aws

Developer Guide

Amazon S3 Encryption Client

Amazon S3 Encryption Client: Developer Guide

Table of Contents

What is the Amazon S3 Encryption Client?

> ⓘ **Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an independent library. For information about previous versions of the Amazon S3 Encryption Client, see the AWS SDK Developer Guide for your programming language.

The Amazon S3 Encryption Client is a client-side encryption library that enables you to encrypt an object locally to ensure its security before passing it to [Amazon Simple Storage Service](#) (Amazon S3). Amazon S3 receives your object already encrypted; it does not play a role in encrypting or decrypting it. After you instantiate the Amazon S3 Encryption Client, your objects are automatically encrypted and decrypted as part of your Amazon S3 `PutObject` and `GetObject` requests. The Amazon S3 Encryption Client is provided free of charge under the Apache 2.0 license.

The Amazon S3 Encryption Client is supported in the following programming languages and platforms. This guide focuses on version 3.*x* of the [Amazon S3 Encryption Client for Java](#) and [Amazon S3 Encryption Client for Go](#). For more information on the remaining language implementations, see their respective AWS SDK Developer Guides.

- C++ ([AWS SDK for C++](#))
- Go ([amazon-s3-encryption-client-go](#) repository)
- Java ([amazon-s3-encryption-client-java](#) repository)
- .NET (v2) ([AWS SDK for .NET](#))
- Ruby (v2) ([AWS SDK for Ruby](#))
- PHP (v3) ([AWS SDK for PHP](#))

The Amazon S3 Encryption Client provides:

A default implementation that adheres to cryptography best practices

By default, the Amazon S3 Encryption Client generates a unique data key for each object that it encrypts. This follows the cryptography best practice of using unique data keys for each encryption operation.

The Amazon S3 Encryption Client encrypts your objects using a secure, authenticated, symmetric key algorithm.

A framework for protecting data keys with wrapping keys

The Amazon S3 Encryption Client protects the data keys that encrypt your objects by encrypting them under a [wrapping key](#). With the Amazon S3 Encryption Client, you define a wrapping key by passing the key to the Amazon S3 Encryption Client, which it uses to optimize its settings.

Amazon S3 Encryption Client concepts

> **ⓘ Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an independent library. For information about previous versions of the Amazon S3 Encryption Client, see the AWS SDK Developer Guide for your programming language.

This topic introduces the concepts and terminology used in the Amazon S3 Encryption Client. It's designed to help you understand how the Amazon S3 Encryption Client works and the terms we use to describe it.

Topics

- [Envelope encryption](#)
- [Data key](#)
- [Wrapping key](#)
- [Keyrings](#)
- [Supported encryption algorithms](#)
- [Cryptographic materials manager](#)
- [Encryption context](#)

Envelope encryption

The security of your encrypted object depends in part on protecting the data key that can decrypt it. One accepted best practice for protecting the data key is to encrypt it. To do this, you need

another encryption key, known as a *key-encryption key* or [wrapping key](#). The practice of using a wrapping key to encrypt data keys is known as *envelope encryption*.

Protecting data keys

The Amazon S3 Encryption Client encrypts each object with a unique [data key](#). Then it encrypts the data key under the wrapping key you specify. It stores the encrypted data key with the encrypted object that the `PutObject` request uploads to Amazon S3.
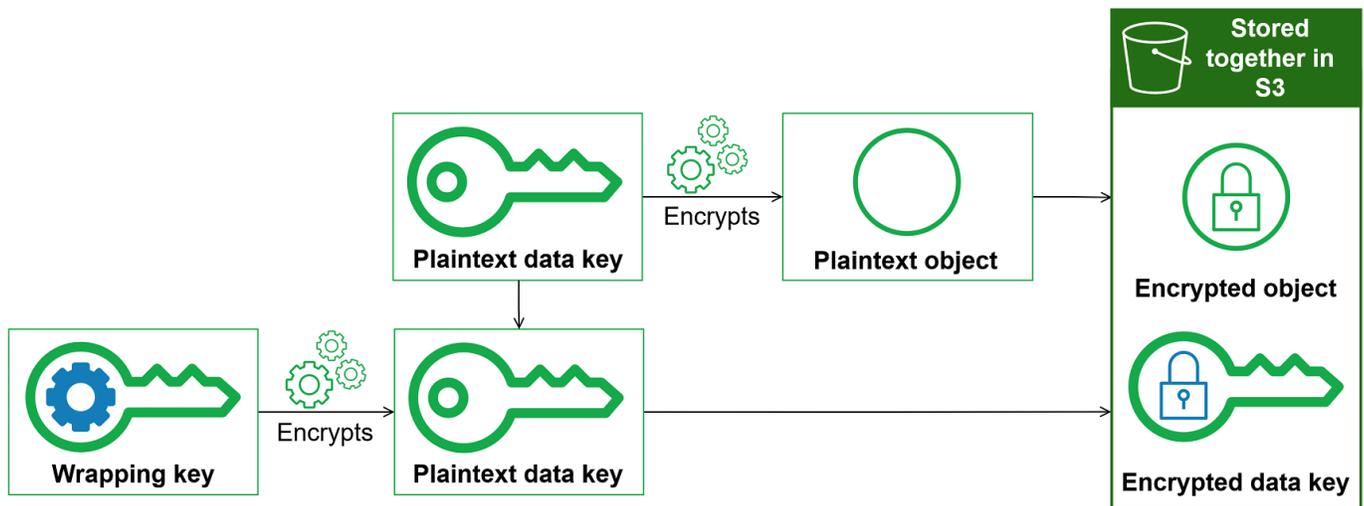
Combining the strengths of multiple algorithms

To encrypt your object, by default, the Amazon S3 Encryption Client uses a sophisticated algorithm suite with AES-GCM symmetric encryption. To encrypt the data key, you can specify a symmetric or asymmetric encryption algorithm appropriate to your wrapping key.

In general, symmetric key encryption algorithms are faster and produce smaller ciphertexts than asymmetric or *public key encryption*. But public key algorithms provide inherent separation of roles and easier key management. To combine the strengths of each, you can encrypt your object with symmetric key encryption, and then encrypt the data key with public key encryption.

Data key

A *data key* is an encryption key that the Amazon S3 Encryption Client uses to encrypt your object. Each data key is a byte array that conforms to the requirements for cryptographic keys. The Amazon S3 Encryption Client uses a unique data key to encrypt each object.

You don't need to specify, generate, implement, extend, protect or use data keys. The Amazon S3 Encryption Client does that work for you.

To protect your data keys, the Amazon S3 Encryption Client encrypts them under a *key-encryption key* known as a wrapping key. When you call `PutObject`, the Amazon S3 Encryption Client uses your plaintext data key to encrypt your object, then removes it from memory as soon as possible. The Amazon S3 Encryption Client encrypts the data key with the wrapping key you provide. Then the Amazon S3 Encryption Client stores the encrypted data key with the encrypted object that the `PutObject` request uploads to Amazon S3. For more information, see How the Amazon S3 Encryption Client works.

Wrapping key

A *wrapping key* is a key-encryption key that the Amazon S3 Encryption Client uses to encrypt the data key that encrypts your object. You specify the wrapping key that is used to protect your data keys when you instantiate your Amazon S3 Encryption Client. Version 3.*x* of the Amazon S3 Encryption Client uses the wrapping key you specify and one of the fully supported wrapping algorithms to encrypt and decrypt data keys.
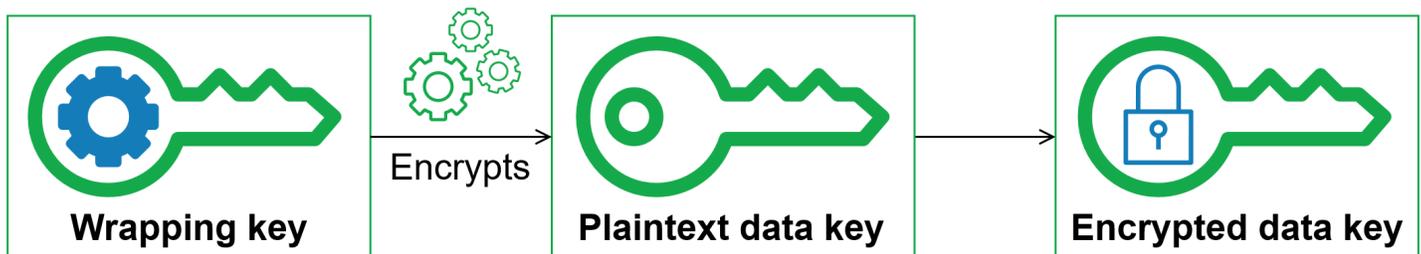
The Amazon S3 Encryption Client supports several commonly used wrapping keys, such as symmetric AWS KMS keys, Raw AES-GCM (Advanced Encryption Standard/Galois Counter Mode) keys, and Raw RSA keys.

