SafeNet KMIP and Google Cloud Storage Integration Guide

Documentation Version: 20130719

Table of Contents

Chapter 1

Google Cloud Storage

This chapter presents an overview of Google Cloud Storage; steps to create a Google APIs console project; steps to activate the Google Cloud Storage service; and how the SafeNet KMIP integration with Google Cloud Storage is implemented.

Introduction

Google Cloud Storage is an Internet service to store data in the Google's cloud. Google Cloud Storage allows world-wide storing and retrieval of any amount of data and at any time. It provides a simple programming interface which enables developers to take advantage of Google's own reliable and fast networking infrastructure to perform data operations in a secure and cost effective manner. If expansion needs arise, developers can benefit from the scalability provided by Google's infrastructure.

Google Cloud Storage provides a range of programming languages to choose from when creating applications. These languages are supported by client libraries that allow applications to communicate with Google Cloud Storage. The libraries take care of the HTTP protocol details when using the Google Cloud Storage APIs.

Google Cloud Storage allows providing browser-based authenticated downloads to individual Google account holders. Authenticated browser-based downloads can be provided by first applying Google account-based ACLs to an object and then sending users a URL that is scoped to the object.

To use the Google Cloud Storage service, users need to activate Google Cloud Storage, create unique buckets to store data (objects), upload data, and control access. On Google Cloud Storage, the data is stored in plaintext.

Prerequisites

Before implementing SafeNet integration, make sure to activate the Google Cloud Storage service. The Google Cloud Storage service can be activated on a project-by-project basis.

Activating the Google Cloud Storage service requires a Google account and a Google APIs console project. At least one project must exist before the Google Cloud Storage service can be activated.

Creating Google APIs Console Project

A Google account is needed to create a Google APIs console project.

To create a Google APIs console:

1. Log on to the **Google APIs Console** (https://code.google.com/apis/console/). If no project already exists, a prompt appears to create a project.
2. Click **Create project...** A new project gets created.
3. Click the **Overview** tab. The Dashboard page displays Project Summary which includes project name, project number, a link to register project ID, and the owner name.
4. Click **Register...** next to Project ID. The Register Project ID dialog box appears.
5. In the **Project ID** field, enter an ID for the project.
6. Click **Check availability**.
7. Click **Choose this ID** if the entered ID is available. The selected project ID now appears next to Project ID under Project Summary.

Activating Google Cloud Storage Service

After creating a project, the Google Cloud Storage service needs to be activated.

To activate the Google Cloud Storage service:

1. Log on to the **Google APIs Console** (if needed). In case of multiple projects, enter the project to activate Google Cloud Storage for.
2. Click the **Services** tab.
3. In the **All Services** table, click **Off** next to Google Cloud Storage. The Google Cloud Storage service is turned on for the project. The Billing and Google Cloud Storage tabs are now visible in the left navigation pane.
4. Click the **Billing** tab.
5. Click **Enable Billing** to enable billing for the project.

Authorizing Requests with OAuth 2.0

Any requests to access private data must be authorized by an authenticated user who has access to that data.

To authorize private data requests:

1. Log on to the **Google APIs Console** (if needed). In case of multiple projects, enter the project to authorize requests for.
2. Click the **API Access** tab.

3 Click **Create an OAuth 2.0 client ID...** The Create Client ID dialog box appears.

4 Specify branding information including product name, product logo, URL to home page.

5 Click **Next**.

6 Select **Installed application** under **Application type**.

7 Select **Other** under **Installed application type**.

8 Click **Create client ID**. The entered branding information and generated Google client secrets (Client ID and Client secret) appear on the screen.

9 Click **Download JSON**. The downloaded `client_secrets.json` file is required while using the sample application. This file contains Google client secrets used to retrieve access token from Google. The retrieved access token is stored in the `credential.json` file in the working directory. The access token is used for Google authentication.

