more data that was encrypted by using the old key, the key can be scheduled for destruction.

Key IAM permissions

When considering key permissions, think about the hierarchy in which the key exists. A key exists in a key ring, a project, a folder in another folder, or under "Cloud Organization".

Recall that there are two fundamental security principles IAM enforces:

- Principle of separation of duties
- Principle of least privilege

A primary role a principal can play is the Cloud KMS CryptoKey Encrypter/Decrypter role at various levels of the hierarchy. There are several other roles Cloud KMS has based on how you structure it. Please refer to the Google Cloud KMS documentation for the list of other IAM roles: `https://packt.link/RyY17`.

We have looked at various IAM roles and permissions for Cloud KMS; let us now look at some best practices on how to manage access:

- Key management roles can be granted based on the culture and process of your enterprises. Traditionally, this role is played by the IT security team.
- For a large or complex organization, you might decide on an approach such as the following:

  - Grant members of your IT security team the Cloud KMS Admin role (`roles/cloudkms.admin`) across all projects. If different team members handle various aspects of a key's lifecycle, you can grant those team members a more granular role, such as the Cloud KMS Importer role (`roles/cloudkms.importer`).
  - Grant the Cloud KMS Encrypter/Decrypter role (`roles/cloudkms.cryptoKeyEncrypterDecrypter`) to users or service accounts that read or write encrypted data.
  - Grant the Cloud KMS Public Key Viewer role (`roles/cloudkms.publicKeyViewer`) to users or service accounts that need to view the public portion of a key used for asymmetric encryption.

- For a small organization, each project team can manage its own Cloud KMS and grant the IT security team access to audit the keys and usage.
- Consider hosting your keys in a separate Google Cloud project from the data protected by those keys. A user with a basic or highly privileged role in one project, such as an editor, cannot use this role to gain unauthorized access to keys in a different project.
- Avoid granting the owner role to any member. Without the owner role, no member in the project can create a key and use it to decrypt data or for signing, unless each of these permissions is granted to that member. To grant broad administrative access without granting the ability to encrypt or decrypt, grant the Cloud KMS Admin (`roles/cloudkms.admin`) role instead.
- To limit access to encrypted data, such as customer data, you can restrict who can access the key and who can use the key for decryption. If necessary, you can create granular custom roles to meet your business requirements.
- For projects using the VPC Service Controls perimeter, we recommend that you have a separate Cloud KMS project for the perimeter.

Now that we have looked at various IAM roles and best practices for key management, let us look at Cloud HSM offerings next.

Cloud HSM

Cloud HSM is a cloud-hosted HSM service that allows you to host encryption keys and perform cryptographic operations in a cluster of FIPS 140-2 Level 3 certified HSMs. The Cloud HSM cluster is managed by Google for you; you do not get access to the physical device. You also do not need to patch or scale it for multi-regional applications. Cloud HSM uses Cloud KMS as its frontend; you use Cloud KMS APIs to interact with the Cloud HSM backend. This abstracts the communication with Cloud HSM, so you do not need to use Cloud HSM-specific code.

When you use HSM-backed keys and key versions, the Google Cloud project that makes the cryptographic request incurs cryptographic operation quota usage, and the Google Cloud project that contains the HSM keys incurs HSM QPM quota usage.

> **Note**
> You can find more information on Cloud KMS in the Google Cloud documentation for quotas.

