ALIBABA CLOUD

阿里云

密钥管理服务

用户指南

文档版本：20200927

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认页面，单击确定。**
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用RAM实现对资源的访问控制

密钥管理服务KMS（Key Management Service）通过访问控制RAM（Resource Access Management）实现对资源的访问控制。本文为您介绍KMS定义的资源类型、操作和策略条件。

阿里云账号对自己的资源拥有完整的操作权限，RAM用户和RAM角色则需要通过显示授权获取对应资源的操作权限。

在了解如何使用RAM授权和访问主密钥之前，请了解以下内容：

- 什么是访问控制
- API概览

KMS定义的资源类型

下表列出了KMS定义的所有资源类型以及对应的资源名称（ARN），用于RAM权限策略的Resource元素。

资源类型	ARN
抽象密钥容器	acs:kms:${region}:${account}:key
抽象凭据容器	acs:kms:${region}:${account}:secret
抽象别名容器	acs:kms:${region}:${account}:alias
密钥	acs:kms:${region}:${account}:key/${key-id}
凭据	acs:kms:${region}:${account}:secret/${secret-name}
别名	acs:kms:${region}:${account}:alias/${alias-name}

KMS定义的操作

针对每一个需要进行访问控制的接口，KMS都定义了用于RAM权限策略的操作（Action），通常为 `kms:${api-name}` 。

> 说明 DescribeRegions不需要进行访问控制，只要请求可以通过认证校验，即可调用。调用者可以是阿里云账号、RAM用户或RAM角色。

以下表格列出了KMS接口的对应RAM权限策略操作，以及接口所访问的资源类型。

- 密钥服务接口

KMS接口	Action	资源类型
ListKeys	kms:ListKeys	抽象密钥容器
CreateKey	kms:CreateKey	抽象密钥容器
DescribeKey	kms:DescribeKey	密钥
UpdateKeyDescription	kms:UpdateKeyDescription	密钥
EnableKey	kms:EnableKey	密钥

KMS接口	Action	资源类型
DisableKey	kms:DisableKey	密钥
ScheduleKeyDeletion	kms:ScheduleKeyDeletion	密钥
CancelKeyDeletion	kms:CancelKeyDeletion	密钥
GetParametersForImport	kms:GetParametersForImport	密钥
ImportKeyMaterial	kms:ImportKeyMaterial	密钥
DeleteKeyMaterial	kms:DeleteKeyMaterial	密钥
ListAliases	kms:ListAliases	抽象别名容器
CreateAlias	kms:CreateAlias	别名和密钥
UpdateAlias	kms:UpdateAlias	别名和密钥
DeleteAlias	kms:DeleteAlias	别名和密钥
ListAliasesByKeyId	kms:ListAliasesByKeyId	密钥
CreateKeyVersion	kms:CreateKeyVersion	密钥
DescribeKeyVersion	kms:DescribeKeyVersion	密钥
ListKeyVersions	kms:ListKeyVersions	密钥
UpdateRotationPolicy	kms:UpdateRotationPolicy	密钥
Encrypt	kms:Encrypt	密钥
Decrypt	kms:Decrypt	密钥
ReEncrypt	○ kms:ReEncryptFrom ○ kms:ReEncryptTo ○ kms:ReEncrypt*	密钥
GenerateDataKey	kms:GenerateDataKey	密钥
GenerateDataKeyWithoutPlaintext	kms:GenerateDataKeyWithoutPlaintext	密钥
ExportDataKey	kms:ExportDataKey	密钥
GenerateAndExportDataKey	kms:GenerateAndExportDataKey	密钥
AsymmetricSign	kms:AsymmetricSign	密钥
AsymmetricVerify	kms:AsymmetricVerify	密钥

KMS接口	Action	资源类型
AsymmetricEncrypt	kms:AsymmetricEncrypt	密钥
AsymmetricDecrypt	kms:AsymmetricDecrypt	密钥
GetPublicKey	kms:GetPublicKey	密钥

- 凭据管家接口

KMS 接口	Action	资源类型
CreateSecret	kms:CreateSecret	抽象凭据容器
ListSecrets	kms:ListSecrets	抽象凭据容器
DescribeSecret	kms:DescribeSecret	凭据
DeleteSecret	kms:DeleteSecret	凭据
UpdateSecret	kms:UpdateSecret	凭据
RestoreSecret	kms:RestoreSecret	凭据
GetSecretValue	kms:GetSecretValue	凭据
PutSecretValue	kms:PutSecretValue	凭据
ListSecretVersionIds	kms:ListSecretVersionIds	凭据
UpdateSecretVersionStage	kms:UpdateSecretVersionStage	凭据
GetRandomPassword	kms:GetRandomPassword	无

- 标签管理接口

KMS 接口	Action	资源类型
ListResourceTags	kms:ListResourceTags	密钥或凭据
UntagResource	kms:UntagResource	密钥或凭据
TagResource	kms:TagResource	密钥或凭据

KMS支持的策略条件

您可以在RAM权限策略中设定条件控制对KMS的访问，只有当条件满足时，权限验证才能通过。例如，您可以使用 `acs:CurrentTime` 条件限制权限策略有效的时间。

除了阿里云全局条件，您也可以使用标签作为条件关键字，限制对Encrypt、Decrypt、GenerateDataKey等密码运算API的使用。条件关键字的格式为 `kms:tag/${tag-key}` 。

详情请参见权限策略基本元素。

常见的授权策略示例

- 允许访问所有的KMS资源

```
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:*"
   ],
   "Resource": [
    "*"
   ]
  }
 ]
}
```

- 只读访问权限，即列出和查看密钥、查看别名以及使用密钥权限

```
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:List*", "kms:Describe*",
    "kms:Encrypt", "kms:Decrypt", "kms:GenerateDataKey"
   ],
   "Resource": [
    "*"
   ]
  }
 ]
}
```

- 允许使用含有下列标签的密钥进行密码运算：
  - 标签键：`Project`
  - 标签值：`Apollo`

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt", "kms:Decrypt", "kms:GenerateDataKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEqualsIgnoreCase": {
          "kms:tag/Project": [
            "Apollo"
          ]
        }
      }
    }
  ]
}
```

2.使用ActionTrail记录操作事件

本文为您介绍如何使用ActionTrail记录密钥管理服务（KMS）操作事件。

KMS已经与操作审计（ActionTrail）服务进行了集成。您可以在ActionTrail中查看所有用户（主账号/RAM用户）对您的资源所进行的操作记录。

ActionTrail记录的KMS信息包括除DescribeRegions的所有API。详情请参见KMS。

关于操作记录的详细信息，请参见操作事件结构定义。

3.管理密钥

您可以使用密钥管理服务KMS（Key Management Service）轻松地创建和管理密钥，使用密钥加密自己的数据。

创建密钥

1. 登录密钥管理服务控制台。
2. 在**密钥列表**中，单击**创建密钥**。
3. 在弹出的**创建密钥**对话框，填写**别名**和**描述**。
4. 单击**高级选项**，选择**密钥材料来源**。
   - **阿里云KMS**：密钥材料将由KMS生成。
   - **外部**：KMS将不会生成密钥材料，您需要将自己的密钥材料导入KMS，详情请参见导入密钥材料。

     > **说明** 此时需要选中**我了解使用外部密钥材料的方法和意义**复选框。
5. 单击**确认**。

> **说明** 密钥创建完成后，您可以查看密钥ID、密钥状态、密钥保护级别等信息。

禁用密钥

密钥创建完成后，默认为启用状态。被禁用的密钥无法用于加密和解密。禁用密钥的步骤如下：

1. 在**密钥列表**中，找到需要禁用的密钥。
2. 在**操作**列，单击**禁用**。
3. 在弹出的**禁用密钥**对话框，单击**确定**。

计划删除密钥

主密钥一旦删除，将无法恢复，使用该主密钥加密的内容及产生的数据密钥也将无法解密。因此，对于主密钥的删除，KMS只提供申请删除的方式，而不提供直接删除的方式。如果您有删除密钥方面的需求，推荐您使用禁用密钥功能。

1. 在**密钥列表**中，找到计划删除的密钥。
2. 在**操作**列，单击**更多 > 计划删除密钥**。
3. 在弹出的**计划删除密钥**对话框，填写**预删除周期**。 **预删除周期**取值为：7~30天，默认值：30天。
4. 单击**确认**。

> **说明**
> - 处于待删除状态的密钥无法用于加密、解密和产生数据密钥。
> - 您可以单击**更多 > 取消删除密钥**来撤销删除密钥的申请。

4.别名概述

别名是用户主密钥的可选标识，在一个阿里云账号、一个地域中具有唯一性。同一个阿里云账号在不同地域下可以拥有相同的别名。 每个别名只能指向同地域的一个用户主密钥，但是每个用户主密钥可以拥有多个别名。

一个别名一定会指向一个用户主密钥，但是别名是独立于用户主密钥存在的一个资源。别名具有以下特点：

- 可以调用UpdateAlias接口更改别名关联的用户主密钥，而不会影响用户主密钥。
- 删除别名不会删除其关联的用户主密钥。
- RAM用户操作别名，需要有别名相关资源的授权。详情请参见使用RAM实现对资源的访问控制。
- 别名不可更改。您可以通过为一个用户主密钥创建新的别名，并且删除旧的别名来达到修改别名的目的。

当一个别名与某一个用户主密钥相关联时，在以下API接口中，可以将访问参数中的密钥ID用别名代替：

- DescribeKey
- Encrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext

当RAM用户使用别名代替密钥ID进行上述操作时，RAM用户必须拥有对应密钥的权限，无需拥有对应别名的权限。

您可以对别名进行以下操作：

- 创建别名
- 更新别名
- 删除别名
- 列出别名
- 列出指定密钥关联的别名

任何情况下使用别名时，都必须使用完整的别名，示例如下。

```
//包含前缀“alias/”的完整别名
alias/example
```

更新别名

更新别名可以将已有的别名关联到其他用户主密钥，您可以调用UpdateAlias接口更新别名。

当RAM用户更新别名时，需要同时拥有原密钥、目标密钥以及别名的权限。

```
//更新别名的RAM Policy示例：用户123456可以将密钥08ec3bb9-034f-485b-b1cd-3459baa8****关联的别名alias
/example关联到新密钥127d2f84-ee5f-4f4d-9d41-dbc1aca28788
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:UpdateAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/08ec3bb9-034f-485b-b1cd-3459baa8****",
    "acs:kms:cn-hangzhou:123456:key/127d2f84-ee5f-4f4d-9d41-dbc1aca2****",
    "acs:kms:cn-hangzhou:123456:alias/example"
   ]
  }
 ]
}

//更新别名
aliyun kms UpdateAlias --AliasName alias/example --KeyId 127d2f84-ee5f-4f4d-9d41-dbc1aca2****
```

列出别名

列出别名可以获取所有的别名信息，您可以调用ListAliases接口列出别名。

当RAM用户访问别名时，需要拥有别名资源类型的权限。

```
//列出别名的RAM Policy示例
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:ListeAliases"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:alias"
   ]
  }
 ]
}

//列出别名
aliyun kms ListAliases
```

列出指定密钥关联的别名

列出指定密钥关联的别名只会返回与指定密钥相关联的别名信息，您可以调用ListAliasesByKeyId接口列出指定密钥关联的别名。

当RAM用户列出与指定密钥关联的别名时，只需拥有指定密钥的权限即可访问。

```
//出指定密钥关联的别名的RAM Policy示例：列出密钥127d2f84-ee5f-4f4d-9d41-dbc1aca2****关联的所有别名
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:DeleteAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/127d2f84-ee5f-4f4d-9d41-dbc1aca2****"
   ]
  }
 ]
}

