ALIBABA CLOUD

阿里云

密钥管理服务

密钥管理服务公共云合集

文档版本：20220711

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.访问控制与审计 08
1.1. 使用RAM实现对资源的访问控制 08
1.2. 使用操作审计查询密钥管理服务的操作事件 15
2.专属KMS基础版 17
2.1. 概述 17
2.2. 快速入门 18
2.3. 服务关联角色 20
2.4. 基础操作 23
2.4.1. 管理专属KMS基础版实例 23
2.4.2. 管理密钥 24
2.4.3. 管理应用接入点 25
2.4.4. 升级实例 28
2.5. SDK参考 28
2.5.1. 专属KMS Java SDK 28
2.5.2. 专属KMS PHP SDK 32
2.5.3. 专属KMS Go SDK 36
2.5.4. 专属KMS Python SDK 40
3.专属KMS标准版 44
3.1. 概述 44
3.2. 快速入门 45
3.3. 服务关联角色 49
3.4. 基础操作 51
3.4.1. 管理专属KMS标准版实例 51
3.4.2. 管理密钥 53
3.4.3. 导入对称密钥材料 54
3.4.4. 导入非对称密钥材料 56

3.4.5. 管理应用接入点 66
3.4.6. 升级实例 69
3.5. 专属KMS SDK 69
3.5.1. 专属KMS Java SDK 69
3.5.2. 专属KMS PHP SDK 73
3.5.3. 专属KMS Go SDK 77
3.5.4. 专属KMS Python SDK 81
4.凭据管家 85
4.1. 凭据管家概述 85
4.2. 管理应用接入点 87
4.3. 应用程序接入凭据管家 90
4.4. 通用凭据 92
4.4.1. 通用凭据概述 93
4.4.2. 管理通用凭据 94
4.4.3. 轮转通用凭据 97
4.5. 动态RDS凭据 99
4.5.1. 动态RDS凭据概述 99
4.5.2. 管理动态RDS凭据 100
4.5.3. 监控动态RDS凭据轮转 101
4.5.4. 动态RDS凭据服务关联角色 104
4.6. 动态RAM凭据 105
4.6.1. 动态RAM凭据概述 105
4.6.2. 授予凭据管家管理RAM用户AccessKey的权限 106
4.6.3. 管理动态RAM凭据 107
4.7. 动态ECS凭据 109
4.7.1. 动态ECS凭据概述 109
4.7.2. 管理动态ECS凭据 109
4.7.3. 监控动态ECS凭据轮转 111

4.7.4. 动态ECS凭据服务关联角色 114
5.证书管家 116
5.1. 证书管家概述 116
5.2. 证书管家快速入门 116
5.3. 导入密钥和证书 119
5.4. 禁用、吊销或删除证书 119
6.云产品与KMS的集成 121
6.1. 服务端集成加密概述 121
6.2. 支持服务端集成加密的云服务 122
7.API参考 126
7.1. API概览 126
8.SDK参考 127
8.1. 概述 127
8.2. 加密SDK 127
8.2.1. 加密SDK概述 128
8.2.2. 快速入门 128
8.2.2.1. 加密SDK快速入门（Java） 128
8.2.2.2. 加密SDK快速入门（Python） 131
8.2.3. 加密场景 133
8.2.3.1. 对OSS进行客户端加解密 133
8.2.3.2. 使用多个地域的CMK加密和解密数据 134
8.2.3.3. 数据签名验签 138
8.2.3.4. 数据库敏感数据加密 139
8.2.4. 原理与数据格式 140
8.2.4.1. 加密解密与签名验签的原理 140
8.2.4.2. 加密与签名数据的格式 142
8.3. 凭据管家SDK 146
8.3.1. 凭据管家客户端 146

8.3.2. 凭据管家JDBC客户端 152
8.3.3. 多种阿里云SDK的托管凭据插件 155
8.3.4. 凭据管家Kubernetes插件 158
8.4. 最佳实践 159
8.4.1. 从ECS实例安全访问KMS 159

1.访问控制与审计

1.1. 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角色则需要通过显式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
抽象证书容器	acs:kms:${region}:${account}:certificate
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/${alias-name}
证书	acs:kms:${region}:${account}:certificate/${id}

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

KMS接口	Action	资源类型
DescribeKey	kms:DescribeKey	密钥
UpdateKeyDescription	kms:UpdateKeyDescription	密钥
EnableKey	kms:EnableKey	密钥
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

KMS接口	Action	资源类型
GenerateAndExportDataKey	kms:GenerateAndExportDataKey	密钥
AsymmetricSign	kms:AsymmetricSign	密钥
AsymmetricVerify	kms:AsymmetricVerify	密钥
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 ○ kms:Decrypt **说明** 仅当您使用自己创建的CMK作为通用凭据的加密主密钥时，需要验证kms:Decrypt。	凭据
PutSecretValue	○ kms:PutSecretValue ○ kms:GenerateDataKey **说明** 仅当您使用自己创建的CMK作为通用凭据的加密主密钥时，需要验证kms:GenerateDataKey。	凭据
ListSecretVersionIds	kms:ListSecretVersionIds	凭据
UpdateSecretVersionStage	kms:UpdateSecretVersionStage	凭据

KMS 接口	Action	资源类型
GetRandomPassword	kms:GetRandomPassword	无

- 证书管家接口

KMS 接口	Action	资源类型
CreateCertificate	kms:CreateCertificate	证书
UploadCertificate	kms:UploadCertificate	证书
GetCertificate	kms:GetCertificate	证书
DescribeCertificate	kms:DescribeCertificate	证书
UpdateCertificateStatue	kms:UpdateCertificateStatue	证书
DeleteCertificate	kms:DeleteCertificate	证书
CertificatePrivateKeySign	kms:CertificatePrivateKeySign	证书
CertificatePublicKeyVerify	kms:CertificatePublicKeyVerify	证书
CertificatePublicKeyEncrypt	kms:CertificatePublicKeyEncrypt	证书
CertificatePrivateKeyDecrypt	kms:CertificatePrivateKeyDecrypt	证书

- 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>` 。

更多信息，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": [
        "*"
      ]
    }
  ]
}
```

- 密钥只读访问权限，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"
                    ]
                }
            }
        }
    ]
}
```

- 允许以下IP地址访问密钥：
  - 指定IP地址段：192.168.0.0/16
  - 指定IP地址：172.16.215.218

```
{
  "Version": "1",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:*"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "IpAddress": {
        "acs:SourceIp": [
          "192.168.0.0/16",
          "172.16.215.218"
        ]
      }
    }
  }]
}
```

- 凭据只读访问权限，即列出和查看凭据、查看凭据版本、查看凭据内容以及生成随机密码的权限。

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

- 证书只读访问权限，即列出和查看证书详情的权限。

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

- 证书签名和验签权限，即使用指定证书生成和验证数字签名的权限。

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

1.2. 使用操作审计查询密钥管理服务的操作事件

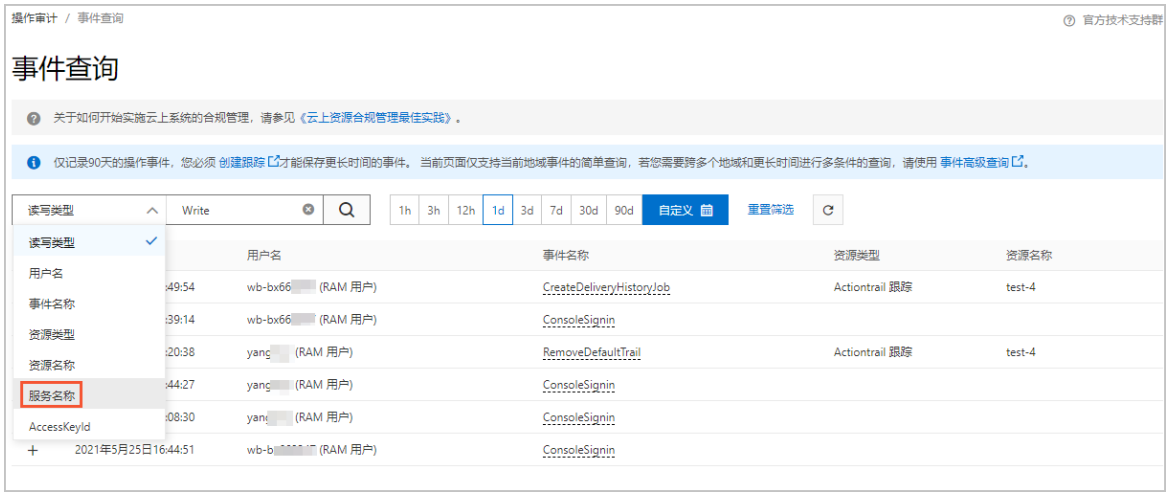
密钥管理服务KMS（Key Management Service）已经集成操作审计（ActionTrail），您可以使用操作审计查询所有用户（阿里云账号或RAM用户）对KMS资源进行的操作事件。本文为您介绍如何使用操作审计控制台查询密钥管理服务的操作事件。

背景信息

操作审计会记录KMS除DescribeRegions以外的所有API调用事件。关于事件代码示例详情，请参见KMS事件代码示例。关于操作事件字段的含义详情，请参见管控事件结构定义。

