AWS KMS Envelope Encryption Using

Android (Java):

**Import all the libraries:**

import android.os.AsyncTask**;**

import android.util.Base64**;**

import com.google.gson.Gson**;**

import com.google.gson.JsonObject**;**

import com.google.gson.JsonParser**;**

import java.io.BufferedReader**;**

import java.io.InputStreamReader**;**

import java.io.OutputStream**;**

import java.net.HttpURLConnection**;**

import java.net.URL**;**

import java.nio.charset.StandardCharsets**;**

import java.util.HashMap**;**

import java.util.Map**;**

import javax.crypto.Cipher**;**

import javax.crypto.spec.IvParameterSpec**;**

import javax.crypto.spec.SecretKeySpec**;**

**Add dependencies in build.gradel(app) to load these libraries and change the compile sdk to 34 and give Internet permission:**

**dependencies {**

**implementation 'androidx.appcompat:appcompat:1.6.1'**

**implementation 'com.google.android.material:material:1.11.0'**

**implementation 'androidx.constraintlayout:constraintlayout:2.1.4'**

**testImplementation 'junit:junit:4.13.2'**

**androidTestImplementation 'androidx.test.ext:junit:1.1.5'**

**androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'**

**implementation 'com.android.volley:volley:1.2.0'**

**implementation 'com.google.code.gson:gson:2.8.8'**

**}**

**Create a Class and add the below method into it and call this class in MainActivity.java**

**Encrypt Method:**

**public static String encryptText(String plaintext, byte[] plaintextKey, String bsy) {**

**try {**

**// Generate a new Initialization Vector (IV) calling generateIV() method**

**IvParameterSpec iv = new IvParameterSpec(*generateIV*());**

**// Create a SecretKeySpec using the provided plaintextKey and AES algorithm**

**SecretKeySpec secretKey = new SecretKeySpec(plaintextKey, "AES");**

**// Instantiate a Cipher instance with AES/CBC/PKCS5Padding transformation**

**Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");**

**// Initialize the cipher for encryption with the secretKey and IV**

**cipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, iv);**

**// Encrypt the plaintext and obtain the ciphertext**

**byte[] encrypted = cipher.doFinal(plaintext.getBytes(StandardCharsets.*UTF\_8*));**

**// Concatenate encrypted data with additional data (cipherkey) and IV**

**byte[] ciphertextWithIV = *concatenateByteArrays*(encrypted,**

**bsy.getBytes(StandardCharsets.*UTF\_8*), iv.getIV());**

**// Encode the concatenated ciphertext with Base64**

**return Base64.*encodeToString*(ciphertextWithIV, Base64.*DEFAULT*);**

**} catch (Exception e) {**

**// Print the stack trace if any exception occurs**

**e.printStackTrace();**

**}**

**// Return null if encryption fails**

**return null;**

**}**

**Decrypt Method:**

**public static String decryptText(String encryptedText) {**

**try {**

**// Decode the Base64 encoded encrypted text**

**byte[] decodedBytes = Base64.*decode*(encryptedText, Base64.*DEFAULT*);**

**// Extract the IV (Initialization Vector) from the decoded bytes**

**byte[] iv = new byte[16];**

**System.*arraycopy*(decodedBytes, decodedBytes.length - 16, iv, 0, 16);**

**// Extract the ciphertext without the IV and cipher key**

**byte[] ciphertext = new byte[decodedBytes.length - 16];**

**System.*arraycopy*(decodedBytes, 0, ciphertext, 0, decodedBytes.length - 16);**

**// Extract the cipher key from the ciphertext**

**byte[] cipherKey = new byte[248];**

**System.*arraycopy*(ciphertext, ciphertext.length - 248, cipherKey, 0, 248);**

**// Extract the ciphertext without the cipher key**

**byte[] cipherTextWithoutKey = new byte[ciphertext.length - 248];**

**System.*arraycopy*(ciphertext, 0, cipherTextWithoutKey, 0, ciphertext.length - 248);**

**// Convert the cipher key to plaintext key and decode it**

**String plaintextKey = new String(cipherKey, StandardCharsets.