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.

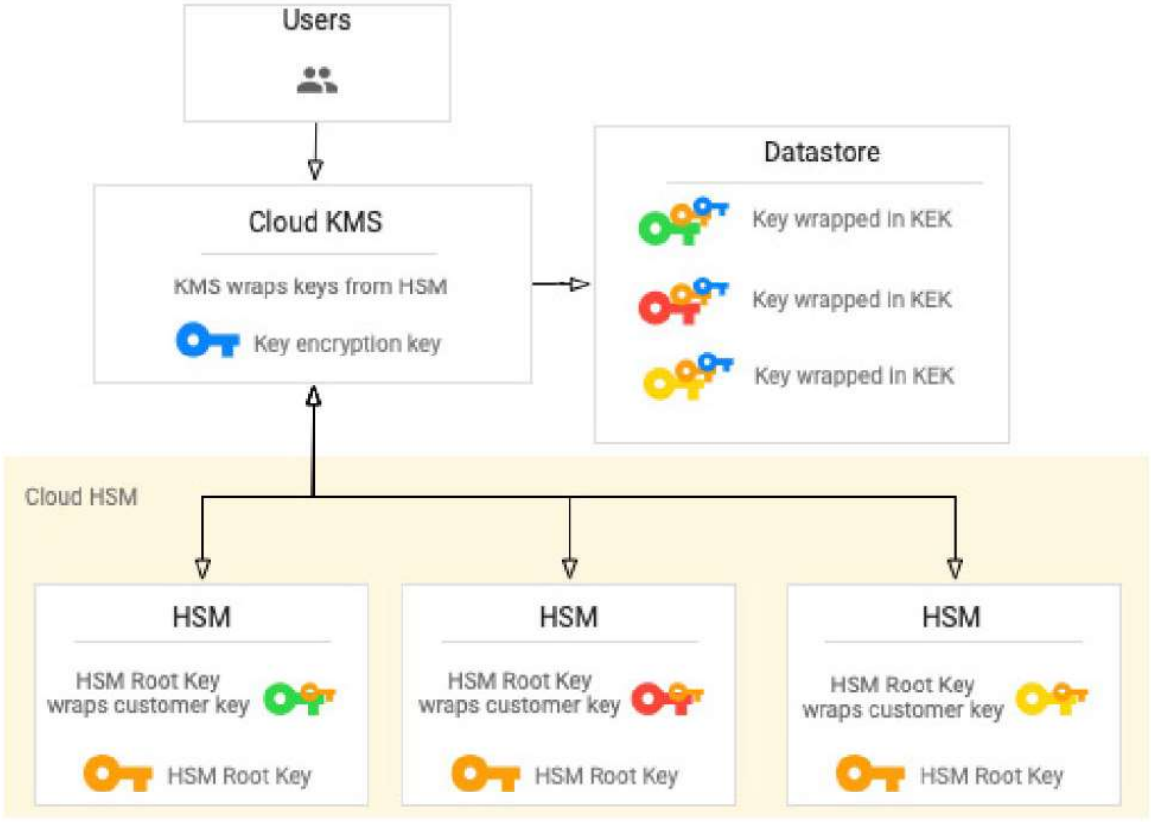

Figure 9.7 – HSM key hierarchy

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.

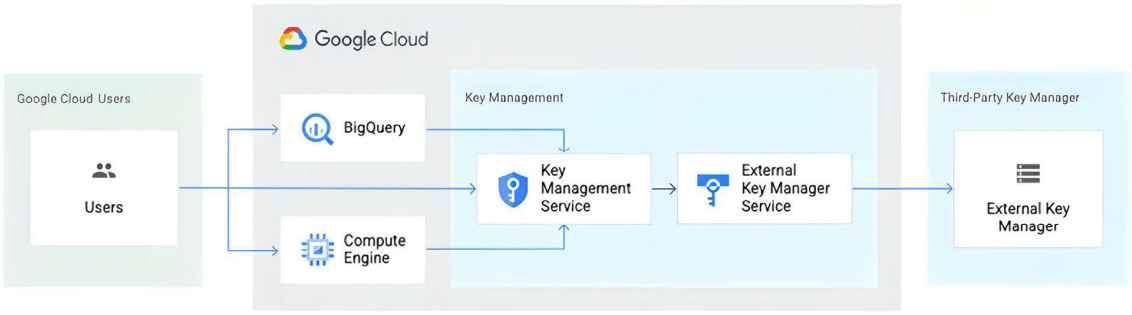

Figure 9.8 – Cloud EKM architecture

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`

10
Cloud Data Loss Prevention

In this chapter, we will look at Google Cloud data loss protection products and capabilities. **Data Loss Prevention** (**DLP**) is a strategy for detecting and preventing the exposure and exfiltration of sensitive data. Google's DLP strategy involves a layered approach. In addition to a proper organizational hierarchy, network security, IAM access, and **VPC Service Controls** (**VPC-SC**), DLP plays a key role in data protection.

Cloud DLP is quite widely used in data pipelines, especially for data warehouses. Protecting confidential data is one of the critical aspects of data workloads, so Cloud DLP helps customers gain visibility of sensitive data risks across the organization. We will look at several features of Cloud DLP, how to configure the product to do inspection and de-identification, and some best practices. There are few tutorials in the chapter, so try out examples to get a solid understanding.

In this chapter, we will cover the following topics:

- Overview of Cloud DLP
- DLP architecture
- How DLP discovery works
- De/re-identification of data
- How to create a DLP scan job
- DLP use cases
- How to mask and tokenize sensitive data
- Best practices and design considerations
- Data exfiltration controls

Overview of Cloud DLP

Cloud DLP offers some key features for Google's customers:

- BigQuery-based data warehouses can be profiled to detect sensitive data, allowing for automated sensitive data discovery. You can scan through the entire Google Cloud organization or choose folders or projects with the profiler's flexibility.
- Over 150 built-in information detectors are available in Cloud DLP. Because DLP is API-based, it can be used to swiftly scan, discover, and classify data from anywhere.
- Support for Cloud Storage, Datastore, and BigQuery is built in: Cloud DLP comes with built-in support for these storage options.
- You can calculate the level of risk to your data privacy: quasi-identifiers are data elements or combinations of data that can be linked to a single individual or a small group of people. Cloud DLP gives you the option to examine statistical features such as k-anonymity and l-diversity, allowing you to assess the risk of data re-identification.
- DLP classification results can be delivered straight to BigQuery for more thorough analysis or exported to your analytical instrument of choice. BigQuery and Data Studio can be used to create bespoke reports.
- Cloud DLP secures your data by storing it in memory. The data in the backend is not persistent. In addition, the product is subjected to various independent third-party audits to ensure data confidentiality, privacy, and safety.
- Cloud DLP may be swiftly deployed using reusable templates, with periodic scans monitoring the data and Pub/Sub notifications for integration into serverless architectures.
- To suit your company's needs, you can add your own custom info types, alter detection levels, and establish detection rules.

Now that you understand Cloud DLP's capabilities at a high level, let us understand the DLP architecture.

DLP architecture options

Cloud DLP primarily is seen in three architecture patterns: the content/streaming and storage methods and a hybrid architecture that combines these two patterns.

Content methods

In this architecture option, the data is streamed to the Cloud DLP APIs for inspection/classification or de-identification/transformation. A synchronous API response is received from Cloud DLP. In this case, the client application is expected to process the response. This architecture is typically seen in data pipelines or call center applications where real-time response is needed.