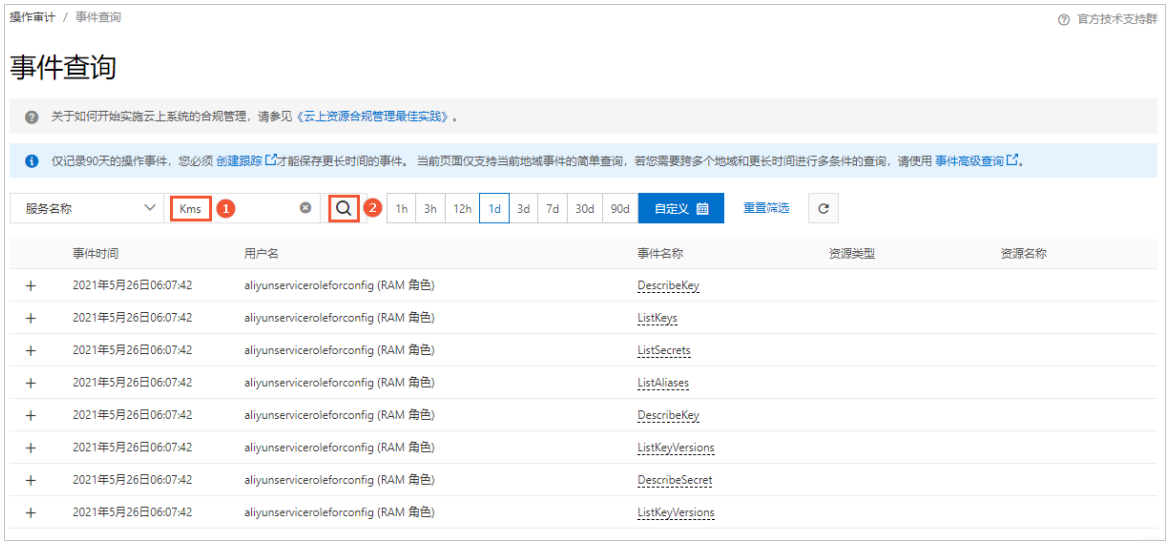
操作步骤

1. 登录操作审计控制台。
2. 在左侧导航栏，单击**事件查询**。
3. 在顶部导航栏选择待查询事件的地域。
4. 在**事件查询**页面，从下拉列表中选择**服务名称**。



5. 在文本框中输入Kms，然后单击

图标。



6. 鼠标悬停至**事件名称**列的事件名称，查询事件详情。
7. （可选）如需查询事件代码记录，您可以单击事件前面的加号，然后单击**事件详情**。

2.专属KMS基础版

2.1. 概述

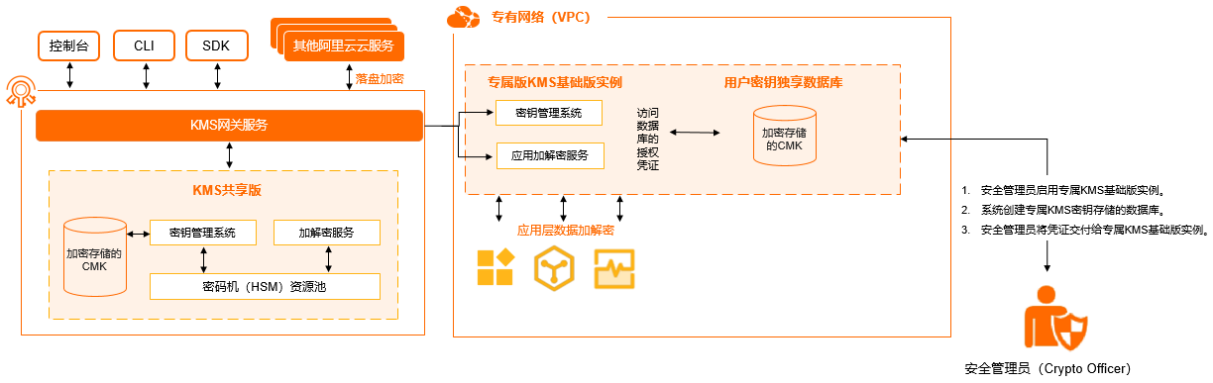
专属KMS基础版具备用户侧数据加密能力，将密钥单独存储在租户独享的数据库中，同时支持云产品原生的数据加密。

应用场景

- 自建应用集成
  自建应用程序可以通过VPC网络使用专属KMS基础版实例的加解密服务进行应用层加解密。
- 密钥运算和存储租户侧独享
  您可以借助密钥运算与租户侧独享存储的专属KMS基础版架构，提升密钥和加密数据安全性。
- 云产品集成
  您可以授权KMS共享版转发云产品的服务端加密请求到专属KMS基础版实例。

产品架构

专属KMS基础版是一个独立部署的实例型服务，产品架构如下图所示。



专属KMS基础版主要组成部分如下：

- 用户独享密钥存储
  您的专属KMS密钥以加密方式单独存储在您独享的数据库中。
- 密钥管理系统
  在您自定义的独享专属KMS基础版实例内，进行密钥的生命周期管理。
- 密码计算服务
  专属KMS基础版实例通过简单易用的API调度密码计算。密码计算过程中密钥不会离开您独享计算资源的安全边界。

使用限制

限制项	最大值
一个实例允许创建的最大用户主密钥（CMK）数量	1000
一个阿里云账号的每秒请求数QPS（Query Per Second）上限	1000

支持的地域

支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。

计费
专属KMS基础版采用预付费（包年包月）的计费方式。更多信息，请参见专属KMS计费说明。

使用专属KMS基础版
关于如何使用专属KMS基础版，请参见快速入门。

2.2. 快速入门
专属KMS分为基础版和标准版，本文为您介绍如何购买并启用专属KMS基础版实例、如何创建密钥、以及如何让应用接入实例。

前提条件
请确保您已经在专属KMS基础版实例所在地域创建专有网络和交换机。具体操作，请参见创建专有网络和交换机和创建交换机。

操作流程


步骤一：购买专属KMS基础版实例
1. 登录密钥管理服务控制台。
2. 在左侧导航栏，单击**专属KMS**，在**专属KMS**页面单击**购买专属KMS**。
3. 在购买页面，设置相关参数。

   > **说明** **定制实例**和**密钥数量**由系统默认填入，无需手动设置。

   - **版本**：选择**基础版**。
   - **地域**：根据需要选择专属KMS实例所在地域。
   - **专有网络数量**：根据需要选择专有网络VPC（Virtual Private Cloud）的数量。同一地域内网络互通的多个VPC支持使用同一个专属KMS实例。

     > **说明** 如果多个VPC之间网络未连通，您可以根据业务需要，选择如下方式之一：使用云企业网实现同地域VPC互通、使用高速通道实现VPC互通或使用VPN网关实现VPC互通。

   - **购买数量**：根据需要选择实例数量。最多支持同时购买20个实例。
   - **购买时长**：根据需要选择购买时长。

     > **说明** 您可以选中**到期自动续费**，当前实例到期后将自动续费。

4. 确认总配置费用后，单击**立即购买**并完成**支付**。

   支付成功后，专属KMS列表中会新增专属KMS基础版实例，**状态**为**未启用**。您需要启用专属KMS基础版实例才能正常使用该实例。

步骤二：启用专属KMS基础版实例
购买专属KMS基础版实例后，您需要先启用专属KMS基础版实例，设置实例所在的专有网络和交换机信息，才能正常使用密钥管理服务。

1. 在**专属KMS**页面，找到目标专属KMS基础版实例，在**操作**列单击**启用**。
2. 在**启动专属KMS实例**对话框，设置相关参数。
   - **VPC ID**：专属KMS基础版实例所在的专有网络ID。
   - **VSwitch ID**：专有网络下的交换机ID。
3. 单击**立即启用**。

   请等待15~30分钟，然后刷新页面，当状态变更为**已启用**时，专属KMS基础版实例启用成功。
4. 如果您需要设置多VPC使用同一专属KMS实例，单击目标实例**专有网络**列下的+，配置专有网络。

   在**待选专有网络**中选中需要设置的专有网络，单击<后，完成**确定**。

> **说明**
>
> - 您可以单击目标实例**操作**列的**详情**，在**服务规格**中会显示当前实例支持关联的VPC数量。
> - 如果选择关联的VPC数量超过了关联资源限制，您可以根据控制台上的提示，单击**前往购买**升级**专有网络数量**的规格。

步骤三：为专属KMS基础版实例创建密钥

当专属KMS基础版实例处于**已启用**状态时，您可以为专属KMS基础版实例创建密钥，以便进行加密解密、签名验签等操作。

1. 在**专属KMS**页面，找到目标专属KMS基础版实例，在**操作**列单击**管理**。
2. 在**用户主密钥**区域，单击**创建密钥**。
3. 在**创建密钥**对话框，设置以下参数。

配置项	说明

密钥类型	取值：
	- 对称密钥的类型：
	  - Aliyun_AES_256
	- 非对称密钥的类型：
	  - RSA_2048
	  - RSA_3072
	  - EC_P256
	  - EC_P256K

密钥用途	取值： - Encrypt/Decrypt：数据加密和解密。 - Sign/Verify：产生和验证数字签名。
别名	用户主密钥的标识符。支持英文字母、数字、下划线（_）、短划线（-）和正斜线（/）。
描述	密钥的说明信息。

4. 单击**确定**。

步骤四：应用接入专属KMS基础版实例

如果您需要使用专属KMS SDK对数据进行密钥运算，才需执行本步骤。当专属KMS基础版实例处于**已启用**状态时，您可以为专属KMS基础版实例快速创建应用接入点（AAP）和应用身份凭证（Client Key），以便应用程序正常访问专属KMS。您也可以获取专属KMS基础版实例CA证书，以便验证专属KMS基础版实例。