Here are some architectural characteristics of Cloud HSM that you should be aware of:

- The Cloud HSM service provides hardware-backed keys to Cloud KMS. It offers Google Cloud customers the ability to manage and use cryptographic keys that are protected by fully managed HSMs in Google data centers.
- The service is available and auto-scales horizontally. It is designed to be extremely resilient to the unpredictable demands of workloads.
- Keys are cryptographically bound to the Cloud KMS region in which the key ring was defined. You can choose to make keys global or enforce strict regional restrictions.
- With Cloud HSM, the keys that you create and use cannot be exported outside of the cluster of HSMs belonging to the region specified at the time of key creation.
- Using Cloud HSM, you can verifiably attest that your cryptographic keys are created and used exclusively within a hardware device.
- No application changes are required for existing Cloud KMS customers to use Cloud HSM: the Cloud HSM services are accessed using the same API and client libraries as the Cloud KMS software backend.
- If you choose to, you can use a PKCS#11-compliant C++ library, `libkmsp11to.so`, to interact with Cloud HSM from your application. You use this library typically for the following purposes:
  - Creating signatures, certifications, or **certificate signing requests** (**CSRs**) using the command line
  - TLS web session signing in web servers such as Apache or NGINX backed by Cloud HSM keys
  - Migrating your application to a Google Cloud project that uses PKCS#11 APIs

  This library authenticates Google Cloud using a service account. The library uses a YAML file to store the Cloud KMS configurations. Use the following link to learn more about this: `https://packt.link/H53BX`.

Let us look at the HSM key hierarchy now.

HSM key hierarchy

In *Figure 9.7*, Cloud KMS is the proxy for HSM. Cloud HSM root keys wrap customer keys, and then Cloud KMS wraps the HSM keys that are passed to Google Datastore for storage.