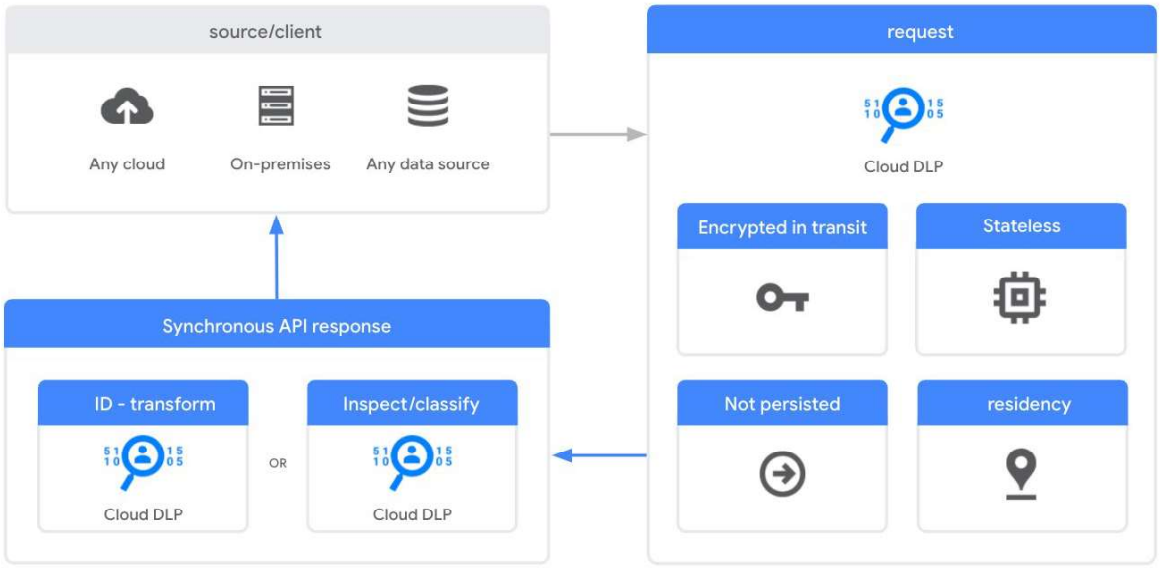

Figure 10.1 – Content method architecture

As shown in *Figure 10.1*, using content inspection, you stream small payloads of data to Cloud DLP along with instructions about what to inspect for. Cloud DLP then inspects the data for sensitive content and **personally identifiable information** (**PII**) and returns the results of its scan back to you.

Storage methods

In this architecture option, a job is set up based on a trigger to scan for sensitive data. The results of the scans are published so that action can be taken based on the results. The results can be pushed to **Security Operations Center** (**SOC**) tools such as Security Command Center so security responders can take action or an automated response detection can be built using Cloud Functions or a more sophisticated product such as SOAR.

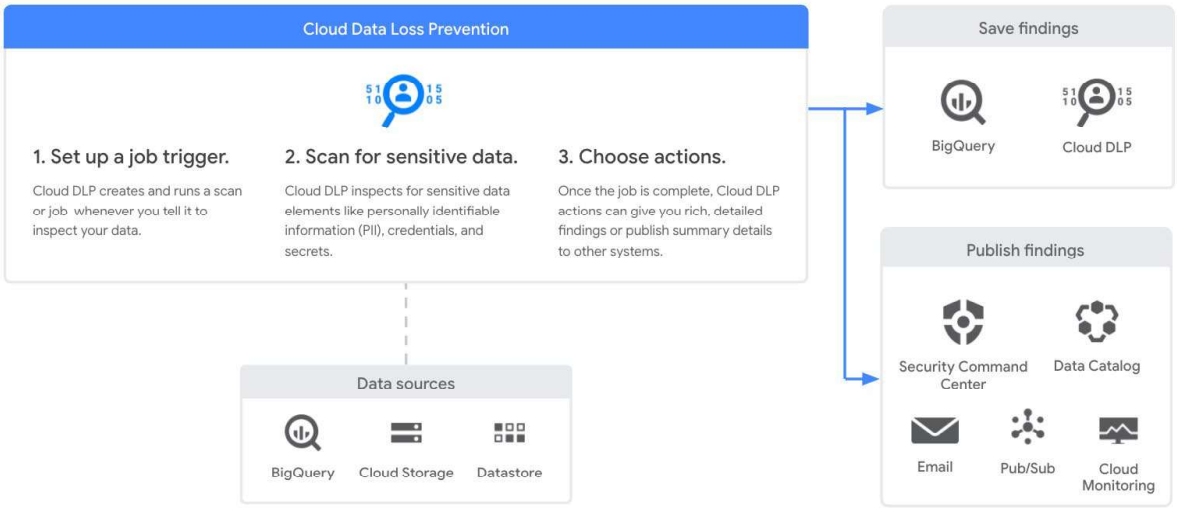

Figure 10.2 – Storage method architecture

As shown in *Figure 10.2*, in this architecture option, you tell Cloud DLP what to inspect and then Cloud DLP runs a job that scans the repository. After the scan is complete, Cloud DLP saves a summary of the results of the scan back to the job. You can additionally specify that the results are sent to another Google Cloud product for analysis, such as a separate BigQuery table or Security Command Center.

The type of scan typically involves those Google Cloud projects where you do not expect to have sensitive data. For projects that are meant to have sensitive data such as cardholder data or PII, you should also invest in data exfiltration controls such as VPC Service Controls for DLP.

Hybrid methods

Hybrid jobs and job triggers enable you to broaden the scope of protection that Cloud DLP provides beyond simple content inspection requests and **Google Cloud Storage** (**GCS**) repository scanning.

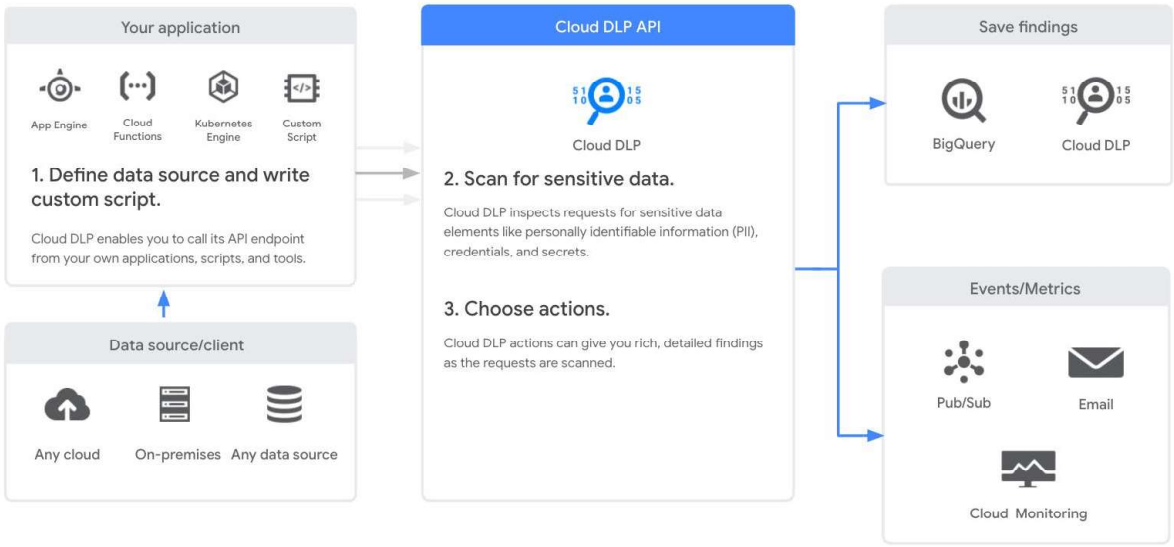

Figure 10.3 – Hybrid inspection architecture

As shown in *Figure 10.3*, you may feed data from practically any source—including sources outside Google Cloud—directly to Cloud DLP via hybrid jobs and job triggers, and Cloud DLP will analyze the data for sensitive information and automatically record and aggregate the scan results for further analysis.