> **ⓘ Note**
>
> Version 3.*x* of the Amazon S3 Encryption Client for Go does not support Raw AES-GCM or Raw RSA wrapping keys.

When you use envelope encryption, you need to protect your wrapping keys from unauthorized access. You can do this in any of the following ways:

- Use a web service designed for this purpose, such as AWS Key Management Service (AWS KMS).

- Use a [hardware security module (HSM)](#) such as those offered by [AWS CloudHSM](#).
- Use other key management tools and services.

If you don't have a key management system, we recommend AWS KMS. The Amazon S3 Encryption Client integrates with AWS KMS to help you protect and use your wrapping keys.

Keyrings

To specify the wrapping keys you use for encryption and decryption, you use a keyring You can use the keyrings that the Amazon S3 Encryption Client provides or design your own implementations. The Amazon S3 Encryption Client provides keyrings that are compatible with each other subject to language constraints.

A *keyring* generates, encrypts, and decrypts data keys. When you define a keyring, you can specify the [wrapping keys](#) that encrypt your data keys. Most keyrings specify at least one wrapping key or a service that provides and protects wrapping keys. You can also define a keyring with no wrapping keys or a more complex keyring with additional configuration options.

For details about specifying wrapping keys, see the examples topic of your [programming language](#).

Supported encryption algorithms

An *algorithm suite* is a collection of cryptographic algorithms and related values. Cryptographic systems use the algorithm implementation to generate the ciphertext message.

To encrypt each Amazon S3 object, the Amazon S3 Encryption Client uses a unique 256-bit symmetric data encryption key and an Advanced Encryption Standard (AES) with Galois/Counter Mode (GCM) algorithm suite. This algorithm suite uses AES-GCM for authenticated encryption with a 12-byte initialization vector, and a 16-byte AES-GCM authentication tag. It does not support a key derivation function.

For information on legacy encryption algorithms, see [Supported encryption algorithms](#).

Cryptographic materials manager

The cryptographic materials manager (CMM) assembles the cryptographic materials that are used to encrypt and decrypt data. The *cryptographic materials* include plaintext and encrypted data keys, and an optional message signing key. You never interact with the CMM directly. The encryption and decryption methods handle it for you.

You can use the default CMM that the Amazon S3 Encryption Client provides or write a custom CMM. You can specify a CMM, but it's not required. When you specify a keyring, the Amazon S3 Encryption Client creates a default CMM for you. The default CMM gets the encryption or decryption materials from the keyring that you specify. This might involve a call to a cryptographic service, such as [AWS Key Management Service](AWS KMS).

Because the CMM acts as a liaison between the Amazon S3 Encryption Client and a keyring, it is an ideal point for customization and extension, such as support for policy enforcement and caching.

Encryption context

If you use a symmetric AWS KMS key as your wrapping key, you can include an encryption context in all requests to encrypt data. Using an encryption context is optional, but it is a cryptographic best practice that we recommend.

An *encryption context* is a set of name-value pairs that contain arbitrary, non-secret additional authenticated data. The encryption context can contain any data you choose, but it typically consists of data that is useful in logging and tracking, such as data about the file type, purpose, or ownership. When you encrypt data, the encryption context is cryptographically bound to the encrypted data so that the same encryption context is required to decrypt the data. The Amazon S3 Encryption Client stores the encryption context in plaintext in the metadata of the encrypted object that it uploads to Amazon S3. The Amazon S3 Encryption Client also uses the encryption context to provide additional authenticated data (AAD) in your AWS KMS calls.

The following example demonstrates how to specify an encryption context in your cryptographic operations.

1. Specify a KMS key as your wrapping key by instantiating your client with the `kmsKeyId` builder parameter.

```
// v3

class v3KMSKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .kmsKeyId(kmsKeyId)
                .build();
    }
}
```

2. Use the `overrideConfiguration` builder parameter to specify the encryption context
    within your `PutObject` request.

```
// v3

class v3EncryptExample {
    public static void main(String[] args) {
        s3Client.putObject(PutObjectRequest.builder()
                        .bucket(bucket)
                        .key(objectKey)

 .overrideConfiguration(withAdditionalConfiguration(encryptionContext)),
                        .build(), RequestBody.fromString(objectContent));
    }
}
```

3. Include the same encryption context in your `GetObject` request.

```
// v3

class v3DecryptExample {
    public static void main(String[] args) {
        ResponseBytes<GetObjectResponse> objectResponse =
 s3Client.getObjectAsBytes(builder -> builder
                .bucket(bucket)
                .key(objectKey));

 .overrideConfiguration(withAdditionalConfiguration(encryptionContext)));
        String output = objectResponse.asUtf8String();
    }
}
```

How the Amazon S3 Encryption Client works

> ### ⓘ Note
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an
> independent library. For information about previous versions of the Amazon S3 Encryption
> Client, see the AWS SDK Developer Guide for your programming language.

The Amazon S3 Encryption Client is designed specifically to protect the data that you store in Amazon S3. The workflows in this section explain how the Amazon S3 Encryption Client encrypts and decrypts your objects.

The Amazon S3 Encryption Client uses envelope encryption to protect your objects. It encrypts each Amazon S3 object under a unique data encryption key. Then it encrypts the data encryption key with a wrapping key that you specify.

Need help with the terminology we use in the Amazon S3 Encryption Client? See the section called "Terms and concepts".

Encrypt and decrypt with the Amazon S3 Encryption Client

The Amazon S3 Encryption Client works as an intermediary between you and Amazon S3 by encrypting your object as you upload it, and decrypting your object as you download it. The following walkthrough specifies an RSA key pair as the wrapping key. For detailed code examples, see the *Examples* topic of your preferred programming language.

1. Specify your wrapping key and create a keyring when you instantiate your client.

2. Encrypt your plaintext object by calling `PutObject`.

    a. The Amazon S3 Encryption Client provides the encryption materials: one plaintext data key and one copy of that data key encrypted by your wrapping key.

    b. The Amazon S3 Encryption Client uses the plaintext data key to encrypt your object, and then discards the plaintext data key.

    c. The Amazon S3 Encryption Client uploads the encrypted data key and the encrypted object to Amazon S3 as part of the `PutObject` call.

3. Decrypt your encrypted object by calling `GetObject`.

    a. The Amazon S3 Encryption Client uses your wrapping key to decrypt the encrypted data key.

    b. The Amazon S3 Encryption Client uses the plaintext data key to decrypt the object, discards the plaintext data key, and returns the plaintext object as part of the `GetObject` call.

Client-side and server-side encryption

> **ⓘ Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an independent library. For information about previous versions of the Amazon S3 Encryption Client, see the AWS SDK Developer Guide for your programming language.

The Amazon S3 Encryption Client supports *client-side encryption*, where you encrypt your objects before you send them to Amazon S3. Amazon S3 provides server-side encryption options that encrypt your objects at their destination before they are saved in Amazon S3.

The tools that you choose depend on your security requirements and the sensitivity of your data. You can use both the Amazon S3 Encryption Client and Amazon S3 server-side encryption. When you send encrypted objects to Amazon S3, Amazon S3 doesn't recognize the objects as being encrypted, it just detects typical objects.

Server-side encryption

Amazon S3 supports encryption at rest with three mutually exclusive server-side encryption options. Amazon S3 encrypts your data at the object level as it writes it to disks in its data centers and decrypts it for you when you access it.

Amazon S3 Encryption Client

Client-side encryption provides end-to-end protection for your object, in transit and at rest, from its source to storage in Amazon S3.

- Your data is protected in transit and at rest. It is never exposed to any third party, including AWS.
- You choose how your cryptographic keys are protected. You specify the wrapping key used to protect the data keys that encrypt your objects.
- Your objects are all encrypted with a unique data key. The Amazon S3 Encryption Client does not use or interact with bucket keys, even if you specify a KMS key as your wrapping key.

Amazon S3 Encryption Client programming languages

The Amazon S3 Encryption Client is supported in the following programming languages and platforms. This guide focuses on version 3.*x* of the Amazon S3 Encryption Client for Java and Amazon S3 Encryption Client for Go. For more information on the remaining language implementations, see their respective AWS SDK Developer Guides.

- Java

- Go

- C++ (AWS SDK for C++)

- .NET (v2) (AWS SDK for .NET)

- Ruby (v2) (AWS SDK for Ruby)

- PHP (v3) (AWS SDK for PHP)

Amazon S3 Encryption Client for Java

> (i) **Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an independent library. For information about previous versions of the Amazon S3 Encryption Client, see the AWS SDK Developer Guide for your programming language.