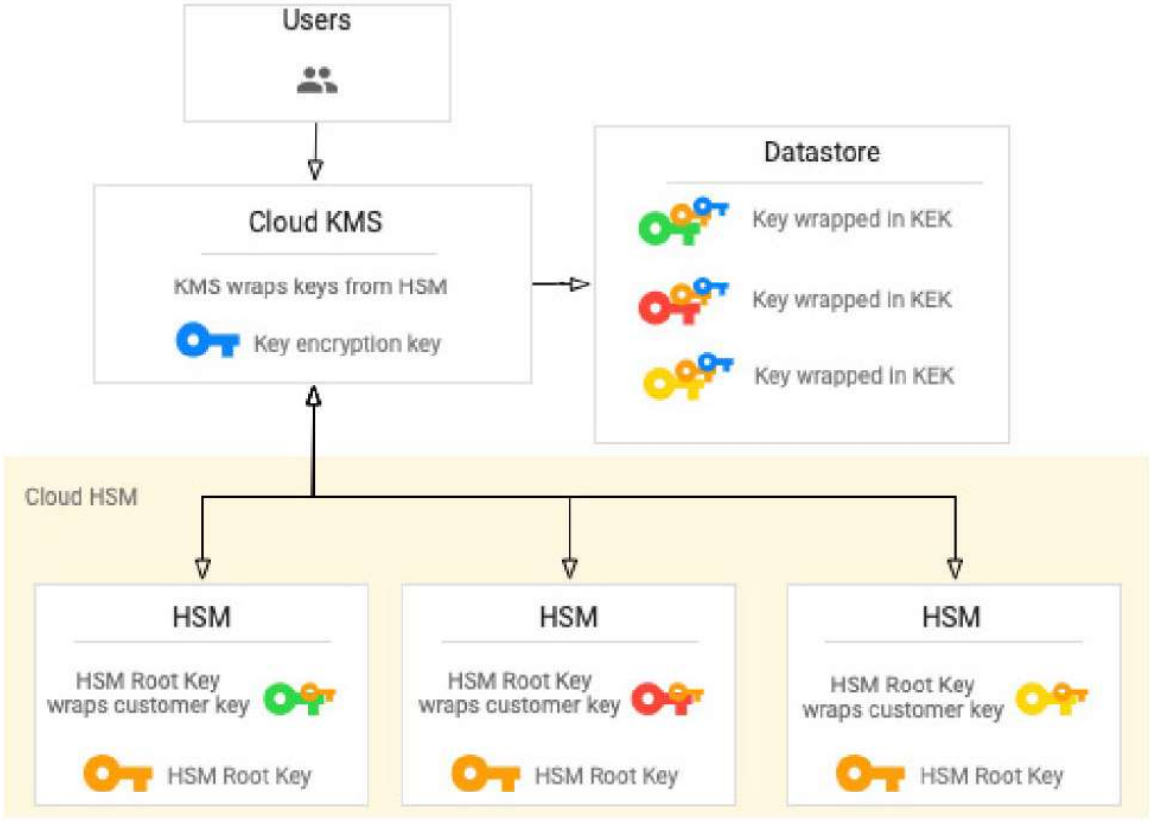

Figure 9.7 – HSM key hierarchy

Cloud HSM has a key (not shown) that controls the migration of material inside the Cloud HSM administrative domain. A region might have multiple HSM administrative domains.

The HSM root key has two characteristics:

- Customer keys wrapped by HSM root keys can be used on the HSM, but the HSM never returns the plaintext of the customer key
- The HSM only uses the customer key for operations

Datastore protection

HSMs are not used as permanent storage for keys; they store keys only while they are being used. Because HSM storage is constrained, the HSM keys are encrypted and stored in the Cloud KMS key datastore.

Rotation policy

Several types of keys are involved in Cloud HSM's key protection strategy. Customers rotate their own keys and rely on Cloud KMS to rotate the HSM keys.

Provisioning and handling process

The provisioning of HSMs is carried out in a lab equipped with numerous physical and logical safeguards, including, for example, multi-party authorization controls to help prevent single-actor compromise.

The following are Cloud HSM system-level invariants:

- The key is generated on an HSM and, throughout its lifetime, never leaves the well-defined boundaries of HSMs
- It may be replicated on other HSMs, or on backup HSMs
- It can be used as a KEK to directly or indirectly wrap customer keys that HSMs use
- Customer keys cannot be extracted as plaintext. Customer keys cannot be moved outside the region of origin
- All configuration changes to provisioned HSMs are guarded through multiple security safeguards
- Administrative operations are logged, adhering to the separation of duties between Cloud HSM administrators and logging administrators
- HSMs are designed to be protected from tampering (such as by the insertion of malicious hardware or software modifications, or unauthorized extraction of secrets) throughout the operational lifecycle

Vendor-controlled firmware

HSM firmware is digitally signed by the HSM vendor. Google cannot create or update the HSM firmware. All firmware from the vendor is signed, including development firmware that is used for testing.

Regionality

Customer keys are assigned to specific geographic regions because of their binding to a specific HSM root key. For example, a key created specifically in the us-west1 region cannot migrate into the us-east1 region or the US multi-region. Similarly, a key created in the US multi-region cannot migrate into or out of the us-west1 region.

Strict security procedures safeguard HSM hardware

As mandated by FIPS 140-2 level 3, HSM devices have built-in mechanisms to help protect against and provide evidence of physical tampering.

In addition to the assurances provided by the HSM hardware itself, the infrastructure for Cloud HSM is managed according to Google infrastructure security design.

> **Note**
> You can find more information at the following link: `https://packt.link/B8pgP`.

Documented, auditable procedures protect the integrity of each HSM during provisioning, deployment, and in production:

- All HSM configurations must be verified by multiple Cloud HSM **site reliability engineers** (**SREs**) before the HSM can be deployed to a data center
- After an HSM is put into service, configuration change can only be initiated and verified by multiple Cloud HSM SREs
- An HSM can only receive firmware that is signed by the HSM manufacturer
- HSM hardware is not directly exposed to any network
- Servers that host HSM hardware are prevented from running unauthorized processes

Service and tenant isolation

The Cloud HSM architecture ensures that HSMs are protected from malicious or inadvertent interference from other services or tenants.

An HSM that is part of this architecture accepts requests only from Cloud HSM, and the Cloud HSM service accepts requests only from Cloud KMS. Cloud KMS enforces that callers have appropriate IAM permissions on the keys that they attempt to use. Unauthorized requests do not reach HSMs.

