VCC-Case Study

Group-40

Team Members:

ID	Name	Worked on below sub topic analysis
M23AID006	Aparna Pundir	Access Control & Identity Management
M23AID026	Nikita Krishna	Encryption & Key Management
M23AID053	Bhuvaneswari J	Security Monitoring & Incident Response

Enhancing Data Security in Cloud-Based Document Management: A Case Study on Encryption and Access Controls in a Financial Firm

Abstract:

In today's digital era, financial institutions handle vast amounts of sensitive documents daily, making secure document management a top priority. To address this need, we implement and test a simple cloud-based Document Management System (DMS) tailored for a financial firm using Google Cloud Platform (GCP). This system ensures not only efficient document handling but also enforces stringent security, compliance, and access controls. This case study explores how a mid-sized financial firm improved the security of its cloud-based document management system using Google Cloud Platform (GCP). The firm faced security concerns related to unauthorized data access, regulatory compliance, and data breaches. To mitigate these risks, they implemented encryption techniques and strict access control mechanisms using Google Cloud Storage, Key Management Service (KMS), and Identity and Access Management (IAM). The solution resulted in improvement in compliance adherence, reduction in unauthorized access incidents, and improved system efficiency. The findings highlight the effectiveness of cloud-native security solutions in protecting financial data.

Introduction

A mid-sized financial firm handling sensitive client data on its cloud-based document management system faced challenges such as:

- **Unauthorized access risks** due to improper access controls.
- **Regulatory compliance** (e.g., GDPR, PCI-DSS) requiring encryption and audit trails.
- **Potential data breaches** due to weak security policies.

To implement robust encryption and access control mechanisms in **Google Cloud Platform (GCP)** to enhance data security, ensure compliance, and minimize unauthorized access.

Scope:

- **Domain:** Financial sector document management system.
- **Cloud Services Analyzed:** Google Cloud Storage, IAM, Cloud KMS, Cloud Audit Logs.

- **Security Measures:** Encryption (AES-256, Google-managed keys), IAM role-based access control, and security auditing.

Literature Review

Document Management Systems (DMS) in the cloud offer financial institutions scalability, cost-efficiency, and remote accessibility. According to Sharma & Choudhury (2018)**,** cloud-based DMS adoption in finance has increased due to the need for digitization, regulatory compliance, and disaster recovery. Google Cloud's infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) offerings have made it easier for businesses to migrate document workflows to the cloud without compromising on security and performance. As per Zhang et al. (2010)**,** cloud systems must implement role-based access control (RBAC) and attribute-based access control (ABAC) to ensure data isolation. GCP's IAM model supports both approaches, enabling organizations to assign specific permissions to users, service accounts, and groups. Studies such as Ferraiolo et al. (2007) emphasize the importance of least privilege and audit logging for compliance and traceability in financial workflows. According to NIST Special Publication 800-57, encryption must be implemented for both data at rest and in transit. Google Cloud Platform natively supports these standards and enhances security through Customer-Managed Encryption Keys (CMEK) **and** Customer-Supplied Encryption Keys (CSEK)**.** The work by Chen et al. (2012) highlights the risks of key exposure and stresses the significance of automated key rotation, which GCP's Cloud KMS provides by default to mitigate cryptographic compromise. Gartner (2021) reports that organizations with integrated security monitoring and automated incident response reduce the average breach impact by over 50%. Tools like Google Cloud Logging**,** Cloud Monitoring, and Security Command Center allow real-time alerts based on anomaly detection and unauthorized access patterns. Research by Liu & Chen (2016) supports using log-based metrics to build alerting systems that act as an early warning mechanism in cloud environments. As observed by Kshetri (2013)**,** cloud platforms that provide transparent auditing, key lifecycle management, and access policies are better suited to handle the rigorous compliance requirements of the financial sector.

Gaps Identified:

- Many financial firms struggle with **automated encryption key management**.
- **Real-time security monitoring** and auditing remain a challenge.

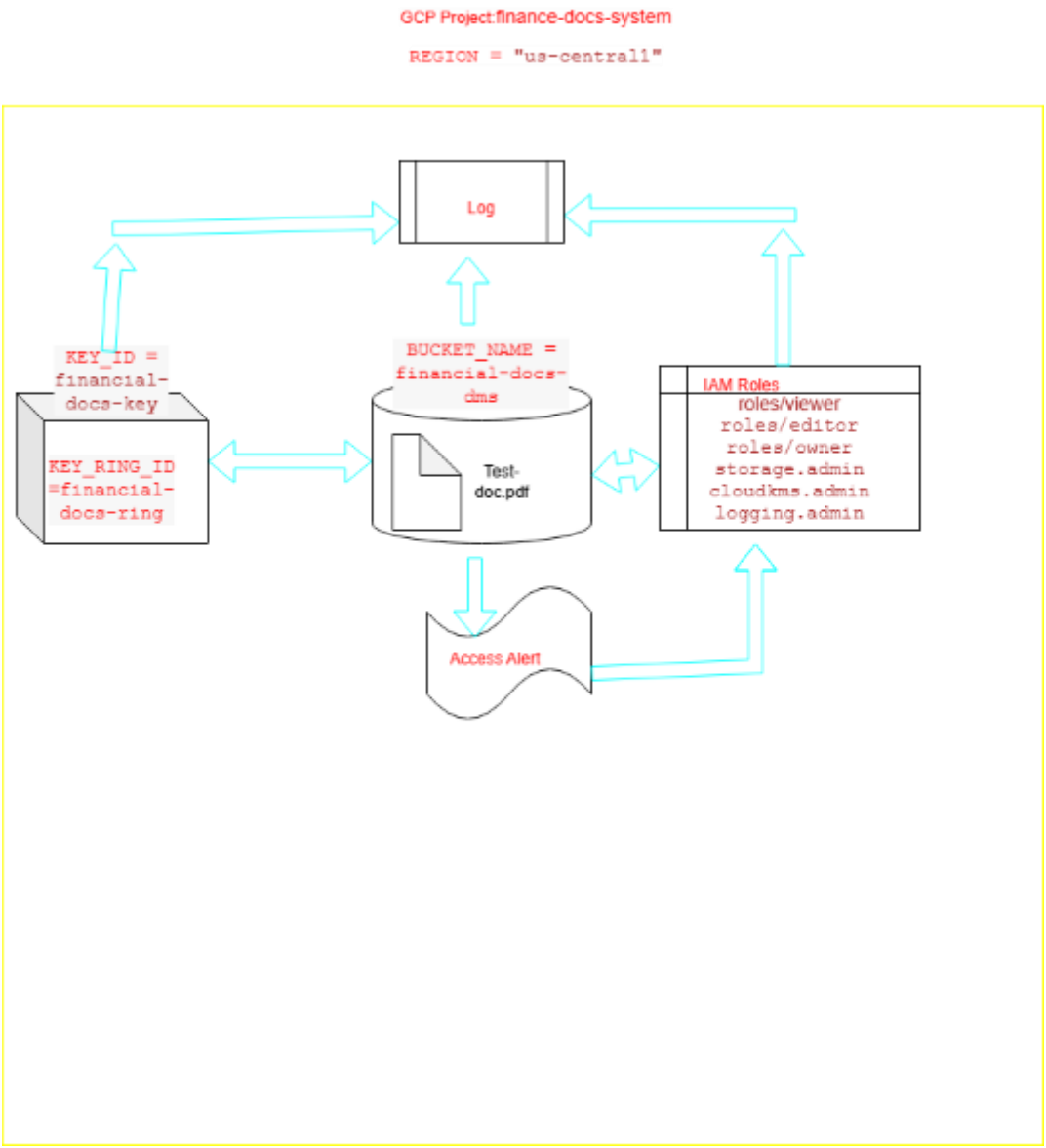
This case study addresses these gaps by integrating encryption with automated key rotation and implementing real-time access monitoring.