*UTF\_8*);**

**String key = *deccode*(plaintextKey, "DE");**

**byte[] bytePlaintext = Base64.*decode*(key, Base64.*DEFAULT*);**

**// Create a SecretKeySpec using the plaintext key and AES algorithm**

**SecretKeySpec secretKey = new SecretKeySpec(bytePlaintext, "AES");**

**// Initialize a Cipher instance for decryption with the secretKey and IV**

**Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");**

**cipher.init(Cipher.*DECRYPT\_MODE*, secretKey, new IvParameterSpec(iv));**

**// Decrypt the ciphertext without the cipher key**

**byte[] decryptedBytes = cipher.doFinal(cipherTextWithoutKey);**

**// Convert the decrypted bytes to plaintext and trim any trailing spaces**

**return new String(decryptedBytes, StandardCharsets.*UTF\_8*).trim();**

**} catch (Exception e) {**

**// Print the stack trace if any exception occurs**

**e.printStackTrace();**

**}**

**// Return null if decryption fails**

**return null;**

**}**

**IV Method:**

**/\*\***

**\* Generates a random Initialization Vector (IV) of length 16 bytes.**

**\***

**\* @return byte array representing the randomly generated IV**

**\*/**

**private static byte[] generateIV() {**

**// Create a byte array to store the IV**

**byte[] iv = new byte[16];**

**// Fill the byte array with random values**

**for (int i = 0; i < 16; ++i) {**

**iv[i] = (byte) (Math.*random*() \* 256);**

**}**

**// Return the generated IV**

**return iv;**

**}**

**Concatenates Method:**

**/\*\***

**\* Concatenates multiple byte arrays into a single byte array.**

**\***

**\* @param arrays byte arrays to concatenate**

**\* @return concatenated byte array**

**\*/**

**public static byte[] concatenateByteArrays(byte[]... arrays) {**

**// Calculate the total length of the concatenated byte array**

**int totalLength = 0;**

**for (byte[] array : arrays) {**

**totalLength += array.length;**

**}**

**// Create a byte array to store the concatenated result**

**byte[] result = new byte[totalLength];**

**// Copy each byte array into the result byte array**

**int currentIndex = 0;**

**for (byte[] array : arrays) {**

**// Copy the current array into the result array**

**System.*arraycopy*(array, 0, result, currentIndex, array.length);**

**// Update the current index to point to the end of the copied array**

**currentIndex += array.length;**

**}**

**// Return the concatenated byte array**

**return result;**

**}**

**“deccode” Method:**

**/\*\***

**\* Sends a request to a specified URL to encode or decode data.**

**\***

**\* @param data the data to encode or decode**

**\* @param kms the type of operation to perform ("EN" for encode, "DE" for decode)**

**\* @return the encoded or decoded data, or null if the operation fails**

**\*/**

**public static String deccode(String data, String kms) {**

**// URL for the API endpoint**

**String url = "https://45i44ltezrqisbucs7rnahe24q0zbwnx.lambda-url.ap-south-1.on.aws/";**

**try {**

**// Create a URL object for the API endpoint**

**URL apiUrl = new URL(url);**

**// Open a connection to the API endpoint**

**HttpURLConnection connection = (HttpURLConnection) apiUrl.openConnection();**

**// Set the request method to GET**

**connection.setRequestMethod("GET");**

**// Set the content type of the request to JSON**

**connection.setRequestProperty("Content-Type", "application/json");**

**// Enable output for the connection**

**connection.setDoOutput(true);**

**// Create a map to hold JSON key-value pairs**

**Map<String, Object> jsonMap = new HashMap<>();**

**// Fill the map based on the operation type**

**if (kms.equals("EN")) {**

**// For encoding, set type to "EN" and data to null**

**jsonMap.put("type", "EN");**

**jsonMap.put("data", null);**

**} else if (kms.equals("DE")) {**

**// For decoding, set type to "DE" and data to the provided data**

**jsonMap.put("type", kms);**

**jsonMap.put("data", data);**

**}**

**// Convert the map to a JSON string**

**String jsonInputString = *mapToJsonString*(jsonMap);**

**// Write the JSON input string to the output stream of the connection**

**try (OutputStream os = connection.getOutputStream()) {**

**byte[] input = jsonInputString.getBytes(StandardCharsets.