1. 在**专属KMS**页面，找到目标专属KMS基础版实例，在**操作**列单击**详情**。
2. 在**应用接入指南**区域，单击**快速创建应用接入点**。
3. 在**快速配置应用身份凭证和权限**面板，设置应用接入点信息。
   i. 输入**应用接入点名称**。
   ii. 设置**访问控制策略**。
   - **允许访问资源**：默认填写`Key/*`，表示允许访问当前专属KMS基础版实例的全部密钥。
   - **允许网络来源**：允许访问的网络类型和IP地址。您可以设置私网IP地址或者私网网段，多个IP地址之间用半角逗号（,）分隔。

   iii. 单击**创建**。
4. 在**应用身份凭证**对话框，获取Client Key的**凭证口令**和**应用身份凭证内容**。
   - **凭证口令**：单击**复制口令**，获取凭证口令。
   - **应用身份凭证内容**：单击**下载应用身份凭证**，保存应用身份凭证信息。
     应用身份凭证信息包含凭证ID（KeyId）和凭证内容（PrivateKeyData），凭证内容为PKCS12格式Base64编码。示例如下：

     ```
     {
       "KeyId": "KAAP.71be72c8-73b9-44e0-bb75-81ee51b4****",
       "PrivateKeyData": "MIIJwwIBAz****ICNXX/pOw=="
     }
     ```

     > **说明** 专属KMS不会保存Client Key的凭证口令和应用身份凭证内容，您只能在创建Client Key时获取到该信息，请妥善保管。

5. 单击**关闭**。

   应用接入点创建成功后，您可以在左侧导航栏单击**应用管理**查看应用接入点信息，包括认证方式、权限策略、网络控制规则、Client Key等。您也可以更新应用接入点。具体操作，请参见管理应用接入点。
6. 在**应用接入指南**区域，单击**获取实例CA证书**下方的**下载**，下载.pem格式的CA证书文件。

后续步骤

您可以使用专属KMS SDK，通过访问专属KMS API调用密钥管理服务。具体操作，请参见专属KMS Java SDK、专属KMS PHP SDK、专属KMS Go SDK和专属KMS Python SDK。

2.3. 服务关联角色

本文为您介绍专属KMS服务关联角色（AliyunServiceRoleForKMSKeyStore）的应用场景、权限策略、创建及删除操作。

应用场景

创建和使用专属KMS基础版实例时，密钥管理服务KMS需要通过服务关联角色访问依赖系统（ECS、VPC等系统）服务。

关于服务关联角色的更多信息，请参见服务关联角色。

权限说明

角色名称：AliyunServiceRoleForKMSKeyStore。

权限策略：AliyunServiceRolePolicyForKMSKeyStore。

权限说明：KMS使用此角色访问ECS和VPC等其他云服务相关资源。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ecs:CreateNetworkInterfacePermission",
        "ecs:DeleteNetworkInterfacePermission",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:DescribeSecurityGroups",
        "ecs:CreateSecurityGroup",
        "ecs:DeleteSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:AuthorizeSecurityGroupEgress",
        "ecs:RevokeSecurityGroup",
        "ecs:RevokeSecurityGroupEgress",
        "ecs:DescribeSecurityGroupAttribute"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "vpc:DescribeVSwitches",
        "vpc:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "yundun-hsm:DescribeInstances",
        "yundun-hsm:DescribeClusters"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "keystore.kms.aliyuncs.com"
        }
      }
    }
  ]
}
```

创建服务关联角色

当您使用阿里云账号在密钥管理控制台首次创建专属KMS基础版实例时，会自动创建服务关联角色（AliyunServiceRoleForKMSKeyStore）。

当您使用的是RAM用户，需要先授权以下自定义策略，才能在密钥管理控制台首次创建专属KMS基础版实例时，自动创建服务关联角色（AliyunServiceRoleForKMSKeyStore）。具体操作，请参见为RAM用户授权。

```
{
    "Action": "ram:CreateServiceLinkedRole",
    "Resource": "*",
    "Effect": "Allow",
    "Condition": {
        "StringEquals": {
            "ram:ServiceName": "keystore.kms.aliyuncs.com"
        }
     }
}
```

删除服务关联角色

删除服务关联角色前，您需要确保当前阿里云账号下的专属KMS基础版实例已经被释放。当专属KMS基础版实例费用到期且没有续费时，KMS会自动释放该专属KMS基础版实例。

您可以在RAM控制台删除服务关联角色。具体操作，请参见删除RAM角色。

2.4. 基础操作

2.4.1. 管理专属KMS基础版实例

当专属KMS基础版实例处于**已启用**状态时，您可以根据需要查询专属KMS基础版实例详情，配置多VPC使用同一专属KMS基础版实例，为专属KMS基础版实例启用安全审计。

查询专属KMS基础版实例

您可以根据需要，查询专属KMS实例ID、实例VPC地址、专有网络等信息。

实例VPC地址是每个专属KMS实例自己专属的访问入口，格式为 `https://{实例ID}.cryptoservice.kms.aliyuncs.com` 。

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，选择专属KMS基础版实例所在的地域。

   支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS基础版实例**操作**列的**详情**。
5. 在**专属KMS实例详情**面板，查询专属KMS基础版实例详情。

配置多VPC使用同一专属KMS基础版实例

配置多VPC使用同一专属KMS基础版实例时，多个VPC之间网络需要联通。如果多VPC之间网络未联通，您可以根据业务需要，选择如下方式之一：使用云企业网实现同地域VPC互通、使用高速通道实现VPC互通或使用VPN网关实现VPC互通。

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，选择专属KMS基础版实例所在的地域。

   支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。

3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS基础版实例**专有网络**列的+。
5. 在**配置专有网络**对话框，选中**待选专有网络**中需要设置的专有网络，单击<后，完成**确定**。

> **说明**
> - 您可以单击目标实例**操作**列的**详情**，在**服务规格**中会显示当前实例支持关联的VPC数量。
> - 如果选择关联的VPC数量超过了关联资源限制，您可以根据控制台上的提示，单击**前往购买**升级**专有网络数量**的规格。

启用安全审计

您在访问专属KMS基础版实例过程中会产生审计日志。审计日志中记录了对实例的访问数据，包括调用实例时的请求信息、用户信息、被访问资源的信息，以及访问结果等。日志文件示例如下：

```
2021-10-19T212021-10-19T21:40:01     [INFO]  - - 3dd60a7a-4587-4c57-8197-d749c3578974 Creat
eKey - TMP.3KfAHseF5DVULM2s8YUhdB8YvwM4nZA1wXr8AcAAhR7YhdyosXG2eSpsRFPMjYbvUArPRtsCWKzxEo88
bC5w5LBfyp**** 111760096384**** 111760096384**** - kst-phzz6108e50c15333w**** - 37 - -40:01
[INFO]  - - 3dd60a7a-4587-4c57-8197-d749c3578974 CreateKey - TMP.3KfAHseF5DVULM2s8YUhdB8Yvw
M4nZA1wXr8AcAAhR7YhdyosXG2eSpsRFPMjYbvUArPRtsCWKzxEo88bC5w5LBfyp**** 111760096384**** 11176
0096384**** - kst-phzz6108e50c15333w**** - 37 - -
```

当您启用安全审计后，专属KMS会将审计日志以小时为单位投递到您指定的OSS存储空间。启用安全审计前请确保您已经创建OSS存储空间。具体操作，请参见创建存储空间。

> **说明** 启用安全审计后，将在1小时内生成并投递审计日志。

1. 单击目标专属KMS基础版实例**操作**列的**详情**。
2. 在**专属KMS实例详情**面板，打开**安全审计**开关。
3. 在**配置安全审计**对话框，选择**日志存储位置**。
4. 单击**确定**。

   成功启用安全审计后，安全审计状态由**未开启**变为**已开启**。您也可以按需修改安全审计配置或者禁用安全审计。

2.4.2. 管理密钥

当您为专属KMS基础版实例创建密钥后，可以根据实际需求禁用密钥、开启删除保护或计划删除密钥。

禁用密钥

密钥创建完成后，默认为启用状态。您可以禁用密钥，被禁用的密钥无法用于加密解密、签名验签。

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，选择专属KMS基础版实例所在的地域。

   支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS基础版实例**操作**列的**管理**。

5. 在**用户主密钥**区域，单击目标密钥**操作**列的**禁用**。
6. 在**禁用密钥**对话框，单击**确定**。

   成功禁用密钥后，密钥状态由**启用中**变为**已禁用**。您也可以单击**启用**，再次启用密钥。

开启删除保护

当您为密钥开启删除保护后，将无法通过控制台或API删除该密钥，从而避免误删除密钥。

> **说明** **待删除**状态的密钥无法开启删除保护。

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，选择专属KMS基础版实例所在的地域。

   支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS基础版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，单击目标密钥名称。
6. 在**密钥详情**区域，单击**开启删除保护**。
7. 在**确认开启**对话框，单击**确定**。

   成功开启删除保护后，删除保护状态由**未开启**变为**已开启**。您也可以单击**关闭删除保护**，为密钥关闭删除保护，以便删除该密钥。

计划删除密钥

密钥一旦删除，将无法恢复，使用该密钥加密的内容及产生的数据密钥也将无法解密。因此，KMS只提供计划删除密钥的方式，而不提供直接删除的方式。如果需要删除密钥，推荐您使用禁用密钥功能。

> **说明** 请确保密钥已经关闭删除保护。

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，选择专属KMS基础版实例所在的地域。

   支持专属KMS基础版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS基础版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，在目标密钥**操作**列选择 > **计划删除密钥**。
6. 在**计划删除密钥**对话框，填写预删除周期。

   预删除周期取值为7~366天，默认值为366天。
7. 单击**确定**。

   成功计划删除密钥后，密钥状态变为**待删除**。处于**待删除**状态的密钥无法用于加密解密、签名验签。您也可以选择 > **取消删除密钥**撤销删除密钥的申请。

2.4.3. 管理应用接入点

当您为专属KMS实例创建应用接入点后，应用接入点权限策略作用域会显示专属KMS。您可以根据需要更新应用接入点、删除应用接入点或者删除Client Key。

创建应用接入点

当专属KMS实例处于**已启用**状态时，您可以为实例快速创建应用接入点（AAP）和应用身份凭证（Client Key），以便应用程序正常访问专属KMS。

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，选择专属KMS实例所在的地域。
3. 在左侧导航栏，单击**专属KMS**。
4. 在**专属KMS**页面，找到目标专属KMS实例，在**操作**列单击**详情**。
5. 在**应用接入指南**区域，单击**快速创建应用接入点**。
6. 在**快速配置应用身份凭证和权限**面板，设置应用接入点信息。
   i. 输入**应用接入点名称**。
   ii. 设置**访问控制策略**。
   - **允许访问资源**：默认填写 `Key/*` ，表示允许访问当前专属KMS实例的全部密钥。
   - **允许网络来源**：允许访问的网络类型和IP地址。您可以设置私网IP地址或者私网网段，多个IP地址之间用半角逗号（,）分隔。

   iii. 单击**创建**。
7. 在**应用身份凭证**对话框，获取**凭证口令**和**应用身份凭证内容**（Client Key）。
   - **凭证口令**：单击**复制口令**，获取凭证口令。
   - **应用身份凭证内容**：单击**下载应用身份凭证**，保存应用身份凭证信息。
     应用身份凭证信息包含凭证ID（KeyId）和凭证内容（PrivateKeyData），示例如下：