Methodology

Technologies Used:

- **Cloud Platform:** Google Cloud Platform (GCP)
- **Services Used:**
    - Google Cloud Storage (Secure file storage)
    - Cloud KMS (Key encryption management)
    - IAM (Identity and access control)
    - Cloud Audit Logs (Access monitoring and logging)

Architecture Design:



Implementation Steps:

Prerequisites:

- Python 3.7+
- Install required packages:

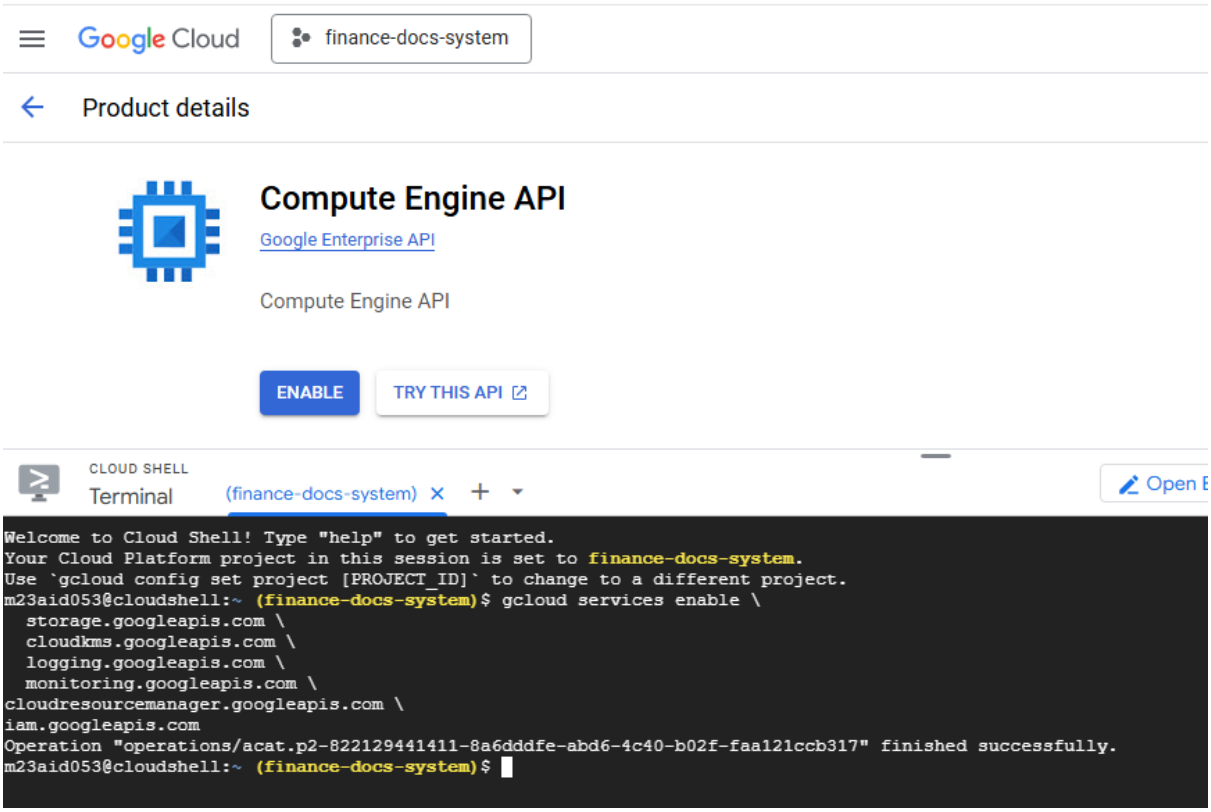
```
pip install google-cloud-storage google-cloud-kms google-cloud-logging google-cloud-monitoring google-api-python-client google-auth google-auth-httplib2
```

Authenticate and Set GCP project:

```
from google.colab import auth
auth.authenticate_user()
import os
os.environ["GOOGLE_CLOUD_PROJECT"] = "finance-docs-system"
PROJECT_ID = "finance-docs-system"
```

Enable Required APIs:

```
gcloud services enable \
  storage.googleapis.com \
  cloudkms.googleapis.com \
  logging.googleapis.com \
  monitoring.googleapis.com \
  cloudresourcemanager.googleapis.com \
  iam.googleapis.com
```



Set Up IAM Roles:

Python Script to Assign IAM Roles to Multiple Users

```
from google.auth import default
from googleapiclient.discovery import build

# Set your project ID
```

```python
PROJECT_ID = "finance-docs-system"

# List of user-role pairs
user_role_pairs = [
    ("m23aid026@iitj.ac.in", "roles/storage.admin"),
    ("m23aid026@iitj.ac.in", "roles/cloudkms.admin"),
    ("m23aid026@iitj.ac.in", "roles/logging.admin"),
    ("m23aid026@iitj.ac.in", "roles/monitoring.admin"),
    ("gcproleuser1@gmail.com", "roles/viewer"),
    ("gcproleuser2@gmail.com", "roles/editor"),
    ("m23aid053@iitj.ac.in", "roles/owner"),
    ("m23aid006@iitj.ac.in", "roles/iam.securityAdmin"),
    ("m23aid006@iitj.ac.in", "roles/cloudfunctions.invoker"),
    ("m23aid006@iitj.ac.in", "roles/iam.serviceAccountTokenCreator"),

]

# Authenticate using ADC (gcloud auth application-default login)
credentials, _ = default()
service = build("cloudresourcemanager", "v1", credentials=credentials)

# Get current IAM policy
policy = service.projects().getIamPolicy(resource=PROJECT_ID,
body={}).execute()

bindings = policy.get("bindings", [])

# Add each user-role binding
for email, role in user_role_pairs:
    member = f"user:{email}"
    print(f"Adding {member} to {role}")

    # Check if role already exists in bindings
    role_binding = next((b for b in bindings if b["role"] == role),
None)

    if role_binding:
        if member not in role_binding["members"]:
            role_binding["members"].append(member)
    else:
        # Create a new binding
        bindings.append({
            "role": role,
            "members": [member]
        })

# Update the IAM policy
policy["bindings"] = bindings
```

```
set_policy_request = {"policy": policy}

response = service.projects().setIamPolicy(
    resource=PROJECT_ID, body=set_policy_request
).execute()

print("\n IAM policy updated successfully.")
```

```
Commands   + Code   + Text

Adding user:m23aid026@iitj.ac.in to roles/storage.admin
Adding user:m23aid026@iitj.ac.in to roles/cloudkms.admin
Adding user:m23aid026@iitj.ac.in to roles/logging.admin
Adding user:m23aid026@iitj.ac.in to roles/monitoring.admin
Adding user:gcproleuser1@gmail.com to roles/viewer
Adding user:gcproleuser2@gmail.com to roles/editor
Adding user:m23aid053@iitj.ac.in to roles/owner
Adding user:m23aid006@iitj.ac.in to roles/iam.securityAdmin
Adding user:m23aid006@iitj.ac.in to roles/cloudfunctions.invoker
Adding user:m23aid006@iitj.ac.in to roles/iam.serviceAccountTokenCreator

 IAM policy updated successfully.
```

Prepare a Test File

```
with open("test-doc.pdf", "w") as f:
    f.write("Confidential Financial Report 2025")
```