If needed, another client ID can be created by clicking **Create another client ID...**

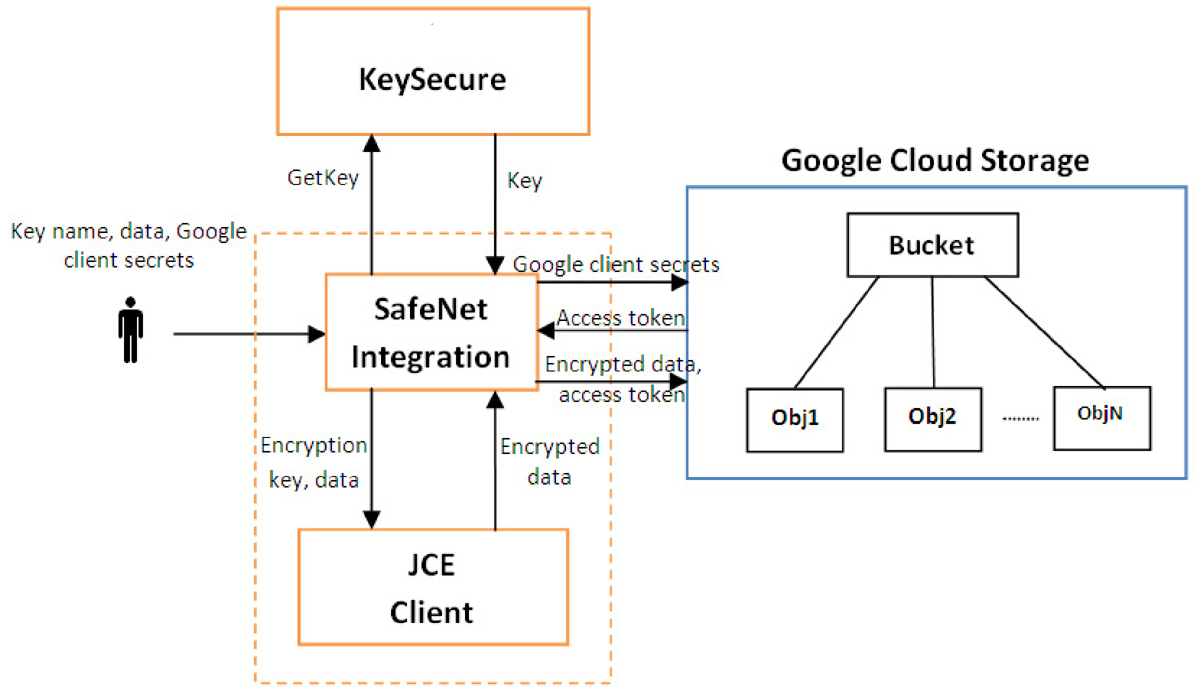
SafeNet Integration

SafeNet integrates Google Cloud Storage with the KeySecure server for key management services. With SafeNet integration, users can manage encryption keys on a KeySecure server. Encryption keys can be generated, managed, and stored on the KeySecure server.

Configuring SafeNet Integration

There are some configuration steps needed before using SafeNet integration. Certain parameters in the `IngrianNAE.properties` (also known as properties file) file need to be modified to define how the provider interacts with KeySecure. SSL setup is also needed for secure communication with KeySecure. For details on SSL setup, refer to Chapter 2, “Setting up SSL”.

