**Encrypting Data in Transit/ A Plugin to Encrypt Data with The Payload, Encryption Key [KMS] and Key Version**

**METHOD 1)🡪 CREATING DIFFERENT FUNCTIONS USING PYTHON**

**ENCRYPTION WORKFLOW:**

MAIN:

* FETCH ROWS (ROW WISE DATA)
* FETCH LATEST KEY WITH VERSION
* NEW TABLE REFERENCE

* ITERATING OVER THE ROWS
* ENCRYPTING THE DATA CELL WISE
* JOINING THE KEY-IV PAIR
* ENCRYPTING THE KEY PAIR USING RSA ALGO.
* STORING THE ENCRYPTED DATA AND ENCRYPTED KEY

Data security plays a critical role in today's digital world, especially with the increasing reliance on cloud platforms like Google Cloud Platform (GCP). This report delves into the implementation of data encryption and decryption in GCP using Python. It focuses on the utilization of both symmetric and asymmetric encryption techniques, with Google Cloud Key Management Service (KMS) managing the keys.

**Symmetric Encryption Function:**

In the symmetric encryption function, we employ the robust AES algorithm to encrypt plaintext data. The function generates a 256-bit random key and a 128-bit initialization vector (iv). To ensure compatibility with AES, the plaintext is padded to fit the block size before encryption using the CBC mode. Finally, the function returns the iv and the encrypted payload, encoded in base64.

**Asymmetric Encryption Function:**

On the other hand, the asymmetric encryption function retrieves a public key from Google Cloud KMS. Utilizing RSA-OAEP encryption, it encrypts a plaintext buffer containing the symmetric key and iv. The resulting ciphertext is then returned, encoded in base64.

**Fetch Latest Key Version Function:**

Ensuring the use of the latest key version is crucial. This function fetches the latest key version from Google Cloud KMS. Inputs include the project ID, location ID, key ring ID, and key ID, and the function returns the latest key version and its version number.

**Fetch Rows Function:**

The fetch rows function queries a specified BigQuery table and returns a BigQuery Table object containing the rows. Inputs required for this operation are the project ID, dataset ID, and table ID.

**Insert Data into Table Function:**

To insert encrypted data into a specified BigQuery table, the insert data into table function is utilized. This function requires the table reference and the data to be inserted as inputs.

**Conclusion:**

In conclusion, this report has explored the process of encrypting and decrypting data in Google Cloud Platform using Python. By leveraging Google Cloud KMS for key management, data can be securely encrypted and stored in BigQuery. This approach ensures the security and integrity of data in cloud-based systems, a critical aspect of modern information systems.Top of Form

**Method 2🡪**

**USING THE INBUILT COLUMN LEVEL ENCRYPTION METHOD OF BIGQUERY:**

A diagram of a computer process

Description automatically generatedWorkflow:

**Column-Level Encryption:**

Column-level encryption is a sophisticated technique used to secure sensitive data stored in databases. Unlike traditional encryption methods that encrypt entire databases or tables, column-level encryption focuses on encrypting individual columns within a table. This approach allows organizations to selectively encrypt only the data that requires protection, such as personally identifiable information (PII), financial data, or other sensitive information.

**How It Works**

At the core of column-level encryption are two types of keys: Data Encryption Keys (DEKs) and Key Encryption Keys (KEKs). DEKs are symmetric keys used to encrypt and decrypt data, while KEKs are asymmetric keys used to encrypt and decrypt the DEKs.

When encrypting data, a new DEK is generated for each encryption operation. This DEK is used to encrypt the data, providing fast and efficient encryption. However, to ensure the security of the DEK, it is encrypted using a KEK before being stored or transmitted. This two-step process ensures that even if the DEK is compromised, the data remains protected.

**DEK and KEK**

**Data Encryption Key (DEK):** The DEK is a symmetric key that is used to encrypt and decrypt data. It is generated randomly for each encryption operation, ensuring that even if one DEK is compromised, other data remains secure.

**Key Encryption Key (KEK):** The KEK is an asymmetric key used to encrypt and decrypt the DEK. It is typically stored securely in a key management system and is used to protect the DEK during storage or transmission.

**AEAD (Authenticated Encryption with Associated Data)**

AEAD is a type of encryption that provides both confidentiality and integrity. It ensures that the encrypted data has not been tampered with during transmission or storage. AEAD algorithms, such as AES-GCM (Advanced Encryption Standard-Galois/Counter Mode), are commonly used in column-level encryption to provide these security guarantees.

**Functions Used in the Code :**

**Create DEK Keyset Table:** This function creates a table in BigQuery to store the DEK key sets. The table has two columns: id, which stores a unique identifier for each table, and keyset, which stores the wrapped DEK key sets.

**Create DEK Wrapped Keyset:** This function inserts a new row into the DEK key set table, containing the wrapped DEK key set for a specific table. It uses the KEYS.NEW\_WRAPPED\_KEYSET function to generate the wrapped key set, specifying the KEK and encryption algorithm.

**Select Key for Encryption:** This function selects the wrapped DEK key set from the DEK key set table for a specific table. It uses a SQL query with a SELECT statement to retrieve the key set.

**Encrypt Table Data**: This function encrypts the data in the specified columns of a table using the selected DEK key set. It uses the DETERMINISTIC\_ENCRYPT function to encrypt each column, providing the KEK and selected DEK key set.

**Conclusion**

Column-level encryption is a powerful tool for protecting sensitive data in databases. By encrypting individual columns with DEKs and KEKs, organizations can enhance their data security and comply with regulatory requirements. The code provided demonstrates a practical implementation of column-level encryption in BigQuery, showcasing how DEKs and KEKs can be used to encrypt and protect sensitive data.