//列出指定密钥关联的别名
aliyun kms ListAliasesByKeyId --KeyId 127d2f84-ee5f-4f4d-9d41-dbc1aca2****
```

5.用户主密钥概述

由密钥管理服务KMS（Key Management Service）托管的用户主密钥有不同的类型。您可以根据需要，使用不同类型的密钥完成特定功能，例如对数据进行加密和解密，或者用于数字签名的生成和验证。

密钥的算法

KMS的密钥支持多种算法，用于实现不同的密码运算。密钥主要分为对称密钥和非对称密钥两大类型。

密码算法大类	密码算法子类	是否支持加密、解密	是否支持签名、验签
对称密钥	AES	支持	不支持
对称密钥	SM4 说明	支持	不支持
非对称密钥	RSA	支持	支持
非对称密钥	ECC	不支持	支持
非对称密钥	SM2 说明	支持	支持

> **说明** 仅中国内地的托管密码机支持SM4 、SM2密钥，详情请参见支持的地域。

对称密钥主要用于数据的加密保护场景。如果您不指定具体的密钥规格（KeySpec），KMS默认创建对称密钥。通过使用KMS加解密的接口，您无需获得密钥的明文材料即可完成对数据的加解密操作。详情请参见对称加密概述。

非对称密钥可用于数据加密和数字签名。您在KMS创建的非对称用户主密钥（CMK），由一对关联的公钥和私钥构成。公钥可以被分发给任何人，而私钥必须进行妥善保管。KMS可以确保私钥的安全性，不提供任何接口导出非对称密钥的私钥。您可以通过私钥运算的接口来使用私钥进行数据解密或数字签名。任何获得公钥的人，都可以使用公钥进行数据加密或验证私钥产生的签名。详情请参见非对称密钥概述。

密钥的保护级别

KMS提供托管密码机，您可以指定用户主密钥的保护级别为HSM，将密钥托管在密码机中，使密钥获得高安全等级的专用硬件保护。保护级别为HSM的主密钥，其密钥材料的明文只会存在于密码机的内部。KMS通过密码机的接口进行密码运算，在运算过程中，KMS和阿里云的运维人员均无法接触到密钥材料的明文。这类密钥无法被明文导出密码机。详情请参见托管密码机简介和使用托管密码机。

如果您指定主密钥的保护级别为Software，则KMS通过软件模块对密钥进行保护，并且使用TPM对软件模块进行根保护。

密钥的托管者

通常情况下，您是KMS中的用户主密钥的托管者。在此情况下，密钥的Creator属性为您的阿里云账号的标识符。

在云服务集成KMS进行服务端加密的场景里，为了更方便您使用入门级的数据加密功能，缩减您在密钥生命周期和权限管理等方面的开销，KMS支持特定云服务为您自动托管一个加密密钥，专门用于相应的云服务对您的数据进行加密保护。这类由云服务托管的密钥被称为服务密钥。为了方便您辨识，服务密钥的Creator属性为相应的云服务代码，同时被关联了特殊的别名，格式为 `acs/<云服务代码>` 。例如：对象存储服务OSS为用户托管的服务密钥，Creator属性为OSS，同时被关联了别名acs/oss。详情请参见服务端集成加密概述。

6.使用对称密钥

6.1. 对称加密概述

对称加密是最常用的数据加密保护方式。KMS提供了简单易用的接口，方便您在云上轻松实现数据加解密功能。

如果您不指定具体的密钥规格（KeySpec），KMS默认创建对称密钥。阿里云支持主流的对称密钥算法并且提供足够的安全强度，保证数据加密的安全性。

对称密钥的类型

KMS支持的对称密钥算法类型如下：

算法	密钥长度	密钥规格	数据加密模式	保护级别
AES	256比特	Aliyun_AES_256	GCM	• Software • HSM
SM4 说明	128比特	Aliyun_SM4	GCM	HSM

> **说明** KMS通过托管密码机提供SM4算法，详情请参见托管密码机简介。

加解密特性

调用Encrypt、ReEncrypt、GenerateDataKey或GenerateDataKeyWithoutPlaintext接口加密时，您只需指定用户主密钥的标识符（或别名），KMS使用指定的用户主密钥完成加密后，返回密文数据。调用Decrypt接口进行解密时，您只需要传入密文数据，而不需要再次指定主密钥的标识符。

使用额外认证数据

KMS的对称密钥使用了分组密码算法的GCM模式，支持您传入额外的认证数据（Additional Authenticated Data，简称AAD），为需要加密的数据提供额外的完整性保护。KMS对额外认证数据的传入进行了封装，帮助您更方便的自定义认证数据，详情请参见EncryptionContext说明。

信封加密

KMS通过GenerateDataKey和GenerateDataKeyWithoutPlaintext接口，一次性产生两级密钥结构，支持更快速的实现信封加密。详情请参见：

- 什么是信封加密？
- 使用KMS信封加密在本地加密或解密数据

轮转对称加密密钥

KMS生成的对称主密钥支持多个密钥版本，同时支持用户主密钥基于密钥版本进行自动轮转，您可以自定义密钥轮转的策略。

在CMK包含多个版本时，KMS的加密动作（例如：Encrypt、GenerateDataKey和GenerateDataKeyWithoutPlaintext）使用指定用户主密钥的最新密钥版本对数据进行加密。解密时，您不需要传入用户主密钥标识符和版本标识符，KMS会自动发现传入密文数据使用的用户主密钥以及对应的加密密钥版本，使用相应版本的密钥材料对数据进行解密。

密钥的自动轮转通过产生新的密钥版本来实现。轮转后，调用KMS接口产生的密文数据自动使用最新版本，而轮转之前产生的密文数据仍然可以使用旧的密钥版本解密。详情请参见自动轮转密钥。

使用自带密钥

为了满足更高的安全合规要求，KMS支持您使用自带密钥（BYOK）进行云上数据的加密保护。对于自带密钥的情形，我们推荐您使用托管密码机对密钥进行保护，将您的密钥导入到保护级别为HSM的用户主密钥中。导入到托管密码机中的密钥只能被销毁，其明文无法被导出。详情请参见导入密钥材料。

6.2. EncryptionContext说明

EncryptionContext是在KMS的Encrypt、GenerateDataKey、Decrypt等API中可能用到的JSON字符串。

EncryptionContext的作用

EncryptionContext只能是String-String形式的JSON字符串，用于保护数据的完整性。

如果加密（Encrypt、GenerateDataKey）时指定了该参数，解密（Decrypt）密文时，需要传入等价的EncryptionContext参数，才能正确的解密。EncryptionContext虽然与解密相关，但是并不会存在密文（CipherBlob）中。

EncryptionContext有效值

EncryptionContext的有效值是一个总长度在8192个字符数以内的JSON字符串，并且只能是String-String形式。当您直接调用API填写EncryptionContext时，请注意转义问题。

有效的EncryptionContext示例

```
{"ValidKey":"ValidValue"}
{"Key1":"Value1","Key2":"Value2"}
```

无效的EncryptionContext（部分）示例

```
[{"Key":"Value"}] //json数组
{"Key":12345} //String-int
{"Key":["value1","value2"]} //String-数组
```

等价的EncryptionContext

EncryptionContext的本质是一个String-String的map（hashtable），因此在作为参数时，只需要保证JSON字符串所表示的key-value含义是一致的，则EncryptionContext是等价的。与加密时输入的EncryptionContext等价的EncryptionContext即可用于正确的解密，而不用保持完全一致的字符串。

等价的EncryptionContext示例

```
{"Key1":"Value1","Key2":"Value2"} 与 {"Key2":"Value2","Key1":"Value1"} 等价
```

6.3. 导入密钥材料

当您创建密钥材料来源为外部的密钥时，KMS不会为您创建的用户主密钥（CMK）生成密钥材料，此时您可以将自己的密钥材料导入到CMK中。本文为您介绍如何导入外部密钥材料。

前提条件

进行操作前，请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

背景信息

用户主密钥（CMK）是KMS的基本资源，由密钥ID、基本元数据（如密钥状态等）以及用于加密、解密数据的密钥材料组成。默认情况下，当您创建主密钥时，会由KMS生成密钥材料。您也可以选择创建密钥材料来源为外部的密钥，此时，KMS将不会为您创建的CMK生成密钥材料，您可以将自己的密钥材料导入到CMK中。

您可以通过调用DescribeKey接口来判断密钥材料来源情况。

- 当KeyMetadata中Origin为*Aliyun_KMS*时，说明密钥材料由KMS生成，称为**普通密钥**。
- 当Origin为*EXTERNAL*时，说明密钥材料由外部导入，称为**外部密钥**。

在您选择密钥材料来源为外部，使用您自己导入的密钥材料时，需要注意以下几点：

- 请确保您使用了符合要求的随机源生成密钥材料。
- 请确保密钥材料的可靠性。
  - KMS可以确保导入密钥材料高可用，但是不能确保导入密钥材料与KMS生成的密钥材料具有相同的可靠性。
  - 您导入的密钥材料可以直接调用DeleteKeyMaterial接口进行删除，也可以设置过期时间，在密钥材料过期后进行删除（CMK不会被删除）。KMS生成的密钥材料无法直接被删除，只能通过调用ScheduleKeyDeletion接口，在等待7到30天后，随着CMK一起被删除。
  - 当导入的密钥材料被删除后，可以再次导入相同的密钥材料使得CMK再次可用，因此您需要自行保存密钥材料的副本。
- 每个CMK只能拥有一个导入密钥材料。当您将一个密钥材料导入CMK时，该CMK将与该密钥材料绑定，即便密钥材料已经过期或者被删除，也不能导入其他密钥材料。如果您需要轮换使用外部密钥材料的CMK，只能创建一个新的CMK然后导入新的密钥材料。
- CMK具有独立性。例如：您使用CMK加密的数据，无法使用其他CMK进行解密，即便这些CMK都使用相同的密钥材料。
- 只能导入128位或256位对称密钥作为密钥材料。128位密钥仅支持部分地域，详情请参见支持的地域。

导入密钥材料（控制台）

1. 创建外部密钥。
   i. 登录密钥管理服务控制台。
   ii. 在页面左上角的地域下拉列表，选择密钥所在的地域。
   iii. 在左侧导航栏单击**密钥列表**。
   iv. 单击**创建密钥**。
   v. 在弹出的**创建密钥**对话框，选择**密钥类型**。请选择对称密钥类型Aliyun_AES_256或Aliyun_SM4。密钥类型详情请参见对称密钥的类型。
   vi. 配置**别名**、**描述**和**轮转周期**。

vii. 单击**高级选项**，选择**密钥材料来源**为**外部**。

viii. 勾选**我了解使用外部密钥材料的方法和意义**，单击**确认**。

2. 获取导入密钥材料参数。导入密钥材料参数包括一个用于加密密钥材料的公钥，以及一个导入令牌。

   i. 在左侧导航栏单击**密钥列表**。

   ii. 找到待导入密钥材料的密钥，单击**别名**，进入密钥管理页面。

   iii. 在**密钥材料**区域，单击**获取导入密钥材料参数**。

   iv. 在弹出的**获取导入密钥材料参数**对话框，选择**加密算法**，单击**下一步**。请选择加密算法RSAES_OAEP_SHA_1、RSAES_OAEP_SHA_256或RSAES_PKCS1_V1_5。

   v. 在弹出的**获取导入密钥材料参数**对话框，下载**加密公钥**和**导入令牌**，单击**关闭**。

3. 使用OPENSSL加密密钥材料。加密公钥是一个2048比特的RSA公钥，使用的加密算法需要与获取导入密钥材料参数时指定的一致。由于加密公钥经过Base64编码，因此在使用时需要先进行Base64解码。您可以通过OPENSSL加密公钥，获取您自己的密钥材料。

   i. 创建一个密钥材料，使用OPENSSL产生一个32字节的随机数进行演示。

   ii. 将加密公钥进行Base64解码。

   iii. 根据指定的加密算法（以RSAES_OAEP_SHA_1为例）加密密钥材料。

   iv. 将加密后的密钥材料进行Base64编码，保存为文本文件。

```
openssl rand -out KeyMaterial.bin 32
openssl enc -d -base64 -A -in PublicKey_base64.txt -out PublicKey.bin
openssl rsautl -encrypt -in KeyMaterial.bin -oaep -inkey PublicKey.bin  -keyform DER  -pubin -out EncryptedKeyMaterial.bin
openssl enc -e -base64 -A -in EncryptedKeyMaterial.bin -out EncryptedKeyMaterial_base64.txt
```

4. 导入密钥材料。

   导入密钥材料时，可以将从未导入过密钥材料的外部密钥进行导入，也可以重新导入已经过期和已被删除的密钥材料，或者重置密钥材料的过期时间。

   导入令牌与加密密钥材料的公钥具有绑定关系，一个令牌只能为其生成时指定的主密钥导入密钥材料。导入令牌的有效期为24小时，在有效期内可以重复使用，失效以后需要获取新的导入令牌和加密公钥。

   i. 在左侧导航栏单击**密钥列表**。

   ii. 找到待导入密钥材料的密钥，单击**别名**，进入密钥管理页面。

   iii. 在**密钥材料**区域，单击**导入密钥材料**。

   iv. 在弹出的**导入密钥材料**对话框，上传**加密密钥材料**和**导入令牌**。

   - **加密密钥材料**：上传步骤3生成的密钥材料文本文件。
   - **导入令牌**：上传步骤2获取的导入令牌文本文件。

   v. 设置**密钥材料过期时间**，单击**确认**。导入密钥材料成功后，密钥状态从**待导入**更新为**启用中**。

导入密钥材料（ALIYUN CLI）

1. 创建外部密钥。通过命令**aliyun kms CreateKey**调用CreateKey接口，指定参数Origin为*EXTERNAL*。

```
aliyun kms CreateKey --Origin EXTERNAL --Description "External key"
```

2. 获取导入密钥材料参数。通过命令**aliyun kms GetParametersForImport**调用GetParametersForImport接口获取导入密钥材料参数。

```
aliyun kms GetParametersForImport --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8**** --WrappingAlgorithm RSAES_OAEP_SHA_1 --WrappingKeySpec RSA_2048
```

3. 导入密钥材料。
   i. 使用加密公钥对密钥材料进行加密。

      加密公钥是一个RSA（2048比特）或者SM2公钥，使用的加密算法需要与获取导入密钥材料参数时指定的一致。由于API返回的加密公钥经过Base64编码，因此在使用时需要先进行Base64解码。目前KMS支持的加密算法有RSAES_OAEP_SHA_1、RSAES_OAEP_SHA_256、RSAES_PKCS1_V1_5和SM2PKE，其中SM2PKE仅部分地域支持，详情请参见支持的地域。

   ii. 将加密后的密钥材料进行Base64编码。

   iii. 通过命令**aliyun kms ImportKeyMaterial**调用ImportKeyMaterial接口，将编码后的密钥材料与导入令牌一起，作为ImportKeyMaterial接口的参数导入KMS。

```
aliyun kms ImportKeyMaterial --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8**** --EncryptedKeyMaterial xxx --ImportToken xxxx
```

代码样例

- **JAVA SDK**

```
//使用最新KMS JAVA SDK。
//KmsClient.java

import com.aliyuncs.kms.model.v20160120.*;
import com.aliyuncs.profile.DefaultProfile;

//KMS API封装。
public class KmsClient {
    DefaultAcsClient client;

    public KmsClient( String region_id, String ak, String secret) {
        DefaultProfile profile = DefaultProfile.getProfile(region_id, ak, secret);
        this.client = new DefaultAcsClient(profile);
    }

    public CreateKeyResponse createKey() throws Exception {
        CreateKeyRequest request = new CreateKeyRequest();
        request.setOrigin("EXTERNAL"); //创建外部密钥。
        return this.client.getAcsResponse(request);
    }
```

```
        }
        //... 省略，其余API类似。
}
//example.java
import com.aliyuncs.kms.model.v20160120.*;
import KmsClient
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.MGF1ParameterSpec;
import javax.crypto.Cipher;
import javax.crypto.spec.OAEPParameterSpec;
import javax.crypto.spec.PSource.PSpecified;
import java.security.spec.X509EncodedKeySpec;
import java.util.Random;
import javax.xml.bind.DatatypeConverter;

public class CreateAndImportExample {
        public static void main(String[] args) {
                String regionId = "cn-hangzhou";
        String accessKeyId = "*** Provide your AccessKeyId ***";
        String accessKeySecret = "*** Provide your AccessKeySecret ***";
        KmsClient kmsclient = new KmsClient(regionId,accessKeyId,accessKeySecret);
        //创建外部密钥。
        try {
                CreateKeyResponse keyResponse = kmsclient.createKey();
                String keyId = keyResponse.KeyMetadata.getKeyId();
                //产生一个32字节随机数。
                byte[] keyMaterial = new byte[32];
                new Random().nextBytes(keyMaterial);
                //获取导入密钥材料参数。
                GetParametersForImportResponse paramResponse = kmsclient.getParametersForImport(k
eyId,"RSAES_OAEP_SHA_256");
                String importToekn = paramResponse.getImportToken();
                String encryptPublicKey = paramResponse.getPublicKey();
                //Base64解码加密公钥。
                byte[] publicKeyDer = DatatypeConverter.parseBase64Binary(encryptPublicKey);
                //解析成RSA的公钥。
                KeyFactory keyFact = KeyFactory.getInstance("RSA");
                X509EncodedKeySpec spec = new X509EncodedKeySpec(publicKeyDer);
                PublicKey publicKey = keyFact.generatePublic(spec);
                //加密密钥材料。
```