This topic explains how to install and use the Amazon S3 Encryption Client for Java. For details about programming with the Amazon S3 Encryption Client for Java, see the amazon-s3-encryption-client-java repository on GitHub.

Topics

- Prerequisites

- Installation

- Amazon S3 Encryption Client for Java examples

- Asynchronous programming in the Amazon S3 Encryption Client for Java

- Migrating to version 3.x of the Amazon S3 Encryption Client for Java

Prerequisites

Before you install the Amazon S3 Encryption Client for Java, be sure you have the following prerequisites.

A Java development environment

You will need Java 8 or later. On the Oracle website, go to Java SE Downloads, and then download and install the Java SE Development Kit (JDK).

If you use the Oracle JDK, you must also download and install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

AWS SDK for Java 2.x

The Amazon S3 Encryption Client for Java requires the Amazon S3 module of the AWS SDK for Java 2.x. You can install the entire SDK or just this module.

For information about updating your version of the AWS SDK for Java, see Migrating from version 1.x to 2.x of the AWS SDK for Java.

To install the AWS SDK for Java, use Apache Maven.

- To import the entire AWS SDK for Java as a dependency, declare it in your `pom.xml` file.
- To create a dependency only for the Amazon S3 module in the AWS SDK for Java, follow the instructions for specifying particular modules. Set the `groupId` to `software.amazon.awssdk` and the `artifactID` to `s3`.

Installation

You can install the latest version of the Amazon S3 Encryption Client for Java in the following ways.

Manually

To install the Amazon S3 Encryption Client for Java, clone or download the amazon-s3-encryption-client-java GitHub repository.

Using Apache Maven

The Amazon S3 Encryption Client for Java is available through Apache Maven with the following dependency definition. Install the latest version offered.

```
<dependency>
   <groupId>software.amazon.encryption.s3</groupId>
   <artifactId>amazon-s3-encryption-client-java</artifactId>
   <version>3.x</version>
</dependency>
```

Optional dependencies

AWS KMS wrapping key

To use use a symmetric KMS key as your wrapping key, install the entire SDK or create a dependency for the AWS KMS module. To create a dependency, follow the instructions for [specifying particular modules](). Set the `groupId` to `software.amazon.awssdk` and the `artifactId` to `kms`.

Multipart upload (high-level API)

To perform multipart uploads with the [high-level API](), create dependencies for the AWS CRT-based Amazon S3 client. For help creating these dependencies, see [Add dependencies to use the AWS CRT-based Amazon S3 client]() in the *AWS SDK for Java 2.x Developer Guide*.

For more information on multipart uploads in the Amazon S3 Encryption Client, see [Multipart upload]().

Amazon S3 Encryption Client for Java examples

The following examples show you how to use the Amazon S3 Encryption Client for Java to encrypt and decrypt Amazon S3 objects. These examples show how to use version 3.*x* of the Amazon S3 Encryption Client for Java. For more detailed examples, see the [amazon-s3-encryption-client-java]() GitHub repository.

Topics

- [Instantiating the Amazon S3 Encryption Client]()
- [Encrypting and decrypting Amazon S3 objects]()
- [Ranged GET requests]()
- [Multipart upload]()

Instantiating the Amazon S3 Encryption Client

After installing the Amazon S3 Encryption Client for Java, you are ready to instantiate your client and begin encrypting and decrypting your Amazon S3 objects. If you have encrypted objects under a previous version of the Amazon S3 Encryption Client, you may need to enable legacy decryption modes when you instantiate the updated client. For more information, see Migrating to version 3.*x* of the Amazon S3 Encryption Client for Java.

With version 3.*x* of the Amazon S3 Encryption Client for Java, you can instantiate your client specifying the builder parameter that identifies your wrapping key. The Amazon S3 Encryption Client supports the following wrapping keys: symmetric AWS KMS keys, Raw AES-GCM keys, and Raw RSA keys. Then, the Amazon S3 Encryption Client automatically configures a keyring based on the wrapping key type with default settings and a default cryptographic materials manager (CMM). If you want to customize your client, you can also manually configure your keyring.

> ⓘ **Note**
>
> If you use Raw RSA or Raw AES-GCM wrapping keys, you are responsible for generating, storing, and protecting the key material, preferably in a hardware security module (HSM) or key management system.

The following examples instantiate the Amazon S3 Encryption Client with the default decryption mode. This means that all objects will be decrypted using the fully supported buffered decryption mode. For more information, see Decryption modes (Version 3.*x* and later).

Topics

- AWS KMS wrapping key
- Raw AES wrapping key
- Raw RSA wrapping key
- Manually instantiate the client

AWS KMS wrapping key

To specify a KMS key as your wrapping key, instantiate your client with the `kmsKeyId` builder parameter.

To use a KMS key as your wrapping key, you need [kms:GenerateDataKey](#) and [kms:Decrypt](#) permissions on the KMS key. The value of the `kmsKeyId` parameter can be any valid KMS key identifier. For details, see [Key identifiers](#) in the *AWS Key Management Service Developer Guide*.

```
// v3

class v3KMSKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .kmsKeyId(kmsKeyId)
                .build();
    }
}
```

Raw AES wrapping key

To specify a Raw AES key (`javax.crypto.SecretKey`) as your wrapping key, instantiate your client with the `aesKey` builder parameter.

```
// v3

class v3AESKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .build();
    }
}
```

Raw RSA wrapping key

To specify a Raw RSA key (`java.security.KeyPair`) as your wrapping key, instantiate your client with the `rsaKeyPair` builder parameter. You can specify an entire RSA key pair or a partial RSA key pair. The value of the `rsaKeyPair` parameter must include both the public and private keys in the key pair to perform both encrypt and decrypt operations. You can specify the public key to enable the Amazon S3 Encryption Client to perform encrypt operations, or the private key to enable decrypt operations as needed. By specifying a partial key pair you can limit the exposure of your keys. For examples using a partial key pair, see the [amazon-s3-encryption-client-java](#) GitHub repository.

RSA key pair

To instantiate version 3.*x* of the client to perform both encrypt and decrypt operations, specify both the public and private keys of your key pair.

```
// v3

class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(rsaKeyPair)
                .build();
    }
}
```

Public key

To instantiate the client to encrypt only, specify the **public key**. If you specify the public key alone, all `GetObject` calls will fail because the private key is required to decrypt.

```
// v3

class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(new PartialRsaKeyPair(null, rsaKeyPair.getPublic()))
                .build();
    }
}
```

Private key

To instantiate the client to decrypt only, specify the **private key**. If you specify the private key alone, all `PutObject` calls will fail because the public key is required to encrypt.

```
// v3

class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(new PartialRsaKeyPair(rsaKeyPair.getPrivate(), null))
                .build();
```

```
        }
}
```

Manually instantiate the client

If you want to customize your client, you can manually configure your own keyring and cryptographic materials manager (CMM). The following example manually configures an AWS KMS keyring using a symmetric encryption AWS KMS wrapping key and passes the custom AWS KMS client to the Amazon S3 Encryption Client.

```
// v3
class v3CustomKeyringExample {
    public static void main(String[] args) {
        KmsKeyring keyring = KmsKeyring.builder()
            .wrappingKeyId(KMS_KEY_ID)
            .kmsClient(customKmsClient)
            .build();

        CryptographicMaterialsManager cmm = DefaultCryptoMaterialsManager.builder()
            .keyring(keyring)
            .build();

        S3Client v3Client = S3EncryptionClient.builder()
            .cryptoMaterialsManager(cmm)
            .build();
    }
}
```

Encrypting and decrypting Amazon S3 objects

The following example shows you how to use the Amazon S3 Encryption Client for Java to encrypt and decrypt Amazon S3 objects.

This example uses a Raw RSA wrapping key and instantiates the Amazon S3 Encryption Client with the default decryption mode.

1. Specify your wrapping key by passing it to the Amazon S3 Encryption Client when you instantiate your client. The Amazon S3 Encryption Client for Java automatically configures a keyring based on the wrapping key you specify.