Python script to

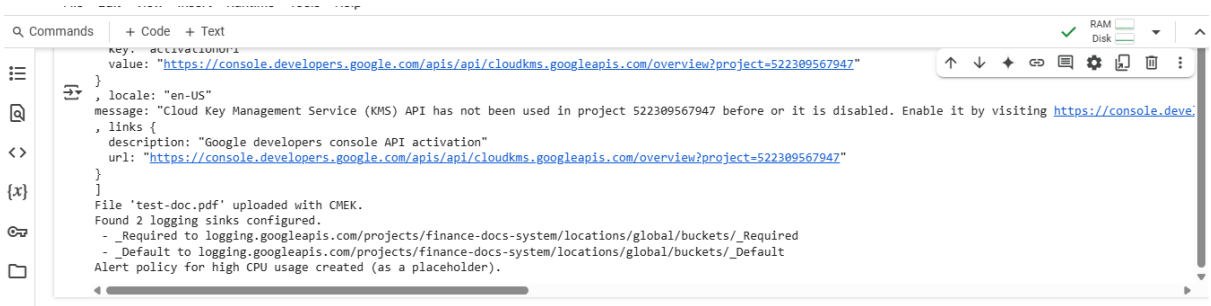
- Create a bucket
- Set up KMS with key rotation
- Upload `test-doc.pdf` using the KMS key
- Check logging sinks
- Create a dummy alert policy for unauthorized access

```
import os
from datetime import datetime, timedelta

from google.cloud import storage, kms_v1
from google.cloud.kms_v1 import KeyManagementServiceClient
from google.cloud.logging_v2 import Client as LoggingClient
from google.cloud.monitoring_v3 import AlertPolicyServiceClient, types
as monitoring_types
from google.api_core.exceptions import Conflict, NotFound
```

```
# Configurations
PROJECT_ID = "finance-docs-system"
BUCKET_NAME = "financial-docs-dms"
REGION = "us-central1"
KEY_RING_ID = "financial-docs-ring"
KEY_ID = "financial-docs-key"

# ---------- STEP 1: CREATE STORAGE BUCKET ----------
def create_bucket():
    storage_client = storage.Client(project=PROJECT_ID)
    try:
        storage_client.get_bucket(BUCKET_NAME)
        print(f"Bucket '{BUCKET_NAME}' already exists.")
    except NotFound:
        bucket = storage_client.bucket(BUCKET_NAME)
        bucket.iam_configuration.uniform_bucket_level_access_enabled =
True
        bucket.create(location=REGION)
        print(f"Bucket '{BUCKET_NAME}' created with uniform access
control.")


# ---------- STEP 2: CREATE KMS KEY RING AND KEY ----------
def create_kms_key():
    client = KeyManagementServiceClient()
    parent = f"projects/{PROJECT_ID}/locations/{REGION}"
    key_ring_path = client.key_ring_path(PROJECT_ID, REGION,
KEY_RING_ID)

    # Create Key Ring
    try:
        client.create_key_ring(request={"parent": parent,
"key_ring_id": KEY_RING_ID})
        print(f"Key ring '{KEY_RING_ID}' created.")
    except Exception as e:
        print(f"Key ring exists or error: {e}")

    # Create Crypto Key with rotation
    next_rotation_time = datetime.utcnow() + timedelta(days=30)
    crypto_key = {
        "purpose": kms_v1.CryptoKey.CryptoKeyPurpose.ENCRYPT_DECRYPT,
        "rotation_period": {"seconds": 60 * 60 * 24 * 30},  # 30 days
        "next_rotation_time": {"seconds":
int(next_rotation_time.timestamp())},
    }

    key_path = client.crypto_key_path(PROJECT_ID, REGION, KEY_RING_ID,
KEY_ID)
```

```
    try:
        client.create_crypto_key(
            request={
                "parent": key_ring_path,
                "crypto_key_id": KEY_ID,
                "crypto_key": crypto_key,
                "skip_initial_version_creation": False,
            }
        )
        print(f"Key '{KEY_ID}' created with 30-day rotation.")
    except Exception as e:
        print(f"Key exists or error: {e}")


# ---------- STEP 3: UPLOAD FILE WITH ENCRYPTION ----------
def upload_file_with_cmek(file_path):
    storage_client = storage.Client(project=PROJECT_ID)
    bucket = storage_client.bucket(BUCKET_NAME)

    blob = bucket.blob(os.path.basename(file_path))
    kms_key_name =
f"projects/{PROJECT_ID}/locations/{REGION}/keyRings/{KEY_RING_ID}/crypt
oKeys/{KEY_ID}"

    blob.kms_key_name = kms_key_name
    blob.upload_from_filename(file_path)

    print(f"File '{file_path}' uploaded with CMEK.")


# ---------- STEP 4: ENABLE LOGGING (Auto-enabled for KMS and Storage)
----------
def check_logging_enabled():
    logging_client = LoggingClient(project=PROJECT_ID)
    sinks = list(logging_client.list_sinks())
    print(f"Found {len(sinks)} logging sinks configured.")
    for sink in sinks:
        print(f" - {sink.name} to {sink.destination}")


# ---------- STEP 5: CREATE ALERT FOR UNAUTHORIZED ACCESS (DEMO) ------
----
def create_dummy_alert_policy():
    client = AlertPolicyServiceClient()
    project_name = f"projects/{PROJECT_ID}"

    policy = monitoring_types.AlertPolicy(
        display_name="DMS: High CPU Usage Alert (Placeholder)",
```

```
        combiner=monitoring_types.AlertPolicy.ConditionCombinerType.AND
,
        conditions=[
            monitoring_types.AlertPolicy.Condition(
                display_name="High CPU Utilization",
                condition_threshold=monitoring_types.AlertPolicy.Condit
ion.MetricThreshold(
                    filter=(
                        'metric.type="compute.googleapis.com/instance/c
pu/utilization" '
                        'resource.type="gce_instance"'
                    ),
                    comparison=monitoring_types.ComparisonType.COMPARIS
ON_GT,
                    threshold_value=0.8,
                    duration={"seconds": 60},
                ),
            )
        ],
        notification_channels=[],
    )

    client.create_alert_policy(name=project_name, alert_policy=policy)
    print("Alert policy for high CPU usage created (as a
placeholder).")


# ------------------ MAIN FLOW ------------------
if __name__ == "__main__":
    create_bucket()
    create_kms_key()
    upload_file_with_cmek("test-doc.pdf")
    check_logging_enabled()
    create_dummy_alert_policy()
```



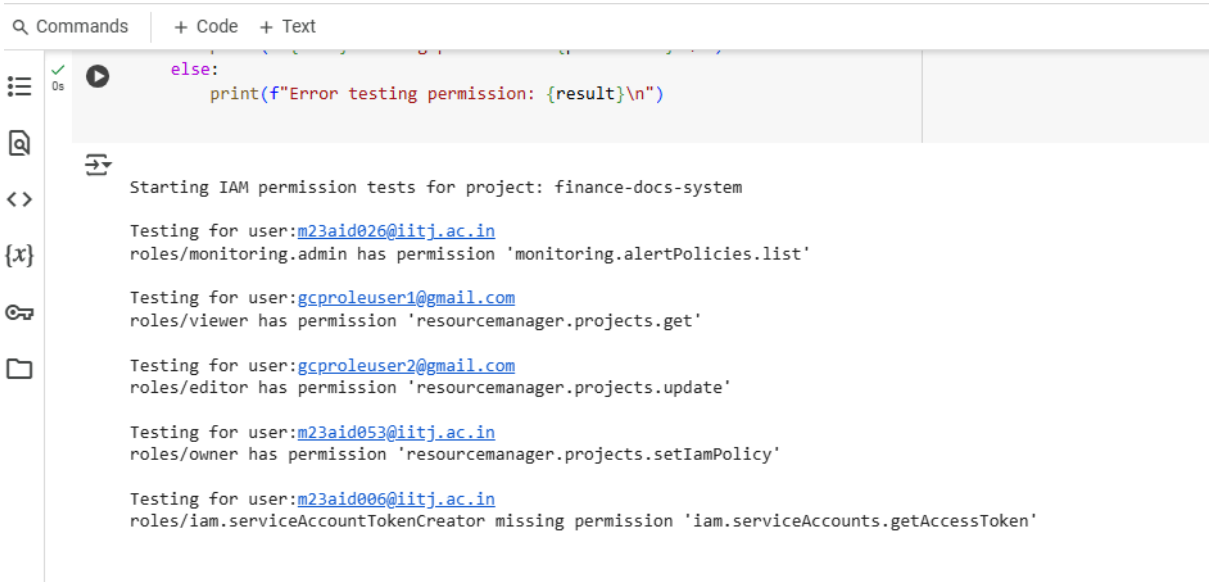
Results and Analysis

Google colab test file link with results:
https://colab.research.google.com/drive/1bgsAQnAr0Uc5hQNLFH6WP1bhVskZnkCt?usp=sharing

