Key Management Service (KMS)

The Basic Terms and Definitions that you should aware before we jump into KMS learning topics

AWS KMS (Key Management Service): A fully managed service that makes it easy to create, control, and manage cryptographic keys used to encrypt data across AWS services and within your applications.

Customer Master Key (CMK): A logical key that represents the top-level key in AWS KMS. CMKs can be used to encrypt and decrypt up to 4 KB of data directly, or to generate data encryption keys (DEKs) for encrypting larger amounts of data. CMKs can be symmetric or asymmetric.

Symmetric Key: A single encryption key that is used for both encryption and decryption of data. AWS KMS symmetric CMKs use a 256-bit Advanced Encryption Standard (AES-256) algorithm.

Asymmetric Key: A key pair (public key and private key) used for encryption and decryption. Asymmetric CMKs can be used for operations like signing, verifying, encrypting, and decrypting data.

Data Encryption Key (DEK): Keys generated by KMS that are used to encrypt data outside of KMS. DEKs can be generated, encrypted, and decrypted by CMKs, but are not stored or managed by AWS KMS.

Envelope Encryption: A method of encrypting data where a DEK is used to encrypt the data, and the DEK itself is encrypted using a CMK. This allows for efficient encryption of large data sets with minimal exposure of the CMK.

Key Policy: A JSON-based document that defines who can use and manage a CMK and under what conditions. Key policies are attached directly to CMKs and are the primary mechanism for controlling access to CMKs.

Grants: Temporary permissions to use a CMK that are more flexible and granular than key policies. Grants can allow specific AWS services, users, or roles to perform specific actions with a CMK.

AWS KMS Alias: A friendly name that you can use to refer to a CMK in your applications instead of using the CMK's key ID or ARN. Aliases make it easier to identify and manage CMKs.

AWS KMS Key ID: A unique identifier for a CMK. Each CMK has a key ID, which is unique across all of your AWS KMS keys.

AWS KMS Key ARN (Amazon Resource Name): A unique identifier for a CMK that includes the AWS region and account. ARNs are used to specify keys in AWS policies and API calls.

Automatic Key Rotation: An AWS KMS feature that automatically rotates a CMK on a scheduled basis. This helps maintain security by periodically generating new cryptographic material for the CMK.

Key Management Service (KMS)

Encryption Context: An optional set of key-value pairs that you can include when encrypting data to provide additional data integrity checks. The encryption context is not encrypted but is cryptographically bound to the ciphertext.

FIPS 140-2 Compliance: AWS KMS is compliant with FIPS 140-2, a U.S. government standard that specifies security requirements for cryptographic modules. AWS KMS provides FIPS-validated endpoints for customers requiring higher security.

Keep in mind...!!

- **Regional service (keys are bound to a region)**
- Provides encryption and decryption of data and manages keys required for it
- Encrypted secrets can be stored in the code or environment variables
- **Encrypt up to 4KB of data per API call (if data > 4 KB, use envelope encryption)**
- Integrated with IAM for authorization
- Audit key usage with CloudTrail
- Need to set **IAM Policy & Key Policy** to allow a user or role to access a KMS key (encrypt or decrypt data using the key)
- Pay for the number of API calls made to KMS
- **Does not support versioning of keys** (cannot get back the old key)

Key Management Service (KMS)

KMS Keys (formerly Customer Master Key)

Symmetric keys

- AES-256 encryption
- **Must call KMS API to encrypt data**
- Necessary for Envelope Encryption
- Two types:
    - **AWS Managed Keys** (free)
        - Default KMS key for each supported service
        - Fully managed by AWS (cannot view, rotate or delete them)
        - Automatic yearly rotation
    - **Customer Managed Keys** (1$ per month)
        - Generated in KMS
            - Optional automatic yearly rotation
        - Generated and imported from outside
            - Must be 256-bit symmetric key
            - Not recommended
            - Manual rotation only
        - Deletion has a waiting period (**pending deletion state**) between **7 - 30 days** (default 30 days). The key can be recovered during the pending deletion state.

Asymmetric Keys

- Public (Encrypt) and Private Key (Decrypt) pair
- Used for Encrypt/Decrypt, or Sign/Verify operations
- The public key is downloadable, but you can't access the Private Key unencrypted
- **No need to call the KMS API to encrypt data** (data can be encrypted by the client)
- **Not eligible for automatic rotation** (use manual rotation)
- Use case: encryption outside of AWS by users who can't call the KMS API

Key Management Service (KMS)

Key Policies

- **Cannot access KMS keys without a key policy attached to them**

Default Key Policy

- Created by default if you don't provide a custom key policy
- Full access to the key for any user or role in the account (most permissible)