*UTF\_8*);**

**os.write(input, 0, input.length);**

**}**

**// Get the response code from the connection**

**int responseCode = connection.getResponseCode();**

**// If the response code indicates success (HTTP\_OK), read and return the response**

**if (responseCode == HttpURLConnection.*HTTP\_OK*) {**

**BufferedReader in = new BufferedReader(new**

**InputStreamReader(connection.getInputStream()));**

**StringBuilder response = new StringBuilder();**

**String line;**

**while ((line = in.readLine()) != null) {**

**response.append(line);**

**}**

**in.close();**

**return response.toString();**

**} else {**

**// If the response code indicates failure, print an error message**

**System.*out*.println("Failed to call API. Status code: " + responseCode);**

**}**

**} catch (Exception e) {**

**// Print the stack trace if any exception occurs**

**e.printStackTrace();**

**}**

**// Return null if the operation fails**

**return null;**

**}**

**JSON Formatter Method:**

**/\*\***

**\* Converts a map of key-value pairs to a JSON-formatted string.**

**\***

**\* @param map the map to convert to JSON**

**\* @return a JSON-formatted string representing the map**

**\*/**

**private static String mapToJsonString(Map<String, Object> map) {**

**// Create a StringBuilder to build the JSON string**

**StringBuilder jsonString = new StringBuilder("{");**

**// Iterate over each entry in the map**

**for (Map.Entry<String, Object> entry : map.entrySet()) {**

**// Append the key surrounded by double quotes**

**jsonString.append("\"").append(entry.getKey()).append("\":");**

**// Check if the value is a String**

**if (entry.getValue() instanceof String) {**

**// If the value is a String, surround it with double quotes**

**jsonString.append("\"").append(entry.getValue()).append("\"");**

**} else {**

**// If the value is not a String, append it directly**

**jsonString.append(entry.getValue());**

**}**

**// Add a comma after each key-value pair except the last one**

**jsonString.append(",");**

**}**

**// Remove the trailing comma and close the JSON object**

**jsonString.deleteCharAt(jsonString.length() - 1);**

**jsonString.append("}");**

**// Return the JSON-formatted string**

**return jsonString.toString();**

**}**

**Main Method:**

**public static void main() {**

**// Example usage:**

**String data = "Test Data To encrypt"; //Pass the data to encrypt**

**String kms = "EN"; // "EN" for encryption**

**new KMSAsyncTask().execute(data, kms);**

**}**

**static class KMSAsyncTask extends AsyncTask<String, Void, String> {**

**@Override**

**protected String doInBackground(String... params) {**

**String data = params[0];**

**String kms = params[1];**

**String getKey = *deccode*(data, kms);**

**// Convert decoded data to JSON**

**Gson gson = new Gson();**

**JsonObject jsonObject = JsonParser.*parseString*(getKey).getAsJsonObject();**

**// Extract and print CiphertextBlob and PlainText**

**String ciphertextBlob = jsonObject.get("CiphertextBlob").getAsString();**

**String plainText = jsonObject.get("PlainText").getAsString();**

**// Decode the PlainText**

**byte[] decodedBytes = Base64.*decode*(plainText, Base64.*DEFAULT*);**

**// Encrypt the data**

**String encryptedData = *encryptText*(data, decodedBytes, ciphertextBlob);**

**System.*out*.println("Encrypted Data: " + encryptedData);**

**return *decryptText*(encryptedData);**

**}**

**// Decrypt the encrypted data**

**@Override**

**protected void onPostExecute(String result) {**

**super.onPostExecute(result);**

**System.