Key creation flow in HSM

When you create an HSM-backed key, the Cloud KMS API does not create the key material but requests that the HSM creates it.

An HSM can only create keys in locations it supports. Each partition on an HSM contains a wrapping key corresponding to a Cloud KMS location. The wrapping key is shared among all partitions that support the Cloud KMS location. The key-creation process looks like this:

1. The KMS API **Google Front End Service** (**GFE**) routes the key creation request to a Cloud KMS server in the location that corresponds to the request.
2. The Cloud KMS API verifies the caller's identity, the caller's permission to create keys in the project, and that the caller has a sufficient write request quota.
3. The Cloud KMS API forwards the request to Cloud HSM.
4. Cloud HSM directly interfaces with the HSM. The HSM does the following:
    I. Creates the key and wraps it with the location-specific wrapping key.
    II. Creates the attestation statement for the key and signs it with the partition signing key.
5. After Cloud HSM returns the wrapped key and attestation to Cloud KMS, the Cloud KMS API wraps the HSM-wrapped key according to the Cloud KMS key hierarchy, then writes it to the project.

This design ensures that the key cannot be unwrapped or used outside of an HSM, cannot be extracted from the HSM, and exists in its unwrapped state only within locations you intend.

Key attestations

In cryptography, an attestation is a machine-readable, programmatically provable statement that a piece of software makes about itself. Attestations are a key component of trusted computing and may be required for compliance reasons.

To view and verify the attestations, you request a cryptographically signed attestation statement from the HSM, along with the certificate chains used to sign it. The attestation statement is produced by the HSM hardware and signed by certificates owned by Google and by the HSM manufacturer (currently, Google uses Marvell HSM devices).

After downloading the attestation statement and the certificate chains, you can check its attributes or verify the validity of the attestation using the certificate chains.

Google has developed an attestation script, an open source Python script that you can use to verify the attestations. You can view the source code for the script to learn more about the attestation format and how verification works, or as a model for a customized solution, at the following link: `https://packt.link/ANloU`.

> **Note**
> To learn more about how to view and verify attestations, see the following Google Cloud documentation: `https://packt.link/2A5Er`.

Cryptographic operation flow in HSM

When you perform a cryptographic operation in Cloud KMS, you do not need to know whether you are using an HSM-backed or software key. When the Cloud KMS API detects that an operation involves an HSM-backed key, it forwards the request to an HSM in the same location:

1. The GFE routes the request to a Cloud KMS server in the appropriate location. The Cloud KMS API verifies the caller's identity, the caller's permission to access the key and perform the operation, and the project's quota for cryptographic operations.
2. The Cloud KMS API retrieves the wrapped key from the datastore and decrypts one level of encryption using the Cloud KMS master key. The key is still wrapped with the Cloud HSM wrapping key for the Cloud KMS location.
3. The Cloud KMS API detects that the protection level is HSM and sends the partially unwrapped key, along with the inputs to the cryptographic operation, to Cloud HSM.
4. Cloud HSM directly interfaces with the HSM. The HSM does the following:
    I. Checks that the wrapped key and its attributes have not been modified.
    II. Unwraps the key and load it into its storage.
    III. Performs the cryptographic operation and returns the result.
5. The Cloud KMS API passes the result back to the caller.

Cryptographic operations using HSM-backed keys are performed entirely within an HSM in the configured location, and only the result is visible to the caller. Now that we have looked at Cloud HSM, let us move on and understand Cloud **External Key Manager** (**EKM**). This is the newest offering from Google for cloud key management.

Cloud EKM

Cloud EKM is one of the newest offerings for data protection. With Cloud EKM, you use the keys that you manage within an EKM partner.

Cloud EKM provides several benefits:

- **Key provenance**: You control the location and distribution of your externally managed keys. Externally managed keys are never cached or stored within Google Cloud. Instead, Cloud EKM communicates directly with the external key management partner for each request.