   ```
   // v3
   ```

```
class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(rsaKeyPair)
                .build();
    }
}
```

2. Encrypt your plaintext object by calling PutObject.

    a. The Amazon S3 Encryption Client provides the encryption materials: one plaintext data
        key and one copy of that data key encrypted by your wrapping key.

    b. The Amazon S3 Encryption Client uses the plaintext data key to encrypt your object, and
        then discards the plaintext data key.

    c. The Amazon S3 Encryption Client uploads the encrypted data key and the encrypted
        object to Amazon S3 as part of the PutObject call.

```
// v3

class v3EncryptExample {
    public static void main(String[] args) {
        s3Client.putObject(PutObjectRequest.builder()
                        .bucket(bucket)
                        .key(objectKey)
                        .build(), RequestBody.fromString(objectContent));
    }
}
```

3. Decrypt your encrypted object by calling GetObject.

    a. The Amazon S3 Encryption Client uses your wrapping key to decrypt the encrypted data
        key.

    b. The Amazon S3 Encryption Client uses the plaintext data key to decrypt the object,
        discards the plaintext data key, and returns the plaintext object as part of the GetObject
        call.

```
// v3
```

```
class v3DecryptExample {
    public static void main(String[] args) {
        ResponseBytes<GetObjectResponse> objectResponse =
 s3Client.getObjectAsBytes(builder -> builder
                .bucket(bucket)
                .key(objectKey));
        String output = objectResponse.asUtf8String();
    }
}
```

> **ⓘ Note**
>
> The default decryption mode cannot decrypt objects larger than 64 MB. This
> decryption mode automatically buffers stream contents into memory to prevent the
> release of unauthenticated objects. If you attempt to decrypt an object larger than 64
> MB, you will receive an exception directing you to enable the delayed authentication
> decryption mode. For more information, see [Decryption modes](#).

4. Optional: verify that the decrypted object matches the original plaintext object that you
   uploaded.

```
assert output.equals(objectContent);
```

5. The Amazon S3 Encryption Client implements the `AutoClosable` interface, which
   automatically calls `close()` when you exit a `try-with-resources` block for which the
   object has been declared in the resource specification header. As a best practice, you should
   either use `try-with-resources` or explicitly call the `close()` method.

```
s3Client.close();
```

Ranged GET requests

With Amazon S3, you can download a specific part of an object by performing a ranged GET
request. In version 3.*x* of the Amazon S3 Encryption Client, you must explicitly enable the
[unauthenticated legacy decryption mode](#) to perform ranged requests.

By default, version 3.*x* of the Amazon S3 Encryption Client encrypts and decrypts your objects
using the AES-GCM algorithm suite. However, you can enable it to use the legacy AES-CTR
algorithm to partially decrypt your object during a ranged GET request. The Amazon S3 Encryption

Client cannot use AES-GCM for ranged gets because it is an authenticated scheme that appends an authentication tag to the encrypted object. When you request a partial object, the client cannot read the entire object stream to reach the authenticated tag. This means that the partial object is not authenticated.

Specifying a range

You can include the Range parameter in your `GetObject` request to download and decrypt a specific byte-range from an object. The start and end indices of the byte range are included in the partial object. The byte-range you specify should reflect to the following format:

```
range("bytes=startIndex-endIndex")
```

The following list details how the Amazon S3 Encryption Client responds to ranged requests that specify an invalid byte-range. For more detailed examples, see the [amazon-s3-encryption-client-java](#) GitHub repository.

- When the start index is within object range but the end index is greater than the object's total length, the Amazon S3 Encryption Client returns the object from the start index to the end of the original plaintext object.

- When the start index is greater than the end index, the Amazon S3 Encryption Client returns the entire object.

- When the range is specified with an invalid format, the Amazon S3 Encryption Client returns the entire object.

  For example, if the range was specified as `range("10-20")`, instead of `range("bytes=10-20")`, then the Amazon S3 Encryption Client will return the entire object.

- When both the start and end indicies are greater than the original plaintext object's total length, but still within the same cipher block, the Amazon S3 Encryption Client returns an empty object.

- When both the start and end indicies are greater than the original plaintext object's total length, and are outside of the object's cipher block, the `GetObject` request fails.

Performing a ranged request

The following walkthrough explains how to perform a ranged request when using version 3.*x* of the Amazon S3 Encryption Client.

1. Enable ranged gets by specifying the `enableLegacyUnauthenticatedModes` parameter
    when you instantiate your client.

    The following example specifies a raw AES key as the wrapping key.

    ```
    // v3

    class v3EnableRangedGetsExample {
        public static void main(String[] args) {
            S3Client v3Client = S3EncryptionClient.builder()
                    .aesKey(aesKey)
                    .enableLegacyUnauthenticatedModes(true)
                    .build();
        }
    }
    ```

2. Partially decrypt your encrypted object by specifying the byte-range in your GetObject
    request.

    The following example specifies a start index at byte 10 and end index at byte 20.

    ```
    // v3

    class v3RangedGetExample {
        public static void main(String[] args) {
            ResponseBytes<GetObjectResponse> objectResponse =
     v3Client.getObjectAsBytes(builder -> builder
                    .bucket(bucket)
                    .range("bytes=10-20")
                    .key(objectKey));
            String output = objectResponse.asUtf8String();
        }
    }
    ```

3. Optional: verify that the decrypted partial object matches the original plaintext object that
    you uploaded at the same range.

    ```
    assert output.equals(objectContent);
    ```

4. The Amazon S3 Encryption Client implements the `AutoClosable` interface, which
    automatically calls `close()` when you exit a `try-with-resources` block for which the

object has been declared in the resource specification header. As a best practice, you should either use `try-with-resources` or explicitly call the `close()` method.

```
s3Client.close();
```

Multipart upload

Amazon S3 allows you to upload a single object as a set of parts using multipart uploads. Amazon S3 recommends that when your object size reaches 100 MB, you should consider using multipart uploads. In version 3.*x* of the Amazon S3 Encryption Client, you can perform multipart uploads with the low-level API or the high-level API. Use the low-level API when you need to vary part sizes during the upload or require more control over the multipart upload process. Use the high-level API to simplify the multipart upload process by enabling the Amazon S3 Encryption Client to automatically perform multipart uploads.

Multipart Upload (high-level API)

When you use the high-level API, the Amazon S3 Encryption Client automatically performs multipart uploads for all objects larger than 5 MB. The Amazon S3 Encryption Client encrypts the object locally and then calls the AWS CRT-based Amazon S3 client to perform the multipart upload to Amazon S3.

> ⓘ **Note**
>
> If your permissions to access required Amazon S3 resources or KMS keys are revoked during a multipart upload using the high-level API, the in-progress request may upload successfully. Subsequent multipart upload requests will fail.

To enable automatic multipart uploads with the high-level API, you must add dependencies for the AWS CRT-based Amazon S3 client to your Maven project file and specify the `enableMultipartPutObject` parameter when you instantiate your client.

Add dependencies

> To use the AWS CRT-based Amazon S3 client with the Amazon S3 Encryption Client, add the following two dependencies. For more information on creating dependencies and installing the Amazon S3 Encryption Client, see Installing the Amazon S3 Encryption Client for Java.

```
<dependency>
   <groupId>software.amazon.awssdk</groupId>
   <artifactId>s3</artifactId>
   <version>2.19.3
<dependency>
   <groupId>software.amazon.awssdk.crt</groupId>
   <artifactId>aws-crt</artifactId>
   <version>0.21.5</version>
</dependency>
```

Enable multipart upload

To enable the Amazon S3 Encryption Client to automatically perform multipart uploads, specify the `enableMultipartPutObject` parameter when you instantiate your client.

The following example specifies a raw AES key as the wrapping key.

```
// v3

class v3EnableMultipartUploadExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .enableMultipartPutObject(true)
                .build();
    }
}
```

Multipart Upload (low-level API)

The Amazon S3 Encryption Client does not require any additional configuration to use the low-level API. Use the following API calls to generate a multipart upload request with version 3.*x* of the Amazon S3 Encryption Client.

1. Start the multipart upload process by calling [CreateMultipartUpload](#).
2. Call [UploadPart](#) to upload each part of your object. When you upload the final part, you must specify `isLastPart` for the Amazon S3 Encryption Client to be able to call `cipher.doFinal()`

```
// v3
```

```
class v3UploadFinalPartExample {
    public static void main(String[] args) {
        UploadPartRequest uploadPartRequest = UploadPartRequest.builder()
                    .bucket(bucket)
                    .key(objectKey)
                    .uploadId(initiateResult.uploadId())
                    .partNumber(partsSent)
                    .overrideConfiguration(isLastPart(true))
                    .build();
    }
}
```

3. Call CompleteMultipartUpload to finish the process.

Asynchronous programming in the Amazon S3 Encryption Client for Java

Version 3.*x* of the Amazon S3 Encryption Client provides a nonblocking asynchronous client that implements high concurrency across a few threads. The asynchronous client enables you to perform requests sequentially without waiting to view results between each request.

The default Amazon S3 Encryption Client uses synchronous methods that block your thread's execution until the client receives a response from Amazon S3. The asynchronous client returns immediately, giving control back to the calling thread without waiting for a response. Because an asynchronous method returns before a response is available, you need a way to get the response when it's ready. The methods in version 3.*x* of the Amazon S3 Encryption Client return *CompletableFuture objects* that allow you to access the response when it's ready.