```
{
  "KeyId": "KAAP.71be72c8-73b9-44e0-bb75-81ee51b4****",
  "PrivateKeyData": "MIIJwwIBAz****ICNXX/pOw=="
}
```

> **说明** 专属KMS不会保存Client Key的凭证内容，因此您只能在创建Client Key时获取到加密的PKCS12文件（凭证内容），请妥善保管。

8. 单击**关闭**。

   应用接入点创建成功后，您可以在左侧导航栏单击**应用管理**查看应用接入点信息，包括认证方式、权限策略、网络控制规则、Client Key等。
9. 在**应用接入指南**区域，单击**获取实例CA证书**下方的**下载**，下载.pem格式的CA证书文件。

更新应用接入点

您可以根据需要更新应用接入点的权限策略，更新应用使用专属KMS实例的权限，从而实现不同应用拥有不同实例访问权限的目的。

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 找到目标应用接入点名称，然后单击右上角的**更新**。
5. 在**更新应用接入点**对话框，更新权限策略。

i. 单击**可选策略**右侧的 ＋ 图标。

ii. 在**创建权限策略**对话框，配置以下参数，然后单击**创建**。

配置项	说明
权限策略名称	权限策略的名称。
作用域	权限策略的适用范围。 选择专属KMS实例的服务ID。
RBAC权限	权限管理模板，表示权限策略对具体资源的操作。 选择CryptoServiceKeyUser。
允许访问资源	权限策略被授权的具体对象。可以通过以下两种方法设置： ■ 方法一：在**可选资源**区域，选择已有资源，然后单击 < 图标。 ■ 方法二：在**已选资源区域**，单击 ＋ 图标，然后手动输入资源，最后单击**添加**。 **说明** 资源支持通配符（*）作为后缀。
网络控制规则	权限策略允许访问的网络类型和IP地址。 您可以在**可选规则**区域，选择已有规则，或者按照以下步骤创建并添加新规则。 a. 单击 ＋ 图标。 b. 在**创建网络访问规则**对话框，设置以下参数： ■ **名称**：网络访问规则的名称。 ■ **网络类型**：应用访问KMS的网络类型。 选择Private，适用于应用程序访问部署到VPC内的专属服务。 ■ **描述信息**：网络访问规则的详细信息。 ■ **允许私网地址**：允许应用程序访问的网络地址。 您可以设置私网IP地址或者网段，多个IP地址之间用半角逗号（,）间隔。 c. 单击**创建**。 d. 选择已有规则，然后单击 < 图标。

iii. 选择已有策略，然后单击 < 图标。

6. 输入描述信息，然后单击**更新**。

删除应用接入点

应用接入点是使用专属KMS实例需要的凭证，删除后您可能无法正常使用专属KMS实例，请谨慎操作。删除应用接入点将同步删除应用接入点绑定的所有Client Key。

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 单击目标应用接入点**操作**列的**删除**。
5. 在**删除应用接入点**对话框，单击**确定**。

删除Client Key

Client Key是应用接入点用于身份认证的凭证，其凭证内容需要在首次创建时妥善保存。如果您忘记了Client Key的凭证内容，可以删除该Client Key，然后创建一个新的Client Key。

Client Key删除方法如下：

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 单击应用接入点名称。
5. 在**Client Key**区域，单击目标Client Key**操作**列的**删除**。
6. 在**删除Client Key**对话框，单击**确定**。

2.4.4. 升级实例

购买专属KMS实例后，如果实例规格无法满足您的实际业务需要，您可以在密钥管理服务控制台升级实例规格。本文介绍如何升级实例规格。

背景信息

升级实例表示提升当前专属KMS实例的规格配置，您需要补齐实例的规格差价。

> **说明** 目前仅支持升级**专有网络数量**规格。同一地域的多个VPC中使用同一个专属KMS实例时，如果选择关联的VPC数量超过了关联资源限制，您需要升级**专有网络数量**。

操作步骤

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，选择专属KMS实例所在的地域。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS实例**操作**列的**详情**。
5. 在**专属KMS实例详情**页签，单击**服务规格**后的**升配**。
6. 在**变配**页签选择**专有网路数量**，勾选**服务协议**。
7. 单击**立即购买**并完成**支付**。

2.5. SDK参考

2.5.1. 专属KMS Java SDK

专属KMS SDK for Java帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS基础版实例，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 您已经获取了专属KMS基础版实例VPC地址，并确保可以通过以下方式访问专属KMS基础版实例VPC地址：
  - 启用专属KMS基础版实例时设置了VPC ID，在该VPC中您可以访问专属KMS基础版实例VPC地址。
  - 您专属KMS SDK程序所在环境的网络，可以正常解析并访问专属KMS基础版实例VPC地址。

  具体操作，请参见查询专属KMS基础版实例。

安装SDK

在项目中添加alibabacloud-dkms-gcs-sdk的依赖，可从Maven仓库中自动下载发布的Java安装包。

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>alibabacloud-dkms-gcs-sdk</artifactId>
  <version>x.x.x</version>
</dependency>
```

> **说明** 专属KMS SDK的最新版本，请参见专属KMS SDK for Java。

初始化SDK

您可以初始化一个专属KMS基础版实例的Java客户端，用于调用专属KMS基础版实例管理的密钥等资源。使用Java SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。
   为保障生产环境通信安全，需要配置Java可信证书。具体配置过程如下：
   i. 将CA证书文件拆分为2个证书文件。
      - 文件1：`rootca.pem`

        ```
        -----BEGIN CERTIFICATE-----
        <Root CA Certificate BASE64 Content>
        -----END CERTIFICATE-----
        ```

      - 文件2：`subca.pem`

        ```
        -----BEGIN CERTIFICATE-----
        <Sub CA Certificate BASE64 Content>
        -----END CERTIFICATE-----
        ```

   ii. 使用keytool工具将拆分后的CA证书导入`JAVA_HOME/lib/security/cacerts`。
      - 导入文件1（`rootca.pem`）

        ```
        keytool -importcert -alias PrivateKmsCA_RootCA -keystore cacerts -storepass changeit -file rootca.pem
        ```

      - 导入文件2（`subca.pem`）

        ```
        keytool -importcert -alias PrivateKmsCA_SubCA -keystore cacerts -storepass changeit -file subca.pem
        ```

iii. 验证代码。

```
URL serviceUrl = new URL("https://<service_id>.cryptoservice.kms.aliyuncs.com");
serviceUrl.openConnection().connect();
```

> **说明** 如果没有 `javax.net.ssl.SSLHandshakeException` 的异常提示，则配置正确。

开发环境可使用 `RuntimeOptions` 设置临时忽略可信证书的验证。示例代码如下：

```
import com.aliyun.dkms.gcs.openapi.util.models.RuntimeOptions;
RuntimeOptions runtimeOptions = new RuntimeOptions();
runtimeOptions.setIgnoreSSL(true);
...
client.encryptWithOptions(encryptRequest, runtimeOptions);
```

2. 创建专属KMS基础版Client。
   创建专属KMS基础版Client时，需要指定实例的Endpoint。EndPoint为专属KMS基础版实例服务地址去掉**https://**。关于专属KMS基础版实例服务地址的更多信息，请参见查询专属KMS基础版实例。

```
import com.aliyun.dkms.gcs.openapi.models.Config;
import com.aliyun.dkms.gcs.sdk.Client;
//连接协议，固定为HTTPS。
String protocol = "https";
//Endpoint，专属KMS基础版实例服务地址去掉https://。
String endpoint = "<service_id>.cryptoservice.kms.aliyuncs.com";
//专属KMS基础版实例Client Key。
String clientKey = "<your client key>";
//专属KMS基础版实例Client Key解密口令。
String clientKeyPass = "<your client key password>";
Client client = new Client(new Config()
                           .setProtocol(protocol)
                           .setEndpoint(endpoint)
                           .setClientKeyContent(clientKey)
                           .setPassword(clientKeyPass));
```