- **Access control**: You manage access to your externally managed keys. Before you can use an externally managed key to encrypt or decrypt data in Google Cloud, you must grant the Google Cloud project access to use the key. You can revoke this access at any time.
- **Centralized key management**: You can manage your keys and access policies from a specific location and user interface, whether the data they protect resides in the cloud or on your premises.

In all cases, the key resides on the external system and is never sent to Google. The following partners are supported for EKM hosting:

- Fortanix
- Futurex
- Thales
- Virtu

Google Cloud supports several cloud services for Cloud EKM for CMEK encryption. Please refer to the Google Cloud documentation for currently supported services.

The architecture of Cloud EKM

Figure 9.8 shows the architecture of Cloud EKM.

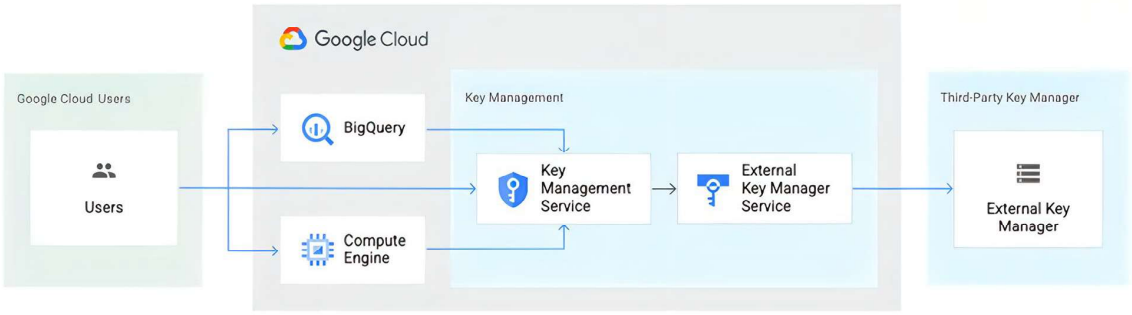

Figure 9.8 – Cloud EKM architecture

The EKM key creation flow is as follows:

1. Establish an account with the partner. They have their own portal where you manage the keys.
2. Create a key or use an existing key. This key has a unique URI or key path. Copy this path to use in *Step 4*.
3. Next, you grant your Google Cloud project workload's service account access to use the key in the external key management partner system.
4. In your Google Cloud project, you create a Cloud EKM key, using the URI or key path for the externally managed key.

While Cloud EKM seems like a great option to use, there are some considerations you should be aware of:

- Cloud EKM can be used with Hosted Private HSM to create a single-tenant HSM solution integrated with Cloud KMS. Choose a Cloud EKM partner that supports single-tenant HSMs.
- When you use a Cloud EKM key, Google has no control over the availability of your externally managed key in the partner system. Google cannot recover your data if you lose keys you manage outside of Google Cloud.
- Review the guidelines about external key management partners and regions when choosing the locations for your Cloud EKM keys.
- Review the Cloud EKM **service-level agreement** (**SLA**).
- Communicating with an external service over the internet can lead to problems with reliability, availability, and latency. For applications with a low tolerance for these types of risks, consider using Cloud HSM or Cloud KMS to store your key material.
- You will need a support contract with the external key management partner. Google Cloud support can only provide support for issues in Google Cloud services and cannot directly assist with issues on external systems. You may need to work with support on both sides to troubleshoot interoperability issues.

Restrictions

Here are some general restrictions you should be aware of while using Cloud EKM:

- Automatic rotation is not supported.
- When you create a Cloud EKM key using the API or the Google Cloud CLI, it must not have an initial key version. This does not apply to Cloud EKM keys created using the Cloud console.
- Cloud EKM operations are subject to specific quotas in addition to the quotas on Cloud KMS operations.