Hybrid jobs and *job triggers* encompass a set of asynchronous API methods that allow you to scan payloads of data sent from virtually any source for sensitive information, and then store the findings in Google Cloud. Hybrid jobs enable you to write your own data crawlers that behave and serve data similarly to the Cloud DLP storage inspection methods.

Hybrid environments are common in enterprises. Many organizations store and process sensitive data using some combination of the following:

- Other cloud providers
- On-premises servers or other data repositories
- Non-native storage systems, such as systems running inside a virtual machine
- Web and mobile apps
- Google Cloud-based solutions

Using hybrid jobs, Cloud DLP can inspect data sent to it from any of these sources. Example scenarios include the following:

- Inspect data stored in Amazon **Relational Database Service** (**RDS**), MySQL running inside a virtual machine, or an on-premises database
- Inspect and tokenize data as you migrate from on-premises to the cloud, or between production, development, and analytics
- Inspect and de-identify transactions from a web or mobile application before storing the data at rest

The basic workflow for using hybrid jobs and job triggers is as follows:

1. You write a script or create a workflow that sends data to Cloud DLP for inspection along with some metadata.
2. You configure and create a hybrid job resource or trigger and enable it to activate when it receives data.
3. Your script or workflow runs on the client side and sends data to Cloud DLP. The data includes an activation message and the job trigger's identifier, which triggers the inspection.
4. Cloud DLP inspects the data according to the criteria you set in the hybrid job or trigger.
5. Cloud DLP saves the results of the scan to the hybrid job resource, along with the metadata that you provide. You can examine the results using the Cloud DLP UI in the Cloud console.
6. Optionally, Cloud DLP can run post-scan actions, such as saving inspection results data to a BigQuery table or notifying you by email or Pub/Sub.

A hybrid job trigger enables you to create, activate, and stop jobs so that you can trigger actions whenever you need them. By ensuring that your script or code sends data that includes the hybrid job trigger's identifier, you don't need to update your script or code whenever a new job is started.

Now that you understand the different inspection methods, let us get started with some foundational aspects of DLP.

Cloud DLP terminology

Before we jump into defining Cloud DLP inspection templates, let us go over some important terminology that you will see in the templates.

DLP infoTypes

Information types, also known as infoTypes, are sensitive data kinds that Cloud DLP is preconfigured to scan and identify—for instance, US Social Security numbers, credit card numbers, phone numbers, zip codes, and names. Both built-in and custom InfoTypes are supported by Cloud DLP.

There is a detector for each infoType defined in Cloud DLP. To identify what to look for and how to transform findings, Cloud DLP employs infoType detectors in its scan configuration. When showing or reporting scan findings, infoType names are also used. Cloud DLP releases new infoType detectors and groups regularly. Call the Cloud DLP REST API's `infoTypes.list` method to receive the most up-to-date list of built-in infoTypes.

Please keep in mind that the built-in infoType detectors aren't always reliable. Google suggests that you evaluate your settings to ensure that they comply with your regulatory requirements.

Here are some examples of infoType detectors (this is not an exhaustive list):

- `ADVERTISING_ID`: Identifiers used by developers to track users for advertising purposes. These include Google Play **Advertising IDs**, Amazon **Advertising IDs**, Apple's **Identifier For Advertising** (**IDFA**), and Apple's **Identifier For Vendor** (**IDFV**).
- `AGE`: An age measured in months or years.
- `CREDIT_CARD_NUMBER`: A credit card number is 12 to 19 digits long. They are used for payment transactions globally.
- `CREDIT_CARD_TRACK_NUMBER`: A credit card track number is a variable-length alphanumeric string. It is used to store key cardholder information.
- `DATE`: A date. This infoType includes most date formats, including the names of common world holidays.
- `DATE_OF_BIRTH`: A date that is identified by context as a date of birth. Note that this is not recommended for use during latency-sensitive operations.
- `DOMAIN_NAME`: A domain name as defined by the DNS standard.
- `EMAIL_ADDRESS`: An email address identifies the mailbox that emails are sent to or from. The maximum length of the domain name is 255 characters, and the maximum length of the local part is 64 characters.
- `US_SOCIAL_SECURITY_NUMBER`: A United States **Social Security number** (**SSN**) is a nine-digit number issued to US citizens, permanent residents, and temporary residents. This detector will not match against numbers with all zeros in any digit group (that is, `000-##-####`, `###-00-####`, or `###-##-0000`), against numbers with 666 in the first digit group, or against numbers whose first digit is nine.

Now we have seen the overview of the Cloud DLP info types, let us understand the technical terms that you will frequently come across while working with Cloud DLP.

Data de-identification

De-identification is a term used to represent a process that removes personal information from a dataset. You will also come across the term *redaction* in relation to Cloud DLP. Redaction in general refers to the removal or masking of information, while de-identification refers to a process of changing it so you no longer identify the original text or are able to derive meaning from it. De-identified data is considered suitable for applications since it doesn't contain any PII that can be used to trace the results back to the user. There are several ways data can be de-identified:

- Masking sensitive data by partially or fully replacing characters with a symbol, such as an asterisk (*) or hash (#)
- Replacing each instance of sensitive data with a token, or surrogate, string
- Encrypting and replacing sensitive data using a randomly generated or pre-determined key

Cloud DLP supports the following methods of de-identification:

- **Masking**: Replaces the original data with a specified character, either partially or completely.
- **Replacement**: Replaces original data with a token or the name of the infoType if detected.
- **Date shifting**: Date-shifting techniques randomly shift a set of dates but preserve the sequence and duration of a period of time. Shifting dates is usually done in relation to an individual or an entity. That is, each individual's dates are shifted by an amount of time that is unique to that individual.
- **Generalization and bucketing**: Generalization is the process of taking a distinguishing value and abstracting it into a more general, less distinguishing value. Generalization attempts to preserve data utility while also reducing the identifiability of the data. One common generalization technique that Cloud DLP supports is bucketing. With bucketing, you group records into smaller buckets in an attempt to minimize the risk of an attacker associating sensitive information with identifying information. Doing so can retain meaning and utility, but it will also obscure the individual values that have too few participants.
- **Pseudonymization**: This is a de-identification technique that replaces sensitive data values with cryptographically generated tokens:
    - **Two-way tokenization pseudonymization**: Replaces the original data with a token that is deterministic, preserving referential integrity. You can use the token to join data or use the token in aggregate analysis. You can reverse or de-tokenize the data using the same key that you used to create the token. There are two methods for two-way tokenization:

- **Deterministic Encryption (DE) using AES-SIV: Advanced Encryption Standard-Synthetic Initialization Vector (AES-SIV)** is a cryptographic mode of operation designed to provide secure and authenticated encryption of data. It is based on the **Advanced Encryption Standard (AES)** block cipher and is intended for applications that require both data confidentiality and data integrity. It is commonly used for protecting data stored in databases, files, or other forms of persistent storage. AES-SIV also provides a way to protect data against replay attacks and can be used to construct secure communication protocols. Using AES-SIV, an input value is replaced with a value that has been encrypted using the AES-SIV encryption algorithm with a cryptographic key, encoded using base64, and then prepended with a surrogate annotation, if specified. This method produces a hashed value, so it does not preserve the character set or the length of the input value. Encrypted, hashed values can be re-identified using the original cryptographic key and the entire output value, including surrogate annotation. Learn more about the format of values tokenized using AES-SIV encryption.
- **Format Preserving Encryption (FPE) with Flexible Format-Preserving Encryption (FFX)**: This is an encryption technique that encrypts data while preserving the format of the original data. This means that FFX encryption can be used to encrypt data such as credit card numbers, SSNs, and other sensitive data without changing the format of the original data. FFX is used to protect data while it is stored or transmitted and to prevent unauthorized access to the data. Using FPE-FFX, an input value is replaced with a value that has been encrypted using the FPE-FFX encryption algorithm with a cryptographic key, and then prepended with a surrogate annotation, if specified. By design, both the character set and the length of the input value are preserved in the output value. Encrypted values can be re-identified using the original cryptographic key and the entire output value, including the surrogate annotation.

- **One-way tokenization using cryptographic hashing pseudonymization**: An input value is replaced with a value that has been encrypted and hashed using **Hash-Based Message Authentication Code Secure Hash Algorithm** 256 or **HMAC-SHA56** on the input value with a cryptographic key. The hashed output of the transformation is always the same length and can't be re-identified. Learn more about the format of values tokenized using cryptographic hashing.

Refer to the following table for a comparison of these three pseudonymization methods.

	Deterministic encryption using AES-SIV	**Format preserving encyption**	**Cryptographic hashing**
Encryption type	AES-SIV	FPR-FFX	HMAC-SHA256
Supported input values	At least 1 char long; no character set limitations.	At least 2 chats long; must be encoded as ASCII.	Must be a string or an integer value.
Surrogate annotation	Optional	Optional	N/A
Context tweak	Optional	Optional	N/A
Character set and length preserved	No	Yes	No
Reversible	Yes	Yes	No
Referential integrity	Yes	Yes	Yes

Table 10.1 – Comparison of the pseudonymization methods

Method selection

Choosing the best de-identification method can vary based on your use case. For example, if a legacy app is processing the de-identified records, then format preservation might be important. If you're dealing with strictly formatted 10-digit numbers, FPE preserves the length (10 digits) and character set (numeric) of an input for legacy system support.

However, if strict formatting isn't required for legacy compatibility, as is the case for values in the cardholder's name column, then DE is the preferred choice because it has a stronger authentication method. Both FPE and DE enable the tokens to be reversed or de-tokenized. If you don't need de-tokenization, then cryptographic hashing provides integrity but the tokens can't be reversed.

Other methods—such as masking, bucketing, date-shifting, and replacement—are good for values that don't need to retain full integrity. For example, bucketing an age value (for example, 27) to an age range (20-30) can still be analyzed while reducing the uniqueness that might lead to the identification of an individual.

Token encryption keys

A cryptographic key, also known as a token encryption key, is necessary for cryptographic de-identification transformations. The same token encryption key that is used to de-identify the original value is also used to re-identify it. This book does not cover the secure development and maintenance of token encryption keys.

However, there are a few key aspects to keep in mind that will be applied later in the lessons:

- In the template, avoid using plaintext keys. Instead, build a wrapped key with Cloud KMS.
- To limit the danger of keys being compromised, use different token encryption keys for each data element.
- Token encryption keys should be rotated. Although the wrapped key can be rotated, the token encryption key cannot be rotated because it compromises the tokenization's integrity. You must re-tokenize the entire dataset when the key is rotated.

Now that we are familiar with the basics of de-identification and which method is better suited for our use case, let us walk through how to create an inspection template. Later we will look at how to use de-identification.

Remember that Cloud DLP requires an inspection of the data before de-identification can be performed. The Cloud DLP inspection process identifies the infoTypes that further can be used in de-identification. The inspection step is *not* optional.

Creating a Cloud DLP inspection template

The first step in using classification capabilities is to create an inspection template. The inspection template will store all the data classification requirements:

1. In the Cloud console, open **Cloud DLP**.
2. From the **CREATE** menu, choose **Template**.

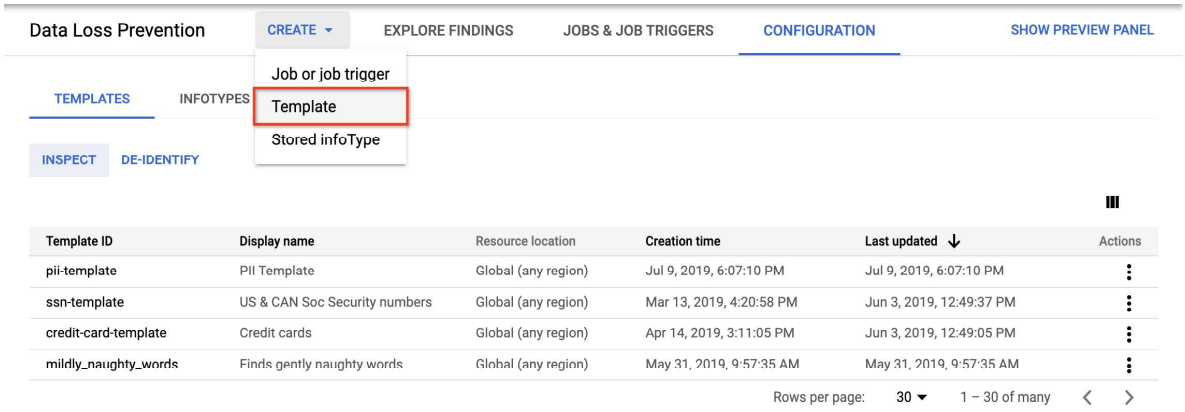

Figure 10.4 – Creating a DLP inspection template

3. Alternatively, click the following button: **Create new template**.

This page contains the following sections:

- **Define template**
- **Configure detection**

Defining the template

Under **Define template**, enter an identifier for the inspection template. This is how you'll refer to the template when you run a job, create a job trigger, and so on. You can use letters, numbers, and hyphens. If you want, you can also enter a more human-friendly display name, as well as a description to better remember what the template does.

Configuring detection

Next, you configure what Cloud DLP detects in your content by choosing an infoType and other options.

Under **InfoTypes**, choose the infoType detectors that correspond to a data type you want to scan for. You can also leave this field blank to scan for all default infoTypes. More information about each detector is provided in the *Further reading* section at the end of this chapter.

You can also add custom infoType detectors in the **Custom infoTypes** section, and customize both built-in and custom infoType detectors in the **Inspection rulesets** section.

Custom infoTypes

Be aware that any custom infoType detector you create here is specific to this workflow and can't be reused elsewhere. The one exception is *Stored infoType*, which requires that you create the stored custom infoType detector before specifying it here.