```
            Cipher oaepFromAlgo = Cipher.getInstance("RSA/ECB/OAEPWithSHA-1AndMGF1Padding");
            String hashFunc = "SHA-256";
            OAEPParameterSpec oaepParams = new OAEPParameterSpec(hashFunc, "MGF1", new MGF1
ParameterSpec(hashFunc), PSpecified.DEFAULT);
            oaepFromAlgo.init(Cipher.ENCRYPT_MODE, publicKey, oaepParams);
            byte[] cipherDer = oaepFromAlgo.doFinal(keyMaterial);
            //加密后的密钥材料，需要进行Base64编码。
            String encryptedKeyMaterial = DatatypeConverter.printBase64Binary(cipherDer);
            //导入密钥材料。
            Long expireTimestamp = 1546272000L; //Unix时间戳，精确到秒，0表示永不过期。
                kmsClient.importKeyMaterial(keyId,encryptedKeyMaterial, expireTimestamp);
        } catch(Exception e) {
            //... 省略。
        }
        }
}
```

- **Go SDK**

```
package main

import (
    "crypto/rand"
    "crypto/rsa"
    "crypto/sha256"
    "crypto/x509"
    "encoding/base64"
    "fmt"
    "log"
    random "math/rand"
    "time"

    "github.com/aliyun/alibaba-cloud-sdk-go/services/kms"
)

//KMS CreateKey API封装。
func kmsCreateKey(client *kms.Client) (string, error) {
    request := kms.CreateCreateKeyRequest()
    request.Scheme = "https"
    request.Origin = "EXTERNAL" //创建外部密钥。
    response, err := client.CreateKey(request)
    if err != nil {
```

```
        return "", fmt.Errorf("CreateKey error:%v", err)
    }
    return response.KeyMetadata.KeyId, nil
}

//KMS GetParametersForImport API封装。
func kmsGetParametersForImport(client *kms.Client, keyId, wrappingKeySpec, wrappingAlgorithm st
ring) (string, string, error) {
    request := kms.CreateGetParametersForImportRequest()
    request.Scheme = "https"
    request.KeyId = keyId
    request.WrappingKeySpec = wrappingKeySpec
    request.WrappingAlgorithm = wrappingAlgorithm
    response, err := client.GetParametersForImport(request)
    if err != nil {
        return "", "", fmt.Errorf("GetParametersForImport error:%v", err)
    }
    return response.PublicKey, response.ImportToken, nil
}

//KMS ImportKeyMaterial API封装。
func kmsImportKeyMaterial(client *kms.Client, keyId, importToken, encryptedKeyMaterial string) err
or {
    request := kms.CreateImportKeyMaterialRequest()
    request.Scheme = "https"
    request.KeyId = keyId
    request.ImportToken = importToken
    request.EncryptedKeyMaterial = encryptedKeyMaterial
    _, err := client.ImportKeyMaterial(request)
    if err != nil {
        return fmt.Errorf("ImportKeyMaterial error:%v", err)
    }
    return nil
}

func randBytes(n int) []byte {
    var r = random.New(random.NewSource(time.Now().UnixNano()))
    bytes := make([]byte, n)
    for i := range bytes {
        bytes[i] = byte(r.Intn(256))
    }
```

```
    return bytes
}

func main() {
    accessKeyId := "*** Provide your AccessKeyId ***"
    accessKeySecret := "*** Provide your AccessKeySecret ***"
    regionId := "cn-hangzhou"
    client, err := kms.NewClientWithAccessKey(regionId, accessKeyId, accessKeySecret)
    if err != nil {
        log.Fatalf("NewClientWithAccessKey error:%+v\n", err)
    }
    //创建一个外部密钥。
    keyId, err := kmsCreateKey(client)
    if err != nil {
        log.Fatalf("kmsCreateKey error:%+v\n", err)
    }
    //以下示例代码产生一个32字节随机数。在实际应用中您需要通过用户的密钥管理系统生成密钥，并使用导入密钥材料参数中的公钥进行加密。
    keyMaterial := randBytes(32)
    //获取导入密钥材料参数。
    encryptPublicKey, importToken, err := kmsGetParametersForImport(client, keyId, "RSA_2048", "RSAES_OAEP_SHA_256")
    if err != nil {
        log.Fatalf("kmsGetParametersForImport error:%v\n", err)
    }
    //Base64解码加密公钥。
    publicKeyDer, err := base64.StdEncoding.DecodeString(encryptPublicKey)
    if err != nil {
        log.Fatalf("base64.StdEncoding.DecodeString error:%v\n", err)
    }
    //解析成RSA的公钥。
    publicKey, err := x509.ParsePKIXPublicKey(publicKeyDer)
    if err != nil {
        log.Fatalf("x509.ParsePKIXPublicKey error:%v\n", err)
    }
    //加密密钥材料。
    cipherDer, err := rsa.EncryptOAEP(sha256.New(), rand.Reader, publicKey.(*rsa.PublicKey), keyMaterial, nil)
    if err != nil {
        log.Fatalf("rsa.EncryptOAEP error:%v\n", err)
```

```
    }
    //加密后的密钥材料，需要进行Base64编码。
    encryptedKeyMaterial := base64.StdEncoding.EncodeToString(cipherDer)
    //导入密钥材料。
    err = kmsImportKeyMaterial(client, keyId, importToken, encryptedKeyMaterial)
    if err != nil {
        log.Fatalf("ImportKeyMaterial error:%v", err)
    }
}
```

6.4. 删除密钥材料

当您导入密钥材料后，可以直接删除密钥材料，此时密钥将无法继续使用，由该密钥加密的密文也无法被解密。本文为您介绍如何删除密钥材料。

前提条件

请确保您已经导入密钥材料，详情请参见导入密钥材料。

背景信息

当您导入密钥材料后，即可像使用普通密钥一样使用外部密钥。与普通密钥不同，外部密钥的密钥材料可能会过期，也可以直接删除。当密钥材料过期或者被删除以后，密钥将无法继续使用，由该密钥加密的密文也无法被解密。您可以导入相同的密钥材料使得CMK再次可用，建议您自行保存密钥材料的副本。

删除密钥材料（控制台）

1. 登录密钥管理服务控制台。
2. 在页面左上角的地域下拉列表，选择密钥所在的地域。
3. 在左侧导航栏，单击**密钥列表**。
4. 找到待删除密钥材料的密钥，单击**别名**，进入密钥管理页面。
5. 在**密钥材料**区域，单击**删除密钥材料**。密钥材料删除成功后，密钥状态从**启用中**更新为**待导入**。

删除密钥材料（ALIYUN CLI）

通过命令**aliyun kms DeleteKeyMaterial**调用DeleteKeyMaterial接口删除密钥材料。

```
aliyun kms DeleteKeyMaterial --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8****
```

7.使用非对称密钥

7.1. 非对称密钥概述

相比对称加密，非对称密钥通常用于在信任程度不对等的系统之间，实现数字签名验签或者加密传递敏感信息。

非对称密钥由一对公钥和私钥组成，他们在密码学上互相关联，其中的公钥可以被分发给任何人，而私钥必须被安全的保护起来，只有受信任者可以使用。阿里云支持主流的非对称密钥算法并且提供足够的安全强度，保证数据加密和数字签名的安全性。

KMS支持对非对称密钥类型的用户主密钥生成证书签名请求CSR（Certificate Signing Request）文件，证书申请者提交CSR给证书颁发机构后，证书颁发机构使用其CA私钥为用户签发数字证书。签发的数字证书可以用于安全电子邮件、安全终端保护、代码签名保护、可信网站服务、身份授权管理等。

非对称密钥的类型

KMS支持的非对称密钥算法类型如下：

算法	密钥规格	说明	用途
RSA	RSA_2048	RSA非对称密码	• 数据的加解密运算 • 数字签名
ECC	• EC_P256：NIST推荐的椭圆曲线P-256 • EC_P256K：SECG椭圆曲线secp256k1	椭圆曲线密码（Elliptic Curve Cryptography）	数字签名
SM2	EC_SM2	标准GBT32918定义的椭圆曲线密码	• 数据的加解密运算 • 数字签名

数据加密

非对称密钥用于数据加密，通常适用于传递敏感信息，典型场景如下：

1. 信息接收者将加密公钥分发给信息传送者。
2. 信息传送者使用公钥对敏感信息进行加密保护。
3. 信息传送者将敏感信息的密文传递给信息接收者。
4. 信息接收者使用私钥将敏感信息的密文解密。

由于解密的私钥只有信息接收者可以使用，因此可以确保敏感信息的明文在传递过程中不被恶意者截获。这种敏感信息传递的方式，被广泛用于各类密钥交换场景。例如在TLS中交换会话密钥、在不同的密码机之间导入导出加密密钥。

详情请参见非对称数据加密。

数字签名

非对称密钥更广泛的用途是数字签名，即使用私钥对消息或者信息产生签名。由于私钥受到严格保护，只有受信者可以使用私钥来产生签名，使用公钥验证签名可以实现以下目的：

- 验证数据的完整性（integrity）：如果数据和签名不匹配，数据可能受到了篡改。
- 验证消息的真实性（authenticity）：如果消息和签名不匹配，消息传送者不是真实持有私钥的用户。
- 为签名提供不可抵赖性（non-repudiation）：如果数据和签名能够匹配，签名者不可以否认此签名。

典型的签名验签场景如下：

1. 签名者将验签公钥分发给消息接收者。
2. 签名者使用签名私钥，对数据产生签名。
3. 签名者将数据以及签名传递给消息接收者。
4. 消息接收者获得数据和签名后，使用公钥针对数据验证签名的合法性。

数字签名被广泛用于数据防篡改、身份认证等相关技术领域。

- 案例1：数字签名用于对二进制代码提供完整性保护，代码执行者可以验证代码未被篡改，以提供可信的执行环境。
- 案例2：数字证书系统中，证书机构（CA）对颁发的数字证书提供签名，证明数字证书的主体信息、公私钥信息、密钥用途、有效期、签发者等信息。证书私钥持有者使用私钥对消息进行签名，消息接收者使用证书中包含的公钥对消息签名进行验证，同时使用证书签发者的公钥，验证证书本身的合法性。

详情请参见非对称数字签名。

密钥版本

由于公私钥使用场景的特殊性，KMS不支持对非对称的用户主密钥进行自动轮转。您可以调用CreateKeyVersion接口，在指定用户主密钥中创建新的密钥版本，生成全新的一对公钥和私钥。如果您使用新的版本进行数字签名或者数据加密，则需要重新分发新版本的公钥。

除此之外，和对称类型的用户主密钥不同，非对称的用于主密钥没有主版本（PrimaryKeyVersion）的概念，因此使用非对称密码运算的接口除需指定用户主密钥标志符（或别名）之外，还需指定密钥版本。

公钥运算的方式

大多数情况下，公钥加密、公钥验签的操作，都可以调用GetPublicKey接口获取公钥，之后分发给公钥使用者。使用者在业务端通过OpenSSL、Java JCE等常用的密码运算库进行本地计算。

KMS也提供公钥运算的接口AsymmetricEncrypt和AsymmetricVerify，满足您的业务需求。和在业务端使用公钥本地运算相比，调用KMS的公钥运算接口可以帮助您更方便的记录调用日志，或者通过访问控制服务对公钥的使用场景进行一些限制，满足您的特定需求。

私钥运算的方式

您仅能通过KMS提供的私钥运算的接口AsymmetricDecrypt和AsymmetricSign，来使用私钥进行数据解密或者数字签名。

7.2. 非对称数据加密

本文以CLI使用为例，简单描述了使用非对称CMK进行数据加密和解密的场景。

非对称加密场景，通常包含以下步骤：

1. 信息接收者将加密公钥分发给信息传送者。
2. 信息传送者使用公钥对敏感信息进行加密保护。
3. 信息传送者将敏感信息的密文传递给信息接收者。
4. 信息接收者使用私钥将敏感信息的密文解密。

前提条件

您需要调用CreateKey接口，在KMS中创建非对称密钥（设定KeySpec参数），并且设定Usage参数为 `ENCRYPT/DECRYPT` 。CreateKey接口详情，请参见CreateKey。

创建RSA加密密钥：

```
$ aliyun kms CreateKey --KeySpec=RSA_2048 --KeyUsage=ENCRYPT/DECRYPT --ProtectionLevel=HSM
```

获取公钥

1. 调用GetPublicKey接口获取非对称密钥的公钥。

```
$ aliyun kms GetPublicKey --KeyId=**** --KeyVersionId=****
```

返回结果如下：

```
{
    "RequestId": "82c383eb-c377-4mf6-bxx8-81hkc1g5g7ab",
    "KeyId": "****",
    "KeyVersionId": "****",
    "PublicKey": "PublicKey-Data****"
}
```

2. 将公钥存入名为rsa_publickey.pub的文件中（PublicKey-Data****是占位符，请替换为真实的公钥）。

```
$ echo PublicKey-Data**** > rsa_publickey.pub
```

使用公钥加密数据

1. 创建一个示例明文文件plaintext-file.txt，包含内容“this is plaintext"。

```
echo "this is plaintext" > plaintext-file.txt
```

2. 使用OpenSSL对文件进行加密，将得到的二进制密文写入文件plaintext-file.enc。

```
openssl pkeyutl -encrypt -in plaintext-file.txt \
 -inkey rsa_publickey.pub -pubin \
 -pkeyopt rsa_padding_mode:oaep \
 -pkeyopt rsa_oaep_md:sha256 \
 -pkeyopt rsa_mgf1_md:sha256 \
 -out plaintext-file.enc
```

调用KMS解密数据

您需要调用KMS接口使用私钥解密数据。

1. 在对密文数据进行网络传输之前，首先需要对其进行Base64编码。

```
$ openssl base64 -in plaintext-file.enc
```

Base64编码后的密文如下：

```
5kdCB06HHeAwgfH9ARY4/9Nv5vlpQ94GXZcmaC9FE59Aw8v8RYdozT6ggSbyZbi+
8STKVq9402MEfmUDmwJLuu0qgAZsCe5wU4JWHh1y84Qn6HT068j0qOy5X2HIlrjs
fCdetgtMtVorSgb3bbERk2RV67nHWrDkecNbUaz+6ik4AlZxv2uWrV62eQ9yUBYm
Jb956LbqnfWdCFxUSHH/qB5QCnLpijzvPmfNlZr653H4nF08gpZjnmlF4FjTu3i2
mGLzK4J3Rh/l7PQHiVMdc4hSnXosg68QmMVdZBGLK9/cD9SYngPDiirU7z0q7Git
dIeloyCAUDFyuQC6a+SqzA==
```

2. 将Base64编码后的密文传入KMS，解密数据。