When it comes to *symmetric encryption keys*, be aware of these restrictions:

- Symmetric encryption keys are only supported for the following:
  - CMEKs in supported integration services
  - Symmetric encryption and decryption using Cloud KMS directly
- Data that is encrypted by Cloud EKM using an externally managed key cannot be decrypted without using Cloud EKM

When it comes to *asymmetric signing keys*, be aware of these restrictions:

- Asymmetric signing keys are limited to a subset of Cloud KMS algorithms
- Asymmetric signing keys are only supported for the following:
  - Asymmetric signing using Cloud KMS directly
  - Custom signing key with Access Approval
- Once an asymmetric signing algorithm is set on a Cloud EKM key, it cannot be modified
- Signing must be done on the `data` field

Now that you understand various Cloud KMS offerings, let us look at some of the best practices for Cloud KMS.

Cloud KMS best practices

Key access and key ring access are managed by organizing keys into key rings and projects, and by granting IAM roles on the keys, key rings, and projects. As you build out your cloud environment, follow the guidance in the following list for how to design your key resource hierarchy to reduce risk:

1. Create a dedicated project for Cloud KMS that is separate from workload projects.
2. Add key rings into the dedicated Cloud KMS project. Create key rings as needed to impose a separation of duties.
3. Monitor privileged admin operations: key deletion operations for out-of-band key creation are considered a privileged operation.
4. Review CMEK-related findings in Security Command Center.
5. Use encryption keys with the appropriate key strength and protection level for data sensitivity or classification. For example, for sensitive data, use keys with a higher strength. Additionally, use encryption keys with different protection levels for different data types.

The following list provides guidance on how to apply IAM roles to support your security and administrative needs:

- Avoid the basic project-wide roles, such as owner, editor, and viewer, on projects that host keys or on enclosing folders or organizations. Instead, designate an organization administrator that is granted at the organization level, as noted on the Cloud KMS separation of duties page.
- The Organization Admin functions as the administrator for all the organization's cryptographic keys. Grant other IAM roles at the project level. If you have further concerns about the separation of duties, grant IAM roles at the key ring level or key level.
- Use Cloud KMS predefined roles for least privilege and separation of duties. For example, do the following:
  - Separate administration roles (`roles/cloudkms.admin` and `roles/cloudkms.importer`) and usage roles for keys
  - Limit the use of `roles/cloudkms.admin` to members of the security administration team and to service accounts that are responsible for creating key rings and key objects through tools such as Terraform
  - For asymmetric keys, grant roles that need private key access (`roles/cloudkms.cryptoKeyDecrypter` and `roles/cloudkms.signer`) separately from roles that do not need private key access (`roles/cloudkms.publicKeyViewer` and `roles/cloudkms.signerVerifier`)
  - In general, grant the most limited set of permissions to the lowest object in the resource hierarchy

Now that you are familiar with the best practices for key management, let us understand some important decisions for key management that you must make for your cloud infrastructure.

Cloud KMS infrastructure decisions

When you are setting up Cloud KMS for a project, you must make several decisions regarding your keys. The following table provides you with guidance on what factors to consider when you create keys and key rings.

Key attribute	**Key attribute guidance**
Key location	Choose the location that is geographically closest to the data on which you will be performing cryptographic operations. Use the same ring location for CMEKs used for the data they are encrypting. For more information, see choosing the best type of location in the Cloud KMS documentation.