To add a custom infoType detector, do the following:

1. Click **Add custom infoType**.
2. Choose the type of custom infoType detector you want to create:
    - **Words or phrases**: Matches on one or more words or phrases that you enter into the field. Use this custom infoType when you have just a few words or phrases to search for. Give your custom infoType a name, and then type the word or phrase you want Cloud DLP to match. To search for multiple words or phrases, press *Enter* after each one.
    - **Dictionary path**: Searches your content for items in a list of words and phrases. The list is stored in a text file in Cloud Storage. Use this custom infoType when you have anywhere from a few to several hundred thousand words or phrases to search for. This method is also useful if your list contains sensitive elements and you don't want to store them inside of a job or template. Give your custom infoType a name, and then, under **Dictionary location**, enter or browse to the Cloud Storage path where the dictionary file is stored.
    - **Regular expression** (**regex**): Matches content based on a regular expression. Give your custom infoType a name, and then, in the **Regex** field, enter a regex pattern to match words and phrases. See the supported regex syntax in Google Cloud documentation at `https://packt.link/dRR06`.
    - **Stored infoType**: This option adds a stored custom dictionary detector, which is a kind of dictionary detector that is built from either a large text file stored in Cloud Storage or a single column of a BigQuery table. Use this kind of custom infoType when you have anywhere from several hundred thousand to tens of millions of words or phrases to search for. Be aware that this is the only option in this menu for which you must have already created the stored infoType to use it. Give your custom infoType a name (different from the name you gave the stored infoType), and then, in the **Stored infoType** field, enter the name of the stored infoType.
3. Click **Add custom infoType** again to add additional custom infoType detectors.

Now let us see how to use rulesets to customize the infoTypes to match your requirements. This helps you to avoid false positives.

Inspection rulesets

Inspection rulesets allow you to customize both built-in and custom infoType detectors using context rules. The two types of inspection rules are as follows:

- **Exclusion rules**, which help exclude false or unwanted findings
- **Hotword rules**, which help adjust the likelihood of the finding based on how near the hotword is to the finding

To add a new ruleset, first, specify one or more built-in or custom infoType detectors in the infoTypes section. These are the infoType detectors that your rulesets will be modifying. Then, do the following:

1. Click in the **Choose infoTypes** field. The infoType or infoTypes you specified previously will appear below the field in a menu, as shown in *Figure 10.5*.

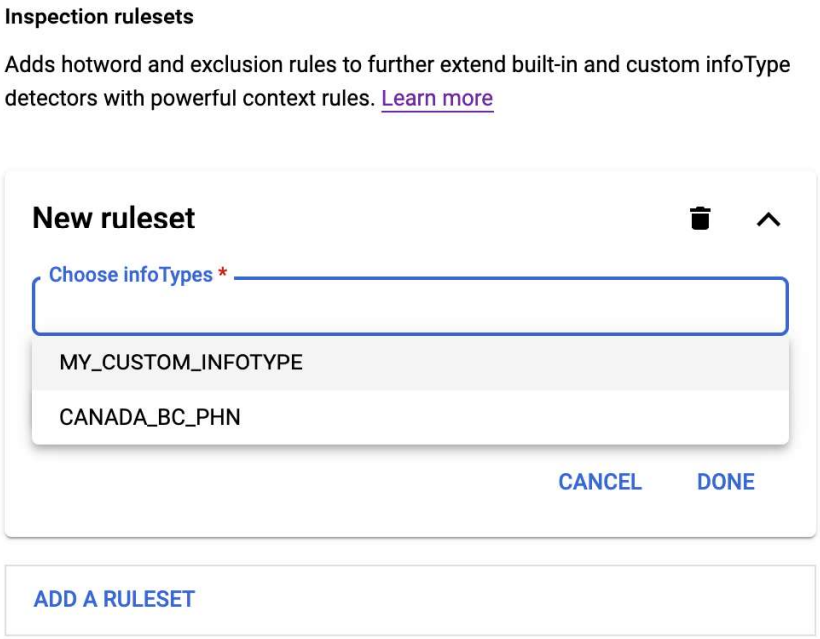

Figure 10.5 – Choosing infoTypes for Inspection rulesets

2. Choose an infoType from the menu, and then click **Add rule**. A menu appears with two options: **Hotword rule** and **Exclusion rule**.

   For hotword rules, choose **Hotword rule**. Then, do the following:

   I. In the **Hotword** field, enter a regular expression that Cloud DLP should look for.

   II. From the **Hotword proximity** menu, choose whether the hotword you entered is found before or after the chosen infoType.

   III. In **Hotword distance from infoType**, enter the approximate number of characters between the hotword and the chosen infoType.

   IV. In **Confidence level adjustment**, choose whether to assign matches to a fixed likelihood level or to increase or decrease the default likelihood level by a certain amount.

   For exclusion rules, choose **Exclusion rules**. Then, do the following:

   V. In the **Exclude** field, enter a regular expression (regex) that Cloud DLP should look for.

   VI. From the **Matching type** menu, choose one of the following:

   - **Full match**: The finding must completely match the regex.
   - **Partial match**: A substring of the finding can match the regex.
   - **Inverse match**: The finding doesn't match the regex.

You can add additional hotword or exclusion rules and rulesets to further refine your scan results.

Confidence threshold

Every time Cloud DLP detects a potential match for sensitive data, it assigns it a likelihood value on a scale from `Very unlikely` to `Very likely`. When you set a likelihood value here, you are instructing Cloud DLP to only match data that corresponds to that likelihood value or higher.

The default value of `Possible` is sufficient for most purposes. If you routinely get matches that are too broad, move the slider up. If you get too few matches, move the slider down.

When you're done, click **Create** to create the template. The template's summary information page appears.

Let us discuss some of the best practices for inspection.

Best practices for inspecting sensitive data

There are several things that you need to consider before starting an inspection. We will go over them now:

- **Identify and prioritize scanning**: It's important to identify your resources and specify which have the highest priority for scanning. When just getting started, you may have a large backlog of data that needs classification, and it'll be impossible to scan it all immediately. Choose data initially that poses the highest risk—for example, data that is frequently accessed, widely accessible, or unknown.
- **Reduce latency**: Latency is affected by several factors: the amount of data to scan, the storage repository being scanned, and the type and number of infoTypes that are enabled. To help reduce job latency, you can try the following:
  - Enable sampling.
  - Avoid enabling infoTypes you don't need. While useful in certain scenarios, some infoTypes—including `PERSON_NAME`, `FEMALE_NAME`, `MALE_NAME`, `FIRST_NAME`, `LAST_NAME`, `DATE_OF_BIRTH`, `LOCATION`, `STREET_ADDRESS`, and `ORGANIZATION_NAME`—can make requests run much more slowly than requests that do not include them.
  - Always specify infoTypes explicitly. Do not use an empty infoTypes list.
  - Consider organizing the data to be inspected into a table with rows and columns, if possible, to reduce network round trips.
- **Limit the scope of your first scans**: For the best results, limit the scope of your first scans instead of scanning all of your data. Start with a few requests. Your findings will be more meaningful when you fine-tune what detectors to enable and what exclusion rules might be needed to reduce false positives. Avoid turning on all infoTypes if you don't need them all, as false positives or unusable findings may make it harder to assess your risk. While useful in certain scenarios, some infoTypes such as `DATE`, `TIME`, `DOMAIN_NAME`, and `URL` detect a broad range of findings and may not be useful to turn on.
- **Limit the amount of content inspected**: If you are scanning BigQuery tables or Cloud Storage buckets, Cloud DLP includes a way to scan a subset of the dataset. This has the effect of providing a sampling of scan results without incurring the potential costs of scanning an entire dataset.