```
aliyun kms AsymmetricDecrypt \
 --KeyId **** \
 --KeyVersionId **** \
 --Algorithm RSAES_OAEP_SHA_256 \
 --CiphertextBlob 5kdCB06HHeAwgfH9ARY4/9Nv5vlpQ94GXZcmaC9FE59Aw8v8RYdozT6ggSbyZbi+8STKV
q9402MEfmUDmwJLuu0qgAZsCe5wU4JWHh1y84Qn6HT068j0qOy5X2HIlrjsfCdetgtMtVorSgb3bbERk2RV67n
HWrDkecNbUaz+6ik4AlZxv2uWrV62eQ9yUBYmJb956LbqnfWdCFxUSHH/qB5QCnLpijzvPmfNlZr653H4nF08g
pZjnmlF4FjTu3i2mGLzK4J3Rh/l7PQHiVMdc4hSnXosg68QmMVdZBGLK9/cD9SYngPDiirU7z0q7GitdIeloyCAU
DFyuQC6a+SqzA==
```

返回结果如下：

```
{
      "KeyId": "****",
      "KeyVersionId": "****",
      "Plaintext": "dGhpcyBpcyBwbGFpbnRleHQgDQo=",
      "RequestId": "6be7a8e4-35b9-4549-ad05-c5b1b535a22c"
}
```

3. 返回结果中的Plaintext经过Base64编码，对其进行Base64解码。

```
echo dGhpcyBpcyBwbGFpbnRleHQgDQo= | openssl base64 -d
```

解密后的明文数据如下：

```
this is plaintext
```

7.3. 非对称数字签名

本文以CLI使用为例，简单描述了使用非对称CMK生成数字签名以及验证签名的场景。您也可以通过KMS的SDK来实现。

非对称加密场景，通常包含以下步骤：

1. 签名者将验签公钥分发给消息接收者。
2. 签名者使用签名私钥，对数据产生签名。
3. 签名者将数据以及签名传递给消息接收者。
4. 消息接收者获得数据和签名后，使用公钥针对数据验证签名的合法性。

开始之前

您需要调用CreateKey接口，在KMS中创建恰当类型的非对称密钥（设定KeySpec参数），并且设定Usage参数为 `SIGN/VERIFY` 。

- 创建RSA签名密钥：

```
$ aliyun kms CreateKey --KeySpec=RSA_2048 --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

- 创建NIST P-256签名密钥：

```
$ aliyun kms CreateKey --KeySpec=EC_P256 --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

- 创建secp256k1签名密钥：

```
$ aliyun kms CreateKey --KeySpec=EC_P256K --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

签名预处理：计算消息摘要

无论是RSA还是ECC的签名运算，都是针对需要签名的消息首先计算摘要，随后对摘要进行签名运算。

> 说明 获取摘要的算法和调用KMS计算签名的算法需要一致。例如：签名算法ECDSA_SHA_256需要结合SHA-256摘要算法使用，如果使用SHA-384算法计算摘要，则和ECDSA_SHA_256算法不匹配。

示例中均使用SHA-256摘要算法。

1. 把需要签名的消息“this is message”存入文件message-file.txt：

```
$ echo "this is message" > message-file.txt
```

2. 计算消息的SHA-256摘要，二进制摘要存入文件message-sha256.bin：

```
$ openssl dgst -sha256 -binary -out message-sha256.bin  message-file.txt
```

调用KMS计算签名

您需要调用KMS接口使用私钥计算消息的签名。

1. 在对消息摘要进行网络传输之前，首先需要对其进行Base64编码：

```
$ openssl base64 -in message-sha256.bin
```

得到Base64编码后的摘要如下：

```
hRP2cuRFSlfEoUXCGuPyi7kZr18VCTZeVOTw0jbUB6w=
```

2. 随后可以将Base64编码后的摘要传入KMS，产生签名。

> 说明 这一步针对不同的密钥和签名算法，调用KMS时传入的参数以及生成的结果均不相同。示例中产生的每种签名结果，被分别存入了不同的文件中。

- RSASSA-PSS

  RSA密钥可以使用RSASSA-PSS算法结合SHA-256摘要进行签名，使用下列命令：

  ```
  $ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
    --Algorithm=RSA_PSS_SHA_256 --Digest=hRP2cu...
  {
      "KeyId": "*****",
      "KeyVersionId": "*****",
      "Value": "J7xmdnZ...",
      "RequestId": "70f78da9-c1b6-4119-9635-0ce4427cd424"
  }
  ```

  对签名结果Value进行Base64解码并输出二进制签名到文件rsa_pss_signature.bin：

  ```
  $ echo J7xmdnZ... | openssl base64 -d -out rsa_pss_signature.bin
  ```

- RSASSA_PKCS1_V1_5

  RSA密钥可以使用RSASSA_PKCS1_V1_5算法结合SHA-256摘要进行签名，使用下列命令：

  ```
  $ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
    --Algorithm=RSA_PKCS1_SHA_256 --Digest=hRP2cu...
  {
      "KeyId": "*****",
      "KeyVersionId": "*****",
      "Value": "qreBkH/u...",
      "RequestId": "4be57288-f477-4ecd-b7be-ad8688390fbc"
  }
  ```

  对签名结果Value进行Base64解码并输出二进制签名到文件rsa_pkcs1_signature.bin：

  ```
  echo qreBkH/u... | openssl base64 -d -out rsa_pkcs1_signature.bin
  ```

- NIST P-256

  NIST曲线P-256可以使用ECDSA算法结合SHA-256摘要进行签名，使用下列命令：

```
$ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
   --Algorithm=ECDSA_SHA_256 --Digest=hRP2cu...
{
      "KeyId": "****",
      "KeyVersionId": "****",
      "Value": "MEYCIQD33Y98...",
      "RequestId": "472d789c-d4be-4271-96bb-367f7f0f8ec3"
}
```

对签名结果Value进行Base64解码并输出二进制签名到文件ec_p256_signature.bin：

```
echo MEYCIQD33Y98... | openssl base64 -d -out ec_p256_signature.bin
```

- **secp256k1**

  SECG曲线secp256k1可以使用ECDSA算法结合SHA-256摘要进行签名，使用下列命令：

```
$ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
   --Algorithm=ECDSA_SHA_256 --Digest=hRP2cu...
{
      "KeyId": "****",
      "KeyVersionId": "****",
      "Value": "MEYCIQDWuuI...",
      "RequestId": "fe41abed-91e7-4069-9f6b-0048f5bf4de5"
}
```

对签名结果Value进行Base64解码并输出二进制签名到文件ec_p256k_signature.bin：

```
echo MEYCIQDWuuI... | openssl base64 -d -out ec_p256k_signature.bin
```

获取公钥

参考获取公钥，从KMS得到相应的非对称密钥的公钥。对应于上述示例，我们做如下假定：

- 将RSA密钥的公钥存入：rsa_publickey.pub
- 将NIST P-256密钥的公钥存入：ec_p256_publickey.pub
- 将secp256k1密钥的公钥存入：ec_p256k_publickey.pub

使用公钥验证签名

根据不同的密钥使用不同类型算法，分别使用如下的命令行进行签名的验证。

- **RSASSA-PSS**

```
$ openssl dgst \
    -verify rsa_publickey.pub \
    -sha256 \
    -sigopt rsa_padding_mode:pss \
    -sigopt rsa_pss_saltlen:-1 \
    -signature rsa_pss_signature.bin \
    message-file.txt
```

- **RSASSA_PKCS1_V1_5**

```
$ openssl dgst \
    -verify rsa_publickey.pub \
    -sha256 \
    -signature rsa_pkcs1_signature.bin \
    message-file.txt
```

- **NIST P-256**

```
$ openssl dgst \
    -verify ec_p256_publickey.pub \
    -sha256 \
    -signature ec_p256_signature.bin \
    message-file.txt
```

- **secp256k1**

```
$ openssl dgst \
    -verify ec_p256k_publickey.pub \
    -sha256 \
    -signature ec_p256k_signature.bin \
    message-file.txt
```

如果验证成功，您应当看到如下输出：

```
Verified OK
```

8.托管密码机

8.1. 托管密码机简介

托管密码机是密钥管理服务KMS（Key Management Service）提供的一项重要功能，助您在阿里云上轻松使用具有合规资质的硬件密码机。

硬件密码机是一种执行密码运算、安全生成和存储密钥的硬件设备。通过将密钥托管在这些高安全等级的硬件设备中，可以保护您在阿里云上最敏感的计算任务和资产。

> 说明 硬件密码机也叫硬件安全模块（Hardware Security Module，简称HSM）。

支持的地域

您可以在下列地域使用托管密码机，同时阿里云计划在更多地域逐步推出该功能。

地域名称	所在城市	密码机检测类型	地域标识符
华北2	北京	国密局商用密码检测认证	cn-beijing
华北3	张家口	国密局商用密码检测认证	cn-zhangjiakou
华东1	杭州	国密局商用密码检测认证	cn-hangzhou
华东2	上海	国密局商用密码检测认证	cn-shanghai
华南1	深圳	国密局商用密码检测认证	cn-shenzhen
中国香港	香港	FIPS 140-2 第三级	cn-hongkong
亚太东南1	新加坡	FIPS 140-2 第三级	ap-southeast-1
亚太东南2	悉尼	FIPS 140-2 第三级	ap-southeast-2
亚太东南3	吉隆坡	FIPS 140-2 第三级	ap-southeast-3
亚太东南5	雅加达	FIPS 140-2 第三级	ap-southeast-5
美国东部1	弗吉尼亚	FIPS 140-2 第三级	us-east-1

除以上公共云地域，KMS还在以下行业云地域支持您使用托管密码机。

地域名称	所在城市	密码机检测类型	地域标识符
华北2阿里政务云 [1]	北京	国密局商用密码检测认证	cn-north-2-gov-1
华东1（杭州金融云） [2]	杭州	国密局商用密码检测认证	cn-hangzhou-finance
华东2（上海金融云） [2]	上海	国密局商用密码检测认证	cn-shanghai-finance-1
华南1（深圳金融云） [2]	深圳	国密局商用密码检测认证	cn-shenzhen-finance-1

> 说明
>
> - 华北2阿里政务云仅供政务云客户使用。详情请参见上云须知。
> - 华东1（杭州金融云）、华东2（上海金融云）和华南1（深圳金融云）仅供金融云客户使用。详情请参见上云须知。

合规

托管密码机帮助您满足严格的合规要求。根据各地区监管机构要求，阿里云提供的多种密码机分别由不同的第三方机构认证，从而适应不同市场的地区性差异，满足您的本地化和国际化需求。

对中国内地的地域：

- 国密检测认证：阿里云运营的密码机，已通过国家密码管理局指定检测机构的检测认证。
- 国密合规：阿里云的托管密码机符合国家密码管理局相关技术要求和规范，为阿里云用户提供符合国家和行业标准的商用密码算法。

对中国内地之外的地域：

- 硬件的FIPS认证：阿里云运营的密码机，包含它们的硬件和固件，已获得FIPS 140-2第三级认证。NIST颁发的证书详情，请参见证书3254。
- FIPS 140-2第三级合规：阿里云的托管密码机运行在FIPS许可的第三级模式下。
- PCI-DSS合规：阿里云的托管密码机符合PCI-DSS合规的要求。

高安全保证

- 硬件保护

  托管密码机通过安全的硬件机制来保护KMS中的密钥。用户主密钥的明文密钥材料只会在密码机的内部被处理，用于密码运算，而不会离开密码机硬件的安全边界。

- 安全的密钥生成

  随机性是密钥强度的关键。通过使用托管密码机，密钥材料的产生基于安全、许可、且以高系统熵值为种子的随机数生成算法，从而保护密钥不被攻击者恢复或者预判。

易运维

阿里云提供密码机硬件的完全托管，免去您自己管理硬件所带来的如下运维开销：

- 硬件生命周期的管理
- 密码机集群管理
- 高可用和可伸缩性管理
- 系统修补（Patching）
- 大部分灾备工作

易集成

通过原生的密钥管理能力，您可以从以下功能中受益：

- 密钥版本管理
- 自动密钥轮转
- 资源标签管理
- 可控制的授权机制

这些功能支持您的应用与托管密码机快速集成，也支持云服务器ECS、关系型数据库RDS等其他云服务与托管密码机集成，实现云上数据静态加密，而您无需为此投入研发成本。

保持对密钥的控制

借助托管密码机，您可以更好地控制云上的加密密钥，将最具敏感性的计算任务和资产移动到云端。

同时使用托管密码机和BYOK（Bring Your Own Key），您可以实现以下功能：

- 完全控制密钥材料的生成方式
- 导入到托管HSM的密钥材料只能被销毁而无法被导出
- 完全控制密钥的生命周期
- 完全控制密钥的持久性

低成本

相比于通过本地密码机自建密钥基础设施，托管密码机采用云计算“用多少花多少”的计费模式，帮助您免去硬件采购的初始成本，以及后续研发和运维带来的持续性投入。

8.2. 使用托管密码机

本文为您介绍如何使用托管密码机创建并使用密钥。

前提条件

进行操作前，请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

背景信息

您需要在当前已经支持的地域使用托管密码机。支持的地域详情，请参见支持的地域。

通过密钥管理服务控制台创建密钥

1. 登录密钥管理服务控制台。
2. 在**密钥列表**中，单击**创建密钥**。
3. 在弹出的**创建密钥**对话框，填写**别名**。
4. 从**保护级别**列表中，选择**HSM**。
5. 填写描述后，单击**确认**。
   创建完成后，在**密钥详情**和**密钥列表**页均可以查看到密钥的**保护级别**。

通过阿里云CLI创建密钥

1. 在阿里云CLI中，输入以下命令。

   ```
   aliyun kms CreateKey --ProtectionLevel HSM --Description "Key1 in Managed HSM"
   ```

2. 调用DescribeKey接口，查看密钥的保护级别。

```
{
 "KeyMetadata": {
  "CreationDate": "2019-07-04T13:14:15Z",
  "Description": "Key1 in Managed HSM",
  "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
  "KeyState": "Enabled",
  "KeyUsage": "ENCRYPT/DECRYPT",
  "DeleteDate": "",
  "Creator": "111122223333",
  "Arn": "acs:kms:cn-hongkong:111122223333:key/1234abcd-12ab-34cd-56ef-12345678****",
  "Origin": "Aliyun_KMS",
  "MaterialExpireTime": "",
  "ProtectionLevel": "HSM"
 },
 "RequestId": "8eaeaa8b-4491-4f1e-a51e-f95a4e54620c"
}
```


将外部密钥导入托管密码机

如果您需要将自建密钥基础设施中的密钥导入到托管密码机中，您只需要在创建外部密钥时，指定**保护级别为HSM**。创建外部密钥操作详情，请参见导入密钥材料（控制台）。