Key attribute	Key attribute guidance
Key protection level	Use protection level `EXTERNAL` when your workloads require keys to be maintained outside of the Google Cloud infrastructure in a partner system.
	Use protection level `HSM` when your workloads require keys to be protected in FIPS 140-2 Level 3-certified hardware security modules (HSMs).
	Use protection level `SOFTWARE` when your workload does not require keys to be maintained outside of Google Cloud and does not require keys to be protected with FIPS 140-2 Level 3-certified hardware.
	Choose appropriate protection levels for development, staging, and production environments. Because the Cloud KMS API is the same regardless of the protection level, you can use different protection levels in different environments, and you can relax protection levels where there is no production data.
Key source	Allow Cloud KMS to generate keys unless you have workload requirements that require keys to be generated in a specific manner or environment. For externally generated keys, use Cloud KMS key import to import them as described in the *Importing keys into Cloud KMS* section.
Key rotation	For symmetric encryption, configure automation key rotation by setting a key rotation period and starting time when you create the key.
	For asymmetric encryption, you must always manually rotate keys, because the new public key must be distributed before the key pair can be used. Cloud KMS does not support automatic key rotation for asymmetric keys.
	If you have indications that keys have been compromised, manually rotate the keys and re-encrypt data that was encrypted by the compromised keys as soon as possible. To re-encrypt data, you typically download the old data, decrypt it with the old key, encrypt the old data using the new key, and then re-upload the re-encrypted data.
Key destruction	Destroy old keys when there is no data encrypted by those keys.
Key attribute	Key attribute guidance

Table 9.1 – KMS infrastructure decision

Now let us look at a few other factors involved in decision-making.

Application data encryption

Your application might use Cloud KMS by calling the API directly to encrypt, decrypt, sign, and verify data. Applications that handle data encryption directly should use the envelope encryption approach, which provides better application availability and scaling behavior.

> **Note**
> In order to perform application data encryption in this way, your application must have IAM access to both the key and the data.

Integrated Google Cloud encryption

By default, Google Cloud encrypts all your data at rest and in transit without requiring any explicit setup by you. This default encryption for data at rest, which is transparent to you, uses Cloud KMS behind the scenes and manages IAM access to the keys on your behalf.

CMEKs

For more control over the keys for encrypting data at rest in a Google Cloud project, you can use several Google Cloud services that offer the ability to protect data related to those services by using encryption keys managed by the customer within Cloud KMS. These encryption keys are called CMEKs.

Google Cloud products that offer CMEK integration might require the keys to be hosted in the same location as the data used with the key. Cloud KMS might use different names for some locations than other services use. For example, the Cloud KMS multi-regional location Europe corresponds to the Cloud Storage multi-region location EU. Cloud KMS also has some locations that are not available in all other services. For example, the Cloud KMS dual-regional location eur5 has no counterpart in Cloud Storage. You need to identify these requirements before you create the Cloud KMS key ring so that the key ring is created in the correct location. Keep in mind that you cannot delete a key ring.

Importing keys into Cloud KMS

Your workloads might require you to generate the keys outside of Cloud KMS. In this case, you can import key material into Cloud KMS. Furthermore, you might need to provide assurance to reliant parties on the key generation and import processes. These additional steps are referred to as a **key ceremony**.

You use a key ceremony to help people trust that the key is being stored and used securely. Two examples of key ceremonies are the DNSSEC root **key signing key** (**KSK**) ceremony and the ceremony used by Google to create new root CA keys. Both ceremonies support high transparency and high assurance requirements because the resulting keys must be trusted by the entire internet community.

During the key ceremony, you generate the key material and encrypt known plaintext into ciphertext. You then import the key material into Cloud KMS and use the ciphertext to verify the imported key. After you have successfully completed the key ceremony, you can enable the key in Cloud KMS and use it for cryptographic operations.

Because key ceremonies require a lot of setup and staffing, you should carefully choose which keys require ceremonies.

> **Note**
> This is a high-level description of the key ceremony. Depending on the key's trust requirements, you might need more steps.

Cloud KMS API

The Cloud KMS service has an endpoint of `cloudkms.googleapis.com`. Here are a few widely used endpoints that you should be aware of:

- `projects.locations`
- `projects.locations.ekmConnections`
- `projects.locations.keyRings`
  - `create`
  - `list`
  - `get`
  - `getIamPolicy`
  - `setIamPolicy`
- `projects.locations.keyRings.cryptoKeys`
  - `create`
  - `decrypt`
  - `encrypt`
  - `get`

    - `getIamPolicy`
    - `list`
    - `setIamPolicy`
    - `updatePrimaryVersion`
- `projects.locations.keyRings.cryptoKeys.cryptoKeyVersions`
- `Projects.locations.keyRings.ImportJobs`

When interacting with Cloud KMS via a programmatic method, you should have a good understanding of these endpoints. Let us move on and understand the Cloud KMS logging components now.