Python script to validate User/Role permission

```
from google.auth import default
from googleapiclient.discovery import build

# Set your GCP project ID
PROJECT_ID = "finance-docs-system"

# List of users and the roles they were assigned
user_role_pairs = {
    "m23aid026@iitj.ac.in": "roles/storage.admin",
    "m23aid026@iitj.ac.in": "roles/cloudkms.admin",
    "m23aid026@iitj.ac.in": "roles/logging.admin",
    "m23aid026@iitj.ac.in": "roles/monitoring.admin",
    "gcproleuser1@gmail.com": "roles/viewer",
    "gcproleuser2@gmail.com": "roles/editor",
    "m23aid053@iitj.ac.in": "roles/owner",
    "m23aid006@iitj.ac.in": "roles/iam.securityAdmin",
    "m23aid006@iitj.ac.in": "roles/cloudfunctions.invoker",
    "m23aid006@iitj.ac.in": "roles/iam.serviceAccountTokenCreator",
}

# Role → Corresponding permission to test
role_permissions = {
    "roles/storage.admin": "storage.buckets.create",
    "roles/cloudkms.admin": "cloudkms.keyRings.create",
    "roles/logging.admin": "logging.sinks.list",
    "roles/monitoring.admin": "monitoring.alertPolicies.list",
    "roles/viewer": "resourcemanager.projects.get",
    "roles/editor": "resourcemanager.projects.update",
    "roles/owner": "resourcemanager.projects.setIamPolicy",
    "roles/iam.securityAdmin": "resourcemanager.projects.setIamPolicy",
    "roles/cloudfunctions.invoker": "cloudfunctions.functions.invoke",
    "roles/iam.serviceAccountTokenCreator":
"iam.serviceAccounts.getAccessToken",
}

# Authenticate and create IAM API client
creds, _ = default()
service = build('cloudresourcemanager', 'v1', credentials=creds)

# Test IAM permission
def check_permission(project_id, permission):
    try:
        response = service.projects().testIamPermissions(
```

```
            resource=project_id,
            body={"permissions": [permission]}
        ).execute()
        return permission in response.get("permissions", [])
    except Exception as e:
        return f"ERROR: {str(e)}"

# Run tests for all users
print(f"\nStarting IAM permission tests for project: {PROJECT_ID}\n")

for user, role in user_role_pairs.items():
    permission = role_permissions[role]
    member = f"user:{user}"

    print(f"Testing for {member}")
    result = check_permission(PROJECT_ID, permission)
    if result is True:
        print(f"{role} has permission '{permission}'\n")
    elif result is False:
        print(f"{role} missing permission '{permission}'\n")
    else:
        print(f"Error testing permission: {result}\n")
```



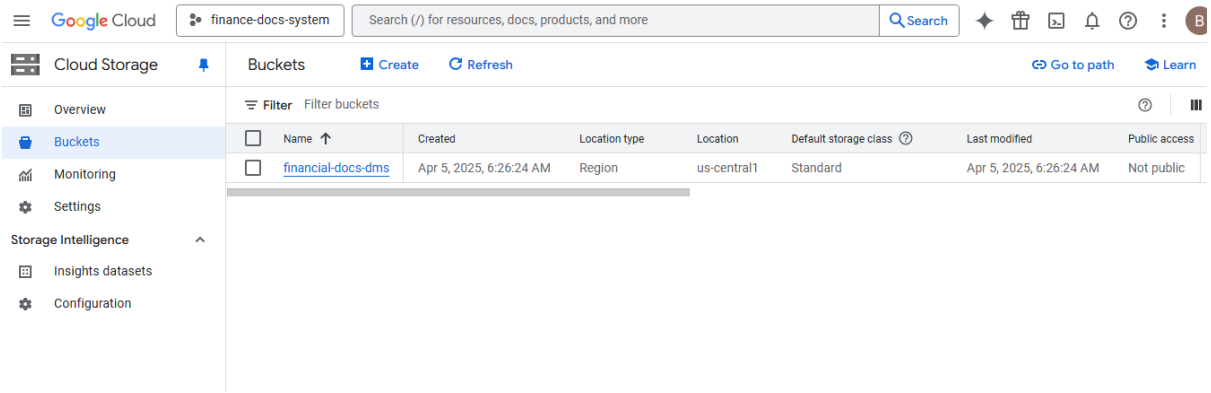
- Above results shows the role level access with respect to user.