> **Note**
>
> Due to the need to scan the entire image for sensitive data, sampling is not supported for image file types. Any images scanned will be billed according to the size of the image file.

- **Inspect data on-premises or in other clouds**: If the data to be scanned resides *on-premises or outside of Google Cloud*, use the API methods `content.inspect` and `content.deidentify` to scan the content to classify findings and pseudonymized content without persisting the content outside of your local storage.

We have seen how to inspect data and best practices around data inspection. Let us now understand how to de-identify sensitive data.

Inspecting and de-identifying PII data

To de-identify sensitive data, use Cloud DLP's `content.deidentify` method.

There are three parts to a de-identification API call:

- **The data to inspect**: A string or table structure (`ContentItem` object) for the API to inspect.
- **What to inspect for**: Detection configuration information (`InspectConfig`) such as what types of data (or infoTypes) to look for, whether to filter findings that are above a certain likelihood threshold, whether to return no more than a certain number of results, and so on. Not specifying at least one infoType in an `InspectConfig` argument is equivalent to specifying all built-in infoTypes. Doing so is not recommended, as it can cause decreased performance and increased cost.
- **What to do with the inspection findings**: Configuration information (`DeidentifyConfig`) that defines how you want the sensitive data de-identified. This argument is covered in more detail in the following section.

The API returns the same items you gave it, in the same format, but any text identified as containing sensitive information according to your criteria is *de-identified*. Now let us look at various de-identification transformations.

De-identification transformations

We saw how to inspect data but many times you want to inspect and de-identify the data. Now we will see how to do that using various transformations supported by DLP. You must specify one or more transformations when you set the de-identification configuration (`DeidentifyConfig`). There are two categories of transformations:

- **InfoTypeTransformations**: Transformations that are only applied to values within the submitted text that are identified as a specific infoType.
- **RecordTransformations**: Transformations that are only applied to values within submitted tabular text data that are identified as a specific infoType, or on an entire column of tabular data.

Now let us go over each deidentification configuration.

replaceConfig

`replaceConfig` will replace any sensitive data with a string you specify.

redactConfig

`redactConfig` redacts a given value by removing it completely.

characterMaskConfig

Setting `characterMaskConfig` to a `CharacterMaskConfig` object partially masks a string by replacing a given number of characters with a fixed character. Masking can start from the beginning or end of the string.

cryptoHashConfig

Setting `cryptoHashConfig` to a `CryptoHashConfig` object performs pseudonymization on an input value by generating a surrogate value using cryptographic hashing.

This method replaces the input value with an encrypted *digest*, or hash value. The digest is computed by taking the SHA-256 hash of the input value. The method outputs a base64-encoded representation of the hashed output. Currently, only string and integer values can be hashed.

dateShiftConfig

Setting `dateShiftConfig` to a `DateShiftConfig` object performs date shifting on a date input value by shifting the dates by a random number of days.

Date-shifting techniques randomly shift a set of dates but preserve the sequence and duration of a period of time. Shifting dates is usually done in relation to an individual or an entity. You might want to shift all of the dates for a specific individual using the same shift differential but use a separate shift differential for each other individual.

Now that you have understood different methods of de-identification, let us go over a tutorial on how to do this in practice. Make sure you have a Google Cloud project ready to execute the steps covered next.

Tutorial: How to de-identify and tokenize sensitive data

Cloud DLP supports both reversible and non-reversible cryptographic methods. In order to re-identify content, you need to choose a reversible method. The cryptographic method described here is called deterministic encryption using **Advanced Encryption Standard in Synthetic Initialization Vector mode** (**AES-SIV**). We recommend this among all the reversible cryptographic methods that Cloud DLP supports because it provides the highest level of security.

In this tutorial, we're going to see how to generate a key to de-identify sensitive text into a cryptographic token. In order to restore (re-identify) that text, you need the cryptographic key that you used during de-identification and the token.

Before you begin, make sure you have the following roles in your Google Cloud project:

- Service account admin, to be able to create service accounts
- Service usage admin, to be able to enable services
- Security admin, to be able to grant roles

Once you have the right roles, follow these steps. The steps walk you through the process of creating an AES key to be able to de-identify sensitive text into a cryptographic token:

1. In the Google Cloud console, on the project selector page, select or create a new Google Cloud project.
2. Make sure that billing is enabled for your cloud project.
3. Enable the Cloud DLP and Cloud KMS APIs.
4. Create a service account by following the next steps:

    I. In the Cloud console, navigate to the **Create service account** page.

    II. In the **Service account name** field, enter a name. The Cloud console fills in the **Service account ID** field based on this name. In the **Service account description** field, enter a description.

    III. Click **Create** and continue.

    IV. To provide access to your project, grant the following role(s) to your service account: **Project > DLP Administrator**.

> **Note**
> In production environments, do not grant the Owner, Editor, or Viewer roles. Instead, grant a predefined role or custom role that meets your needs.

    V. Click **Continue**.

    VI. Click **Done** to finish creating the service account.

    Do not close your browser window. You will use it in the next step.

5. Create a service account key:

    I. In the Cloud console, click the email address for the service account that you just created.

    II. Click **Keys**.

    III. Click **Add key**, then click **Create new key**.

    IV. Click **Create**. A JSON key file is downloaded to your computer.

> **Note**
> It is against security practice to create a key and keep it forever. You should delete this key as soon as this tutorial is finished.

    V. Click **Close**.

6. Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the path of the JSON file that contains your service account key. This variable only applies to your current shell session, so if you open a new session, set the variable again.

Now that you have the right service account, we will create a KMS key that will be used to wrap the AES key used for the actual encryption of the DLP token.

Step 1: Creating a key ring and a key

Before you start this procedure, decide where you want Cloud DLP to process your de-identification and re-identification requests. When you create a Cloud KMS key, you must store it either globally or in the same region that you will use for your Cloud DLP requests. Otherwise, the Cloud DLP requests will fail.

You can find a list of supported locations in Cloud DLP locations. Take note of the name of your chosen region (for example, `us-west1`).

This procedure uses global as the location for all API requests. If you want to use a different region, replace global with the region name.

Create a key ring:

```
gcloud kms keyrings create "dlp-keyring" --location "global"
```

Create a key:

```
gcloud kms keys create "dlp-key" --location "global"  --keyring
"dlp-keyring" --purpose "encryption
```

List your key ring and key:

```
gcloud kms keys list --location "global" --keyring "dlp-keyring"
```

You will get the following output:

```
NAME: projects/PROJECT_ID/locations/global/keyRings/dlp-keyring/
cryptoKeys/dlp-key
PURPOSE: ENCRYPT_DECRYPT
ALGORITHM: GOOGLE_SYMMETRIC_ENCRYPTION
PROTECTION_LEVEL: SOFTWARE
LABELS:
PRIMARY_ID: 1
PRIMARY_STATE: ENABLED
```

In this output, `PROJECT_ID` is the ID of your project.

The path under `NAME` is the full resource name of your Cloud KMS key. Take note of it because the de-identify and re-identify requests require it.

Step 2: Creating a base64-encoded AES key

This section describes how to create an **Advanced Encryption Standard** (**AES**) key and encode it in base64 format. This key shall be used to encrypt the actual sensitive data. As you can see, you have complete control over the generation and maintenance of this key.

> **Note**
>
> These steps use the `openssl` and `base64` commands, but there are a few other ways to perform this task based on your security policies.