代码示例

- 使用专属KMS基础版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS基础版实例加密密钥的ID或别名（Alias）。
String cipherKeyId = "<your cipher key id>";
//待加密数据。
byte[] originData = <your origin data to encrypt>;
EncryptRequest encryptRequest = new EncryptRequest();
encryptRequest.setKeyId(cipherKeyId);
encryptRequest.setPlaintext(originData);
EncryptResponse encryptResponse = client.encrypt(encryptRequest);
//加密数据。
byte[] cipherData = encryptResponse.getCiphertextBlob();
//Cipher初始向量，用于解密数据。
byte[] iv = encryptResponse.getIv();
//请求ID。
String requestId = encryptResponse.getRequestId();
```

- 使用专属KMS基础版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS基础版实例解密密钥的ID或别名（Alias）。
String cipherKeyId = "<your cipher key id>";
//待解密数据。
byte[] cipherData = <your cipher data to decrypt>;
//Cipher初始向量，必须与加密时一致。
byte[] iv = <IV value>;
DecryptRequest decryptRequest = new DecryptRequest();
        decryptRequest.setKeyId(cipherKeyId);
        decryptRequest.setCiphertextBlob(cipherData);
        decryptRequest.setIv(iv);
DecryptResponse decryptResponse = client.decrypt(decryptRequest);
//原始数据。
byte[] originData = decryptResponse.getPlaintext();
//请求ID。
String requestId = decryptResponse.getRequestId();
```

- 使用专属KMS基础版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS基础版实例签名密钥的ID或别名（Alias）。
String signerKeyId = "<the signer key id>";
//待签名数据。
byte[] message = <the data to sign>;
SignRequest signRequest = new SignRequest();
signRequest.setKeyId(signKeyId);
signRequest.setMessage(message);
signRequest.setMessageType("RAW");
SignResponse signResponse = client.sign(signRequest);
//签名值。
byte[] signature = signResponse.getSignature();
//请求ID。
String requestId = signResponse.getRequestId();
```

- 使用专属KMS基础版Client调用Verify接口验证数字签名
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS基础版实例签名密钥的ID或别名（Alias）。
String signerKeyId = "<the signer key id>";
//待验证签名的数据。
byte[] message = <the data to sign>;
VerifyRequest verifyRequest = new VerifyRequest();
verifyRequest.setKeyId(signerKeyId);
verifyRequest.setMessage(message);
verifyRequest.setMessageType("RAW");
verifyRequest.setSignature(signature);
VerifyResponse verifyResponse = client.verify(verifyRequest);
//验签结果。
boolean valid = verifyResponse.getValue();
//请求ID。
String requestId = verifyResponse.getRequestId();
```

2.5.2. 专属KMS PHP SDK

专属KMS SDK for PHP帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS基础版实例，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 您已经获取了专属KMS基础版实例VPC地址，并确保可以通过以下方式访问专属KMS基础版实例VPC地址：
  - 启用专属KMS基础版实例时设置了VPC ID，在该VPC中您可以访问专属KMS基础版实例VPC地址。
  - 您专属KMS SDK程序所在环境的网络，可以正常解析并访问专属KMS基础版实例VPC地址。

  具体操作，请参见查询专属KMS基础版实例。

安装SDK

- **方式一（推荐）：通过Composer来管理项目依赖**

  i. 在终端中切换到项目目录，直接执行以下代码安装AlibabaCloud DKMS-GCS SDK for PHP作为依赖项。

  ```
  composer require alibabacloud/dkms-gcs-sdk
  ```

  ii. 在composer.json中添加以下内容，声明对AlibabaCloud DKMS-GCS SDK for PHP的依赖。

  ```
  "require": {
    "alibabacloud/dkms-gcs-sdk": "^0.2.1"
    }
  ```

  iii. 在终端中切换到项目目录下，执行以下代码安装依赖。

  ```
  composer install
  ```

  iv. 使用Composer安装完成后，在PHP代码中引入依赖。

  ```
  require_once __DIR__ . '/vendor/autoload.php';
  ```

- **方式二：直接下载SDK源码**

  访问开源代码仓库下载SDK源码，在PHP代码中引入SDK目录下的autoload.php文件。

  ```
  require_once '/path/to/dkms-gcs-sdk/autoload.php';
  ```

初始化SDK

您可以初始化一个专属KMS基础版实例的PHP客户端，用于调用专属KMS基础版实例管理的密钥等资源。使用PHP SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。为保障生产环境通信安全，需要配置Java可信证书。
   设置 `RuntimeOptions` 的 `verify` 字段。示例代码如下：

   ```
   <?php
   use AlibabaCloud\Dkms\Gcs\OpenApi\Util\Models\RuntimeOptions;
   $runtimeOptions = new RuntimeOptions();
   $runtimeOptions->verify = 'path/to/caCert.pem';
   ...
   $encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
   ```

   开发环境可使用 `RuntimeOptions` 的 `ignoreSSL` 设置临时忽略可信证书的验证。示例代码如下：

   ```
   <?php
   use AlibabaCloud\Dkms\Gcs\OpenApi\Util\Models\RuntimeOptions;
   $runtimeOptions = new RuntimeOptions();
   $runtimeOptions->ignoreSSL = true;
   ...
   $encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
   ```

2. 创建专属KMS基础版Client。
   创建专属KMS基础版Client时，需要指定实例的Endpoint。EndPoint为专属KMS基础版实例VPC地址去掉HTTPS。关于专属KMS基础版实例VPC地址的更多信息，请参见查询专属KMS基础版实例。

```
<?php
use AlibabaCloud\Dkms\Gcs\Sdk\Client as AlibabaCloudDkmsGcsSdkClient;
use AlibabaCloud\Dkms\Gcs\OpenApi\Models\Config as AlibabaCloudDkmsGcsOpenApiConfig;
$config = new AlibabaCloudDkmsGcsOpenApiConfig();
//连接协议，固定为HTTPS。
$config->protocol = 'https';
//专属KMS基础版实例Client Key。
$config->clientKeyContent = '<your client key content>';
//专属KMS基础版实例Client Key解密口令。
$config->password = '<your client key password>';
//Endpoint，专属KMS基础版实例的服务地址去掉"HTTPS://"。
//示例：<service_id>.cryptoservice.kms.aliyuncs.com;
$config->endpoint = '<your dkms instance service address>';
$client = new AlibabaCloudDkmsGcsSdkClient($config);
```

代码示例

- 使用专属KMS基础版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\EncryptRequest;
$encryptRequest = new EncryptRequest();
//专属KMS基础版实例加密密钥的ID或别名（Alias）。
$encryptRequest->keyId = '<your cipher key id>';
//待加密数据。
$encryptRequest->plaintext = \AlibabaCloud\Dkms\Gcs\OpenApi\Util\Utils::toBytes('encrypt
plaintext');
$encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
//密文。
$ciphertextBlob = $encryptResponse->ciphertextBlob;
//Cipher初始向量，用于解密数据。
$iv = $encryptResponse->iv;
//请求ID。
$requestId = $encryptResponse->requestId;
```

- 使用专属KMS基础版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\DecryptRequest;
$decryptRequest = new DecryptRequest();
//专属KMS基础版实例解密密钥的ID或别名（Alias）。
$decryptRequest->keyId = '<your cipher key id>';
//待解密数据，加密返回的密文。
$decryptRequest->ciphertextBlob = null;
//Cipher初始向量，必须与加密时一致。
$decryptRequest->iv = null;
$decryptResponse = $client->decryptWithOptions($decryptRequest, $runtimeOptions);
//原始明文数据。
$plaintext = $decryptResponse->plaintext;
//请求ID。
$requestId = $decryptResponse->requestId;
```

- 使用专属KMS基础版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\SignRequest;
$signRequest = new SignRequest();
//专属KMS基础版实例签名密钥的ID或别名（Alias）。
$signRequest->keyId = 'your cipher key id';
//待签名数据。
$signRequest->message = <the data to sign>;
//待签名数据类型，RAW-原始数据，DIGEST-原始数据摘要。
$signRequest->messageType = 'RAW';
$signResponse = $client->signWithOptions($signRequest, $runtimeOptions);
//签名值。
$signature = $signResponse->signature;
//请求ID。
$requestId = $signResponse->requestId;
```

- 使用专属KMS基础版Client调用Verify接口验证数字签名
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\VerifyRequest;
$verifyRequest = new VerifyRequest();
//专属KMS基础版实例签名密钥的ID或别名（Alias）。
$verifyRequest->keyId = 'your cipher key id';
//待验证签名的数据。
$verifyRequest->message = <the data to sign>;
//待签名数据类型，RAW表示原始数据，DIGEST表示原始数据摘要。
$verifyRequest->messageType = 'RAW';
//待验证签名值。
$verifyRequest->signature = null;
$verifyResponse = $client->verifyWithOptions($verifyRequest, $runtimeOptions);
//验签结果。
$value = $verifyResponse->value;
//请求ID。
$requestId = $verifyResponse->requestId;
```

2.5.3. 专属KMS Go SDK

专属KMS SDK for Go帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS基础版实例，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 您已经获取了专属KMS基础版实例VPC地址，并确保可以通过以下方式访问专属KMS基础版实例VPC地址：
  - 启用专属KMS基础版实例时设置了VPC ID，在该VPC中您可以访问专属KMS基础版实例VPC地址。
  - 您专属KMS SDK程序所在环境的网络，可以正常解析并访问专属KMS基础版实例VPC地址。

  具体操作，请参见查询专属KMS基础版实例。

安装SDK

- 方式一：使用 `go mod` 管理您的依赖。
  在 `go mod` 中添加以下内容安装依赖包。

```
require (
    github.com/aliyun/alibabacloud-dkms-gcs-go-sdk v0.2.0
)
```

- 方式二：使用 `go get` 命令获取远程代码包。

```
$ go get -u github.com/aliyun/alibabacloud-dkms-gcs-go-sdk
```

初始化SDK

您可以初始化一个专属KMS基础版实例的Go客户端，用于调用专属KMS基础版实例管理的密钥等资源。使用Go SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。