将外部密钥导入托管密码机时，阿里云会进行如下操作：

- 当您调用GetParametersForImport接口时：阿里云将根据您指定的保护级别**HSM**，在托管密码机中生成一个用于导入外部密钥的密钥对，并把密钥对的公钥返回给您。
- 当您调用ImportKeyMaterial接口时：阿里云将加密的外部密钥材料导入到托管密码机的内部，并通过HSM的密钥反打包（Unwrap）机制获取密钥材料本身，而导入的密钥材料明文不能被任何人导出。

管理和使用密钥

密钥管理服务支持的所有管理类功能和密码运算功能都适用于您在托管密码机中创建的密钥，具体功能如下：

- 密钥状态的开启和禁用
- 密钥的生命周期管理
- 密钥的别名管理
- 密钥的云标签管理
- 密码运算接口的调用

和其他云产品的集成

托管密码机中的密钥，可以通过密钥管理服务的标准接口和ECS、RDS、OSS等其它云产品实现无缝集成，用于对阿里云上的原生数据进行保护。云产品需要支持对用户自选密钥进行服务端加密的能力。您只需在云产品中配置用于服务端加密的CMK，选择一个创建在托管密码机中的密钥即可。

9.密钥的轮转

9.1. 密钥轮转概述

密钥常用于保护特定的数据，因此，数据的安全依赖于密钥的安全。您可以通过密钥版本化和定期轮转来加强密钥使用的安全性，实现数据保护的安全策略和最佳实践。

实现安全目标

您可以通过周期性轮转密钥，达成以下安全目标：

- 减少每个密钥加密的数据量

  一个密钥的安全性与被它加密的数据量呈反相关。数据量通常是指同一个密钥加密的数据总字节数或总消息数。例如，NIST将GCM模式下一个密钥的安全生命周期定义为基于其加密的总消息数。通过定期轮转密钥而改变加密密钥的方式，可以使得每个密钥具有更高的安全阈值和更小的密码分析攻击面。

- 提前具备响应安全事件的能力

  在系统设计的早期，引入密钥轮转的功能并将其作为日常运维手段。这样可以使系统在特定安全事件发生时具备实际执行能力，符合软件工程中Fail Early、Fail Often的原则。如果第一次执行（突发性）密钥轮转是在响应具体事件的情形下，并且发生在运行中的系统上，则发生故障的概率会被无限放大。

- 对数据形成逻辑上的隔离

  轮转加密密钥使得轮转前后产生的密文数据形成事实上的隔离效果。特定密钥的安全事件可以被快速定义影响范围，从而采取进一步措施。

- 减小破解密钥的时间窗口

  如果在定期轮转加密密钥的基础上，将旧的加密密钥产生的密文数据用新的加密密钥轮转加密，则轮转周期即为一个密钥的破解时间窗口。这意味着恶意者只有在两次轮转事件之间完成破解，才能拿到数据。这对于保护数据不受密码分析（Cryptanalysis）攻击具有很强的实践意义。

满足合规规范

密钥的周期性轮转功能可以方便企业符合各种合规规范。与密钥轮转相关的合规规范包含（但不限于）：

- 支付卡行业数据安全标准（PCI DSS）
- 中国国家密码管理局发布的密码行业相关标准，例如GM/T 0051-2016
- 美国国家标准技术委员会（NIST）发布的密码使用相关标准，例如NIST Publication 800-38D

9.2. 自动轮转密钥

本文为您介绍如何对密钥管理服务（KMS）中的用户主密钥（CMK）进行自动轮转。

密钥版本

KMS中的CMK支持多个密钥版本。每一个密钥版本是一个独立生成的密钥，同一个CMK下的多个密钥版本在密码学上互不相关。KMS通过生成一个新的密钥版本来实现密钥的自动轮转。

密钥版本的类别

CMK的密钥版本分为以下2种：

- 主版本（Primary Key Version）

- 主版本是CMK的活跃加密密钥（Active Encryption Key）。每个CMK在任何时间点上有且仅有一个主版本。
- 调用GenerateDataKey、Encrypt等加密API时，KMS使用指定CMK的主版本对明文进行加密。
- 您可以通过DescribeKey返回的响应查看PrimaryKeyVersion属性。

- 非主版本（Non-primary Key Version）
  - 非主版本是CMK的非活跃加密密钥（Inactive Encryption Key）。每个CMK可以有零到多个非主版本。
  - 非主版本历史上曾经是主版本，在当时被用作活跃加密密钥。
  - 密钥轮转产生新的主版本后，KMS不会删除或禁用非主版本，它们需要被用作解密数据。

> 说明 加密API使用指定CMK的主版本，解密API使用传入密文对应的加密密钥版本。

密钥版本的生成

您可以通过以下两种方式生成密钥版本：

- 创建CMK

  通过调用CreateKey创建CMK。如果指定参数Origin为*Aliyun_KMS*，则KMS会生成初始密钥版本，并且将其设置为主版本。

- KMS执行自动轮转策略

  CMK配置自动轮转策略，配置的策略由KMS定期执行，周期性产生新的密钥版本。

自动轮转

配置和查询轮转策略

当您调用CreateKey创建CMK时，可以指定CMK的自动轮转策略，也可以调用UpdateRotationPolicy变更当前的自动轮转策略。调用时，需配置以下参数：

- EnableAutomaticRotation：是否开启自动轮转。
- RotationInterval：自动轮转的周期。

您可以通过调用DescribeKey查询所配置的轮转策略，返回的相关参数为：

- AutomaticRotation：自动轮转的开启状态。详情请参见主密钥状态对轮转的影响。
  - *Disabled*：用户未开启自动轮转。
  - *Enabled*：用户已开启自动轮转。
  - *Suspended*：用户开启了自动轮转，但是KMS暂停执行。
- RotationInterval：自动轮转的周期。

KMS执行策略

如果您指定开启自动轮转，则KMS会按照以下方式计算下次轮转的时间。

```
${NextRotationTime} = ${LastRotationTime} + ${RotationInterval}
```

参数含义如下：

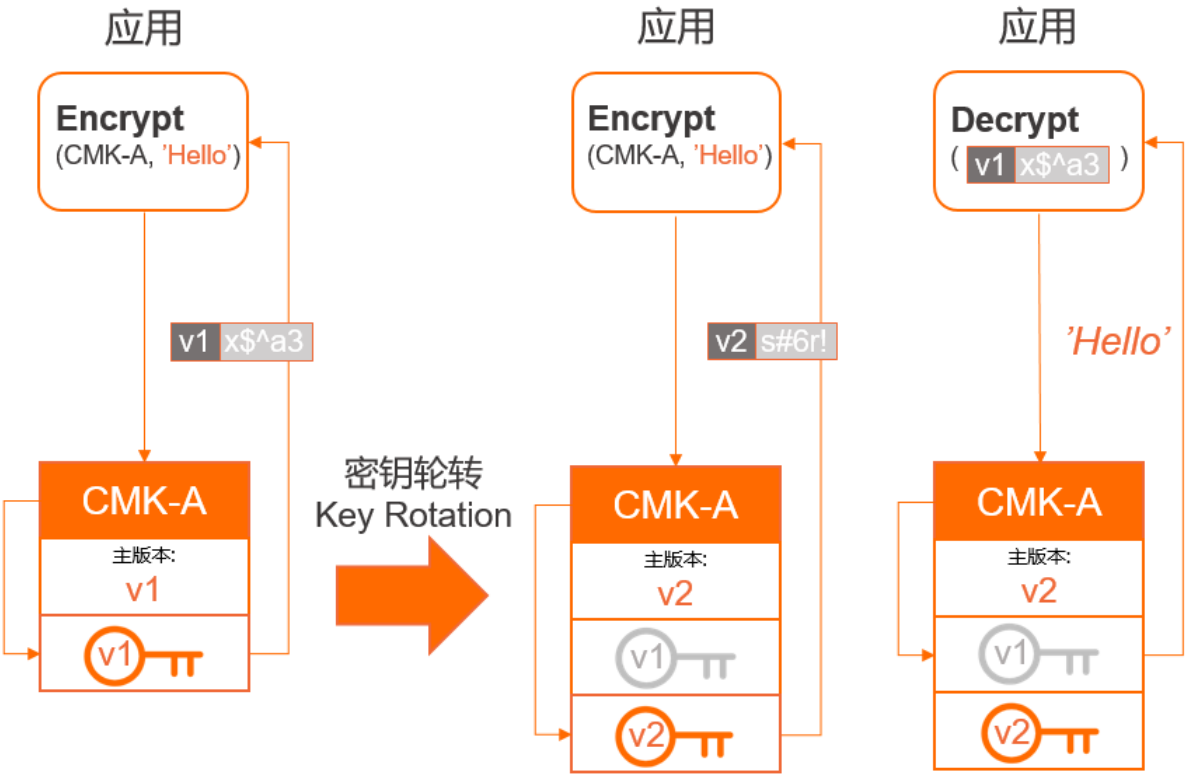
- `LastRotationTime`：生成上一个密钥版本的时间。您可以通过DescribeKey返回的`LastRotationDate`字段查询此参数值。

- `NextRotationTime`：KMS会在计算出来的下一次轮转时间到来之后，执行轮转任务，创建新的密钥版本。您可以通过DescribeKey返回的`NextRotationDate`字段查询此参数值。

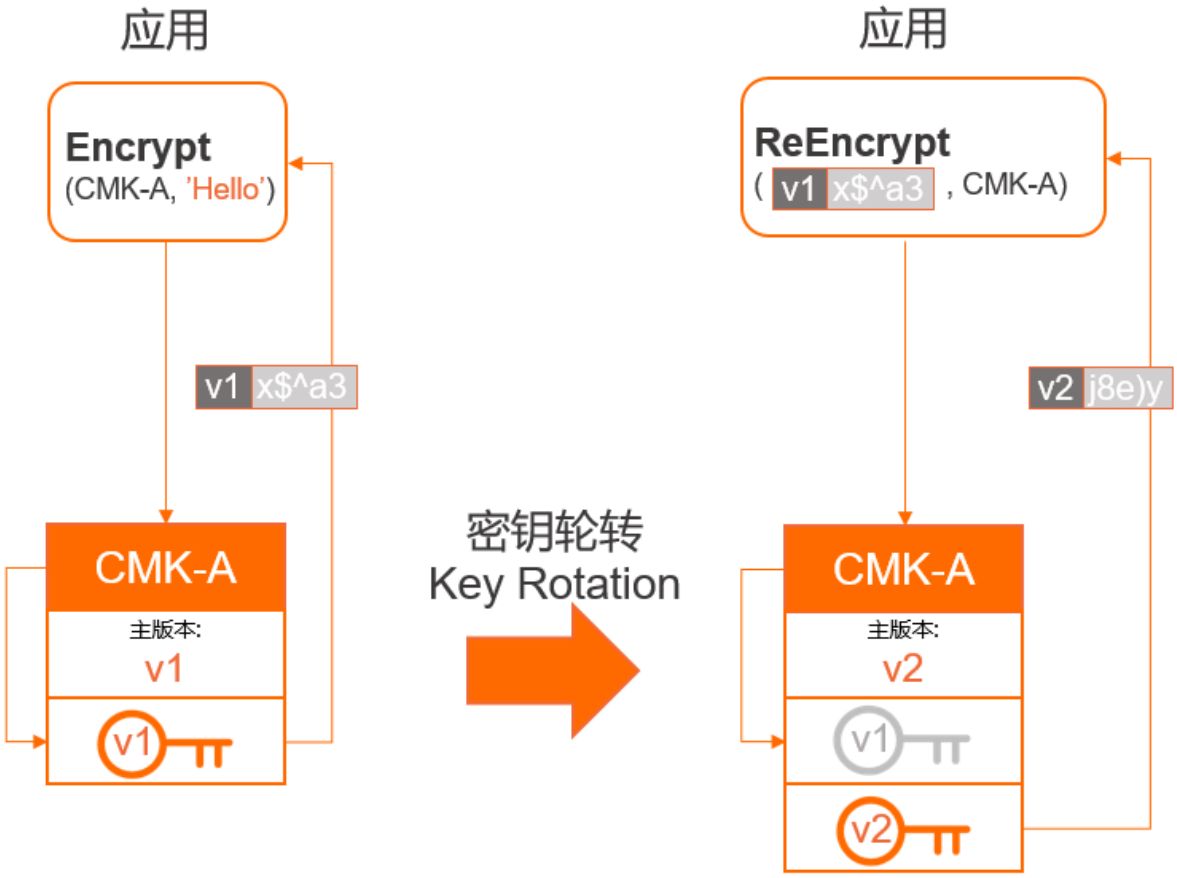
> 注意 更新轮转策略所指定的轮转周期，可能会导致KMS计算出来的下次轮转时间成为过去的某个时间点。这并不影响KMS执行轮转策略的规则，即在下一次轮转时间到来之后生成新的密钥版本。如果更新后的策略指向的下一次轮转时间已经在当前时间之前，则立即满足了KMS触发密钥轮转的标准。

轮转对加密解密的影响

对于密钥轮转后的加密请求，KMS会使用当前版本的密钥进行加密。对于历史版本密钥加密后密文的解密请求，您可以使用历史版本密钥进行解密。



对于历史版本密钥加密后密文，您可以通过ReEncrypt将数据转为换当前版本密钥进行加密。



主密钥状态对轮转的影响

只有在启用状态的主密钥（KeyState为*Enabled*）才能产生新的密钥版本。您需要注意以下情形：

- 如果一个主密钥处于*Disabled*或者*PendingDeletion*状态，请勿调用UpdateRotationPolicy变更主密钥的轮转策略。
- 如果一个主密钥在用户开启自动轮转后，进入到*Disabled*或者*PendingDeletion*状态，则KMS暂停执行自动轮转。此时通过DescribeKey查看到的轮转状态（AutomaticRotation属性）为*Suspended*。当主密钥重新回到启用状态时，之前配置的轮转策略会重新生效。

不适用范围

KMS管理的以下类型的密钥不支持多个版本：

- 云产品托管密钥（Service Managed Key）：特定云产品托管在KMS上的、用于加密保护您的数据的默认密钥。这类密钥由特定云产品为用户代为管理，为您的数据提供最基本的加密保护。
- 用户自带密钥（Customer Supplied Key）：您导入到KMS中的密钥。这类CMK的Origin属性为*External*，KMS不负责为用户生成密钥材料，无法自动发起轮转行为。导入密钥材料详情，请参见导入密钥材料。

因此，以上两种类型的密钥不支持以自动或人工的方式进行基于版本的密钥轮转。此外，KMS也不支持自带密钥的多版本功能，原因如下：

- 自带密钥的持久性和生命周期由用户强管控，本身就具有较高的管理难度和易错风险（例如：您需要有云下的密钥管理设施，云上云下信息需要同步，云上删除密钥材料没有任何缓冲期），而多版本带来的复杂度升级会超线性地升高易错性，从而带来数据风险。