Topics

- Instantiating the asynchronous client
- Encrypt and decrypt with the asynchronous client

Instantiating the asynchronous client

To use the asynchronous client, you must specify the S3AsyncEncryptionClient builder and the builder parameter that identifies your wrapping key when you instantiate your client. The Amazon S3 Encryption Client supports the following wrapping keys: symmetric AWS KMS keys, raw AES-GCM keys, and raw RSA keys.

> ⓘ **Note**
>
> If you use Raw RSA or Raw AES-GCM wrapping keys, you are responsible for generating, storing, and protecting the key material, preferably in a hardware security module (HSM) or key management system.

The following examples instantiate the asynchronous Amazon S3 Encryption Client with the default decryption mode. This means that all objects will be decrypted using the fully supported buffered decryption mode. For more information, see Decryption modes (Version 3.*x* and later).

KMS key

To specify a KMS key as your wrapping key, instantiate your client with the kmsKeyId builder parameter. The value of the kmsKeyId parameter can be any valid KMS key identifier. For details, see Key identifiers in the *AWS Key Management Service Developer Guide*.

```
// v3

class v3KMSKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .kmsKeyId(kmsKeyId)
                .build();
    }
}
```

Raw AES key

To specify a raw AES key (javax.crypto.SecretKey) as your wrapping key, instantiate your client with the aesKey builder parameter.

```
// v3

class v3AESKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .build();
    }
}
```

Raw RSA key

To specify a raw RSA key (`java.security.KeyPair`) as your wrapping key, instantiate your client with the `rsaKeyPair` builder parameter. You can specify an entire RSA key pair or a partial RSA key pair. The value of the `rsaKeyPair` parameter must include both the public and private keys in the key pair to perform both encrypt and decrypt operations. You can specify the public key to enable the Amazon S3 Encryption Client to perform encrypt operations, or the private key to enable decrypt operations as needed. By specifying a partial key pair you can limit the exposure of your keys. For examples using a partial key pair, see the amazon-s3-encryption-client-java GitHub repository.

To instantiate version 3.*x* of the client to perform both encrypt and decrypt operations, specify both the public and private keys of your key pair.

```
// v3

class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(rsaKeyPair)
                .build();
    }
}
```

To instantiate the client to encrypt only, specify the **public key**. If you specify the public key alone, all `GetObject` calls will fail because the private key is required to decrypt.

```
// v3

class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(new PartialRsaKeyPair(null, rsaKeyPair.getPublic()))
                .build();
    }
}
```

To instantiate the client to decrypt only, specify the **private key**. If you specify the private key alone, all `PutObject` calls will fail because the public key is required to encrypt.

```
// v3
```

```
class v3RSAKeyPairExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .rsaKeyPair(new PartialRsaKeyPair(rsaKeyPair.getPrivate(), null))
                .build();
    }
}
```

You can customize your asynchronous client by specifying different builder parameters to enable the features you need. By default, version 3.*x* of the Amazon S3 Encryption Client does not support legacy decryption, ranged 'GET' requests, or multipart uploads (via the high-level API).

For example, if you need to decrypt data keys that were encrypted with a legacy wrapping algorithm, specify the enableLegacyWrappingAlgorithms parameter when you instantiate your client. The following example specifies a raw AES key as the wrapping key.

```
// v3

class v3EnableLegacyModesAsyncClientExample {
    public static void main(String[] args) {
        S3AsyncClient v3Client = S3AsyncEncryptionClient.builder()
                .aesKey(AES_KEY)
                .enableLegacyWrappingAlgorithms(true)
                .build();
    }
}
```

Encrypt and decrypt with the asynchronous client

The following walkthrough demonstrates how encrypt and decrypt asynchronously with version 3.*x* of the Amazon S3 Encryption Client.

1. Instantiate your asynchronous client with the S3AsyncEncryptionClient builder.

    The following example specifies a raw AES key as the wrapping key.

    ```
    // v3

    class v3EnableAsyncClientExample {
        public static void main(String[] args) {
    ```

```
            S3AsyncClient v3Client = S3AsyncEncryptionClient.builder()
                    .aesKey(AES_KEY)
                    .build();
    }
}
```

2. Call PutObject to encrypt a plaintext object and upload it to Amazon S3.

   The asynchronous client stores the response to confirm that the PutObject request
   completed when you call GetObject in the future.

```
// v3

class v3PutObjectAsyncClientExample {
    public static void main(String[] args) {
        CompletableFuture<PutObjectResponse> futurePut =
 v3AsyncClient.putObject(builder -> builder
                .bucket(bucket)
                .key(objectKey)
                .build(), AsyncRequestBody.fromString(objectContent));
        // Block on completion of the futurePut
        futurePut.join();
    }
}
```

3. Call GetObject to download and decrypt the encrypted object.

```
// v3

class v3GetObjectAsyncClientExample {
    public static void main(String[] args) {
        CompletableFuture<ResponseBytes<GetObjectResponse>> futureGet =
 v3AsyncClient.getObject(builder -> builder
                .bucket(bucket)
                .key(objectKey)
                .build(), AsyncResponseTransformer.toBytes());
        // Wait for the future to complete
        ResponseBytes<GetObjectResponse> getResponse = futureGet.join();
    }
}
```

4. Optional: verify that the decrypted object matches the original plaintext object that you
   uploaded.

```
assert output.equals(objectContent);
```

5. The Amazon S3 Encryption Client implements the `AutoClosable` interface, which
    automatically calls `close()` when you exit a `try-with-resources` block for which the
    object has been declared in the resource specification header. As a best practice, you should
    either use `try-with-resources` or explicitly call the `close()` method.

```
s3Client.close();
```

> **ⓘ Note**
>
> The default decryption mode cannot decrypt objects larger than 64 MB. This decryption
> mode automatically buffers stream contents into memory to prevent the release of
> unauthenticated objects. If you attempt to decrypt an object larger than 64 MB, you will
> receive an exception directing you to enable the delayed authentication decryption mode.
> For more information, see Decryption modes.

Migrating to version 3.*x* of the Amazon S3 Encryption Client for Java

The `v3Client` constructor does not use the `EncryptionMaterialsProvider` that was required
in versions 1.*x* and 2.*x* of the Amazon S3 Encryption Client. Instead, you use a parameter of the
`v3Client` builder to specify your wrapping key. The Amazon S3 Encryption Client supports the
following wrapping keys: AWS Key Management Service (AWS KMS) symmetric AWS KMS keys,
raw AES-GCM (Advanced Encryption Standard/Galois Counter Mode) keys, and raw RSA keys. The
Amazon S3 Encryption Client optimizes its settings based on the wrapping key type.

When updating from earlier versions of the Amazon S3 Encryption Client to version 3.*x*, you need
to update your client builder code to use the new, simpler interface for the `v3Client`. If you're
decrypting ciphertext that was encrypted by earlier versions of the Amazon S3 Encryption Client,
you might also need to allow the Amazon S3 Encryption Client to decrypt legacy encryption
algorithms.

To update to Amazon S3 Encryption Client version 3.*x*, delete the code that instantiates the
`EncryptionMaterialsProvider`. Then replace the code that calls the `v1Client` or `v2Client`

builder with code that calls the `v3Client` builder. Use a parameter of the `v3Client` builder to specify your wrapping key.

The following examples show the equivalent code required to specify a KMS key as the wrapping key in versions 1.*x*, 2.*x*, and 3.*x* of the Amazon S3 Encryption Client.

Version 1.*x*

In Amazon S3 Encryption Client version 1.*x*, you instantiate an `EncryptionMaterialsProvider` with your wrapping key, and then specify that materials provider when instantiating the `v1Client` object.

```
// v1

class v1KMSKeyExample {
    public static void main(String[] args) {

        EncryptionMaterialsProvider materialsProvider = new
 KMSEncryptionMaterialsProvider(kmsKeyId);
        AmazonS3Encryption v1Client = AmazonS3EncryptionClient.encryptionBuilder()
                .withEncryptionMaterialsProvider(materialsProvider)
                .build();
    }
}
```

Version 2.*x*

In Amazon S3 Encryption Client version 2.*x*, you instantiate an `EncryptionMaterialsProvider` with your wrapping key, and then specify that materials provider when instantiating the `v2Client` object.

```
// v2

class v2KMSKeyExample {
    public static void main(String[] args) {

        EncryptionMaterialsProvider materialsProvider = new
 KMSEncryptionMaterialsProvider(kmsKeyId);
        AmazonS3EncryptionV2 v2Client =
 AmazonS3EncryptionClientV2.encryptionBuilder()
                .withEncryptionMaterialsProvider(materialsProvider)
                .build();
```

```
        }
    }
```

Version 3.*x*

In Amazon S3 Encryption Client version 3.*x*, the `v3Client` constructor requires only a parameter that identifies the wrapping key. For a KMS key, use the `kmsKeyId` parameter. The value of the `kmsKeyId` parameter can be any valid KMS key identifier. For details, see Key identifiers in the *AWS Key Management Service Developer Guide*.

```
// v3

class v3KMSKeyExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .kmsKeyId(kmsKeyId)
                .build();
    }
}
```

Enable legacy decryption modes

If you need to decrypt objects or data keys that were encrypted with a legacy algorithm, or you need to partially decrypt an AES-GCM encrypted object when performing a ranged request, you need to explicitly enable this behavior when you instantiate the client.

