AWS KMS Envelope Encryption Using Python3

**Import all the libraries:**

|  |
| --- |
| import os  from Crypto.Cipher import AES  from Crypto.Util.Padding import pad  import boto3  from Crypto.Util.Padding import unpad |

**Generate Data Key:**

|  |
| --- |
| def generate\_data\_key(key\_id, key\_spec='AES\_256'):  """  Generate a data key using AWS KMS  :param key\_id: ID or ARN of the KMS key  :param key\_spec: The length of the data key  :return: Plaintext and ciphertext data key  """  kms\_client = boto3.client('kms')  # Generate data key  response = kms\_client.generate\_data\_key(  KeyId=key\_id,  KeySpec=key\_spec  )  return response['Plaintext'], response['CiphertextBlob'] |

**Encrypt File function:**

|  |
| --- |
| def encrypt\_file(file\_path, plaintext\_key):  """  Encrypts a file using AES encryption  :param file\_path: Path to the file to be encrypted  :param plaintext\_key: Plaintext data key  """  # Generate a random initialization vector (IV)  iv = os.urandom(16)  # Read the plaintext from the file  with open(file\_path, 'rb') as file:  plaintext = file.read()  # Pad the plaintext to be a multiple of 16 bytes (AES block size)  plaintext = pad(plaintext, AES.block\_size)  # Create AES cipher object  cipher = AES.new(plaintext\_key, AES.MODE\_CBC, iv)  # Encrypt the plaintext  ciphertext = cipher.encrypt(plaintext)  # Write the encrypted data (ciphertext + IV) to a new file  encrypted\_file\_path = file\_path + '.enc'  with open(encrypted\_file\_path, 'wb') as encrypted\_file:  encrypted\_file.write(ciphertext)  encrypted\_file.write(iv)  print(f"File '{file\_path}' encrypted successfully. Encrypted file saved as '{encrypted\_file\_path}'") |

**To Encrypt the file, use the main code.**

|  |
| --- |
| # Replace 'key-id' with the actual ID or ARN of your KMS key  key\_id = 'arn:aws:kms:ap-south-1:6xxxxxxxxxxxxxx3:key/6985a680-150b-4f17-a613-b31ac8eb64c9'  # Generate plaintext data key  plaintext\_key, \_ = generate\_data\_key(key\_id)  # Replace 'file\_path' with the path to your file  file\_path = 'demo\_file.txt'  # Encrypt the file  encrypt\_file(file\_path, plaintext\_key) |

**Decrypt File function:**

|  |
| --- |
| def decrypt\_file(encrypted\_file\_path, plaintext\_key):  with open(encrypted\_file\_path, 'rb') as encrypted\_file:  # Read the ciphertext  ciphertext = encrypted\_file.read()  # Separate the ciphertext and the IV  iv = ciphertext[-16:]  ciphertext = ciphertext[:-16]  # Create AES cipher object  cipher = AES.new(plaintext\_key, AES.MODE\_CBC, iv)  # Decrypt the ciphertext  decrypted\_data = cipher.decrypt(ciphertext)  # Remove the padding  decrypted\_data = unpad(decrypted\_data, AES.block\_size)  return decrypted\_data |

**To decrypt the file, use the main code.:**

|  |
| --- |
| # Replace 'key-id' with the actual ID or ARN of your KMS key  key\_id = 'arn:aws:kms:ap-south-1:6xxxxxxxxxxxxxx3:key/6985a680-150b-4f17-a613-b31ac8eb64c9'  # Generate plaintext data key  plaintext\_key, \_ = generate\_data\_key(key\_id)  # Call the decrypt\_file function with the encrypted file path and the plaintext data key  decrypted\_data = decrypt\_file(‘demo\_file.txt.enc', plaintext\_key)  # output\_location  file\_path = "output.txt"  # Open the file in write mode ('w')  with open(file\_path, "w") as file:  # Write the decoded data to the file  file.write(decrypted\_data.decode('utf-8'))  # Print a message to indicate that the file has been saved  print("Data has been saved to", file\_path) |