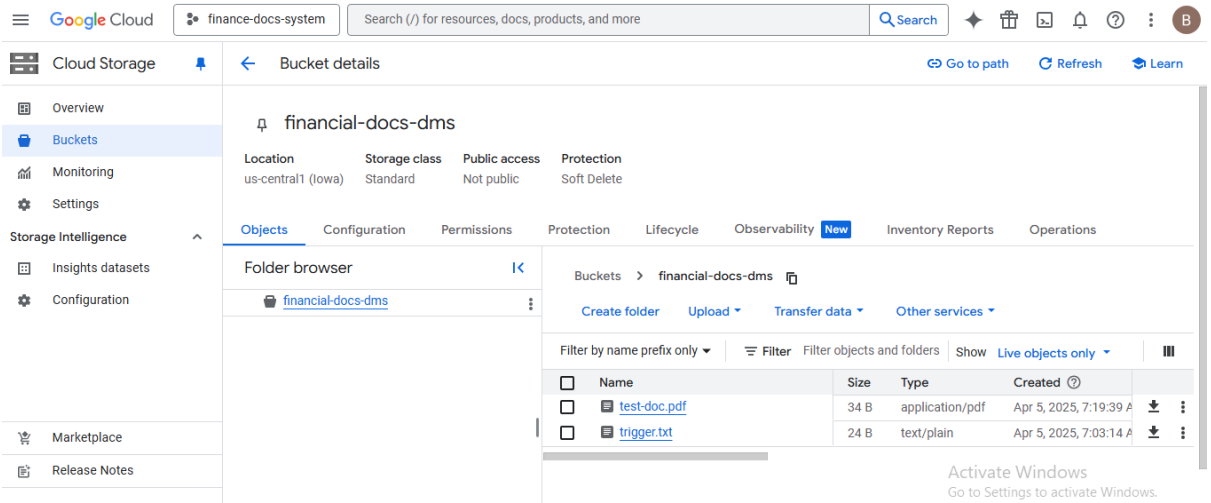
Verify Bucket & File

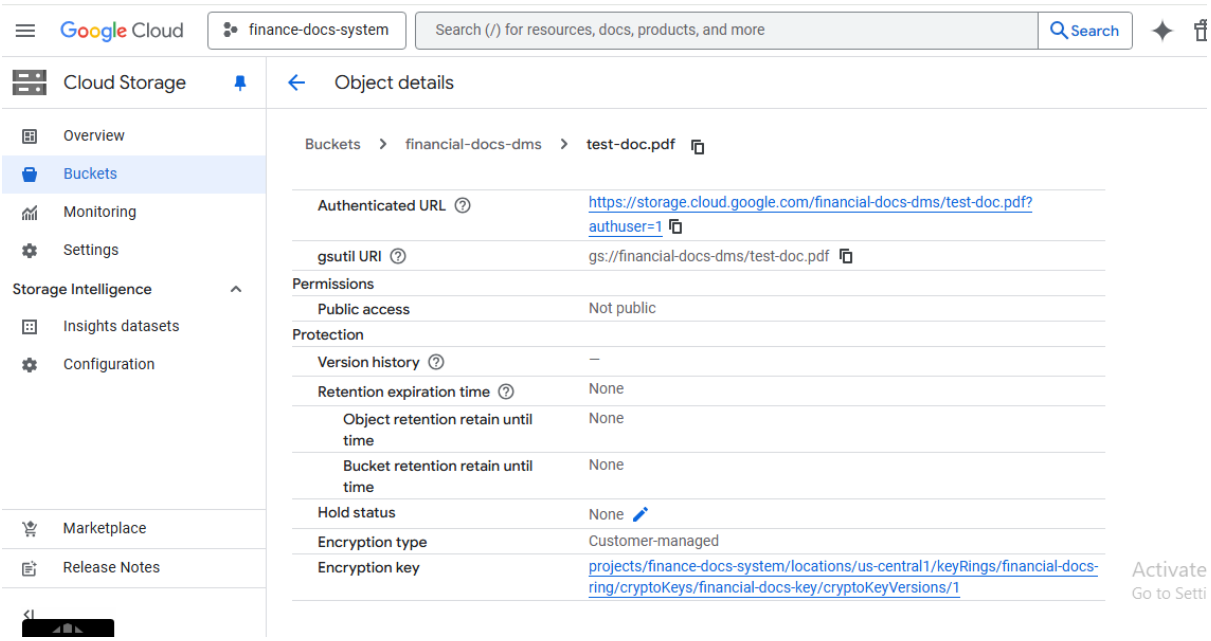
Go to **Storage > Browser** on GCP console:

- Check that `financial-docs-dms` exists
- Check that `test-doc.pdf` is uploaded

Click on the file → "Encryption" tab → should show **Customer-managed key (CMEK)**



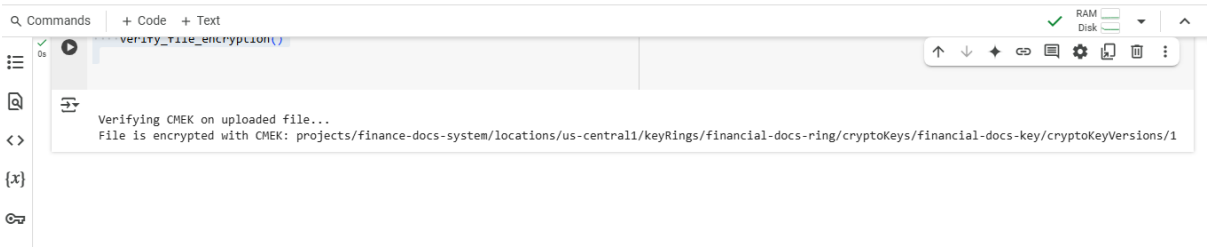




Verify KMS

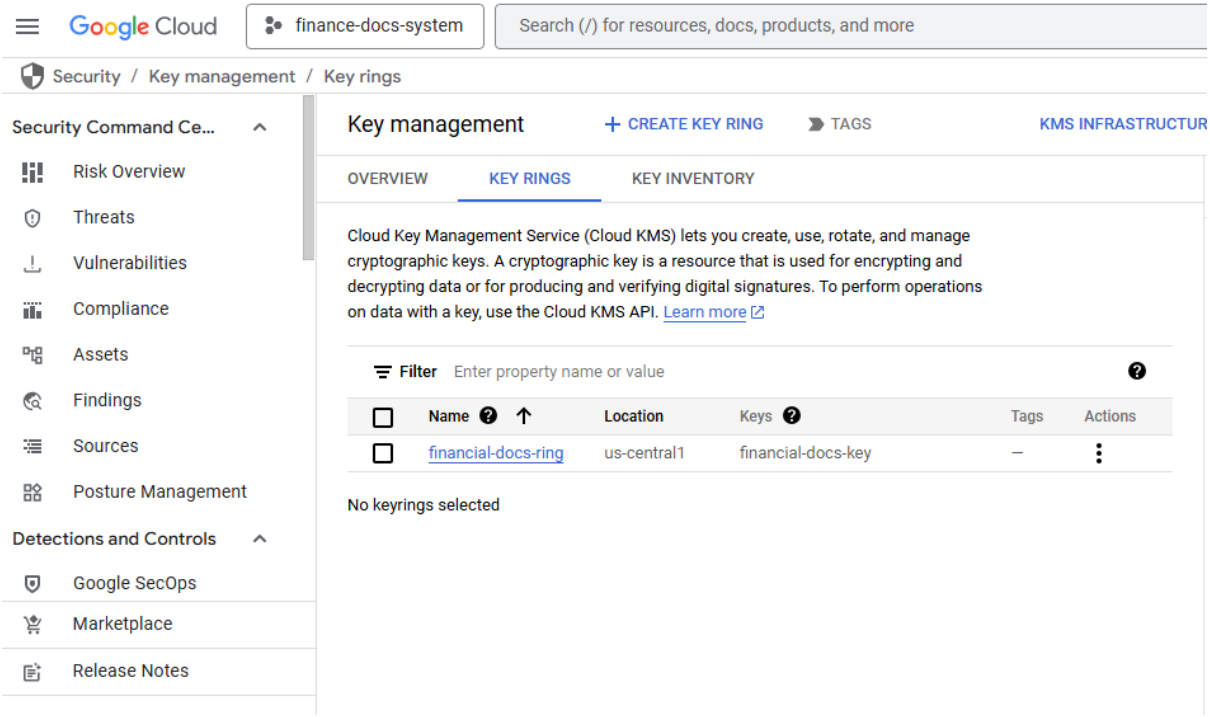
Python script to verify file encryption

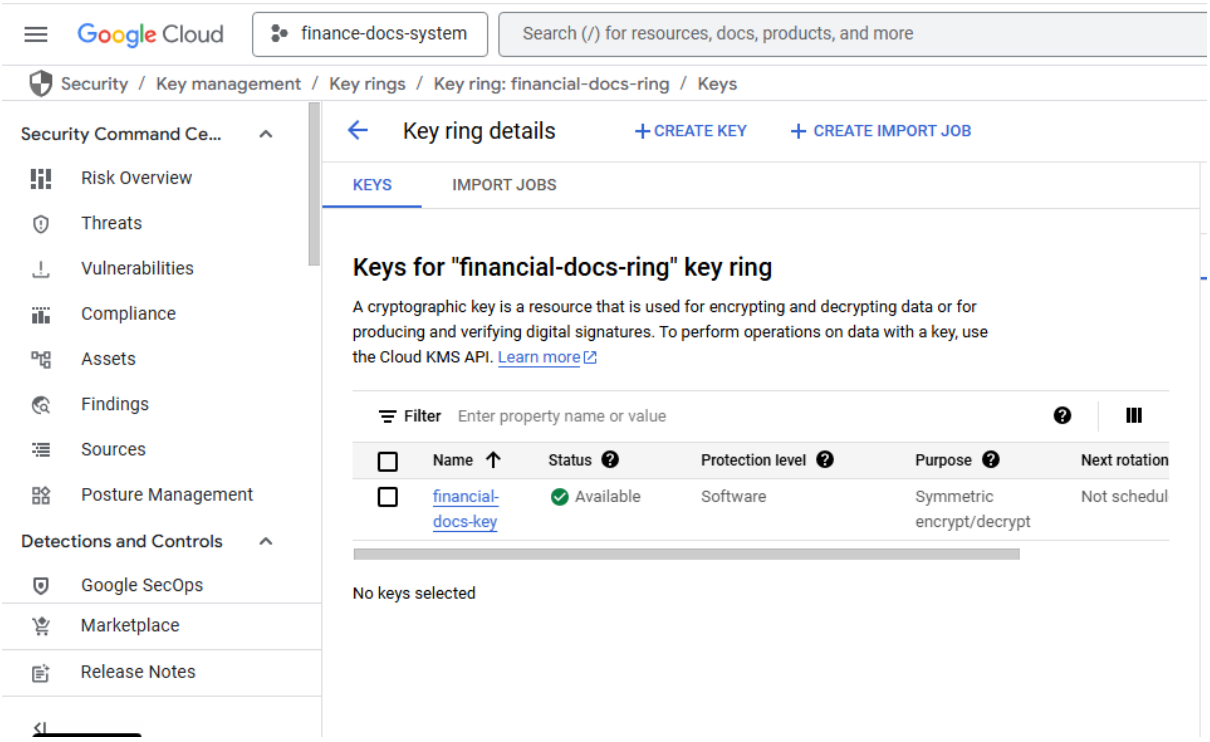
```
from google.cloud import storage, kms_v1
from google.cloud.kms_v1 import KeyManagementServiceClient

# Configuration
PROJECT_ID = "finance-docs-system"
BUCKET_NAME = "financial-docs-dms"
REGION = "us-central1"
KEY_RING_ID = "financial-docs-ring"
KEY_ID = "financial-docs-key"
OBJECT_NAME = "test-doc.pdf"

def verify_file_encryption():
    print("\n Verifying CMEK on uploaded file...")
    storage_client = storage.Client(project=PROJECT_ID)
    bucket = storage_client.bucket(BUCKET_NAME)
    blob = bucket.get_blob(OBJECT_NAME)

    if blob is None:
        print(f" File '{OBJECT_NAME}' not found in bucket
'{BUCKET_NAME}'.")
        return

    if blob.kms_key_name:
        print(f" File is encrypted with CMEK: {blob.kms_key_name}")
    else:
        print("File is not encrypted using a CMEK.")



if __name__ == "__main__":
    verify_file_encryption()
```