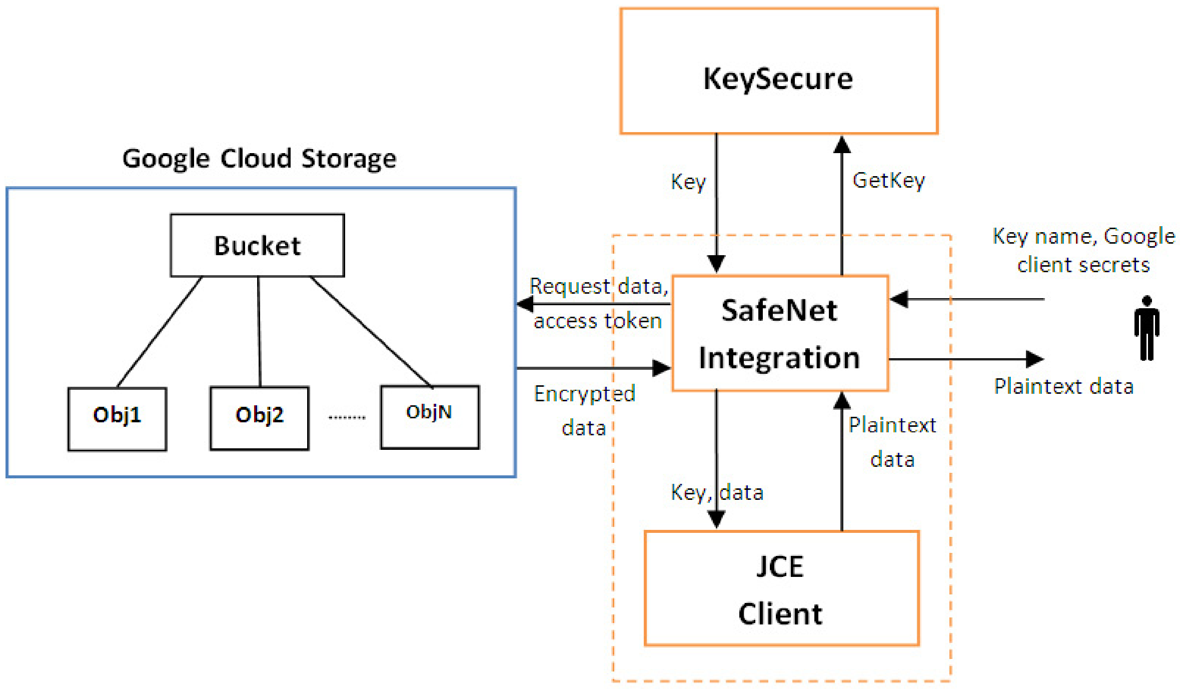
Uploading Data Using Keys Stored on KeySecure



With SafeNet integration, the encryption process is as follows:

1. User provides the key name, data, and Google client secrets to SafeNet integration.
2. SafeNet integration sends Google client secrets to retrieve access token from Google.
3. SafeNet integration connects to KeySecure, locates the key name on KeySecure, and fetches the key.
4. SafeNet integration sends the data and the encryption key to the JCE client.
5. JCE client encrypts the data using the encryption key.
6. JCE client sends the encrypted data to SafeNet integration.
7. SafeNet integration uploads the encrypted data to Google Cloud Storage under a specified bucket.

Downloading Data Using Keys Stored on KeySecure



With SafeNet integration, the decryption process is as follows:

1 User provides the key name and Google client secrets to SafeNet integration.

2 SafeNet integration connects to KeySecure, locates the key name on KeySecure, and fetches the key.

3 SafeNet integration requests the data from Google Cloud Storage using the already retrieved access token.

4 SafeNet integration retrieves encrypted data from Google Cloud Storage.

5 SafeNet integration sends the encrypted data and the key retrieved from KeySecure to the JCE client.

6 JCE client decrypts the encrypted data using the key received from SafeNet integration.

7 JCE client sends the decrypted data to SafeNet integration.

8 SafeNet integration sends the decrypted data to the user.

Chapter 2

Setting up SSL

This chapter provides an overview of SafeNet SSL and SSL with Client Certificate Authentication features.

SSL Configuration Procedures

This section describes the procedures you will follow when configuring SSL. It explains the following processes:

1. Creating a Local CA
2. Creating a Server Certificate Request on the Management Console
3. Signing a Server Certificate Request with a Local CA
4. Downloading the Local CA Certificate
5. Adding the CA Certificate to the Java Keystore

Creating a Local CA

To create a local CA:

1. Log on to the Management Console as an administrator with Certificate Authorities access control.
2. Navigate to the Create Local Certificate Authority section on the Certificate and CA Configuration page (Security, Local CAs).
3. Modify the fields as needed.
4. Select either *Self-signed Root CA* or *Intermediate CA Request* as the **Certificate Authority Type**.
5. Click **Create**.

Note: Only a local CA can sign certificate requests on KeySecure. If you are using a CA that does not reside on KeySecure, you *cannot* use the Management Console to sign certificate requests.

Important! Local CA certificates must be added to a trusted CA list to be recognized by the Cryptographic Key Server. Local CA certificates should be backed up for protection.

To add a local CA to a trusted CA list:

1. Navigate to the Certificate and CA Configuration page. (Security, Trusted CA Lists).
2. Select **Default** under Profile Name.
3. Click **Edit** under the Trusted Certificate Authority List section.
4. Select the local CA in the **Available CAs** list.