*out*.println("Decrypted Data: " + result);**

**}**

**}**

**}**

**AWS Lambda/API code for the URL used in “deccode” method:**

**import json**

**import boto3**

**import base64**

**# AWS KMS key ID**

**key\_id = "alias/AES-256-encryption-demo" kms ARN**

**# Initialize AWS KMS client**

**client = boto3.client('kms', region\_name='ap-south-1')**

**# Function to generate a data key using AWS KMS**

**def generate\_data\_key(key\_id, key\_spec="AES\_256"):**

**response = client.generate\_data\_key(KeyId=key\_id, KeySpec=key\_spec)**

**data = {**

**"PlainText": base64.b64encode(response['Plaintext']),**

**"CiphertextBlob": base64.b64encode(response['CiphertextBlob'])**

**}**

**return data**

**# Lambda handler function**

**def lambda\_handler(event, context):**

**# Extract symmetric type from the HTTP request body**

**symmetrictype = json.loads(event["body"])["type"]**

**# Check if it's an encryption request**

**if symmetrictype == "EN":**

**# Generate a data key and return plaintext and ciphertext versions**

**return generate\_data\_key(key\_id)**

**# Check if it's a decryption request**

**elif symmetrictype == "DE":**

**# Extract encrypted text from the request body**

**encrypted\_text = json.loads(event["body"])["data"]**

**# Decrypt the text using AWS KMS**

**decrypted = client.decrypt(CiphertextBlob=base64.b64decode(encrypted\_text))**

**# Return the plaintext version of the decrypted data**

**return base64.b64encode(decrypted["Plaintext"])**

**AWS KMS Key Setup:**

**Access AWS Management Console:**

* Log in to your AWS Management Console.

**Navigate to AWS KMS:**

* Go to the AWS Key Management Service (KMS) console.

**Create a Customer Master Key (CMK):**

* Click on "Create key" to create a new customer master key.
* Select "Symmetric" key type.
* Choose the key administrator (the AWS account that will manage the key).
* Optionally, you can add tags to the key for better organization.
* Click "Next" to proceed.

**Configure Key Options:**

* Provide a key alias (a human-readable name for the key).
* Define key administrative permissions.
* Define key usage permissions.
* Click "Next" to proceed.

**Review and Create:**

* Review the key configuration details.
* If everything looks correct, click "Finish" or "Create key" to create the symmetric key.

**Create Your Lambda Function:**

* Go to the AWS Management Console and navigate to the Lambda service.
* Click on "Create function".
* Fill in the necessary details like Function name, Runtime (e.g., Python), and Execution role (add KMS Related Role here).
* Upload the above given code into this function.
* Once your function is created, note down its ARN (Amazon Resource Name) for later use.

**Create an API Gateway:**

* Navigate to the API Gateway service in the AWS Management Console.
* Click on "Create API".
* Choose the type of API you want to create (ex. REST).
* Select "New API" and provide a name for your API.
* Click on "Create API".

**Create a Resource and Method:**

* After creating the API, you'll see a list of resources. Click on "Actions" and select "Create Resource".
* Enter a Resource Name and Resource Path.
* Click on "Create Resource".
* Now, with the resource selected, click on "Actions" and choose "Create Method".
* Select the HTTP method (GET).
* Choose "Lambda Function" as the integration type.
* Select the Lambda Region where your function is deployed and provide the Lambda Function ARN.
* Click on "Save" to integrate your Lambda function with the API Gateway.

**Deploy Your API:**

* After integrating the method with your Lambda function, click on "Actions" and select "Deploy API".
* Choose the Deployment Stage (e.g., "Prod").
* Click on "Deploy" to deploy your API.

**Test Your API:**

* Once deployed, you'll see an Invoke URL for your API.
* Use tools like c-URL, Postman, or simply a web browser to send requests to your API and verify the responses.