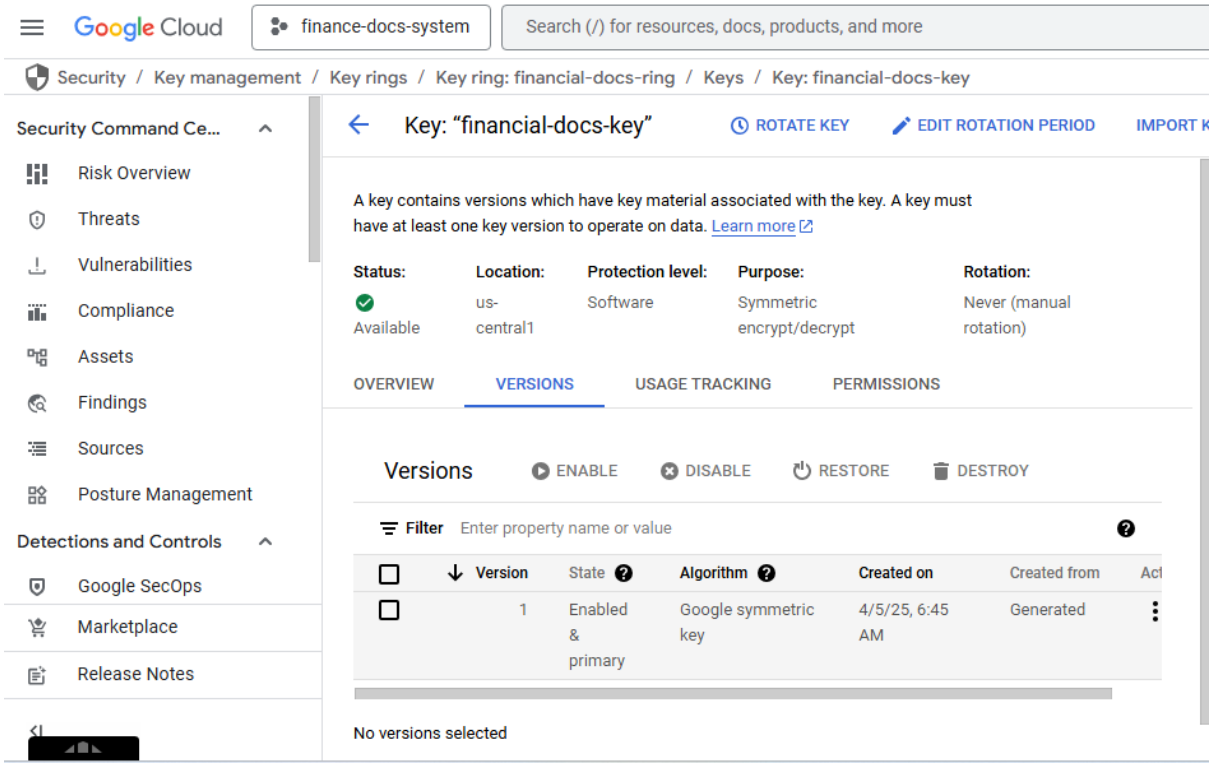


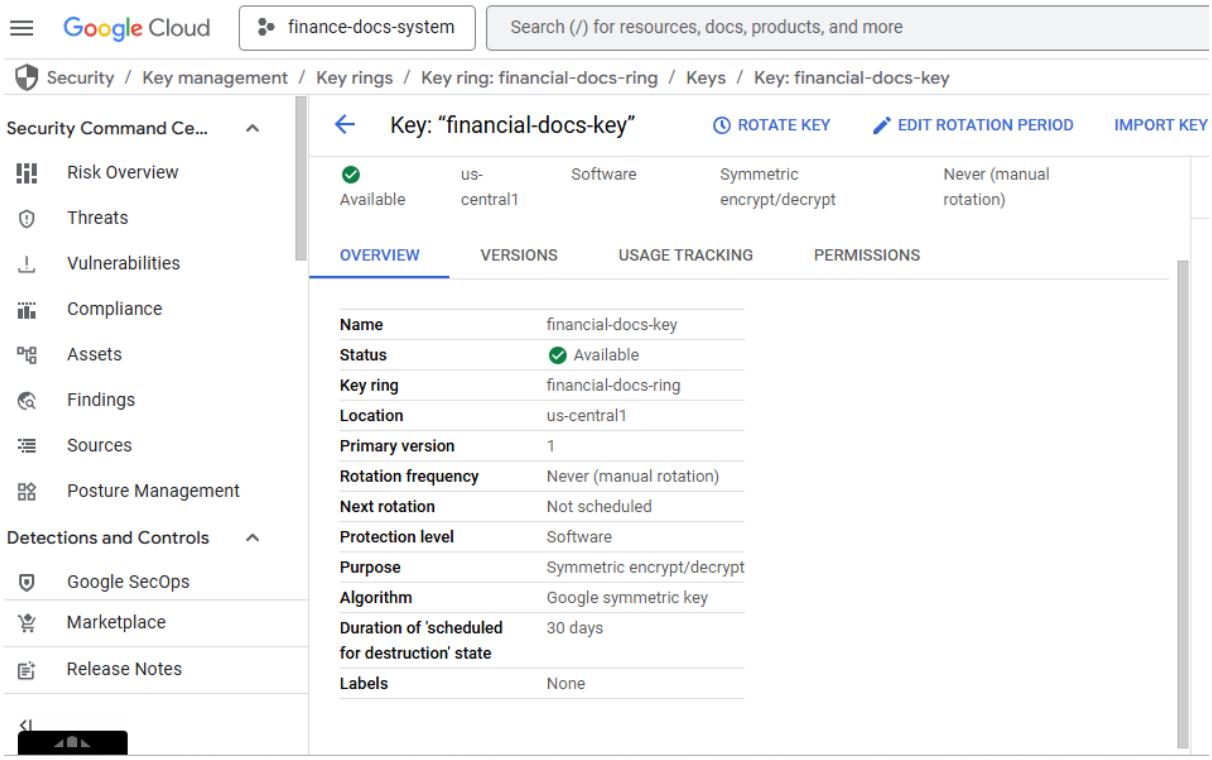
Go to **Security > Key Management > Key Rings**

- Open `financial-docs-ring` → `financial-docs-key`
- Check for key rotation policy