5 Click **Add**. The local CA now appears in the **Trusted CAs** list.

6 Click **Save**.

Creating a Server Certificate Request on the Management Console

To create a server certificate request on the Management Console:

1 Log on to the Management Console as an administrator with Certificates access control.

2 Navigate to the Create Certificate Request section of the Certificate and CA Configuration page (Security, SSL Certificates) and modify the fields as needed.

3 Click **Create Certificate Request**. This creates the certificate request and places it in the Certificate List section of the Certificate and CA Configuration page. The new entry shows that the **Certificate Purpose** is *Certificate Request* and that the **Certificate Status** is *Request Pending*.

Signing a Server Certificate Request with a Local CA

To sign a server certificate request with a local CA:

1 Log on to the Management Console as an administrator with Certificates and Certificate Authorities access control.

2 Navigate to the Certificate List section on the Certificate and CA Configuration page (Security, SSL Certificates).

3 Select the certificate request and click **Properties**.

4 Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and footer (-----END CERTIFICATE REQUEST-----).

5 Navigate to the Local Certificate Authority List (Security, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.

6 Modify the fields as shown:

- **Sign with Certificate Authority** - Select the CA that signs the request.
- **Certificate Purpose** - Select *Server*.
- **Certificate Duration (days)** - Enter the life span of the certificate.
- **Certificate Request** - Paste all text from the certificate request, including the header and footer.

7 Click **Sign Request**. This will take you to the CA Certificate Information section.

8 Copy the actual certificate. The copied text must include the header (-----BEGIN CERTIFICATE-----) and footer (-----END CERTIFICATE-----).

9 Navigate back to the Certificate List section (Security, SSL Certificates). Select your certificate request and click **Properties**.

10 Click **Install Certificate**.

11 Paste the actual certificate in the Certificate Response text box. Click **Save**. The Management Console returns you to the Certificate List section. The section will now show that the **Certificate Purpose** is *Server* and that the **Certificate Status** is *Active*.

The certificate can now be used as the server certificate for the NAE Server.

Downloading the Local CA Certificate

To download a local CA certificate from KeySecure:

1 Log on to the Management Console as an administrator with Certificate Authorities access control.

2 Navigate to the Local Certificate Authority List section of the Certificates and CA Configuration page (Security, Local CAs).

3 Select the Local CA and click the **Download** button to download the file to your client.

Adding the CA Certificate to the Java Keystore

To add the CA certificate to the keystore:

1 Open a command prompt window on the client and navigate to the Java security directory. This is typically `<JRE-Home>/lib/security`.

2 Use the keytool utility to import the CA certificate by issuing the command below. This statement selects `cacerts` as the keystore. You create an alias for the CA at this time.

```
keytool -import -keystore cacerts -alias <CAAlias> -file <CertFileName.crt>
```

Here, `<CertFileName.crt>` represents the name of the certificate file with the path.

3 Enter the keystore password when prompted. By default, the password is “changeit”.

4 Indicate that the CA is trusted, when prompted.

The utility will then issue a message confirming that the certificate has been added to the keystore. For information about the keytool utility, please refer to Sun’s documentation at: http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html

SSL with Client Certificate Authentication Procedures

This section describes the procedures you will follow when configuring SSL with Client Certificate Authentication. It explains the following processes:

1 Generating a Client Certificate Request with Keytool

2 Signing a Certificate Request and Downloading the Certificate

3 Adding the CA and Client Certificates to the Java Keystore

Generating a Client Certificate Request with Keytool

Note: You cannot authenticate the client IP address if you use keytool to generate the client certificate request.

To generate a client certificate request:

1 Open a command prompt window on your client and navigate to the Java security directory (`<Java_Home>\lib\security`).

2 Generate a public/private key pair by issuing the command below. You create an alias for the key pair at this time.

```
keytool -keystore <KeystoreName> -genkey -alias <KeyPairAlias> -keyalg RSA
```

The key generation process will then request the following data:

- A keystore password.
- The distinguished name.

  This is a series of fields whose values are incorporated into the certificate request. These fields include country name, state or province name, city or locality name, organization name, organizational unit name, and the users first and last name.