Now use the following command to create a 256-bit key in the current directory:

```
openssl rand -out "./aes_key.bin" 32
```

The `aes_key.bin` file is added to your current directory.

Encode the AES key as a `base64` string:

```
base64 -i ./aes_key.bin
```

You will get an output similar to the following:

```
uEDo6/yKx+zCg2cZ1DBwpwvzMVNk/c+jWs7OwpkMc/s=
```

> **Warning**
> Do not use this example key to protect actual sensitive workloads. This key is provided only to serve as an example. Because it's shared here, this key is not safe to use.

Step 3: Wrapping the AES key using the Cloud KMS key

This section describes how to use the Cloud KMS key that you created in *Step 1* to wrap the base64-encoded AES key that you created in *Step 2: Creating a base64-encoded AES key*.

To wrap the AES key, use `curl` to send the following request to the Cloud KMS API `projects.locations.keyRings.cryptoKeys.encrypt`:

```
curl "https://cloudkms.googleapis.com/v1/projects/PROJECT_ID/
locations/global/keyRings/dlp-keyring/cryptoKeys/dlp-key:encrypt"
--request "POST"
  --header "Authorization:Bearer $(gcloud auth application-default
print-access-token)" --header "content-type: application/json" --data
"{\"plaintext\": \"BASE64_ENCODED_AES_KEY\"}"
```

Replace the following:

- `PROJECT_ID`: The ID of your project.
- `BASE64_ENCODED_AES_KEY`: The base64-encoded string returned in *Step 2: Creating a base64-encoded AES key*.

The response that you get from Cloud KMS is similar to the following JSON:

```
{
  "name": "projects/PROJECT_ID/locations/global/keyRings/dlp-keyring/
cryptoKeys/dlp-key/cryptoKeyVersions/1",
  "ciphertext":
"CiQAYuuIGo5DVaqdE0YLioWxEhC8LbTmq7Uy2G3qOJlZB7WXBw0SSQAj
dwP8ZusZJ3Kr8GD9W0vaFPMDksmHEo6nTDaW/
j5sSYpHa1ym2JHk+lUgkC3Zw5bXhfCNOkpXUdHGZKou1893O8BDby/82HY=",
  "ciphertextCrc32c": "901327763",
  "protectionLevel": "SOFTWARE"
}
```

In this output, `PROJECT_ID` is the ID of your project.

Take note of the value of ciphertext in the response that you get. That is your wrapped AES key.

Step 4: Sending a de-identify request to the Cloud DLP API

This section describes how to de-identify sensitive data in text content.

To complete this task, you need the following:

- The full resource name of the Cloud KMS key that you created in *Step 1: Creating a key ring and a key*
- The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*

To de-identify sensitive data in text content, follow these steps:

1. Create a JSON request file with the following text:

```
{
  "item": {
    "value": "My name is Alicia Abernathy, and my email address
is aabernathy@example.com."
  },
  "deidentifyConfig": {
    "infoTypeTransformations": {
      "transformations": [
        {
          "infoTypes": [
            {
              "name": "EMAIL_ADDRESS"
            }
          ],
          "primitiveTransformation": {
            "cryptoDeterministicConfig": {
              "cryptoKey": {
                "kmsWrapped": {
                  "cryptoKeyName": "projects/PROJECT_ID/
locations/global/keyRings/dlp-keyring/cryptoKeys/dlp-key",
                  "wrappedKey": "WRAPPED_KEY"
                }
              },
              "surrogateInfoType": {
                "name": "EMAIL_ADDRESS_TOKEN"
              }
            }
          }
        }
      ]
    }
```

```
    },
    "inspectConfig": {
      "infoTypes": [
        {
          "name": "EMAIL_ADDRESS"
        }
      ]
    }
  }
```

2. Replace the following:

- `PROJECT_ID`: The ID of your project.
- `WRAPPED_KEY`: The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*.

  Make sure that the resulting value of `cryptoKeyName` forms the full resource name of your Cloud KMS key.

3. Save the file as `deidentify-request.json`.

Step 5: Sending a de-identity request to the Cloud DLP API

Now let us send the de-identification request to the DLP API based on the file you created in the previous step:

```
curl -s -H "Authorization: Bearer $(gcloud auth application-
default print-access-token)" -H "Content-Type: application/json"
https://dlp.googleapis.com/v2/projects/dlp-tesing/locations/global/
content:deidentify -d @deidentify-request.json
```

You should see output similar to this:

```
{
  "item": {
    "value": "My name is Alicia Abernathy, and my email
address is EMAIL_ADDRESS_TOKEN(52):ARa5jvGRxjop/UOzU9DZQAlCT/
yoTOjcOws7I/2IrzxrZsnlnjUB."
  },
  "overview": {
    "transformedBytes": "22",
    "transformationSummaries": [
      {
        "infoType": {
          "name": "EMAIL_ADDRESS"
        },
```

```
        "transformation": {
          "cryptoDeterministicConfig": {
            "cryptoKey": {
              "kmsWrapped": {
                "wrappedKey": "CiQAolK1/0r6aNktZPVngvs2ml/
ZxWAMXmjssvZgzSTui4keEgQSSQBaC4itVweyjdz5vdYFO3k/gh/
Kqvf7uEGYkgmVF98ZIbSffI3QRzWtR6zwLK8ZpXaDuUaQRgOuhMZJR2jf9Iq2f68aG0y
WKUk=",
                "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key"
              }
            },
            "surrogateInfoType": {
              "name": "EMAIL_ADDRESS_TOKEN"
            }
          }
        },
        "results": [
          {
            "count": "1",
            "code": "SUCCESS"
          }
        ],
        "transformedBytes": "22"
      }
    ]
  }
}
```

A couple of things to note in this output are as follows:

- `wrappedKey` is the KMS-wrapped key used to encrypt the token.
- `EMAIL_ADDRESS_TOKEN` is the encrypted token of the sensitive data (email address in our example). This is the data that you will store as well as the wrapped key, so if needed you can re-identify it later. We will see how to do that in the next step.

Step 6: Sending a re-identify request to the Cloud DLP API

This section describes how to re-identify tokenized data that you de-identified in the previous step.

To complete this task, you need the following:

- The full resource name of the Cloud KMS key that you created in *Step 1: Creating a key ring and a key*.
- The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*.
- The token that you received in *Step 4: Sending a de-identify request to the Cloud DLP API*.

To re-identify tokenized content, follow these steps:

1. Create a JSON request file with the following text:

```
{
  "reidentifyConfig":{
    "infoTypeTransformations":{
      "transformations":[
        {
          "infoTypes":[
            {
              "name":"EMAIL_ADDRESS_TOKEN"
            }
          ],
          "primitiveTransformation":{
            "cryptoDeterministicConfig":{
              "cryptoKey":{
              "kmsWrapped": {
                "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key",
                "wrappedKey": "WRAPPED_KEY"
              }
            },
              "surrogateInfoType":{
                "name":"EMAIL_ADDRESS_TOKEN"
              }
            }
          }
        }
      ]
    }
  },
  "inspectConfig":{
```

```
    "customInfoTypes":[
      {
        "infoType":{
          "name":"EMAIL_ADDRESS_TOKEN"
        },
        "surrogateType":{

        }
      }
    ]
  },
  "item":{
    "value": "My name is Alicia Abernathy, and my email address
is TOKEN."
  }
}
```

2. Replace the following:

   - `PROJECT_ID`: The ID of your project
   - `WRAPPED_KEY`: The wrapped key that you created in *Step 3*
   - `TOKEN`: The token that you received in *Step 4*—for example, `EMAIL_ADDRESS_TOKEN(52):AVAx2eIEnIQP5jbNEr2j9wLOAd5m4kpSBR/0jjjGdAOmryzZbE/q`

   Make sure that the resulting value of `cryptoKeyName` forms the full resource name of your Cloud KMS key.