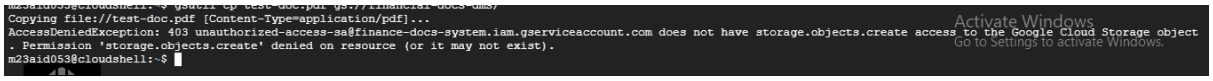




Simulate Unauthorized Access

```
simulate_unauthorized_access.sh

#!/bin/bash

# CONFIGURATION
```

```
PROJECT_ID="finance-docs-system"
SERVICE_ACCOUNT_NAME="unauthorized-access-sa"
BUCKET_NAME="financial-docs-dms"
FILE_TO_UPLOAD="test-doc.pdf"

# STEP 1: Create a new service account with NO KMS permissions
echo "Creating service account..."
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME \
  --project=$PROJECT_ID \
  --display-name="Unauthorized Access SA"

# STEP 2: Generate a key file for the service account
echo "Creating service account key..."
gcloud iam service-accounts keys create ${SERVICE_ACCOUNT_NAME}-
key.json \
  --iam-
account="${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com"
\
  --project=$PROJECT_ID

# STEP 3: Activate service account with the key (simulate user login)
echo "Authenticating as the new service account..."
gcloud auth activate-service-account \
  --key-file=${SERVICE_ACCOUNT_NAME}-key.json

# STEP 4: Attempt to upload a file to a CMEK-protected bucket
echo "Attempting to upload to CMEK bucket (should fail)..."
gsutil cp $FILE_TO_UPLOAD gs://$BUCKET_NAME/

# STEP 5: Cleanup auth (switch back to default user)
echo "Restoring original credentials..."
gcloud auth revoke

# Optional: remove key
# rm ${SERVICE_ACCOUNT_NAME}-key.json
```



View Logs

```
from google.cloud import logging_v2

# Set your GCP Project ID
PROJECT_ID = "finance-docs-system"

def view_recent_logs(log_filter: str = None, limit: int = 10):
    """Fetch and display recent logs from Google Cloud Logging."""
```

```
    client = logging_v2.Client(project=PROJECT_ID)
    logger = client.list_entries(order_by=logging_v2.DESCENDING,
page_size=limit, filter_=log_filter)

    print(f"\nFetching up to {limit} log entries...\n")

    for i, entry in enumerate(logger, start=1):
        print(f"--- Log Entry #{i} ---")
        print(f"Timestamp: {entry.timestamp}")
        print(f"Severity: {entry.severity}")
        print(f"Log Name: {entry.log_name}")
        print(f"Resource: {entry.resource.type}")
        print(f"Payload: {entry.payload}")
        print("----------------------\n")

if __name__ == "__main__":
    # Optional filter to narrow down results (e.g., only KMS logs)
    # Examples:
    #   log_filter = 'resource.type="gcs_bucket"'
    #   log_filter = 'resource.type="cloudkms_cryptokey"'
    #   log_filter =
'logName="projects/YOUR_PROJECT_ID/logs/cloudaudit.googleapis.com%2Fact
ivity"'

    log_filter = None  # or customize your filter
    view_recent_logs(log_filter=log_filter, limit=10)
```

```
Commands   + Code   + Text
view_recent_logs(log_filter=log_filter, limit=10)

Fetching up to 10 log entries...

--- Log Entry #1 ---
Timestamp: 2025-04-05 07:15:43.129744+00:00
Severity: NOTICE
Log Name: projects/finance-docs-system/logs/cloudaudit.googleapis.com%2Factivity
Resource: service_account
Payload: OrderedDict([('@type', 'type.googleapis.com/google.cloud.audit.AuditLog'), ('status', {}), ('authenticationInfo', {'principalEmail': 'm23aid053@iitj.
----------------------

--- Log Entry #2 ---
Timestamp: 2025-04-05 07:15:40.281577+00:00
Severity: NOTICE
Log Name: projects/finance-docs-system/logs/cloudaudit.googleapis.com%2Factivity
Resource: service_account
Payload: OrderedDict([('@type', 'type.googleapis.com/google.cloud.audit.AuditLog'), ('status', {}), ('authenticationInfo', {'principalEmail': 'm23aid053@iitj.
----------------------

--- Log Entry #3 ---
Timestamp: 2025-04-05 06:56:13.058735+00:00
Severity: NOTICE
Log Name: projects/finance-docs-system/logs/cloudaudit.googleapis.com%2Factivity
```

Verify Alert Policy

Create unauthorised access:

```
m23aid053@cloudshell:~ (finance-docs-system)$ gcloud config set account m23aid053@iitj.ac.in
Updated property [core/account].
m23aid053@cloudshell:~ (finance-docs-system)$ gcloud logging metrics create unauthorized_access_metric \
  --description="Metric for unauthorized access attempts to GCS" \
  --log-filter='resource.type="gcs_bucket" AND protoPayload.status.code=7'
Created [unauthorized_access_metric].
```

Create Alert policy:

```
from google.cloud import monitoring_v3

PROJECT_ID = "finance-docs-system"
ALERT_POLICY_NAME = "DMS: Unauthorized Access Attempt"
METRIC_TYPE = "logging.googleapis.com/user/unauthorized_access_metric"

def create_alert_policy():
    client = monitoring_v3.AlertPolicyServiceClient()
    project_name = f"projects/{PROJECT_ID}"

    # Define the condition for the alert
    condition = monitoring_v3.AlertPolicy.Condition(
        display_name="Unauthorized GCS Access Attempt",
        condition_threshold=monitoring_v3.AlertPolicy.Condition.MetricT
hreshold(
            filter=f'metric.type="{METRIC_TYPE}" AND
resource.type="gcs_bucket"',
            comparison=monitoring_v3.ComparisonType.COMPARISON_GT,
            threshold_value=0,
            duration={"seconds": 60},
            aggregations=[monitoring_v3.Aggregation(
                alignment_period={"seconds": 60},
                per_series_aligner=monitoring_v3.Aggregation.Aligner.AL
IGN_RATE
            )],
        ),
    )

    # Define the full alert policy
    alert_policy = monitoring_v3.AlertPolicy(
        display_name=ALERT_POLICY_NAME,
        combiner=monitoring_v3.AlertPolicy.ConditionCombinerType.AND,
        conditions=[condition],
        enabled=True
    )

    # Create the alert policy
    policy = client.create_alert_policy(name=project_name,
alert_policy=alert_policy)
    print(f"Alert policy created: {policy.name}")

if __name__ == "__main__":
    create_alert_policy()
```

Verify Alert

```
from google.cloud import monitoring_v3

# Your Google Cloud Project ID
PROJECT_ID = "finance-docs-system"

# The display name of the alert policy you want to verify
TARGET_ALERT_NAME = "DMS: Unauthorized Access Attempt"

def verify_alert_policy():
    client = monitoring_v3.AlertPolicyServiceClient()
    project_name = f"projects/{PROJECT_ID}"

    # Fetch all alert policies in the project
    policies = client.list_alert_policies(name=project_name)

    print(f"\n🔍 Verifying alert policy: '{TARGET_ALERT_NAME}'\n")

    found = False
    for policy in policies:
        if policy.display_name == TARGET_ALERT_NAME:
            found = True
            print("✅ Alert policy found!")
            print(f"ID: {policy.name}")
            print(f"Display Name: {policy.display_name}")
            print(f"Combiner: {policy.combiner}")
            print(f"Conditions:")
            for condition in policy.conditions:
                print(f"  - {condition.display_name}")
                if condition.condition_threshold is not None:
                    print(f"    Metric Filter:
{condition.condition_threshold.filter}")
                    print(f"    Threshold Value:
{condition.condition_threshold.threshold_value}")
                    print(f"    Duration:
{condition.condition_threshold.duration}")
            print(f"Notification Channels:
{policy.notification_channels}")
            break

    if not found:
        print("❌ Alert policy not found. Please make sure it exists.")
```

```python
if __name__ == "__main__":
    verify_alert_policy()
```

```
Commands   + Code   + Text

Verifying alert policy: 'DMS: Unauthorized Access Attempt'

Alert policy found!
ID: projects/finance-docs-system/alertPolicies/11744076689601395964
Display Name: DMS: Unauthorized Access Attempt
Combiner: 1
Conditions:
  - Unauthorized GCS Access Attempt
    Metric Filter: metric.type="logging.googleapis.com/user/unauthorized_access_metric" AND resource.type="gcs_bucket"
    Threshold Value: 0.0
    Duration: 0:01:00
Notification Channels: []
```

Comparative Analysis:

- **Before:** Lacked encryption, inconsistent access control, and no audit logs.
- **After:** Strong encryption, role-based access control, and real-time monitoring.