  Important! The user name specified here must exist on KeySecure. If the user does not exist, create it on the User & Group Configuration page. (Security, Local Authentication, Local Users & Groups)
- The key password.

  The certificate password *must* be the same as the keystore password. You can simply press Return to set the password. You need not retype the keystore password.

  The sample output looks similar to the following:

```
C:\Program Files\Java\jdk1.7.0_07\jre\lib\security>keytool -keystore ca-
certs -genkey -alias KeyPairAlias1 -keyalg RSA

Enter keystore password:

What is your first and last name?

  [Unknown]:  AB

What is the name of your organizational unit?

  [Unknown]:  ENGG

What is the name of your organization?

  [Unknown]:  SFNT

What is the name of your City or Locality?

  [Unknown]:  NOIDA

What is the name of your State or Province?

  [Unknown]:  UP
```

```
What is the two-letter country code for this unit?
  [Unknown]:  IN
Is CN=AB, OU=ENGG, O=SFNT, L=NOIDA, ST=UP, C=IN correct?
  [no]:  yes


Enter key password for <KeyPairAlias1>
        (RETURN if same as keystore password):
```

Important! The user name, **AB**, specified above must exist on KeySecure. If the user does not exist, create it on the User & Group Configuration page. (Security, Local Authentication, Local Users & Groups)

3 Create the certificate request by issuing the command below. Reference the Key Pair Alias you created above.

```
keytool -certreq -alias <KeyPairAlias> -file <CertReqFileName>
-keystore <KeystoreName>
```

You will now have a certificate request in the `<CertReqFileName>` file.

Signing a Certificate Request and Downloading the Certificate

This section describes how to sign a certificate request with a local CA and then download the certificate. You must download the certificate *immediately* after it is signed by the CA.

To sign a certificate request with a local CA:

1 Open the certificate request in a text editor.

2 Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and the footer (-----END CERTIFICATE REQUEST-----).

3 Log on to KeySecure as an administrator with Certificate Authorities access control.

4 Navigate to the Local Certificate Authority List (Security, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.

5 Modify the fields as shown:

- **Sign with Certificate Authority** – Select the CA that signs the request.
- **Certificate Purpose** – Select *Client*.
- **Certificate Duration (days)** – Enter the life span of the certificate.
- **Certificate Request** – Paste all text from the request, including the header and footer.

6 Click **Sign Request**. This will take you to the CA Certificate Information section.

7 Click the **Download** button to save the certificate on your local machine.

Adding the CA and Client Certificates to the Java Keystore

To add the client certificate to the Java keystore:

1. Open a command prompt window on your client and navigate to the Java security directory (`<Java_Home>\lib\security`).
2. Import the CA certificate that signed the client certificate using the command below. You create an alias for the CA at this time. When prompted, enter the keystore password and indicate that the CA is trusted.

   ```
   keytool -keystore <KeystoreName> -import -alias <CAAlias> -file
   <CertFileName.crt>
   ```

   The above step is required if a CA other than the Local CA is to be used.
3. Import the signed client certificate using the following command. Use the key pair alias you used to create the certificate request. When prompted, enter the keystore password.

   ```
   keytool -keystore <KeystoreName> -alias <KeyPairAlias> -import -file
   <CertFileName.crt>
   ```
4. Verify that the client certificate was properly imported by executing the following command. Reference the key pair alias you used above. The system should display the certificate.

   ```
   keytool -keystore <KeystoreName> -alias <KeyPairAlias> -list -v
   ```

Important! To enable Client Certificate Authentication, your keystore must have a copy of the CA certificate that signed the server certificate.

Updating the Properties File

After setting up the SSL, the SSL-related parameters must be updated in the properties file. The following parameters in the `IngrianNAE.properties` file need to be updated as follows:

- `NAE_IP.1=<IP address of the server.>`
- `KMIP_Port=<KMIP SSL port on the server.>`
- `Protocol=ssl`
- `Key_Store_Location=<path and name of keystore that contains a copy of the server’s local CA, the client certificate, and the CA that signed the client certificate.>`
- `Key_Store_Password=<keystore password>`
- `Client_Cert_Alias=<client certificate alias>`
- `Client_Cert_Passphrase=<client certificate password, if used>`

Chapter 3

Sample Application

A sample application, `SafeNetKmipGCSSample`, is included to demonstrate the data upload/download with and without SafeNet key management.