The `v3Client` encrypts only with fully supported algorithms. It will never encrypt with a legacy algorithm. By default, it decrypts only with fully supported algorithms, but you can enable it to decrypt with both fully supported and legacy algorithms. For more information, see Decryption modes (Amazon S3 Encryption Client for Java version 3.*x* and later).

The `enableLegacyUnauthenticatedModes` parameter of the `v3Client` builder enables the Amazon S3 Encryption Client to decrypt encrypted objects with a fully supported or legacy encryption algorithm.

Version 3.*x* of the Amazon S3 Encryption Client uses one of the fully supported wrapping algorithms and the wrapping key you specify to encrypt and decrypt the data keys. The `enableLegacyWrappingAlgorithms` parameter of the `v3Client` builder enables the Amazon S3 Encryption Client to decrypt encrypted data keys with a fully supported or legacy wrapping algorithm.

If your `v3Client` doesn't include the necessary legacy decryption mode with a value of `true`, and it encounters an object encrypted with a legacy algorithm, it throws `S3EncryptionClientException`.

For example, this code builds a `v3Client` object with a user-provided raw AES wrapping key. This client always encrypts only with fully supported algorithms. However, it can decrypt objects and data keys encrypted with fully supported or legacy algorithms.

```
// v3

class v3EnableLegacyDecryptionModesExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .enableLegacyUnauthenticatedModes(true)
                .enableLegacyWrappingAlgorithms(true)
                .build();
    }
}
```

The legacy decryption modes are designed to be a temporary fix. After you've re-encrypted all of your objects with fully supported algorithms, you can eliminate it from your code.

Amazon S3 Encryption Client for Go

> **(i) Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an independent library. For information about previous versions of the Amazon S3 Encryption Client, see the AWS SDK Developer Guide for your programming language.

This topic explains how to install and use the Amazon S3 Encryption Client for Go. For details about programming with the Amazon S3 Encryption Client for Go, see the amazon-s3-encryption-client-go repository on GitHub.

Topics

- Prerequisites
- Installation

- Amazon S3 Encryption Client for Go examples
- Migrating to version 3.x of the Amazon S3 Encryption Client for Go

Prerequisites

Before you install the Amazon S3 Encryption Client for Go, be sure you have the following prerequisites.

A Go development environment

> The Amazon S3 Encryption Client for Go requires Go 1.2 or later, but we recommend that you use the latest version.
>
> You can view your current version of Go by running the following command.

```
go version
```

AWS SDK for Go 2.x

> The Amazon S3 Encryption Client for Go requires the Amazon S3 and AWS KMS service clients of the AWS SDK for Go 2.x. Service clients can be constructed using either the NewFromConfig or New function. For more information, see Using the AWS SDK for Go V2 with AWS services in the *AWS SDK for Go Developer Guide*.
>
> For information about updating your version of the AWS SDK for Go, see Migration to the AWS SDK for Go V2 in the *AWS SDK for Go Developer Guide*.

Installation

To install the Amazon S3 Encryption Client for Go and its dependencies, run the following Go command.

```
go get github.com/aws/amazon-s3-encryption-client-go/v3
```

Amazon S3 Encryption Client for Go examples

The following examples show you how to use the Amazon S3 Encryption Client for Go to encrypt and decrypt Amazon S3 objects. These examples show how to use version 3.*x* of the Amazon S3 Encryption Client for Go.

> ℹ **Note**
>
> The Amazon S3 Encryption Client for Go does not support asynchronous programming, multipart uploads, or ranged GET requests. To use these features with version 3.*x* of the Amazon S3 Encryption Client, you must use the Amazon S3 Encryption Client for Java.

Topics

- Instantiating the Amazon S3 Encryption Client
- Encrypting and decrypting Amazon S3 objects

Instantiating the Amazon S3 Encryption Client

After installing the Amazon S3 Encryption Client for Go, you are ready to instantiate your client and begin encrypting and decrypting your Amazon S3 objects. If you have encrypted objects under a previous version of the Amazon S3 Encryption Client, you may need to enable legacy decryption modes when you instantiate the updated client. For more information, see Migrating to version 3.*x* of the Amazon S3 Encryption Client for Go.

The Amazon S3 Encryption Client for Go supports keyrings that use symmetric encryption KMS keys as the wrapping key. The Amazon S3 Encryption Client for Go does not support keyrings that use Raw AES-GCM or Raw RSA wrapping keys. To use Raw AES-GCM or Raw RSA wrapping keys, you must use the Amazon S3 Encryption Client for Java. For more information, see Instantiating the Amazon S3 Encryption Client for Java.

To use a KMS key as your wrapping key, you need kms:GenerateDataKey and kms:Decrypt permissions on the KMS key. To specify a KMS key, use any valid KMS key identifier. For details, see Key identifiers in the *AWS Key Management Service Developer Guide*.

The following example instantiates the Amazon S3 Encryption Client with the default decryption mode. This means that all objects will be decrypted using the fully supported buffered decryption mode. For more information, see Decryption modes (Version 3.*x* and later).

```
import (
    ...
    // Import the materials and client package
    "github.com/aws/amazon-s3-encryption-client-go/v3/client"
    "github.com/aws/amazon-s3-encryption-client-go/v3/materials"
    ...
```

```
)
s3EncryptionClient, err := client.New(s3Client, cmm)

// Create the keyring and cryptographic materials manager (CMM)
cmm, err :=
 materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsClient, kmsKeyArn,
  func(options *materials.KeyringOptions) {
   options.EnableLegacyWrappingAlgorithms = false
}))
if err != nil {
 t.Fatalf("error while creating new CMM")
}

s3EncryptionClient, err := client.New(s3Client, cmm)
```

Encrypting and decrypting Amazon S3 objects

The following example shows you how to use the Amazon S3 Encryption Client for Go to encrypt
and decrypt Amazon S3 objects.

1. Create a keyring with a KMS key as your wrapping key when you instantiate your client.

   ```
   s3Client = s3.NewFromConfig(cfg)
   kmsClient := kms.NewFromConfig(cfg)
   cmm, err :=
    materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsClient, kmsKeyArn,
     func(options *materials.KeyringOptions) {
        options.EnableLegacyWrappingAlgorithms = false
   }))
   if err != nil {
       t.Fatalf("error while creating new CMM")
   }
   ```

2. Encrypt your plaintext object by calling PutObject. To include an optional material
   description, add an EncryptionContext value to the context and supply this value to the
   PutObject request.

   a. The Amazon S3 Encryption Client provides the encryption materials: one plaintext data
      key and one copy of that data key encrypted by your wrapping key.

   b. The Amazon S3 Encryption Client uses the plaintext data key to encrypt your object, and
      then discards the plaintext data key.

c. The Amazon S3 Encryption Client uploads the encrypted data key and the encrypted
    object to Amazon S3 as part of the PutObject call.

```
ctx := context.Background()
...
encryptionContext := context.WithValue(ctx, "EncryptionContext",
 map[string]string{"ec-key": "ec-value"})

s3EncryptionClient, err := client.New(s3Client, cmm)
_, err = s3EncryptionClient.PutObject(encryptionContext, &s3.PutObjectInput{
    Bucket: aws.String(bucket),
    Key:    aws.String(objectKey),
    Body:   bytes.NewReader([]byte(plaintext)),
})

if err != nil {
    t.Fatalf("error while encrypting: %v", err)
}
```

3. Decrypt your encrypted object by calling GetObject.

    a. The Amazon S3 Encryption Client uses your wrapping key to decrypt the encrypted data
        key.

    b. The Amazon S3 Encryption Client uses the plaintext data key to decrypt the object,
        discards the plaintext data key, and returns the plaintext object as part of the GetObject
        call.

```
result, err := s3EncryptionClient.GetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  return fmt.Errorf("error while decrypting: %v", err)
 }

 decryptedPlaintext, err := io.ReadAll(result.Body)
 if err != nil {
  return fmt.Errorf("failed to read decrypted plaintext into byte array")
 }
```

4. Optional: verify that the decrypted object matches the original plaintext object that you uploaded.

```
if e, a := []byte(plaintext), decryptedPlaintext; !bytes.Equal(e, a) {
  return fmt.Errorf("expect %v text, got %v", e, a)
 }
```

Migrating to version 3.*x* of the Amazon S3 Encryption Client for Go

With version 3.*x* of the Amazon S3 Encryption Client for Go, you create one client for both encryption and decryption. Version 3.*x* replaces the cipher data generators with the cryptographic materials manager (CMM), and replaces the KMS key providers, NewKMSContextKeyGenerator, with the NewKmsKeyring.