Custom Key Policy

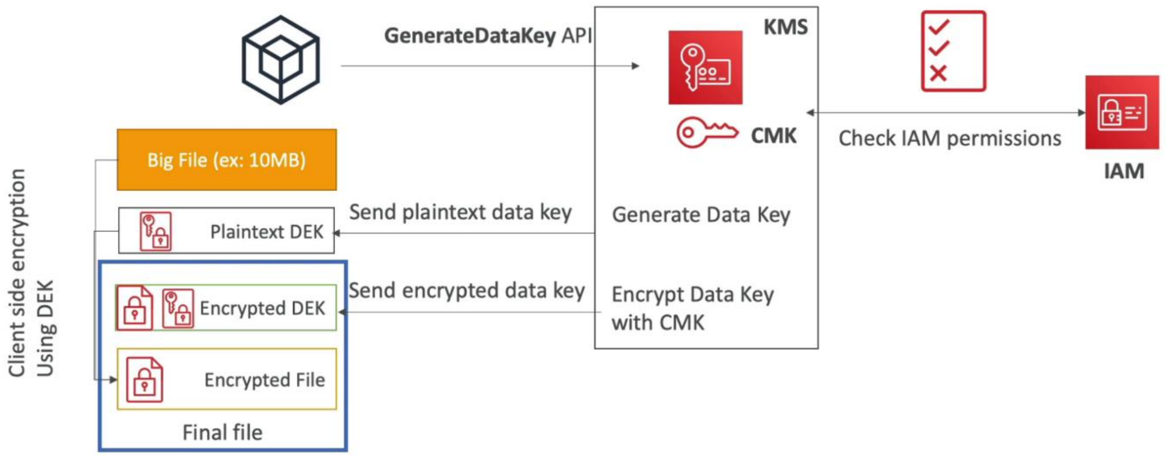
- **Can only be applied to customer owned keys**
- Define users, roles that can access the KMS key
- Define who can administer the key
- Useful for **cross-account access** of your KMS key

Key Management Service (KMS)

Whatever we discussed till this moment is useful when we are dealing with the small sets of data, when we have large sets of data, we go for the Envelop Encryption

Envelop Encryption – has 2 steps as usual encryption and decryption

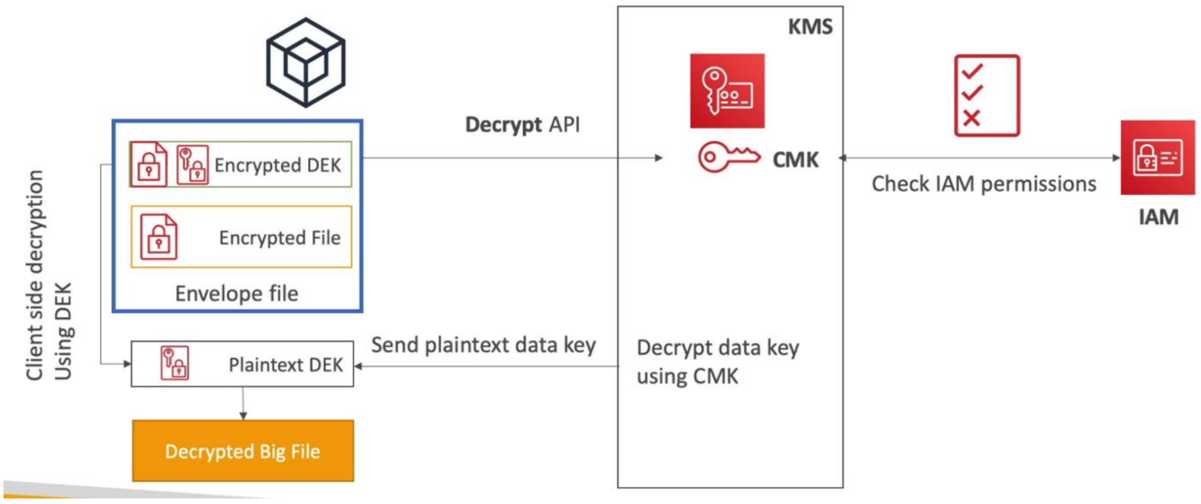
Encryption Phase



To encrypt a file larger than 4 KB, we call the GenerateDataKey API which returns a **Data Encryption Key (DEK)** in both plaintext and encrypted form (using the KMS key specified in the command, requires the user to have IAM permissions to encrypt using the KMS key). The DEK (symmetric) is used to **encrypt the large file client-side**. Then, an envelope is created which contains the encrypted file as well as the encrypted DEK. The data is now encrypted and the plaintext DEK can be discarded.

Key Management Service (KMS)

Decryption Phase



The encrypted DEK (< 4 KB) is extracted from the envelope and passed to the Decrypt API which returns the decrypted DEK (only if the user has permissions to use the KMS key to decrypt data). The plaintext DEK is then used to **decrypt the large file client-side**.