Extracting SafeNet Integration

Extracting the `SafeNetKMIPGCS.zip` file creates a `SafeNetKMIPGCS` folder. This folder contains the following items:

- `src` – Source file for the sample application.
- `bin` – Package and class file generated by compiling the source file.
- `lib` – Library (jar) dependencies, Google API, SafeNet KMIP library (jar), and the properties file.
- `doc` – JavaDoc html and resource files.
- `UserGuide` – SafeNet KMIP and Google Cloud Storage Integration Guide.
- `readme.txt` – Instructions to implement SafeNet integration.

Using Sample Application

1 Add the `lib` and `bin` folder to the `CLASSPATH` environment variable.

   For example, set `CLASSPATH=`
   `%CLASSPATH%;"D:\SafeNetKMIPGCS\lib\*;D:\SafeNetKMIPGCS\bin";`

2 Set the properties file `IngrianNAE.properties`. For details, refer to “Updating the Properties File” on page 12.

3 Copy the downloaded `client_secrets.json` file to the working directory.

4 Check the command usage for available options.

   a Navigate to the path where SafeNet integration is extracted.

   b Execute `java SafeNetKMIPGCS`. The command usage is displayed:

```
D:\SafeNetKMIPGCS\bin>java SafeNetKmipGCSSample
Usage:

-createENCKey <KeyName> <KeySize>
-createBucket <Project ID> <BucketName>
```

```
    -listBucket   <Project ID>
    -getBucket <BucketName>
    -deleteBucket <BucketName>
    -putObject <BucketName> <FileName> <ENCKey>*
    -getObject <BucketName> <ObjectName> <FilePath> <ENCKey>*
    -listObject <BucketName>
    -deleteObject <BucketName> <ObjectName>


    *ENCKey should be provided in case of encrypting data before uploading to
    Google storage.
```

5 Use the sample `SafeNetKmipGCSSample` with required parameters shown in the command usage above.

Using SafeNetKmipGCSSample

Use `SafeNetKmipGCSSample` to upload and download data from Google Cloud Storage with or without SafeNet KeySecure. All user data is stored in buckets in the form of objects. Symmetric keys can also be created on KeySecure using sample application `SafeNetKmipGCSSample`.

The following topics describe the tasks that can be performed using the sample application `SafeNetKmipGCSSample`.

Important! Performing operations on Google Cloud Storage requires access token for authentication. While using the sample application, the application may prompt to retrieve access token from Google (if the access token does not already exist). Allow the application to retrieve the access token. The retrieved access token is stored in the `credential.json` file downloaded at the working directory. The same access token will be used for all future authentications.

Creating Encryption Keys

To create an encryption key, execute:

```
java SafeNetKmipGCSSample -createENCKey <KeyName> <KeySize>
```

Where:

- `<KeyName>`: Name for the symmetric key.
- `<KeySize>`: Size for the symmetric key.

Managing Buckets

Creating Buckets

To create a bucket in a project, execute:

```
java SafeNetKmipGCSSample -createBucket <Project ID> <BucketName>
```

Where:

- `<Project ID>` – ID of the project in which the bucket is to be created. To view a project ID, click the **Overview** tab. The Dashboard page displays the project ID.
- `<BucketName>`: Name for the bucket.

Listing Buckets in a Project

To list buckets in a project, execute:

```
java SafeNetKmipGCSSample -listBucket   <Project ID>
```

Where:

- `<Project ID>` – ID of the project.

Viewing Properties of a Bucket

To view properties (metadata) of a bucket, execute:

```
java SafeNetKmipGCSSample -getBucket <BucketName>
```

Where:

- `<BucketName>` – Name of the bucket.

Deleting Buckets

Note: Only empty buckets can be deleted. Before deleting a bucket, delete all the objects stored in the bucket, as described in “Deleting Objects from a Bucket” on page 16.

To delete a bucket from a project, execute:

```
java SafeNetKmipGCSSample -deleteBucket <BucketName>
```

Where:

- `<BucketName>` – Name of the bucket to be deleted.