When updating from earlier versions of the Amazon S3 Encryption Client to version 3.*x*, you need to update your client builder code to use the new, simpler client. If you're decrypting ciphertext that was encrypted by earlier versions of the Amazon S3 Encryption Client, you might also need to allow the Amazon S3 Encryption Client to decrypt legacy encryption algorithms.

The following examples show the equivalent code required to specify a KMS key provider with a KMS key ID in versions 1.*x*, 2.*x*, and 3.*x* of the Amazon S3 Encryption Client.

Version 1.x

> In version 1.*x*, you use the NewKMSKeyGeneratorWith function to construct the cipherDataGenerator.
>
> ```
> sess := session.Must(session.NewSession())
> kmsClient := kms.New(sess)
> cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"
>
> cipherDataGenerator := s3crypto.NewKMSKeyGenerator(kmsClient, kmsKeyID)
> ```

Version 2.x

> In version 2.*x*, you use the NewKMSContextKeyGenerator function to construct the cipherDataGenerator.
>
> ```
> sess := session.Must(session.NewSession())
> kmsClient := kms.New(sess)
> ```

```
cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"
var matDesc s3crypto.MaterialDescription

// changed NewKMSKeyGenerator to NewKMSContextKeyGenerator
cipherDataGenerator := s3crypto.NewKMSContextKeyGenerator(kmsClient, kmsKeyID,
 matDesc)
```

Version 3.x

In version 3.*x*, you use the `NewKmsKeyring` function to construct your cryptographic materials manager (CMM).

```
s3Client := s3.NewFromConfig(cfg)
kmsClient := kms.NewFromConfig(cfg)
cmm, err :=
 materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsClient, kmsKeyID))
 if err != nil {
  return fmt.Errorf("error while creating new CMM")
 }
```

Migrating from version 2.*x*

The following example demonstrates how to migrate a version 2.*x* application that uses the `NewKMSContextKeyGenerator` KMS key provider with a material description and `AESGCMContentCipherBuilderV2` content cipher to version 3.*x* of the Amazon S3 Encryption Client for Go.

```
func KmsContextV2toV3GCMExample() error {
 bucket := LoadBucket()
 kmsKeyAlias := LoadAwsKmsAlias()

 objectKey := "my-object-key"
 region := "us-west-2"
 plaintext := "This is an example.\n"

 // Create an S3EC Go v2 encryption client
 // using the KMS client from AWS SDK for Go v1
 sessKms, err := sessionV1.NewSession(&awsV1.Config{
  Region: aws.String(region),
 })
```

```go
kmsSvc := kmsV1.New(sessKms)
handler := s3cryptoV2.NewKMSContextKeyGenerator(kmsSvc, kmsKeyAlias,
s3cryptoV2.MaterialDescription{})
builder := s3cryptoV2.AESGCMContentCipherBuilderV2(handler)
encClient, err := s3cryptoV2.NewEncryptionClientV2(sessKms, builder)
if err != nil {
 log.Fatalf("error creating new v2 client: %v", err)
}

// Encrypt using KMS+Context and AES-GCM content cipher
_, err = encClient.PutObject(&s3V1.PutObjectInput{
 Bucket: aws.String(bucket),
 Key:    aws.String(objectKey),
 Body:   bytes.NewReader([]byte(plaintext)),
})
if err != nil {
 log.Fatalf("error calling putObject: %v", err)
}
fmt.Printf("successfully uploaded file to %s/%s\n", bucket, key)

// Create an S3EC Go v3 client
// using the KMS client from AWS SDK for Go v2
ctx := context.Background()
cfg, err := config.LoadDefaultConfig(ctx,
 config.WithRegion(region),
)

kmsV2 := kms.NewFromConfig(cfg)
cmm, err := materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsV2,
kmsKeyAlias))
if err != nil {
 t.Fatalf("error while creating new CMM")
}

s3V2 := s3.NewFromConfig(cfg)
s3ecV3, err := client.New(s3V2, cmm)

result, err := s3ecV3.GetObject(ctx, &s3.GetObjectInput{
 Bucket: aws.String(bucket),
 Key:    aws.String(objectKey),
})
if err != nil {
 t.Fatalf("error while decrypting: %v", err)
```

```
    }
```

Enable legacy decryption modes

If you need to decrypt objects or data keys that were encrypted with a legacy algorithm, or you need to partially decrypt an AES-GCM encrypted object when performing a ranged request, you need to explicitly enable this behavior when you instantiate the client.

Version 3.*x* of the Amazon S3 Encryption Client encrypts only with fully supported algorithms. It will never encrypt with a legacy algorithm. By default, it decrypts only with fully supported algorithms, but you can enable it to decrypt with both fully supported and legacy algorithms. For more information, see Decryption modes (Amazon S3 Encryption Client for Java version 3.*x* and later).

The `enableLegacyUnauthenticatedModes` flag enables the Amazon S3 Encryption Client to decrypt encrypted objects with a fully supported or legacy encryption algorithm.

Version 3.*x* of the Amazon S3 Encryption Client uses one of the fully supported wrapping algorithms and the wrapping key you specify to encrypt and decrypt the data keys. The `enableLegacyWrappingAlgorithms` flag enables the Amazon S3 Encryption Client to decrypt encrypted data keys with a fully supported or legacy wrapping algorithm.

If your client doesn't include the necessary legacy decryption mode with a value of `true`, and it encounters an object encrypted with a legacy algorithm, it throws `S3EncryptionClientException`.

The following example enables the `enableLegacyUnauthenticatedModes` and `enableLegacyWrappingAlgorithms` flags. This client always encrypts only with fully supported algorithms. However, it can decrypt objects and data keys encrypted with fully supported or legacy algorithms.

```
cmm, err :=
 materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsClient, ,
 func(options *materials.KeyringOptions) {
    options.EnableLegacyWrappingAlgorithms = true
})

if err != nil {
 t.Fatalf("error while creating new CMM")
}
```

```
client, err := client.New(s3Client, cmm, func(clientOptions
 *client.EncryptionClientOptions) {
  clientOptions.EnableLegacyUnauthenticatedModes = true
})

if err != nil {
 // handle error
 }
```

The legacy decryption modes are designed to be a temporary fix. After you've re-encrypted all of your objects with fully supported algorithms, you can eliminate it from your code.

Supported encryption algorithms

> ⓘ **Note**
>
> This documentation describes the Amazon S3 Encryption Client version 3.*x*, which is an
> independent library. For information about previous versions of the Amazon S3 Encryption
> Client, see the AWS SDK Developer Guide for your programming language.

The Amazon S3 Encryption Client supports industry-standard algorithms for encrypting objects
and data keys. As our knowledge evolves, we adjust our support for encryption algorithms to
ensure that your sensitive data is protected. The following topic provides context on which
encryption algorithms are fully supported and the different decryption modes supported in version
3.*x* of the Amazon S3 Encryption Client.

Topics

- [Encryption algorithms (Version 3.x and later)](#)
- [Decryption modes (version 3.x and later)](#)
- [Encryption algorithms (Version 2.x and earlier)](#)

Encryption algorithms (Version 3.*x* and later)

In versions 3.*x* and later, the Amazon S3 Encryption Client will use fully supported algorithms to
encrypt and decrypt objects and data keys. You can enable the Amazon S3 Encryption Client to
decrypt objects and data keys with a legacy encryption algorithm, but it will not encrypt with a
legacy encryption algorithm. It encrypts only with a fully supported encryption algorithm.

The following tables list the object encryption algorithms and wrapping algorithms that are
supported in version 3.*x* of the Amazon S3 Encryption Client. Use these tables to determine if
any of your objects or data keys were encrypted with an algorithm that is no longer supported. If
you need to decrypt objects or data keys that were encrypted with a legacy algorithm, see [Enable
legacy decryption modes](#).

Encrypting objects — The following table lists the fully supported (Full) and previously supported
(Legacy) encryption algorithms that are used to encrypt objects.

Algorithm	Support
AES-GCM	Full
AES-CBC	Legacy

Encrypting data keys — The following table lists the fully supported (Full) and previously supported (Legacy) wrapping algorithms that are used to encrypt the data keys that encrypt your objects. Version 3.*x* of the Amazon S3 Encryption Client uses one of the fully supported wrapping algorithms and the wrapping key you specify to encrypt and decrypt the data keys.