- 每个密钥版本，包括用于加密和解密的主版本以及仅用于解密的非主版本，都可能在不同的时间点上不可用（例如：在不同时间点上过期而被KMS删除，或者过期后被重新分别导入），导致无法同步主密钥和被保护数据的可用性，很难保证系统设计的完整性。

> **说明** 对于上述不支持基于版本进行密钥轮转的场景，替代方案请参见人工轮转主密钥。

9.3. 人工轮转主密钥

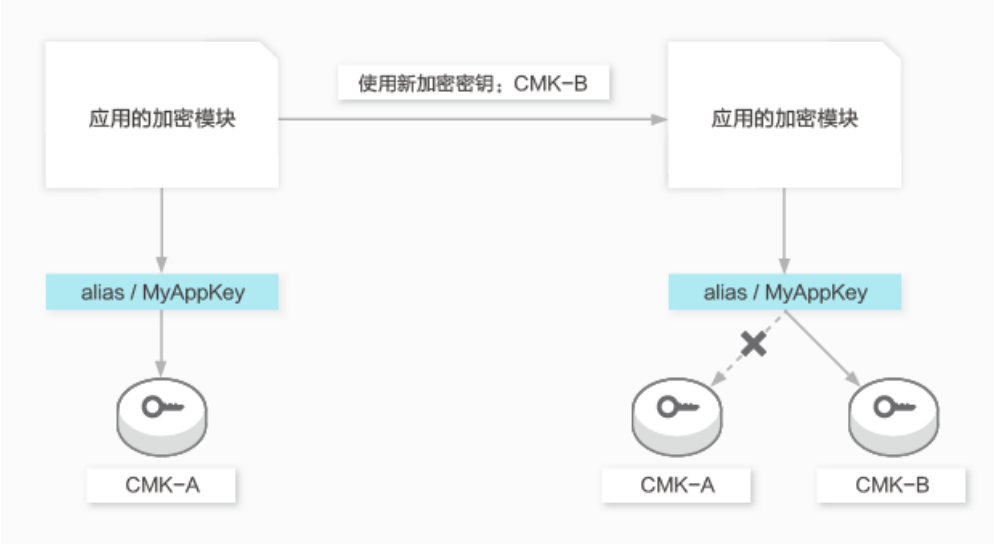
如果您使用的密钥类型不支持基于密钥版本的自动轮转，您可以基于使用场景，直接轮转使用的用户主密钥（CMK），通过人工的方式实现密钥轮转。由于这种方式建立在CMK之上，对于支持和不支持自动轮转的CMK而言，都可以作为特殊场景下的替代技术方案。

应用自定义加密场景

云上或者云下的应用可以调用KMS API实现自定义的数据加密，例如：

- 在将身份证号、信用卡号、家庭地址等敏感信息写入数据库之前进行加密
- 在上传数据到OSS之前，进行客户端加密
- 对包含敏感信息的应用配置文件、SSL私钥证书等服务配置文件进行加密

您可以通过KMS提供的密钥别名功能，实现对应用中加密密钥的轮转。KMS的解密API不需要您提供密钥的ID或者别名。



这一抽象的轮转场景包括以下步骤：

1. 初始设置
   i. 管理员创建CMK，假定ID为CMK-A。
   ii. 管理员为上述CMK-A绑定别名alias/MyAppKey。
   iii. 应用加密模块调用Encrypt接口时，在KeyId参数传入别名*alias/MyAppKey*：KMS发现alias/MyAppKey绑定了ID为CMK-A的主密钥，因此使用CMK-A执行加密。
   iv. 应用解密模块调用Decrypt接口时，不传入KeyId参数，KMS使用实际用于加密数据的CMK执行解密。
2. 人工轮转密钥
   i. 管理员创建新的CMK，假定ID为CMK-B。

ii. 管理员调用UpdateAlias，将别名alias/MyAppKey绑定到CMK-B。

iii. 应用加密模块调用Encrypt接口时，在KeyId参数传入别名*alias/MyAppKey*：KMS发现alias/MyAppKey绑定了ID为CMK-B的主密钥，因此使用CMK-B执行加密。

iv. 应用解密模块调用Decrypt接口时，不传入KeyId参数，KMS使用实际用于加密数据的CMK执行解密。

云产品服务端加密场景

云产品通过集成KMS API，可以对其管理的数据进行服务端加密。云产品服务端加密的场景，从密钥轮转角度可能出现以下情形：

- KMS侧配置自动轮转策略对云产品服务端加密产生效果

  这种情形通常是因为云产品配置指定CMK加密之后，会持续性调用KMS的GenerateDataKey产生新的数据密钥，因此KMS产生新的主版本后，云产品会使用到新的主版本来加密新产生的数据密钥。这类云产品包括OSS等。对于这类情况，如果您希望具备自动轮转密钥的能力，则不能使用服务托管的默认密钥，也不能使用自带密钥（您导入到KMS的外部密钥），因为它们并不支持密钥自动轮转。

- KMS侧配置自动轮转策略对云产品服务端加密不产生效果

  这种情形通常是因为云产品配置指定CMK加密之后，只会对特定资源调用一次KMS的加密API，因此KMS即便轮转CMK产生新的密钥版本，云产品也无法使用。例如ECS对云盘的加密，只会一次性调用KMS的GenerateDataKey产生数据密钥用于卷加密，后续不会再对整个磁盘的卷加密密钥（Volume Encryption Key）进行更新。

如果您在第一种情形中使用了自带密钥或者希望对数据密钥也进行变更轮转，或者应用场景属于第二种情形，则可以通过变更配置、拷贝数据等方式达成密钥轮转的目的。这类解决方案依赖于云产品的特定功能，详情请参见相关云产品文档。

10.云产品与KMS的集成

10.1. 服务端集成加密概述

阿里云为您提供云产品落盘存储加密服务，并统一使用密钥管理服务（KMS）进行密钥管理。阿里云的存储加密功能提供了256位密钥的存储加密强度（AES256），满足敏感数据的加密存储需求。

云产品和KMS在服务端的集成为您带来了如下收益：

- 增加云上数据的安全和隐私

  通过使用您在KMS中管理的密钥，云产品可以加密任何归属于您的数据。这些数据既可以是您可以直接访问的数据，也可以是您无法直接访问的云产品内部数据（例如数据库引擎产生的文件），从而为您提供对云上数据更大的控制权，保证数据的安全性和隐私性。

- 降低自研数据加密带来的研发成本

  自研数据加密往往会带来如下研发成本：

  - 需要考虑合理的密钥层次结构和对应的数据划分方式，以平衡加密性能和安全性
  - 需要考虑密钥轮转、数据重加密
  - 需要掌握密码学技术，保证加密算法的健壮性、安全性以及防篡改能力等
  - 需要考虑自研系统的工程健壮性和可靠性，以确保数据持久性

  云服务端集成加密为您考虑和解决了所有这些复杂的工程和安全问题，降低了研发成本。

选择合适的密钥

您可以根据数据保护需求，选择托管在KMS中的不同类型密钥用于服务端加密：

- 服务托管的密钥

  如果您介意密钥管理带来的开销，云产品可以为您在KMS中托管一个用于服务端加密的默认密钥，称为服务密钥。服务密钥由对应云产品管理，您对云产品使用此服务密钥的授权是隐式的。通过操作审计服务，您可以对云产品使用服务密钥的情况进行审计。

  为了方便用户识别，服务密钥的Creator属性为对应云产品的代码，同时服务密钥被关联了特殊的别名，格式为 `acs/<云产品代码>` 。例如，对象存储服务OSS为用户创建的服务密钥，Creator属性为OSS，同时被关联了别名acs/oss。

- 用户托管的密钥

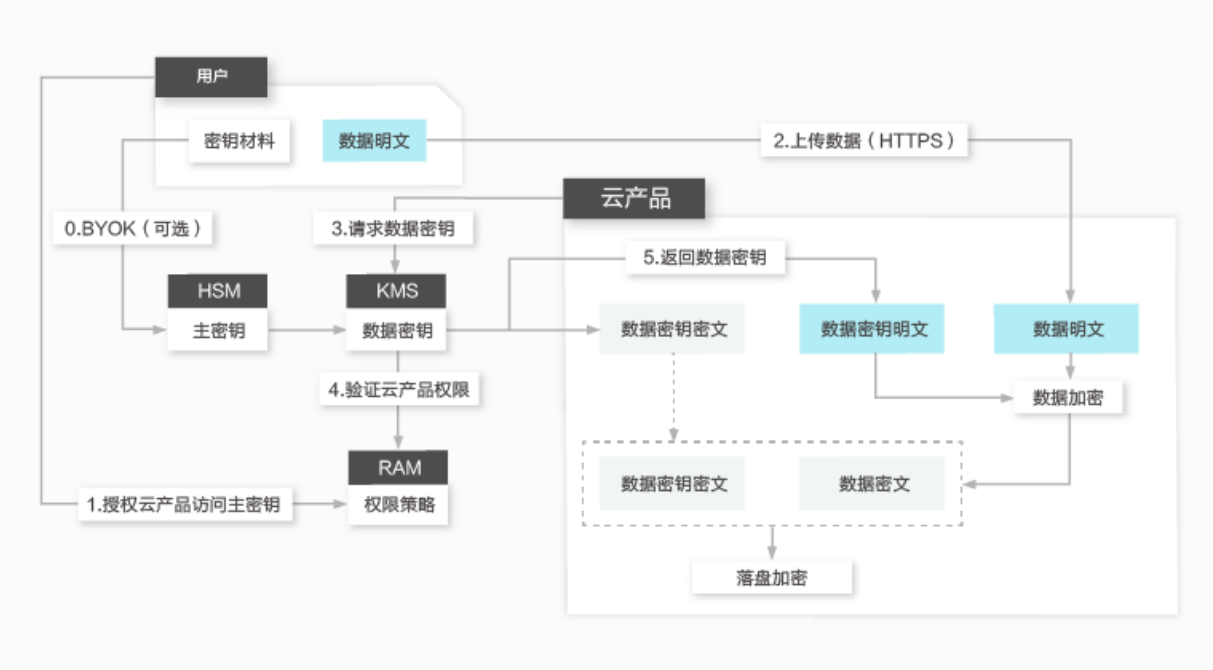
  您可以选择自己托管的密钥进行云产品服务端加密，从而获取对数据加密行为更大的控制权。由于云产品并不是密钥托管者，对密钥的使用需要获得您的显式授权。RAM服务是云产品和KMS之间的权限鉴别仲裁者，您通过RAM服务配置权限策略，允许或者拒绝云产品使用特定的主密钥，云产品访问KMS时，KMS向RAM服务验证云产品是否具备使用特定主密钥的权限。

  您除了可以让KMS生成密钥，还可以通过导入自带密钥（BYOK）的方式，将线下的密钥材料安全的导入KMS的主密钥，以获得对密钥的更大控制权。例如，您并不可以立即删除KMS生成的密钥材料，但是可以立即删除导入到KMS中的密钥材料。使用自带密钥将产生额外的管理成本，需谨慎使用，详情请参见导入密钥材料。

云产品加密的方式

不同产品基于业务形态和客户需求，其加密的具体设计略有不同。通常，存储加密中密钥层次结构会至少分为两层，并通过信封加密的机制实现对数据的加密。

第一层为KMS中的用户主密钥（CMK），第二层为数据密钥（DK）。CMK对DK进行加解密操作和保护，DK对业务数据进行加解密操作和保护。在业务数据落盘存储时，云产品会将DK的密文（由KMS使用CMK加密）与业务数据的密文（由云产品使用DK加密）一同写入持久化存储介质中。信封加密中的“信封”是指在概念上DK的密文和业务数据的密文被打包在一个“信封”（Envelope）中。在读取加密数据时，DK的密文也会一同被读取，并先于业务数据进行解密。只有在DK被解密后，密文的业务数据才能够被正常解密读取。



在信封加密机制中，CMK受KMS提供的密钥管理基础设施的保护。云产品必须通过恰当的用户授权，才可以使用对应的主密钥来产生DK，用于业务数据的加密，也只有通过用户授权，才可以使用对应的主密钥来解密DK的密文，用于业务数据的解密。DK的明文仅会在您使用的云产品服务实例所在的宿主机的内存中使用，不会以明文形式存储在永久介质上。

10.2. 支持服务端集成加密的云服务

本文介绍了当前支持服务端集成加密的云服务。云服务通过使用服务托管密钥或者用户自选密钥（包括BYOK-自带密钥）对数据进行服务端加密保护。

服务端集成加密功能可以低成本实现云上数据的加密保护，为业务数据提供安全边界，提升阿里云对您业务安全的保障能力。这不仅适用于您可直接获取的业务数据，也适用于您只能间接访问的业务数据。

服务名称	描述	相关文档

服务名称	描述	相关文档
云服务器ECS	ECS云盘加密功能默认使用服务密钥为用户数据进行加密，也支持使用用户自选密钥为用户的数据进行加密。云盘的加密机制中，每一块云盘（Disk）会有相对应的用户主密钥（CMK）和数据密钥（DK），并通过信封加密机制对用户数据进行加密。 使用ECS云盘加密功能，系统会将从ECS实例传输到云盘的数据自动进行加密，并在读取数据时自动解密。加密解密操作在ECS实例所在的宿主机上进行。在加密解密的过程中，云盘的性能几乎没有衰减。 在创建加密云盘并将其挂载到ECS实例后，系统将对以下数据进行加密： • 云盘中的静态数据 • 云盘和实例间传输的数据（实例操作系统内数据不加密） • 从加密云盘创建的所有快照（即加密快照）	加密概述
对象存储OSS	OSS支持在服务器端对上传的数据进行加密（Server-Side Encryption）： • 上传数据时，OSS对收到的用户数据进行加密，然后将得到的加密数据持久化保存。 • 下载数据时，OSS自动对保存的加密数据进行解密并把原始数据返回给用户。在返回的HTTP请求Header中，声明该数据进行了服务器端加密。 OSS在支持集成KMS之前就支持了SSE-OSS，即：使用OSS私有密钥体系进行服务端加密。这种方式并不使用归属用户的密钥，因此用户无法通过操作审计服务审计密钥使用情况。 OSS支持集成KMS进行服务端加密，称之为SSE-KMS。OSS支持使用服务密钥和用户自选密钥两种方式进行服务端加密。OSS既支持在桶级别配置默认加密CMK，也支持在上传每个对象时使用特定的CMK。	• 服务器端加密 • SDK参考
容器服务Kubernetes版ACK	在Kubernetes集群中，我们通常使用Secrets密钥模型存储和管理业务应用涉及的敏感信息，例如应用密码、TLS证书、Docker镜像下载凭据等敏感信息。Kubernetes会将所有的这些Secrets密钥对象数据存储在集群对应的ETCD中。 在ACK Pro托管集群中，您可以使用在KMS中创建的密钥加密Kubernetes Secret密钥。	使用阿里云KMS进行Secret的落盘加密