Managing Objects

Listing Objects in a Bucket

To list objects in a bucket, execute:

```
java SafeNetKmipGCSSample -listObject <BucketName>
```

Where:

- `<BucketName>` – Name of the bucket.

Uploading Objects to a Bucket

To upload an object to a bucket, execute:

```
java SafeNetKmipGCSSample -putObject <BucketName> <FileName> <ENCKey>*
```

Where:

- `<BucketName>` – Name of the bucket.
- `<FileName>` – Name and path of the file to be uploaded.
- `<ENCKey>` – Name of the key for encryption.

Downloading Objects from a Bucket

To download an object from a bucket, execute:

```
java SafeNetKmipGCSSample -getObject <BucketName> <ObjectName> <FilePath>
<ENCKey>*
```

Where:

- `<BucketName>` – Name of the bucket.
- `<ObjectName>` – Name of the object to be downloaded.
- `<FilePath>` – Path to store the downloaded object.
- `<ENCKey>` – Name of the key used for encryption.

Deleting Objects from a Bucket

To delete an object from a bucket, execute:

```
java SafeNetKmipGCSSample -deleteObject <BucketName> <ObjectName>
```

Where:

- `<BucketName>` – Name of the bucket.

- `<ObjectName>` – Name of the object to be deleted.

Use Cases

This section describes how data is uploaded to and downloaded from Google Cloud Storage in various real life scenarios.

Uploading/Downloading Data without KeySecure

This use case describes how data is uploaded to and downloaded from Google Cloud Storage without using SafeNet integration.

1. Uploading Data to Google Cloud Storage without KeySecure

This use case describes how the data is uploaded to Google Cloud Storage without using SafeNet integration.

To upload data to Google Cloud Storage, execute:

```
java SafeNetKmipGCSSample -putObject <BucketName> <FileName>
```

Where:

- `<BucketName>` – Name of the bucket.
- `<FileName>` – Name and path of the file to be uploaded.

2. Downloading Data from Google Cloud Storage without KeySecure

This use case describes how the data is downloaded from Google Cloud Storage without using SafeNet integration.

To download data from Google Cloud Storage, execute:

```
java SafeNetKmipGCSSample -getObject <BucketName> <ObjectName> <FilePath>
```

Where:

- `<BucketName>` – Name of the bucket.
- `<ObjectName>` – Name of the object to be downloaded.
- `<FilePath>` – Path to store the downloaded object.

Uploading/Downloading Data Using KeySecure

This use case describes how data is uploaded to and downloaded from Google Cloud Storage using SafeNet integration when the key is created on KeySecure.

1. Creating Symmetric Key on KeySecure

To create a symmetric key on KeySecure using SafeNet integration, execute the following command:

```
java SafeNetKmipGCSSample -createENCKey <KeyName> <KeySize>
```

Where:

- `<KeyName>` – Name for the symmetric key.
- `<KeySize>` – Size for the symmetric key.

2. Uploading Data to Google Cloud Storage Using KeySecure

This use case describes how the data is uploaded to Google Cloud Storage using SafeNet integration.

To upload data to Google Cloud Storage, execute:

```
java SafeNetKmipGCSSample -putObject <BucketName> <FileName> <ENCKey>*
```

Where:

- `<BucketName>` – Name of the bucket.
- `<FileName>` – Name and path of the file to be uploaded.
- `<ENCKey>` – Name of the key for encryption.

Note: *`ENCKey` is needed to encrypt the data before uploading it to Google Cloud Storage.

3. Downloading Data from Google Cloud Storage Using KeySecure

This use case describes how the data is downloaded from Google Cloud Storage using SafeNet integration.

To download data from Google Cloud Storage, execute:

```
java SafeNetKmipGCSSample -getObject <BucketName> <ObjectName> <FilePath>
<ENCKey>*
```

Where:

- `<BucketName>` – Name of the bucket.
- `<ObjectName>` – Name of the object to be downloaded.
- `<FilePath>` – Path to store the downloaded object.
- `<ENCKey>` – Name of the key used for encryption.

Note: *`ENCKey` is needed to decrypt the data before downloading it from Google Cloud Storage.