为保障生产环境通信安全，需要配置可信证书。具体配置如下：
将CA证书内容设定为 `RuntimeOptions` 的verify字段，示例代码如下：

```
import (
    dedicatedkmsopenapiutil "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi-uti
l"
    "github.com/alibabacloud-go/tea/tea"
    "io/ioutil"
)
// 验证服务端证书
ca, err := ioutil.ReadFile("path/to/caCert.pem")
if err != nil {
    panic(err)
}
runtimeOptions := &dedicatedkmsopenapiutil.RuntimeOptions{
    Verify: tea.String(string(ca)),
}
...
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
```

开发环境可使用 `RuntimeOptions` 的ignoreSSL字段临时忽略可信证书的验证。示例代码如下：

```
import (
    dedicatedkmsopenapiutil "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi-uti
l"
    "github.com/alibabacloud-go/tea/tea"
)
// 忽略服务端证书
runtimeOptions := &dedicatedkmsopenapiutil.RuntimeOptions{
    IgnoreSSL: tea.Bool(true),
}
...
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
```

2. 创建专属KMS基础版Client。
创建专属KMS基础版Client时，需要指定基础版实例的Endpoint。EndPoint为专属KMS基础版实例服务地址去掉HTTPS。关于专属KMS基础版实例服务地址的更多信息，请参见查询专属KMS基础版实例。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
  "github.com/alibabacloud-go/tea/tea"
)
config := &dedicatedkmsopenapi.Config{
  // 连接协议，固定为HTTPS
  Protocol: tea.String("https"),
  // 专属KMS基础版实例Client Key
    ClientKeyContent: tea.String("<your client key content>"),
  // 专属KMS基础版实例Client Key解密口令
    Password: tea.String("<your client key password>"),
  // Endpoint，专属KMS基础版实例的服务地址去掉'HTTPS://'
    Endpoint: tea.String("<service_id>.cryptoservice.kms.aliyuncs.com"),
}
client, err := dedicatedkmssdk.NewClient(config)
```

示例

- 使用专属KMS基础版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 待加密数据。
plaintext := []byte("encrypt plaintext")
// 专属KMS基础版实例加密密钥的ID或别名（Alias）。
keyId := "<your cipher key id>"
encryptRequest := &dedicatedkmssdk.EncryptRequest{
    KeyId:     tea.String(keyId),
    Plaintext: plaintext,
}
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 密文。
cipher := encryptResponse.CiphertextBlob
// Cipher初始向量，用于解密数据。
iv := encryptResponse.Iv
// 请求ID。
requestId := tea.StringValue(encryptResponse.RequestId)
```

- 使用专属KMS基础版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS基础版实例解密密钥的ID或别名（Alias）。
keyId := "<your cipher key id>"
// 待解密数据，加密返回的密文。
ciphertextBlob := []byte("<your cipher data to decrypt>")
// Cipher初始向量，必须与加密时一致。
iv := []byte("<IV value>")
decryptRequest := &dedicatedkmssdk.DecryptRequest{
    KeyId:          tea.String(keyId),
    CiphertextBlob: ciphertextBlob,
    Iv:             iv,
}
decryptResponse, err := client.DecryptWithOptions(decryptRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 原始明文数据。
plaintext := decryptResponse.Plaintext
// 请求ID。
requestId := tea.StringValue(decryptResponse.RequestId)
```

- 使用专属KMS基础版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS实例签名密钥的ID或别名（Alias）。
signerKeyId := "<the signer key id>"
// 待签名数据。
message := []byte("<the data to sign>")
// 待签名数据类型，RAW-原始数据，DIGEST-摘要。
messageType := "RAW"
signRequest := &dedicatedkmssdk.SignRequest{
    KeyId:       tea.String(signerKeyId),
    Message:     message,
    MessageType: tea.String(messageType),
}
signResponse, err := client.SignWithOptions(signRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 签名值。
signature := signResponse.Signature
// 请求ID。
requestId := tea.StringValue(signResponse.RequestId)
```

- 使用专属KMS基础版Client调用Verify接口验证数字签名

详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS基础版实例签名密钥的ID或别名（Alias）。
signerKeyId := "<the signer key id>"
// 待验证签名的数据。
message := []byte("<the data to sign>")
// 待签名数据类型，RAW-原始数据，DIGEST-摘要。
messageType := "RAW"
// 待验证签名值。
signature := []byte("<the signature>")
verifyRequest := &dedicatedkmssdk.VerifyRequest{
    KeyId:       tea.String(signerKeyId),
    Message:     message,
    MessageType: tea.String(messageType),
    Signature:   signature,
}
verifyResponse, err := client.VerifyWithOptions(verifyRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 验签结果。
value := tea.BoolValue(verifyResponse.Value)
// 请求ID。
requestId := tea.StringValue(verifyResponse.RequestId)
```

2.5.4. 专属KMS Python SDK

专属KMS SDK for Python帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

- 如果您使用Python3访问专属KMS SDK，可以查看Python3开源代码仓库。
- 如果您使用Python2访问专属KMS SDK，可以查看Python2开源代码仓库。

同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS基础版实例，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 您已经获取了专属KMS基础版实例VPC地址，并确保可以通过以下方式访问专属KMS基础版实例VPC地址：
  - 启用专属KMS基础版实例时设置了VPC ID，在该VPC中您可以访问专属KMS基础版实例VPC地址。
  - 您专属KMS SDK程序所在环境的网络，可以正常解析并访问专属KMS基础版实例VPC地址。

  具体操作，请参见查询专属KMS基础版实例。

安装SDK

- 如果您通过Python3使用专属KMS SDK安装alibabacloud-dkms-gcs模块，安装命令如下：

```
pip install alibabacloud-dkms-gcs
```

- 如果您通过Python2使用专属KMS SDK安装alibabacloud-dkms-gcs-python2模块，安装命令如下：

```
pip install alibabacloud-dkms-gcs-python2
```

初始化SDK

您可以初始化一个专属KMS基础版实例的Python客户端，用于调用专属KMS基础版实例管理的密钥等资源。使用Python SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。
   为保障生产环境通信安全，需要配置可信证书。具体配置如下：
   将CA证书路径设定为 `RuntimeOptions` 的verify字段，示例代码如下：

```
# -*- coding: utf-8 -*-
from openapi_util.models import RuntimeOptions
runtime_options = RuntimeOptions()
# 忽略SSL验证
# runtime_options.ignore_ssl = True
# ca证书路径
runtime_options.verify = "<your-ca-certificate-file-path>"
...
response = client.encrypt_with_options(request, runtime_options)
```

2. 创建专属KMS基础版Client。
   创建专属KMS基础版Client时，需要指定实例的Endpoint。EndPoint为专属KMS基础版实例服务地址去掉HTTPS。关于专属KMS基础版实例服务地址的更多信息，请参见查询专属KMS基础版实例。

```
# -*- coding: utf-8 -*-
from openapi.models import Config
from sdk.client import Client
config = Config()
# 连接协议，固定为HTTPS
config.protocol = "https"
# 专属KMS基础版实例Client Key
config.client_key_content = "<your-client-key-content>"
# 专属KMS基础版实例Client Key解密口令
config.password = "<your-password>"
# Endpoint，专属KMS基础版实例服务地址去掉HTTPS
config.endpoint = "<your-endpoint>"
client = Client(config)
```

示例

- 使用专属KMS基础版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import EncryptRequest
request = EncryptRequest()
# 待加密数据
request.plaintext = "<your-plaintext>".encode("utf-8")
# 专属KMS基础版实例加密密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
encrypt_response = client.encrypt_with_options(request, runtime_options)
# 加密数据
ciphertext_blob = encrypt_response.ciphertext_blob
# cipher初始向量，用于解密数据
iv = encrypt_response.iv
# 请求ID
request_id = encrypt_response.request_id
```

- 使用专属KMS基础版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import DecryptRequest
request = DecryptRequest()
# 待解密数据
request.ciphertext_blob = "<your-ciphertext-blob>"
# 专属KMS基础版实例解密密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
# Cipher初始向量，必须与加密时一致
request.iv = "<your-iv>"
decrypt_response = client.decrypt_with_options(request, runtime_options)
// 原始明文数据
plaintext = decrypt_response.plaintext;
// 请求ID
request_id = decrypt_response.request_id;
```

- 使用专属KMS基础版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import SignRequest
request = SignRequest()
# 专属KMS基础版实例签名密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
# 待签名数据
request.message = "<your_raw_message>"
# RAW-原始消息，DIGEST-摘要
request.message_type = "<RAW>"
# 签名算法
request.algorithm = "<your-algorithm>"
sign_response = client.sign_with_options(request, runtime_options)
# 签名值
signature = sign_response.signature
# 请求ID
request_id = sign_response.request_id
```