3. Save the file as `reidentify-request.json`. Now we will use this file to send a request to DLP API via `curl`.

4. Use `curl` to make a `projects.locations.content.reidentify` request:

```
curl -s \
-H "Authorization: Bearer $(gcloud auth application-default
print-access-token)" \
-H "Content-Type: application/json" \
https://dlp.googleapis.com/v2/projects/PROJECT_ID/locations/
global/content:reidentify \
-d @reidentify-request.json
```

5. Replace `PROJECT_ID` with the ID of your project.

6. To pass a filename to `curl`, you use the `-d` option (for data) and precede the filename with an @ sign. This file must be in the same directory where you execute the `curl` command.

> **Note**
> This example request explicitly targets the global location. This is the same as calling the `projects.content.reidentify` API, which defaults to the global location.

The response that you get from Cloud DLP is similar to the following JSON:

```
{
 "item": {
   "value": "My name is Alicia Abernathy, and my email address
is aabernathy@example.com."
 },
 "overview": {
   "transformedBytes": "70",
   "transformationSummaries": [
     {
       "infoType": {
         "name": "EMAIL_ADDRESS"
       },
       "transformation": {
         "cryptoDeterministicConfig": {
           "cryptoKey": {
             "kmsWrapped": {
               "wrappedKey":
"CiQAYuuIGo5DVaqdE0YLioWxEhC8LbTmq7Uy2G3qOJlZB7WXBw0SSQAjdwP
8ZusZJ3Kr8GD9W0vaFPMDksmHEo6nTDaW/
j5sSYpHa1ym2JHk+lUgkC3Zw5bXhfCNOkpXUdHGZKou1893O8BDby/82HY=",
               "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key"
             }
           },
           "surrogateInfoType": {
             "name": "EMAIL_ADDRESS_TOKEN"
           }
         }
       },
       "results": [
         {
           "count": "1",
           "code": "SUCCESS"
         }
       ],
```

```
        "transformedBytes": "70"
      }
    ]
  }
}
```

In the `item` field, `EMAIL_ADDRESS_TOKEN` from the previous step is replaced with the plain email address from the original text.

You've just de-identified and re-identified sensitive data in text content using deterministic encryption. Now let us move on and understand a few use cases of workloads where DLP can be used.

DLP use cases

You have seen how DLP can be used to inspect, identify, and re-identify sensitive data in your workloads. Now let us understand various use cases to see how DLP fits:

- **Automatically discover sensitive data**: With Cloud DLP, you can automatically understand and manage your data risk across your entire enterprise. Continuous data visibility can assist you in making more informed decisions, managing and reducing data risk, and being compliant. Data profiling is simple to set up on the Cloud console, and there are no jobs or overhead to worry about, so you can focus on the results and your business.
- **Classify data across your enterprise**: Cloud DLP can help you categorize your data, whether it's on or off the cloud, and provide the insights you need to ensure correct governance, management, and compliance. Publish summary findings to other services such as Data Catalog, Security Command Center, Cloud Monitoring, and Pub/Sub or save comprehensive findings to BigQuery for study. In the Cloud console, you may audit and monitor your data, or you can use Google Data Studio or another tool to create custom reports and dashboards.

- **Protect sensitive data as you migrate to the cloud**: You can evaluate and classify sensitive data in both structured and unstructured workloads with Cloud DLP.

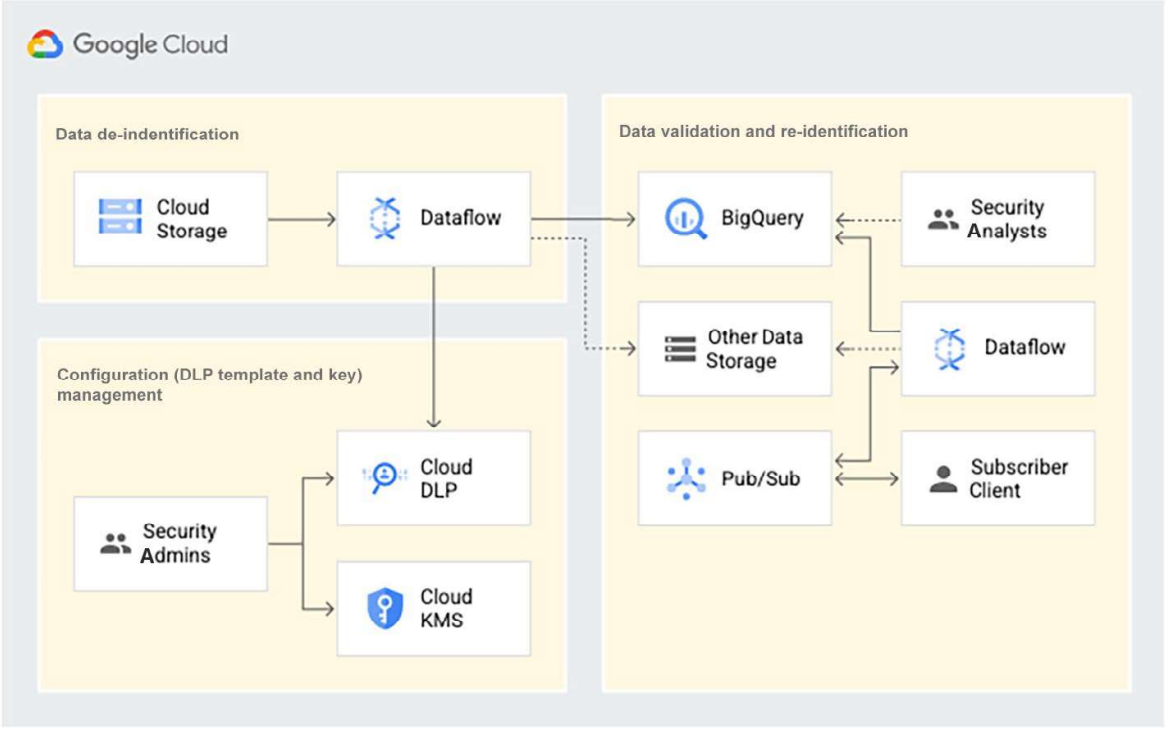

Figure 10.6 – DLP pattern for Dataflow and Data Fusion pipeline

As in *Figure 10.6*, here are a couple of patterns that you can employ to use Cloud DLP:

- Use Cloud DLP + Dataflow to tokenize data before loading it into BigQuery
- Use Cloud DLP + Cloud Data Fusion to tokenize data from Kafka Streams

By obfuscating the raw sensitive identifiers, de-identification techniques such as tokenization (pseudonymization) preserve the utility of your data for joining or analytics while decreasing the danger of handling the data. If you don't want the data to end up in the data warehouse, you can create a quarantine pipeline to send it to a sensitive dataset (with very limited permissions).

- **Use Cloud DLP in contact center chats to de-identify PII**: Cloud DLP can be used to mask any PII in customer support chats to make sure the agent doesn't get hold of any customer data.

We have seen some examples of workloads that you can use DLP for; now let us learn some best practices while using DLP.