Algorithm	Support
AES-GCM	Full
AWS KMS (with an encryption context)	Full
RSA-OAEP-MGF1 and SHA-1	Full
AES	Legacy
AESWrap	Legacy
AWS KMS (without an encryption context)	Legacy
RSA-OAEP-MGF-1 and SHA-256	Legacy
RSA	Legacy

Decryption modes (version 3.*x* and later)

Version 3.*x* of the Amazon S3 Encryption Client defines four modes of support for decryption that you can use to enable the client to decrypt objects and data keys with either fully supported or legacy algorithms.

Fully supported

By default, version 3.*x* of the Amazon S3 Encryption Client encrypts and decrypts your objects using the AES-GCM algorithm suite. AES-GCM is an authenticated scheme. This means that an authentication tag is appended to the encrypted object. The default behavior for versions 1.*x* and 2.*x* allowed streaming decryption of AES-GCM encrypted objects. Because authentication happens at the end of the decryption process, the entire object must be read before the cipher can validate the integrity of it. This allows plaintext objects to be released and used before the authentication tag is validated.

Version 3.*x* of the Amazon S3 Encryption Client supports streaming decryption of AES-GCM encrypted objects, but we recommend using the default decryption mode to prevent the release of unauthenticated plaintext objects.

Buffered (default)

By default, version 3.*x* of the Amazon S3 Encryption Client automatically buffers the stream contents into memory as the decrypted object is read to prevent the release of unauthenticated objects. If the client reaches the end of the stream, and the authentication fails, your [GetObject](#) request will throw an exception and the unauthenticated object will not be returned.

When you use the buffered decryption mode with the Amazon S3 Encryption Client for Java, the default maximum object size that can be decrypted is 64 MB. However, you can use the optional `setBufferSize` parameter to customize the maximum object size that the mode will buffer. You can use the `setBufferSize` parameter to specify any integer between 16 bytes and 64 GB as the maxiumum object size.

The following Java example instantiates the `v3Client` with a raw AES wrapping key and a maximum object size of 32 MiB.

```
// v3

class v3DefaultDecryptionModeExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .setBufferSize(32 * 1024 * 1024) // OPTIONAL
                .build();
    }
```

```
}
```

When you use the buffered decryption mode with the Amazon S3 Encryption Client for Go, the default maximum object size that can be decrypted is 63 GiB. You cannot set a custom buffer size with the Amazon S3 Encryption Client for Go. As a result, the buffered decryption mode does not require any additional configuration when you instatiate your client with the Amazon S3 Encryption Client for Go.

We recommend that you use the buffered decryption mode whenever possible. Since this is the default mode, you do not need to specify the buffered decryption mode when you instantiate your client.

Delayed authentication

> **ⓘ Note**
>
> The Amazon S3 Encryption Client for Go does not support the delayed authentication mode. To decrypt objects under the delayed authentication mode, you must use the Amazon S3 Encryption Client for Java.

The delayed authentication mode also supports streaming decryption of AES-GCM encrypted objects, but it does not buffer or interrupt the stream to prevent unauthenticated objects from being returned. We recommend using the Buffered (default) decryption mode whenever possible. However, you might want to use the delayed authentication mode if you established your own method of buffering the stream while using versions 1.*x* and 2.*x* of the client.

If you use the delayed authentication mode and are processing the plaintext data from the stream before reading to the end, you must account for the *delayed* authentication. Read the entire object to the end before you start using the decrypted object. When using this decryption mode, the Amazon S3 Encryption Client will not authenticate any object until it reaches the end of the stream. You will need to manually roll back any data from the stream if an exception is thrown at the end of the stream.

To enable the delayed authentication mode, specify the `enableDelayedAuthenticationMode` parameter when you instantiate the `v3Client`.

The following example specifies a raw AES key as the wrapping key. This client only encrypts with fully supported algorithms and decrypts using the delayed authentication mode.

```
// v3

class v3DelayedAuthenticationModeExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .enableDelayedAuthenticationMode(true)
                .build();
    }
}
```

Legacy

Legacy wrapping algorithms

By default, the Amazon S3 Encryption Client uses the wrapping key you specify and one of the fully supported wrapping algorithms to encrypt and decrypt the data keys that encrypt your objects. If you need to decrypt data keys that were encrypted with a legacy wrapping algorithm, you must specify the `enableLegacyWrappingAlgorithms` parameter when you instantiate your client.

Java

The following example specifies a raw AES key as the wrapping key. This client only encrypts with fully supported wrapping algorithms. However, it can decrypt data keys encrypted with fully supported or legacy wrapping algorithms.

```
// v3

class v3LegacyWrappingAlgorithmsExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .enableLegacyWrappingAlgorithms(true)
                .build();
    }
}
```

Go

> The following example creates a keyring that uses a KMS key as the wrapping key and only
> encrypts with fully supported wrapping algorithms. However, it can decrypt data keys
> encrypted with fully supported or legacy wrapping algorithms.

```
cmm, err :=
 materials.NewCryptographicMaterialsManager(materials.NewKmsKeyring(kmsClient, kmsKeyArn,
 func(options *materials.KeyringOptions) {
    options.EnableLegacyWrappingAlgorithms = true
})
```

Unauthenticated legacy object encryption algorithms

If you need to decrypt objects that were encrypted with a legacy algorithm, or you need to
partially decrypt an AES-GCM encrypted object by performing a [ranged request](#), you need to
use the unauthenticated legacy mode. The Amazon S3 Encryption Client will decrypt objects
with a legacy encryption algorithm, but will use the fully supported AES-GCM algorithm to
encrypt any objects that you upload to Amazon S3. The decryption of AES-CBC encrypted
objects and ranged requests are considered *unauthenticated* because the algorithms do not
provide any form of authentication to ensure the integrity of the object.

To enable the unauthenticated legacy mode, specify the
`enableLegacyUnauthenticatedModes` parameter when you instantiate the `v3Client`.

Java

> The following example specifies an AES key as the wrapping key. This client only encrypts
> with fully supported algorithms. However, it can decrypt objects encrypted with fully
> supported or legacy algorithms.

```
// v3

class v3UnauthenticatedLegacyModesExample {
    public static void main(String[] args) {
        S3Client v3Client = S3EncryptionClient.builder()
                .aesKey(aesKey)
                .enableLegacyUnauthenticatedModes(true)
                .build();
    }
}
```

Go

The following example creates a cryptographic materials manager that only encrypts with fully supported wrapping algorithms. However, it can decrypt objects encrypted with fully supported or legacy algorithms.

```
client, err := NewS3EncryptionClientV3(s3Client, cmm, func(clientOptions
 *client.EncryptionClientOptions) {
   clientOptions.EnableLegacyUnauthenticatedModes = true
 })
 if err != nil {
  // handle error
 }
```

The enableLegacyModes parameter is designed to be a temporary fix. After you've re-encrypted all of your objects with fully supported algorithms, you can remove it from your code.

Encryption algorithms (Version 2.*x* and earlier)

The following tables list the object encryption and wrapping algorithms that are supported in versions 2.*x* and earlier of the Amazon S3 Encryption Client. Versions 1.x and 2.x of the Amazon S3 Encryption Client are included in the following AWS SDKs.

Encrypting objects — The following table lists encryption algorithms that are used to encrypt objects.

Algorithm	C++	Go	Java	.NET	PHP v3	Ruby v2
AES-GCM	Full	Full	Full	Full	Full	Full
AES-CBC	Legacy	Legacy	Legacy	No	No	Legacy

Encrypting data keys — The following table lists encryption algorithms that are used to encrypt the data keys that were used to encrypt objects.

Algorithm	C++	Go	Java	.NET	PHP v3	Ruby v2
AES-ECB	No	No	Legacy	Legacy	No	Legacy
AES-GCM	Full	No	Full	Full	No	Full
AESWrap	Legacy	No	Legacy	Legacy	No	Legacy
KMS	Legacy	Legacy	Legacy	Legacy	Legacy	Legacy
KMS +context	Full	Full	Full	Full	Full	Full
RSA	No	No	Legacy	No	No	Legacy
RSA-OAEP-SHA1	No	No	Full	Full	No	Full

Document history for the Amazon S3 Encryption Client Developer Guide

The following table describes significant changes to the Amazon S3 Encryption Client Developer Guide. In addition to these major changes, we also update the documentation frequently to improve the descriptions and examples, and to address the feedback that you send to us.

Change	Description	Date
Amazon S3 Encryption Client for Go version 3.*x*	Added and updated documentation for version 3.*x* of the Amazon S3 Encryption Client for Go.	November 16, 2023
Initial release	The initial release of this documentation describes version 3.*x* of the Amazon S3 Encryption Client for Java.	April 5, 2023