- 使用专属KMS基础版Client调用Verify接口验证数字签名

详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import SignRequest
request = VerifyRequest()
# 专属KMS基础版实例签名密钥的ID或别名（Alias）。
request.key_id = "<your-key-id>"
# 待验证签名的数据。
request.message = "<your_raw_message>"
# RAW-原始消息，DIGEST-摘要
request.message_type = "RAW"
# 签名算法
request.algorithm = "<your-algorithm>"
# 签名值
request.signature = "<your-signature>"
verify_response = client.verify_with_options(request, runtime_options)
# 验签结果
valid = verify_response.valid
# 请求ID
request_id = verify_response.request_id
```

3.专属KMS标准版

3.1. 概述

专属KMS标准版具备用户侧数据加密能力和加密服务密码机集群加密能力，将密钥存储在您的独享密码机集群内，同时支持云产品原生的数据加密。

应用场景

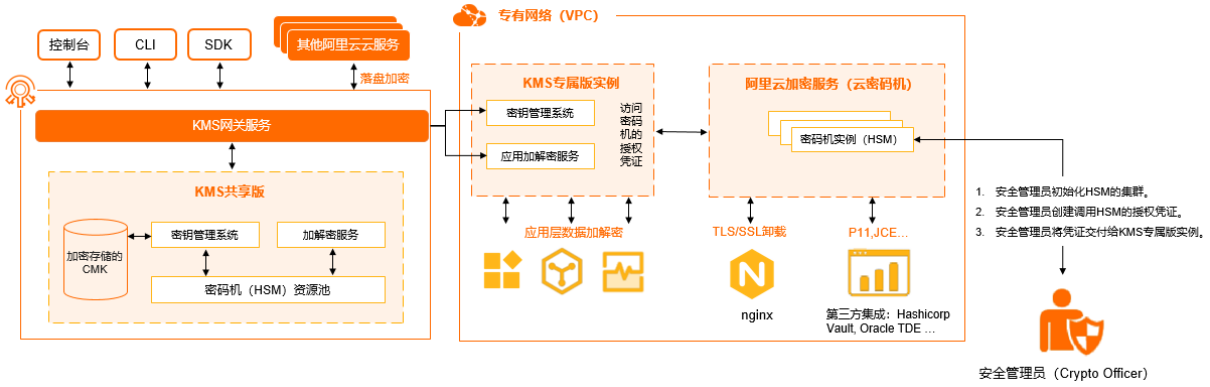
- 自建应用集成
  自建应用程序可以通过VPC网络使用专属KMS标准版实例的加解密服务进行应用层加解密。
- 第三方ISV应用集成
  第三方ISV（Independent Software Vendors）应用程序可以使用专属KMS标准版实例的密码计算接口。
- 云产品集成
  您可以授权KMS共享版转发云产品的服务端加密请求到专属KMS标准版实例。

产品优势

- 专属KMS标准版提供租户独享的服务实例，并部署到租户的VPC内，满足私有网络接入需求。
- 专属KMS标准版使用租户独享的密码资源池（HSM集群），实现资源隔离和密码学隔离，以获得更高的安全性。
- 专属KMS标准版可以降低使用HSM的复杂度，为您的HSM提供稳定、易用的上层密钥管理途径和密码计算服务。
- 专属KMS标准版可以将您的HSM与云服务无缝集成，为云服务加密提供更高的安全性和可控制性。更多信息，请参见支持服务端集成加密的云服务。

产品架构

专属KMS标准版实例是一个独立部署的实例型服务，产品架构如下图所示。



专属KMS主要组成部分如下：

- 密码资源池
  您在加密服务CloudHSM中管理的、租户独享的HSM集群。它是用于密钥存储和计算的安全设备。
  关于加密服务CloudHSM的更多信息，请参见什么是加密服务。
- 密钥管理系统
  在您自定义的独享HSM集群内，进行密钥的生命周期管理。
- 密码计算服务
  专属KMS标准版实例通过简单易用的API调度密码计算。密码计算过程中密钥不会离开HSM的安全边界。

支持的地域

支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。

计费

专属KMS标准版采用预付费（包年包月）的计费方式。更多信息，请参见专属KMS计费说明。

3.2. 快速入门

专属KMS使用租户独享的密码资源池（密码机集群），实现资源隔离和密码学隔离，以获得更高的安全性。本文介绍如何快速使用专属KMS标准版实例。

前提条件

专属KMS标准版实例需要关联同一阿里云账号下加密服务CloudHSM的密码机集群。请确保加密服务CloudHSM已经完成以下设置：

1. 已经购买两个不同可用区的GVSM密码机实例。具体操作，请参见购买加密服务。
2. 已经创建并激活密码机集群，集群中已经添加两个不同可用区的GVSM密码机实例。具体操作，请参见使用密码机实例集群。

为密码机GVSM配置双向TLS认证前，您需要提交工单，获取证书文件生成工具。

操作流程

步骤一：购买专属KMS标准版实例

1. 登录密钥管理服务控制台。
2. 在左侧导航栏，单击**专属KMS**。
3. 单击**购买专属KMS**。
4. 在购买页面，设置相关参数。

   > **说明** **定制实例**和**密钥数量**由系统默认填入，无需手动设置。

   - **版本**：选择**标准版**。
   - **地域**：根据需要选择专属KMS实例所在地域。
   - **专有网络数量**：根据需要选择专有网络VPC（Virtual Private Cloud）的数量。同一地域内网络互通的多个VPC支持使用同一个专属KMS实例。

     > **说明** 如果多个VPC之间网络未连通，您可以根据业务需要，选择如下方式之一：使用云企业网实现同地域VPC互通、使用高速通道实现VPC互通或使用VPN网关实现VPC互通。

   - **购买数量**：根据需要选择实例数量。最多支持同时购买20个实例。
   - **购买时长**：根据需要选择购买时长。

> **说明**
>
> - 建议您为专属KMS标准版实例和密码机实例设置相同的购买时长，并为专属KMS标准版实例设置自动续费，以确保专属KMS标准版实例正常运行。
> - 您也可以选中**到期自动续费**，当前实例到期后将自动续费。

5. 确认总配置费用后，单击**立即购买**并完成**支付**。

步骤二：为密码机GVSM配置双向TLS认证

为了确保安全性，专属KMS标准版实例连接密码机GVSM时采用TLS双向认证，需进行双向TLS认证配置。本文以Windows系统的ECS实例为例。

1. 登录ECS实例。具体操作，请参见通过密码或密钥认证登录Windows实例。
2. 运行证书文件生成工具，生成CA证书、客户端证书和服务端证书。
   - Windows系统：双击鼠标左键运行windows文件夹中的hsm_certificate_generate.exe文件，在certs文件夹获取证书文件。
   - Linux系统和macOS系统：切换到Linux或darwin文件夹所在路径，然后运行以下命令。

     ```
     ./hsm_certificate_generate -dir ./certs -pswd 12345678
     -dir 用于指定生成证书文件的保存路径。
     -pswd 用于指定生成的PKCS格式证书口令。
     ```

     命令执行成功后，会在certs文件夹生成以下证书文件：

     ```
     certs/
     ├── client.p12 (客户端PKCS12格式证书，包含客户端证书以及私钥)
     ├── client.pem (PEM格式客户端证书)
     ├── rootca.key (CA私钥)
     ├── rootca.pem (PEM格式CA证书)
     └── server.p12 (服务端PKCS12格式证书)
     ```

3. 使用服务器密码机管理工具连接主密码机GVSM。
   i. 提交工单，获取服务器密码机管理工具。
   ii. 双击 `PKIManager-1.1.2.9.exe` 打开密码机管理工具。
   iii. 选择**系统** > **TCP/IP连接**。
   iv. 在**TCP/IP连接**对话框，输入密码机GVSM**IP地址**（创建主密码机时生成的IP地址）和**端口号**（固定为8020）。
   v. 单击**连接**。
   vi. 单击**登录**。
4. 使用服务器密码机管理工具配置客户端证书（ `client.pem` ）。
   i. 选择**密钥管理** > **客户端管理**。
   ii. 在**客户端证书管理**对话框，单击**导入证书**。
   iii. 在**导入证书**对话框，选择证书文件生成工具生成的客户端证书（ `client.pem` ），然后单击**导入**。
5. 配置服务端证书（ `server.p12` ）。
   i. 选择**密钥管理** > **服务端管理**。

ii. 在**服务端p12证书管理**对话框，单击**导入证书**。

iii. 在**导入证书**对话框，选择证书文件生成工具生成的服务端证书（`server.p12`），然后输入**证书口令**（默认为12345678）。

iv. 单击**导入**。

v. 单击**立即重启**，然后单击**确定**。

6. 配置主密码机GVSM设备端口属性。

i. 选择**设备管理 > 主机端口属性**。

ii. 在**主机端口属性**对话框，配置以下参数：

- **Socket KeepAlive时间**：60。
- **消息报文头长度**：0。
- **消息报文编码格式**：ASCII。
- **主机服务通讯方式**：双向通讯。
- **主机服务通讯协议**：TLSv1.2。

iii. 单击**重置**。

iv. 单击**立即重启**，然后单击**确定**。

7. 登录加密服务管理控制台，在**集群详情**页面的**密码机实例列表**区域单击**一键同步集群**，将主密码机GVSM中的数据同步到密码机集群中的其他密码机。具体操作，请参见同步集群中密码机实例的数据。

步骤三：启用专属KMS标准版实例

1. 在**专属KMS**页面，找到目标专属KMS标准版实例，在**操作**列单击**启用**。
2. 在**连接密码机**对话框，指定密码机集群。

> **说明** 一个密码机集群只能绑定一个专属KMS标准版实例。

3. 配置访问凭据。
   - **客户端保护口令**：客户端证书保护口令，默认为12345678。
   - **客户端证书**：经过Base64编码的PKCS12格式证书。请获取步骤二：为密码机GVSM配置双向TLS认证中的`client.p12`文件，进行Base64编码后输入文本框。
   - **安全域证书**：PEM格式CA证书。请将步骤二：为密码机GVSM配置双向TLS认证中的`rootca.pem`文件内容输入文本框。
4. 单击**连接密码机**。

   请等待几分钟，然后刷新页面，当状态变更为**已启用**时，专属KMS标准版实例启用成功。
5. 如果您需要设置多个VPC使用同一专属KMS实例，单击目标实例**专有网络**列下的+，配置专有网络。

   在**待选专有网络**中选中需要设置的专有网络，单击<后，完成**确定**。

> **说明**
> - 您可以单击目标实例**操作**列的**详情**，在**服务规格**中会显示当前实例支持关联的VPC数量。
> - 如果选择关联的VPC数量超过了关联资源限制，您可以根据控制台上的提示，单击**前往购买**升级**专有网络数量**的规格。

步骤四：为专属KMS标准版实例创建密钥

1. 在**专属KMS**页面，找到目标专属KMS标准版实例，在**操作**列单击**管理**。
2. 在**用户主密钥**区域，单击**创建密钥**。
3. 在**创建密钥**对话框，设置以下参数。

参数名称	说明
密钥类型	取值： ○ 对称密钥的类型： ■ Aliyun_AES_256 ■ Aliyun_AES_128 ■ Aliyun_AES_192 ■ Aliyun_SM4 ○ 非对称密钥的类型： ■ RSA_2048 ■ RSA_3072 ■ RSA_4096 ■ EC_SM2 ■ EC_P256 ■ EC_P256K
密钥用途	取值： ○ Encrypt/Decrypt：数据加密和解密。 ○ Sign/Verify：产生和验证数字签名。
别名	用户主密钥的标识符。支持英文字母、数字、下划线（_）、短划线（-）和正斜线（/）。
描述	密钥的说明信息。

4. （可选）单击**高级选项**，选择**密钥材料来源**和**附加密钥用途**。
   - **密钥材料来源**
     - **阿里云KMS**：密钥材料将由专属KMS生成。
     - **外部**：专属KMS将不会生成密钥材料，您需要将自己的密钥材料导入专属KMS。更多信息，请参见导入对称密钥材料。

       > **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**。

   - **附加密钥用途**：当密钥类型为非对称密钥时，支持设置附加密钥用途，让一个密钥具有多个用途。
5. 单击**确定**。

步骤五：应用接入专属KMS标准版实例

1. 创建应用接入点AAP（Application Access Point），控制应用正常访问专属KMS标准版实例。

i. 在**专属KMS**页面，找到目标专属KMS标准版实例，在**操作**列单击**详情**。

ii. 在**应用接入指南**区域，单击**快速创建应用接入点**。

iii. 在**快速配置应用身份凭证和权限**面板，设置应用接入点信息。

a. 输入**应用接入点名称**。

b. 设置**访问控制策略**。

- **允许访问资源**：默认填写 `Key/*` ，表示允许访问当前专属KMS实例的全部密钥。
- **允许网络来源**：允许访问的网络类型和IP地址。您可以设置私网IP地址或者私网网段，多个IP地址之间用半角逗号（,）分隔。

c. 单击**创建**。

iv. 在**应用身份凭证**对话框，获取**凭证口令**和**应用身份凭证内容**（Client Key）。

- **凭证口令**：单击**复制口令**，获取凭证口令。
- **应用身份凭证内容**：单击**下载应用身份凭证**，保存应用身份凭证信息。
  应用身份凭证信息包含凭证ID（KeyId）和凭证内容（PrivateKeyData），凭证内容为PKCS12格式Base64编码。示例如下：