Challenges and Solutions

Challenges Faced:

1. Integration with existing IAM policies.
2. High costs of encryption key management.
3. Performance impact of real-time logging.

Solutions Implemented:

1. **Mapped IAM policies** to existing users and groups.
2. Used **Google-managed encryption keys** to reduce costs.
3. Optimized logging filters to focus on **high-risk access events only**.

Future Scope

- **AI-driven security monitoring:** Implement machine learning to detect anomalous access patterns.
- **Multi-cloud security integration:** Extend security mechanisms to AWS/Azure.
- **Zero Trust Security Model:** Further reduce risk with stricter identity verification.

Conclusion

This case study demonstrates how **Google Cloud Storage, IAM, and Cloud KMS** enhance data security in a financial firm's document management system. By implementing **strong encryption, access control, and audit logging**, the firm successfully mitigated security risks, ensured compliance, and improved operational efficiency. **Cloud-native security solutions are essential for protecting sensitive financial data in modern cloud environments.**

References

- A. Ikuomola, E. A. Oyekan, and O. M. Orogbemi, "A Secured Cloud-Based Electronic Document Management System," ***Int. J. Innov. Res. Dev.***, vol. 11, no. 12, Dec. 2022. [Online]. Available: **https://www.researchgate.net/publication/369768638_A_Secured_Cloud-Based_Electronic_Document_Management_System**

- Y. Wang, M. Zhu, J. Yuan, G. Wang, and H. Zhou, "The Intelligent Prediction and Assessment of Financial Information Risk in the Cloud Computing Model," ***arXiv preprint arXiv:2404.09322***, Apr. 2024. [Online]. Available: https://arxiv.org/abs/2404.09322

- M. A. AlZain, B. Soh, and E. Pardede, "A Survey on Data Security Issues in Cloud Computing: From Single to Multi-Clouds," ***J. Softw.***, vol. 8, no. 5, pp. 1068–1078, May 2013. [Online]. Available: https://www.academia.edu/3555558/A_Survey_on_Data_Security_Issues_in_Cloud_Computing_From_Single_to_Multi_Clouds

- J. Li, "Cloud Computing and Its Impact on the Security of Financial Systems," ***Int. J. Comput. Sci. Eng.***, vol. 14, no. 6, pp. 150–160, Jun. 2024. [Online]. Available: **https://article.sapub.org/10.5923.j.computer.20241406.01.html**

- R. K. L. Ko et al., "TrustCloud: A Framework for Accountability and Trust in Cloud Computing," ***HP Labs Singapore***, 2011. [Online]. Available: **https://www.hpl.hp.com/techreports/2011/HPL-2011-38.pdf**

- S. Subashini and V. Kavitha, "A Survey on Security Issues in Service Delivery Models of Cloud Computing," ***J. Netw. Comput. Appl.***, vol. 34, no. 1, pp. 1–11, Jan. 2011. [Online]. Available: **https://www.sciencedirect.com/science/article/pii/S1084804510001281**

- M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "On Technical Security Issues in Cloud Computing," in ***Proc. IEEE Int. Conf. Cloud Comput.***, Bangalore, India, 2009, pp. 109–116. [Online]. Available: **https://ieeexplore.ieee.org/document/5280678**

- S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising from Cloud Computing," in ***Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.***, Indianapolis, IN, USA, 2010, pp. 693–702. [Online]. Available: **https://ieeexplore.ieee.org/document/5708510**

- W. Jansen and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing," ***NIST Special Publication 800-144***, Dec. 2011. [Online]. Available: **https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-144.pdf**

- P. Mell and T. Grance, "The NIST Definition of Cloud Computing," ***NIST Special Publication 800-145***, Sep. 2011. [Online]. Available: **https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf**

Appendices

Appendix A: Sample Log Entry

```
--- Log Entry #9 ---

Timestamp: 2025-04-05 01:25:17.724246+00:00

Severity: ERROR

Log Name: projects/finance-docs-
system/logs/cloudaudit.googleapis.com%2Factivity

Resource: cloudkms_cryptokey

Payload: OrderedDict([('@type',
'type.googleapis.com/google.cloud.audit.AuditLog'), ('status', {'code':
6, 'message': 'CryptoKey projects/finance-docs-system/locations/us-
central1/keyRings/financial-docs-ring/cryptoKeys/financial-docs-key
already exists.'}), ('authenticationInfo', {'principalEmail':
'm23aid053@iitj.ac.in', 'principalSubject':
'user:m23aid053@iitj.ac.in'}), ('requestMetadata', {'callerIp':
'35.240.234.53', 'callerSuppliedUserAgent': 'google-cloud-sdk
gcloud/517.0.0 command/gcloud.kms.keys.create invocation-
id/0271756f1bfd4492b34d98139c562696 environment/devshell environment-
version/None client-os/LINUX client-os-ver/6.6.72 client-pltf-
arch/x86_64 interactive/True from-script/False python/3.12.8
term/screen (Linux 6.6.72+),gzip(gfe)', 'requestAttributes': {'time':
'2025-04-05T01:25:17.739410061Z', 'auth': {}}, 'destinationAttributes':
{}}), ('serviceName', 'cloudkms.googleapis.com'), ('methodName',
'CreateCryptoKey'), ('authorizationInfo', [{'resource':
'projects/finance-docs-system/locations/us-central1/keyRings/financial-
docs-ring', 'permission': 'cloudkms.cryptoKeys.create', 'granted':
True, 'resourceAttributes': {'service': 'google.cloud.kms', 'name':
'projects/finance-docs-system/locations/us-central1/keyRings/financial-
docs-ring/cryptoKeys/financial-docs-key', 'type':
'cloudkms.googleapis.com/CryptoKey'}, 'permissionType':
'ADMIN_WRITE'}]), ('resourceName', 'projects/finance-docs-
system/locations/us-central1/keyRings/financial-docs-
ring/cryptoKeys/financial-docs-key'), ('request', {'cryptoKey':
{'versionTemplate': {'algorithm': 'GOOGLE_SYMMETRIC_ENCRYPTION',
'protectionLevel': 'SOFTWARE'}, 'purpose': 'ENCRYPT_DECRYPT'},
'parent': 'projects/finance-docs-system/locations/us-
central1/keyRings/financial-docs-ring', '@type':
'type.googleapis.com/google.cloud.kms.v1.CreateCryptoKeyRequest',
'cryptoKeyId': 'financial-docs-key'}), ('metadata', {}),
('resourceLocation', {'currentLocations': ['us-central1']})])
```

Appendix B: Sample alert policy

```
Verifying alert policy: 'DMS: Unauthorized Access Attempt'

✅ Alert policy found!
ID: projects/finance-docs-system/alertPolicies/11744076689601395964
Display Name: DMS: Unauthorized Access Attempt
Combiner: 1
Conditions:
  - Unauthorized GCS Access Attempt
```

```
    Metric Filter:
metric.type="logging.googleapis.com/user/unauthorized_access_metric"
AND resource.type="gcs_bucket"
    Threshold Value: 0.0
    Duration: 0:01:00
Notification Channels: []
```