服务名称	描述	相关文档
关系型数据库RDS	RDS数据加密提供以下两种方式： • 云盘加密 针对RDS云盘版实例，阿里云免费提供云盘加密功能，基于块存储对整个数据盘进行加密。云盘加密使用的密钥由KMS服务加密保护，RDS只在启动实例和迁移实例时，动态读取一次密钥。 • 透明数据加密TDE RDS提供MySQL和SQL Server的透明数据加密TDE（Transparent Data Encryption）功能。TDE加密使用的密钥由KMS服务加密保护，RDS只在启动实例和迁移实例时动态读取一次密钥。当RDS实例开启TDE功能后，用户可以指定参与加密的数据库或者表。这些数据库或者表中的数据在写入到任何设备（磁盘、SSD、PCIe卡）或者服务（对象存储OSS）前都会进行加密，因此实例对应的数据文件和备份都是以密文形式存在的。	• MySQL：云盘加密、设置透明数据加密TDE • SQL Server：云盘加密、设置透明数据加密TDE • Postgre SQL：设置数据加密
云数据库MongoDB	MongoDB加密和RDS类似。	设置透明数据加密TDE
云数据库PolarDB	PolarDB加密和RDS类似。	设置透明数据加密TDE
文件存储NAS	NAS的加密功能默认使用服务密钥为用户的数据进行加密。NAS的加密机制中，每一卷（Volume）会有相对应的用户主密钥（CMK）和数据密钥（DK），并通过信封加密机制对用户数据进行加密。	数据加密
表格存储Tablestore	表格存储的加密功能默认使用服务密钥为用户的数据进行加密，同时也支持使用用户自选密钥为用户的数据进行加密。表格存储的加密机制中，每一个表格（Table）会有相对应的用户主密钥（CMK）和数据密钥（DK），并通过信封加密机制对用户数据进行加密。	无
大数据计算MaxCompute	MaxCompute支持使用服务密钥作为用户主密钥（CMK）进行数据加密。	数据加密
云存储网关CSG	云存储网关CSG支持两种方式进行加密： • 网关侧加密：文件会在网关侧缓存盘进行加密后上传至OSS。 • 基于OSS对数据进行加密。	• 网关侧加密 • 管理共享
媒体处理MTS	MTS支持私有加密和HLS标准加密两种方式，均可以集成KMS对视频内容进行保护。	加密

服务名称	描述	相关文档
视频点播VOD	VOD支持阿里云视频加密和HLS标准加密两种方式，均可以集成KMS对视频内容进行保护。	• 阿里云视频加密 • HLS标准加密
Web应用托管服务Web+	Web+使用KMS对应用托管服务中使用的敏感配置数据进行加密保护，这包含了类似RDS数据库的访问凭证等机密信息。	云数据库RDS
应用配置管理ACM	ACM通过和KMS集成，对应用配置进行加密，确保敏感配置（数据源、Token、用户名、密码等）的安全性，降低用户配置的泄露风险。ACM服务端和KMS的集成有以下两种方式： • 在KMS服务端直接加密 ACM服务通过KMS的数据加密API，将配置传送到KMS端，指定CMK完成对配置的加密。 • 在ACM服务端信封加密 通过KMS的API，使用指定CMK来保护生成的数据密钥（DK），使用数据密钥（DK）在ACM服务端完成对配置的加密。	创建和使用加密配置

11.凭据管家

11.1. 凭据管家概述

本文为您介绍凭据加密方式、安全分发凭据的实现原理、访问控制和凭据的使用，以及凭据定期轮转方式。

凭据管理是企业IT系统运维安全的核心诉求之一。KMS凭据管家为您提供凭据的创建、检索、更新、删除等全生命周期的管理服务，轻松实现对敏感凭据的统一管理。用户或应用程序可以通过调用凭据管理API，规避账号口令、访问密钥、服务器证书硬编码等风险问题，有效避免敏感信息泄密以及权限失控的业务风险。

加密保护凭据

使用凭据管家，您可以托管各种类型的凭据，包括但不限于访问密钥、API密钥、服务器证书和私钥、账号口令等。凭据管家对您托管的凭据进行加密存储，保证其在服务端的安全性。凭据支持以下两种加密方式：

- 默认系统加密

  如果您不指定特定的KMS主密钥作为凭据的保护机制，凭据管家会在您首次托管凭据时，自动生成一个凭据管家专用的主密钥，用于默认加密保护您阿里云账号内的托管凭据。这个自动生成的默认加密密钥归属于凭据管家，您无需管理、查看、审计其使用，也无需为其付费。

  > **说明** 凭据管家会为每一个阿里云账号分配一个独立的KMS主密钥用作默认加密密钥。

- 自选密钥加密

  您可以指定一个KMS主密钥作为凭据对象的加密密钥，凭据管家会在您存入凭据明文时，生成一个数据密钥用于加密凭据明文，并且将凭据的密文、数据密钥的密文存入持久化存储中。通过这种方式，您可以在KMS管理主密钥的生命周期，在操作审计查看和审计KMS主密钥的调用记录。操作审计详细介绍，请参见什么是操作审计。

> **说明** 自选密钥加密凭据时，您需要注意以下事项：
>
> - 除了凭据的相应权限，在以下情形，您需要额外具备关联的KMS主密钥的使用权限：
>   - 当您调用CreateSecret接口或PutSecretValue接口存入凭据值时，需要具备相应加密主密钥的kms:GenerateDataKey权限，以便凭据管家以调用者的身份调用GenerateDataKey接口产生数据密钥，随后使用数据密钥加密凭据值。
>
>   API接口详情，请参见：
>
>   - CreateSecret
>   - PutSecretValue
>   - GenerateDataKey
>   - 当您调用GetSecretValue接口读取凭据值时，需要具备相应加密主密钥的kms:Decrypt权限，以便凭据管家以调用者的身份调用Decrypt接口解密数据密钥，随后使用数据密钥解密凭据值的密文。
>
>   API接口详情，请参见：
>
>   - GetSecretValue
>   - Decrypt
> - 凭据的加密主密钥，可以被独立管理，其密钥状态会影响您使用凭据。例如：如果凭据的加密主密钥处于禁用（Disabled）或者待删除（PendingDeletion）状态，则不能对被保护的凭据执行上一条说明中提到的存入凭据值，或者读取凭据值的相应操作。

安全分发凭据

凭据管家可以帮助您安全的分发敏感的凭据。您无需将凭据的明文作为静态信息配置在代码、配置文件或数据库中，只需为应用程序配置凭据管家中的凭据名称，在应用系统需要时动态的获取凭据的明文。实现原理如下：

1. 服务端使用相应的KMS主密钥解密凭据对应的数据密钥密文。
2. 凭据管家使用数据密钥的明文解密凭据的密文。
3. 凭据管家通过HTTPS信道加密的方式返回凭据给客户端。
4. 客户端通过HTTPS信道解密获取凭据的明文。

控制访问和审计使用

企业的信息系统可能有成千上万的凭据需要管理，请您使用凭据管家时，合理的分配每个凭据的使用权限。KMS凭据管家与访问控制和操作审计服务集成，为您提供如下功能：

- 控制对凭据的访问

  方便您自定义细粒度的权限策略，防范对敏感信息的非法访问和泄漏。
- 审计对凭据的使用

  您可以通过持续监控对凭据的访问，还可以将审计日志接入SIEM解决方案等方式进一步提升威胁检测和响应能力。

定期轮转凭据

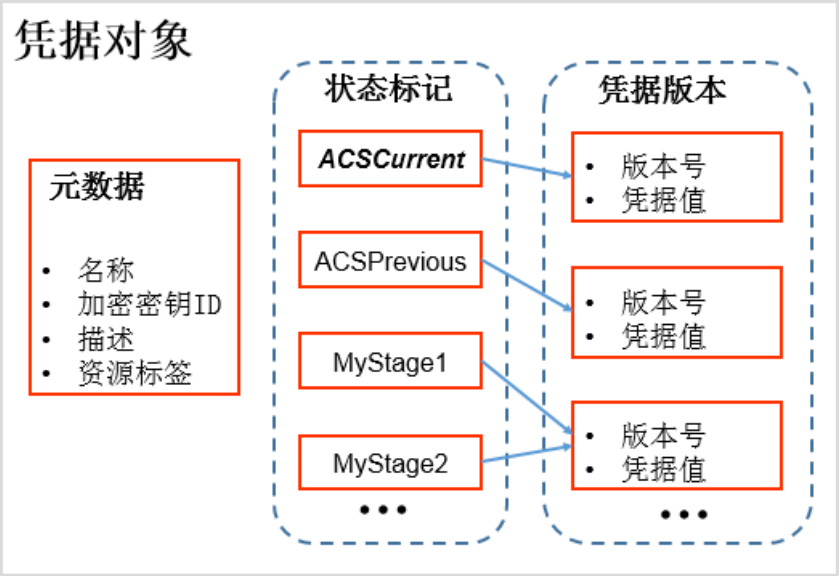
凭据的定期轮转可以极大的降低安全风险。通过KMS凭据管家，凭据的管理者和凭据的使用者（应用系统）采用发布、订阅（pub-sub）的模式对凭据进行管理分发，使得管理者可以从发布侧进行定期轮转，而使用者（应用系统）通过定时重新请求或采用实时请求的方式，保证使用最新的凭据。

KMS凭据管家为定期轮转凭据进行数据模型的优化，支持凭据对象内保存多个版本的凭据值。每个版本通过状态标记（Stage Label）的方式，来辅助自动化轮转。最常见的轮转场景里，通常有两个版本的凭据值：“当前版本”和“先前版本”，凭据管家内建了ACSCurrent和ACSPrevious两个状态的标记以及相应的变换规则，方便您切换“当前版本”。客户端调用GetSecretValue接口时，如果不指定状态标记或版本号，则默认获取被标记为ACSCurrent状态的版本。

11.2. 凭据的对象模型

通过本文您可以了解凭据的元数据、版本和版本状态。

为了更方便的支持对凭据进行轮换，KMS凭据管家对凭据对象内建了支持多版本以及对版本进行状态标记。如果您仅使用凭据托管和保护的功能，而不关心凭据的轮转，则您托管的凭据对象内只有一个版本。凭据管家默认将第一个版本标记为当前版本。凭据对象的完整模型如下图所示。



凭据的元数据

凭据的元数据主要有以下属性：

- 凭据名称

  当您访问KMS凭据管家相关接口时，用来指代凭据。

- 加密密钥标识符

  您托管的凭据包含了敏感信息，凭据管家通过加密对其进行保护。您可以指定一个KMS主密钥或依赖凭据管家的默认系统加密，详情请参见加密保护凭据。

- 描述信息

  凭据的描述信息用来帮助您更方便的管理凭据。

- 资源标签

  凭据和KMS的其他资源一样，也支持标签管理。

凭据的版本

凭据对象支持多个版本，通过版本号标识。您存入到凭据对象中的敏感数据（凭据值）被存储为一个凭据版本。凭据的版本有以下特征：

- 凭据管家加密保护的是每个版本的凭据值：凭据值调用CreateSecret或PutSecretValue接口的 `SecretData` 参数指定，在服务端被加密后存储。可以调用GetSecretValue接口获取被解密后的明文。
- 每个版本号所标识的凭据版本只能被写入一次：您不能调用PutSecretValue接口修改某个版本的凭据值。如果您使用同一个版本号多次存入同一个凭据值，则效果是幂等的；如果您使用已有版本的版本号，存入一个新的凭据值，则请求会被拒绝。

版本的状态

凭据管家的使用方式如下：

- 仅托管凭据

  您只使用凭据托管的功能而不对凭据进行轮转，因此凭据对象只有一个凭据值，即当前版本。

- 托管且轮转凭据

  您需要对凭据进行轮转，轮转过程中新的凭据值被存入到一个新的版本，当前版本从老版本切换为新的版本。此后应用程序获取的当前版本会自动更新。

在上述场景中，使用凭据的应用程序仅需要获取当前版本的凭据值。当前版本指版本的状态（Version Stage），凭据管家通过将状态标记（Stage Label）关联到凭据版本的方式来设置版本的状态。凭据管家内建了**ACSCurrent**和**ACSPrevious**两个状态标记，用来表示当前和先前两个版本状态。凭据管家内建状态标记的规则如下：

- 有且仅有一个当前版本

  当前版本被标记为**ACSCurrent**。GetSecretValue接口默认情况下返回**ACSCurrent**标记的版本凭据值。

- 自动变更当前版本和先前版本

  当您调用PutSecretValue接口存入新的凭据值时，如果您不指定新版本的状态，则新版本被默认标记为**ACSCurrent**。

  当您调用PutSecretValue或UpdateSecretVersionStage接口变更了**ACSCurrent**标记的版本，**ACSCurrent**标记的上一个版本会被标记为**ACSPrevious**，即先前版本。

除了**ACSCurrent**和**ACSPrevious**内建版本状态之外，您可以使用自定义状态标记来定义额外的版本状态。内建和自定义的状态标记都适用以下规则：

- 指针规则

  一个状态标记类似一个指针，只能用来标记（指向）一个版本；而每个版本可以被标记零个到多个版本状态。

- 回收规则

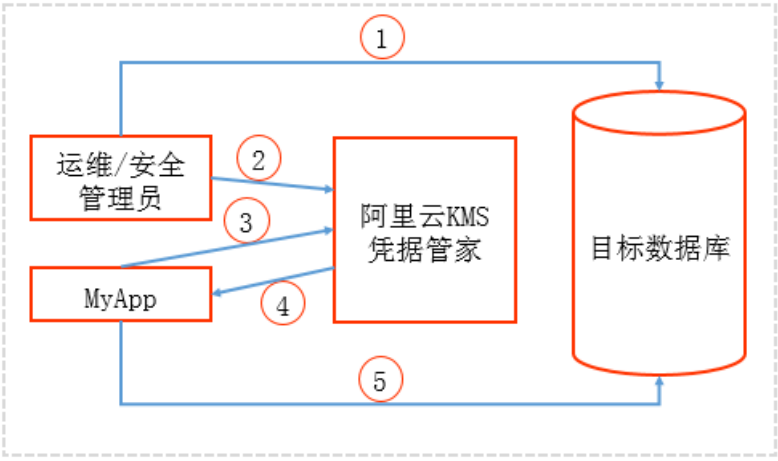
  如果一个版本没有被标记版本状态，则被认为是可回收的（Deprecated）。当凭据内的版本数超过上限时，最老的可回收版本会被删除（回收）。