Cloud KMS logging

The following types of audit logs are available for Cloud KMS:

- **Admin Activity audit logs**: Include `admin write` operations that write metadata or configuration information. You cannot disable Admin Activity audit logs.

  Admin Activity audit logs cover the following Cloud KMS operations:

```
cloudkms.projects.locations.keyRings.create
cloudkms.projects.locations.keyRings.setIamPolicy
cloudkms.projects.locations.keyRings.cryptoKeys.create
cloudkms.projects.locations.keyRings.cryptoKeys.patch
cloudkms.projects.locations.keyRings.cryptoKeys.setIamPolicy
cloudkms.projects.locations.keyRings.cryptoKeys.
updatePrimaryVersion
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.create
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.destroy
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.patch
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.restore
cloudkms.projects.locations.keyRings.importJobs.create
cloudkms.projects.locations.keyRings.importJobs.setIamPolicy
```

- **Data Access audit logs**: These include `admin read` operations that read metadata or configuration information. They also include `data read` and `data write` operations that read or write user-provided data. To receive Data Access audit logs, you must explicitly enable them.

  Data Access audit logs cover the following Cloud KMS operations:

  - `ADMIN_READ` for the following API operations:

```
cloudkms.projects.locations.get
cloudkms.projects.locations.list
cloudkms.projects.locations.keyRings.get
cloudkms.projects.locations.keyRings.getIamPolicy
cloudkms.projects.locations.keyRings.list
cloudkms.projects.locations.keyRings.testIamPermissions
cloudkms.projects.locations.keyRings.cryptoKeys.get
cloudkms.projects.locations.keyRings.cryptoKeys.getIamPolicy
cloudkms.projects.locations.keyRings.cryptoKeys.list
cloudkms.projects.locations.keyRings.cryptoKeys.
testIamPermissions
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.get
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.list
cloudkms.projects.locations.keyRings.importJobs.get
cloudkms.projects.locations.keyRings.importJobs.getIamPolicy
cloudkms.projects.locations.keyRings.importJobs.list
cloudkms.projects.locations.keyRings.importJobs.
testIamPermissions
kmsinventory.organizations.protectedResources.search
kmsinventory.projects.cryptoKeys.list
kmsinventory.projects.locations.keyRings.cryptoKeys.
getProtectedResourcesSummary
```

  - `DATA_READ` for the following API operations:

```
cloudkms.projects.locations.keyRings.cryptoKeys.decrypt
cloudkms.projects.locations.keyRings.cryptoKeys.encrypt
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.asymmetricDecrypt
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.asymmetricSign
cloudkms.projects.locations.keyRings.cryptoKeys.
cryptoKeyVersions.getPublicKey
```

This concludes the logging section and the chapter.

Summary

In this chapter, we went over the details of Cloud KMS, its supported operations, and how to use them. We also looked at bringing your own encryption key to the cloud. We went over advanced options such as Cloud HSM and Cloud EKM. In addition to this, we saw the best practices and Cloud KMS infrastructure decisions while setting up your project on Google Cloud. As a security engineer, you should be able to define the right architecture for key management for your organization and recommend the right compliance options for project teams.

In the next chapter, we will look at data security, specifically how to use Google Cloud's **Data Loss Prevention** (**DLP**) services. Cloud KMS and DLP should bring you one step closer to creating the right strategy for data security.

Further reading

For more information on Google Cloud KMS, refer to the following link:

- Key attestations and verifications: `https://packt.link/V0Fki`