Key Management Service (KMS)

Let me simplify for you both encryption and decryption

Steps in Envelope Encryption:

1. **Generate a Data Encryption Key (DEK):**
    - First, a **Data Encryption Key (DEK)**, typically a **symmetric key** (like AES-256), is generated. This key will be used to encrypt the actual data (e.g., files, database contents, etc.).
2. **Encrypt the Data with the DEK:**
    - The generated DEK is used to **encrypt the data.** This is efficient because symmetric key encryption is fast and well-suited for encrypting large data.
3. **Encrypt the DEK with a Master Key (CMK):**
    - The DEK itself is **encrypted** using a **Customer-Managed Key (CMK)** or an **AWS-Managed Key** in **AWS KMS.** This second layer of encryption protects the DEK, ensuring that even if someone gains access to the encrypted data and the DEK, they won't be able to decrypt the data without access to the **CMK.**
4. **Store the Encrypted DEK:**
    - The **encrypted DEK** (the output from KMS) is stored alongside the encrypted data. This way, whenever the data needs to be decrypted, the encrypted DEK can be retrieved and **decrypted** using KMS to obtain the original DEK.
5. **Decrypting the Data:**
    - When you need to access the encrypted data:
        1. The **encrypted DEK** is sent to KMS, which decrypts it using the **Customer-Managed Key (CMK).**
        2. The decrypted DEK is used to **decrypt the actual data.**

Key Management Service (KMS)

Main Parameters to keep in mind

APIs

- Encrypt - encrypt up to 4 KB of data
- GenerateDataKey - generate a unique **symmetric** DEK for **Envelope Encryption** (returns both plaintext and encrypted data key using the KMS key specified in the command)
- GenerateDataKeyWithoutPlaintext - generate a DEK to use in the **future** (returns the encrypted DEK only)
- Decrypt - decrypt up to 4 KB of data (could be DEK)
- GenerateRandom - generate a random byte string.

AWS Encryption SDK

- **Implements Envelope Encryption** (difficult to implement manually)
- Available in Java, Python, C, JS and also as a CLI tool
- **Data Key Caching**
    - Re-use data keys (DEK) (instead of generating them for each encryption)
    - **Reduces the number of API calls to KMS** (cheaper) but not as secure as generating distinct data keys for each encryption.
    - Leverages LocalCryptoMaterialsCache feature

Limits

- All cryptographic operations within the AWS account that leverage keys managed by KMS (eg. SSE-KMS) share the same request quota (on a per second basis)
- Exceeding the request quota gives ThrottlingException (status code: 400). Possible solutions:
    - Exponential backoff
    - For GenerateDataKey, **enable DEK Caching** in Encryption SDK
    - Request quota increase
- Example: if a lot of objects are being uploaded to an S3 bucket with SSE-KMS, there can be performance degradation due to KMS API throttling.

Key Management Service (KMS)

Why we need to go for Envelop Encryption (CMK + DMK) for the large datasets

1. Efficiency and Performance

- **CMKs and Data Size Limitations**: CMKs are designed for managing encryption keys and are not optimized for encrypting large volumes of data. Encrypting large data sets directly with a CMK would involve multiple interactions with AWS KMS, which can be slow and costly.
- **Envelope Encryption**: With envelope encryption, the CMK encrypts a much smaller DEK. The DEK is then used to encrypt the actual large data set. This approach is more efficient because encryption operations on large data sets are handled locally using the DEK.

2. Cost Considerations

- **KMS Operations Cost**: AWS KMS charges for each cryptographic operation (e.g., encrypt, decrypt). If you encrypt large data sets directly with a CMK, the cost would be high due to frequent KMS interactions.
- **Envelope Encryption**: By encrypting large data with the DEK (and only using the CMK to encrypt and decrypt the DEK), you significantly reduce the number of KMS operations, thus lowering costs.

3. Scalability

- **Handling Large Data**: Encrypting large data directly with a CMK can become a bottleneck as the data size increases. KMS is optimized for managing and rotating encryption keys, not for performing large-scale data encryption.
- **Envelope Encryption**: Allows you to handle large data sets efficiently. You only interact with KMS for key management (encrypting and decrypting DEKs), while the actual data encryption and decryption are handled using the DEK.

4. Security and Key Management

- **CMK Role**: The CMK is responsible for key management, including key rotation, access control, and auditing. It's designed to be secure and managed centrally.
- **DEK Role**: The DEK is a temporary key used for encrypting data. Since the DEK is used for encryption and decryption of data locally, it is kept in memory and is discarded after use. This minimizes exposure of sensitive data encryption keys.

Key Management Service (KMS)

Scenario based Questions

https://lisireddy.medium.com/aws-kms-key-scenario-based-questions-7e54052e6bbb