11.3. 凭据托管和使用

通过本文您可以了解凭据托管和使用的基本场景，以及您可以对凭据执行的相关操作。

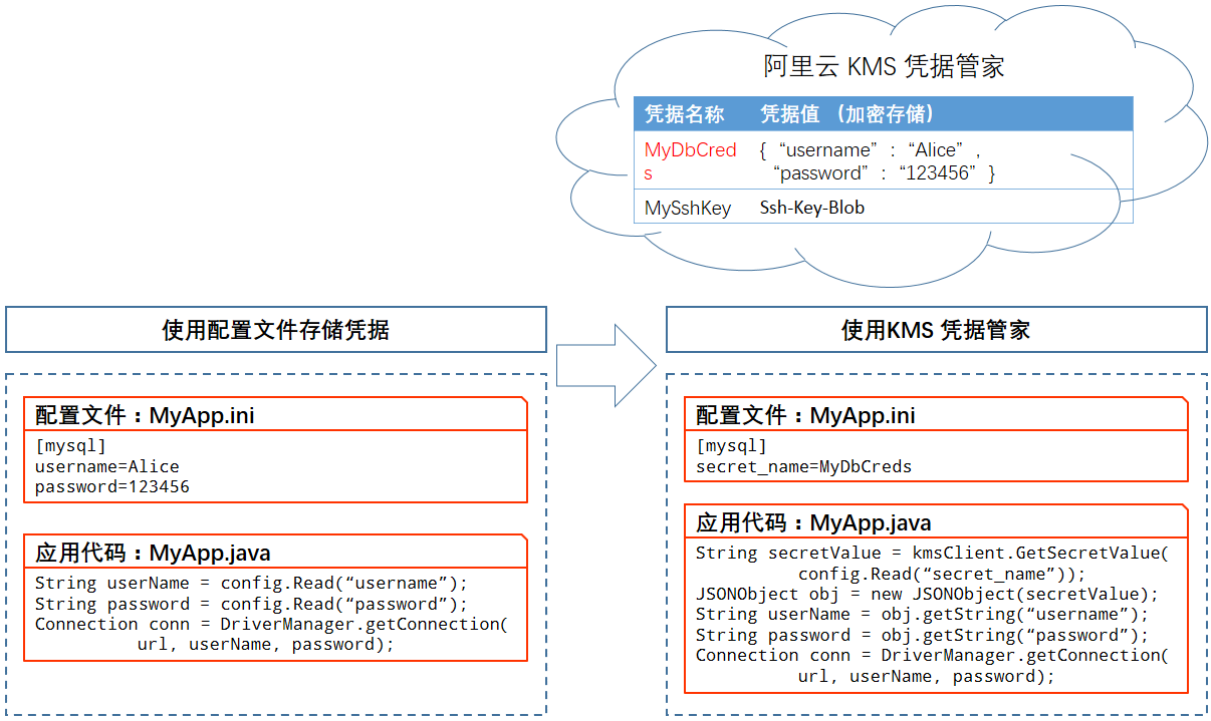
基本场景

以数据库用户名和口令的托管为例，介绍一个基本的凭据托管和使用场景。



1. 管理员在目标数据库配置MyApp访问数据库所需的用户名和密码。
2. 管理员在KMS凭据管家创建一个凭据对象MyDbCreds，用来加密存储上述用户名和密码。
3. 当MyApp需要访问数据库时，需要向KMS凭据管家请求凭据MyDbCreds。
4. KMS凭据管家读取到存储的凭据密文，解密后将明文通过HTTPS返回给MyApp。
5. MyApp读取并解析KMS凭据管家返回的凭据明文，获取到用户名和密码，使用该账号可以访问目标数据库。

对应用MyApp而言，通过调用KMS凭据管家的API来获取敏感的凭据，避免了在程序中硬编码凭据带来的信息泄露风险。硬编码凭据和使用KMS凭据管家的应用程序之间的差异，如下图所示。



托管和保护凭据

- 示例1：创建凭据时不指定加密密钥

  使用如下CLI命令创建凭据，KMS凭据管家会使用系统默认加密的方式保护凭据值。

```
$ aliyun kms CreateSecret \
  --SecretName db_cred \
  --SecretData "{\"uname\": \"alice\", \"pwd\": \"12****\"}" \
  --VersionId v1
```

KMS返回如下结果：

```
{
 "Arn": "acs:kms:us-east-1:123456:secret/db_cred",
 "SecretName": "db_cred",
 "VersionId": "v1",
 "RequestId": "0993b6bb-08de-43b1-a93f-f7dbacaf****"
}
```

- 示例2：创建凭据时指定加密密钥

  使用如下CLI命令创建凭据，KMS凭据管家使用指定的KMS主密钥对凭据进行加密。

```
$ aliyun kms CreateSecret \
  --SecretName ssh_key \
  --SecretData ssh-key-blob \
  --VersionId v1 \
  --EncryptionKeyId Example-CMK-Id
```

> 说明
> - KMS凭据管家使用指定的CMK产生数据密钥，用于加密保护凭据的明文数据。
> - 此时CreateSecret的调用者还需要具备指定CMK的kms:GenerateDataKey权限。

KMS返回如下结果：

```
{
  "Arn": "acs:kms:us-east-1:123456:secret/ssh_key",
  "SecretName": "ssh_key",
  "VersionId": "v1",
  "RequestId": "990bf04d-09a5-44de-925a-2da1fb43****"
}
```

列出和查看凭据元数据

调用ListSecrets接口列出凭据。

```
$ aliyun kms ListSecrets
{
  "SecretList": {
    "Secret": [
      {
        "SecretName": "db_cred",
        "CreateTime": "2020-01-22T03:55:18Z",
        "UpdateTime": "2020-01-22T03:55:18Z"
      },
      {
        "SecretName": "ssh_key",
        "CreateTime": "2020-01-22T03:57:09Z",
        "UpdateTime": "2020-01-22T03:57:09Z"
      }
    ]
  },
  "RequestId": "75aebbde-be68-4cab-ba6e-e4925b61****",
  "PageNumber": 1,
  "PageSize": 10,
  "TotalCount": 2
}
```

调用DescribeSecret接口，查看指定凭据的元数据。此接口不返回凭据值的明文数据。

```
$ aliyun kms DescribeSecret --SecretName ssh_key
{
  "Arn": "acs:kms:us-east-1:123456:secret/ssh_key",
  "SecretName": "ssh_key",
  "EncryptionKeyId": "Example-CMK-Id",
  "Description": "",
  "CreateTime": "2020-01-22T03:57:09Z",
  "UpdateTime": "2020-01-22T03:57:09Z",
  "RequestId": "ca61398f-e61e-4552-aa7e-957955f6****"
}
```

获取凭据值的明文数据

调用GetSecretValue接口，获取被KMS凭据管家加密保护的凭据明文数据。

```
$ aliyun kms GetSecretValue --SecretName db_cred
{
  "SecretName": "db_cred",
  "VersionId": "v1",
  "SecretData": "{\"uname\": \"alice\", \"pwd\": \"12****\"}",
  "SecretDataType": "text",
  "VersionStages": {
    "VersionStage": [
      "ACSCurrent"
    ]
  },
  "CreateTime": "2020-01-22T03:55:18Z",
  "RequestId": "06a380d8-a43c-4cd5-bdd4-e4a8e986****"
}
```

上述返回值中，参数SecretData为解密后的凭据明文数据。

> 说明 凭据对象支持多个版本以及对版本进行状态标记，CreateSecret存入的初始版本会被标记为ACSCurrent，GetSecretValue默认返回被标记为ACSCurrent的版本的凭据值。

删除和恢复凭据

删除的凭据，默认可以在30天内恢复。

```
$ aliyun kms DeleteSecret --SecretName ssh_key
{
  "SecretName": "ssh_key",
  "RequestId": "3e54b02b-6461-46bb-afd5-dbd29d96****",
  "PlannedDeleteTime": "2020-02-21T04:24:04.58616562Z"
}
```

删除的凭据，指定恢复窗口为7天。

```
$ aliyun kms DeleteSecret --SecretName ssh_key --RecoveryWindowInDays 7
{
  "SecretName": "ssh_key",
  "RequestId": "95ec4f18-8f97-4fd5-b7c6-1588979d****",
  "PlannedDeleteTime": "2020-01-29T04:25:14.165242211Z"
}
```

在恢复窗口期内，可以恢复凭据。

```
$ aliyun kms RestoreSecret --SecretName ssh_key
{
  "RequestId": "12770cee-92af-42f5-88e0-cbaa7e0c****",
  "SecretName": "ssh_key"
}
```

强制删除的凭据，不允许恢复。

```
$ aliyun kms DeleteSecret --SecretName ssh_key --ForceDeleteWithoutRecovery true
{
  "SecretName": "ssh_key",
  "RequestId": "75efc9c3-8e21-4e38-b6e4-486886be****",
  "PlannedDeleteTime": "2020-01-22T12:28:22.006884739+08:00"
}
```

轮转凭据

如果您需要对凭据进行轮转，请参见凭据的对象模型和凭据轮转。

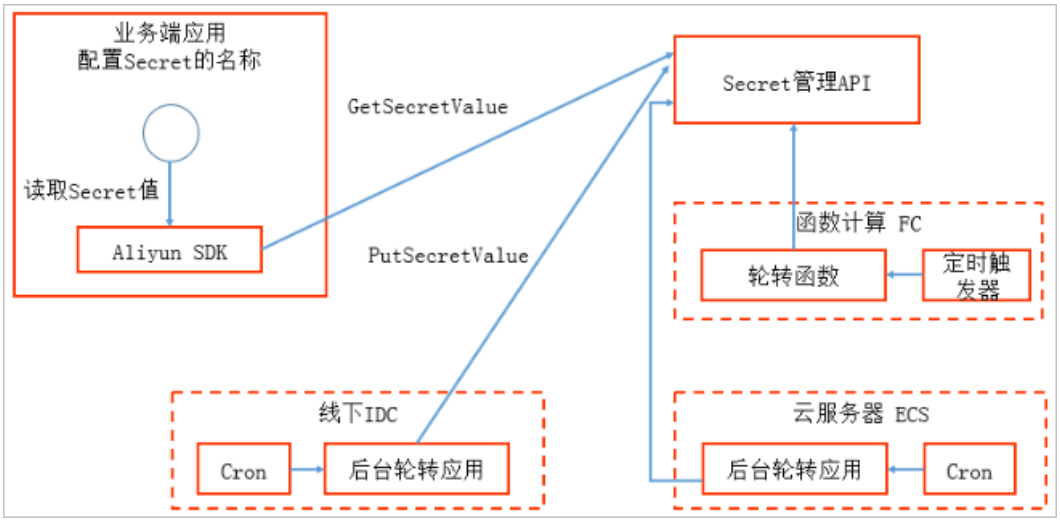
11.4. 凭据轮转

通过本文您可以了解凭据轮转的典型场景，以及通过一个或多个API调用完成凭据轮转的操作方法。

如果一个业务系统使用了凭据管家，其架构中通常有如下两类角色：

- 凭据的消费者：也就是使用凭据的业务应用。业务应用通过持续调用GetSecretValue接口获取最新版本的凭据明文数据，将其用于访问目标系统。
- 凭据的产生者：通常是企业IT运维系统，或者安全运维管理员。调用CreateSecret接口创建凭据对象并且存入初始值；如果需要周期性轮转凭据，则可以调用PutSecretValue接口为凭据对象存入新的版本。

本文以两个典型的场景作为范例，为您介绍如何使用凭据管家API接口实现凭据轮转，如下图所示。



模拟场景

凭据轮转模拟场景的前提条件如下：

- 您的运维系统已经为凭据存入了初始版本，初始版本默认被凭据管家标记为**ACSCurrent**。
- 您需要使用凭据的应用，调用GetSecretValue接口获取所需的凭据值，客户端仅需要指定凭据的名称，服务端返回被标记为**ACSCurrent**的版本凭据值。

您的运维系统对凭据进行轮转，在凭据对象中存入新版本的凭据值，当新的版本被标记为**ACSCurrent**之后，客户端会获取到新的凭据值。

场景1：一个API调用完成轮转

适用于在目标系统中新生成的凭据已经生效，随后再存入凭据管家的情景。例如：常见的*OAuth 2.0*系统中的应用密钥（App Secret）通常由系统生成，您可以将它们托管在凭据管家。轮转流程如下：

1. 管理员在OAuth 2.0应用管理系统中，生成新的App Secret。
2. 将新的App Secret存入托管的凭据中，同时新存入的App Secret所在版本会被标记为**ACSCurrent**。**ACSCurrent**标记的上一个版本被自动标记为**ACSPrevious**。

```
$ aliyun kms PutSecretValue \
  --SecretName MyOAuthAppSecret
  --SecretData sample-app-secret \
  --VersionId v2
```

最后，GetSecretValue接口返回被标记为**ACSCurrent**状态的版本的凭据值即为最新的App Secret。

场景2：多个API调用完成轮转

更多情况下，目标系统并不会自动生成凭据。例如：RDS数据库的账号和ECS实例的SSH Key，您可以将创建的新凭据存储到凭据管家中，作为已有凭据的待定状态的版本；随后，在将新的凭据值注册到目标系统之后，将KMS凭据的待定版本更新为当前版本。

更新托管在凭据管家的数据库用户的口令通常包含如下操作：

1. 轮转程序需要为新的口令产生一个随机的字符串。

   通过凭据管家的GetRandomPassword接口来实现。

```
$ aliyun kms GetRandomPassword --ExcludePunctuation true
{
  "RequestId": "e36ca295-6e47-4dfb-9df1-48d19df4****",
  "RandomPassword": "v2GwsgcuNylyYw9JGJNE5yBViGSi****"
}
```

2. 将数据库的新用户和新密码，作为一个新的版本存入凭据中。

   如下的CLI 命令会保持**ACSCurrent**和**ACSPrevious**标记的版本不变，将新创建的版本标记为**MyPendingLabel**。

```
$ aliyun kms PutSecretValue \
  --SecretName db_cred
  --SecretData "{\"uname\": \"alice\", \"pwd\": \"v2GwsgcuNylyYw9JGJNE5yBViGSiZ****"}" \
  --VersionId v2 \
  --VersionStages "[\"MyPendingLabel\"]"
```

3. 在目标数据库中，注册上述新的用户名和口令。

   凭据对象db_cred内的版本号为v2的凭据值，才可以用于访问目标数据库。

4. 注册新的账号后，您可以通过以下CLI命令，将凭据对象中新存入的凭据版本设置为ACSCurrent状态。

```
$ aliyun kms UpdateSecretVersionStage \
  --SecretName db_cred \
  --VersionStage ACSCurrent \
  --MoveToVersion v2
```

最后，GetSecretValue接口返回被标记为ACSCurrent状态的版本的凭据值即是最新的账号口令。