```
{
  "KeyId": "KAAP.71be72c8-73b9-44e0-bb75-81ee51b4****",
  "PrivateKeyData": "MIIJwwIBAz****ICNXX/pOw=="
}
```

> **说明** KMS不会保存Client Key的凭证口令和应用身份凭证内容，您只能在创建Client Key时获取到该信息，请妥善保管。

v. 单击**关闭**。

2. 获取CA证书，以便验证专属KMS标准版实例。

在**应用接入指南**区域，单击**获取实例CA证书**下方的**下载**，下载.pem格式的CA证书文件。

后续步骤

您可以使用专属KMS SDK，通过访问专属KMS API调用密钥管理服务。具体操作，请参见专属KMS Java SDK、专属KMS PHP SDK、专属KMS Go SDK和专属KMS Python SDK。

3.3. 服务关联角色

本文为您介绍专属KMS标准版服务关联角色（AliyunServiceRoleForKMSKeyStore）的应用场景、权限策略、创建及删除操作。

应用场景

创建和使用专属KMS标准版实例时，密钥管理服务KMS需要通过服务关联角色访问加密服务CloudHSM的密码机集群。

关于服务关联角色的更多信息，请参见服务关联角色。

权限说明

角色名称：AliyunServiceRoleForKMSKeyStore。

权限策略：AliyunServiceRolePolicyForKMSKeyStore。

权限说明：KMS使用此角色访问加密服务CloudHSM的密码机集群、ECS和VPC等其他云服务相关资源。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ecs:CreateNetworkInterfacePermission",
        "ecs:DeleteNetworkInterfacePermission",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:DescribeSecurityGroups",
        "ecs:CreateSecurityGroup",
        "ecs:DeleteSecurityGroup",
        "ecs:AuthorizeSecurityGroup",
        "ecs:AuthorizeSecurityGroupEgress",
        "ecs:RevokeSecurityGroup",
        "ecs:RevokeSecurityGroupEgress",
        "ecs:DescribeSecurityGroupAttribute"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "vpc:DescribeVSwitches",
        "vpc:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "yundun-hsm:DescribeInstances",
        "yundun-hsm:DescribeClusters"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "keystore.kms.aliyuncs.com"
        }
      }
    }
  ]
}
```

创建服务关联角色

当您使用阿里云账号在密钥管理控制台首次创建专属KMS标准版实例时，专属KMS会自动创建服务关联角色（AliyunServiceRoleForKMSKeyStore）。

当您使用的是RAM用户，需要先授权以下自定义策略，才能在密钥管理控制台首次创建专属KMS标准版实例时，自动创建服务关联角色（AliyunServiceRoleForKMSKeyStore）。具体操作，请参见为RAM用户授权。

```
{
    "Action": "ram:CreateServiceLinkedRole",
    "Resource": "*",
    "Effect": "Allow",
    "Condition": {
        "StringEquals": {
            "ram:ServiceName": "keystore.kms.aliyuncs.com"
        }
     }
}
```

删除服务关联角色

删除服务关联角色前，您需要确保当前阿里云账号下的专属KMS标准版实例已经被释放。当专属KMS标准版实例费用到期且没有续费时，KMS会自动释放该专属KMS标准版实例。

您可以在RAM控制台删除服务关联角色。具体操作，请参见删除RAM角色。

3.4. 基础操作

3.4.1. 管理专属KMS标准版实例

当专属KMS标准版实例处于**已启用**状态时，您可以根据需要查询专属KMS标准版实例、断开或者重新建立专属KMS标准版实例与密码机的连接、配置多VPC使用同一专属KMS标准版实例、启用安全审计功能。

查询专属KMS标准版实例

您可以根据需要，查询专属KMS标准版实例ID、实例VPC地址、专有网络、加密服务集群等信息。

实例VPC地址是每个专属KMS标准版实例自己专属的访问入口，格式为 `https://{实例ID}.cryptoservice.kms.aliyuncs.com` 。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 找到目标专属KMS标准版实例，在**操作**列下单击**详情**。

断开专属KMS标准版实例与密码机的连接

如果您需要解除专属KMS标准版实例和密码机集群的绑定关系，可以断开专属KMS标准版实例与密码机集群的连接。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。

3. 在左侧导航栏，单击**专属KMS**。
4. 找到目标专属KMS标准版实例，在**操作**列下单击**详情**。
5. 单击**专属密码机集群**右侧的**断开**。
6. 在**断开**对话框，单击**断开**。

   状态变更为**未启用**，表示断开连接成功。

重新连接专属KMS标准版实例与密码机

假设您已经配置了专属KMS标准版实例并连接到密码机，如果手动断开连接后需要重新连接，无需指定密码机集群，只需配置访问凭据并单击**连接密码机**。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**启用**。
5. 在**连接密码机**对话框，配置**访问凭据**，然后单击**连接密码机**。

   几分钟后，状态从**正在创建连接**变更为**已启用**，表示重新连接成功。

配置多VPC使用同一专属KMS标准版实例

如果您需要在同一地域的多个VPC访问专属KMS标准版实例，需要执行本操作。执行操作前请确保多个VPC之间网络已连通。如果多个VPC之间网络未连通，您可以根据业务需要，选择如下方式之一：使用云企业网实现同地域VPC互通、使用高速通道实现VPC互通或使用VPN网关实现VPC互通。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**专有网络**列的图标。
5. 在**配置专有网络**对话框，选中**待选专有网络**中需要设置的专有网络，单击后，完成**确定**。

> **说明**
>
> - 您可以单击目标实例**操作**列的**详情**，在**服务规格**中会显示当前实例支持关联的VPC数量。
> - 如果选择关联的VPC数量超过了关联资源限制，您可以根据控制台上的提示，单击**前往购买**升级**专有网络数量**的规格。

启用安全审计

您在访问专属KMS标准版实例过程中会产生审计日志。审计日志中记录了实例的访问数据，包括调用实例时的请求信息、用户信息、被访问资源信息，以及访问结果等。日志文件示例如下：

```
2021-10-19T212021-10-19T21:40:01     [INFO]  - - 3dd60a7a-4587-4c57-8197-d749c3578974 Creat
eKey - TMP.3KfAHseF5DVULM2s8YUhdB8YvwM4nZA1wXr8AcAAhR7YhdyosXG2eSpsRFPMjYbvUArPRtsCWKzxEo88
bC5w5LBfyp**** 111760096384**** 111760096384**** - kst-phzz6108e50c15333w**** - 37 - -40:01
[INFO]  - - 3dd60a7a-4587-4c57-8197-d749c3578974 CreateKey - TMP.3KfAHseF5DVULM2s8YUhdB8Yvw
M4nZA1wXr8AcAAhR7YhdyosXG2eSpsRFPMjYbvUArPRtsCWKzxEo88bC5w5LBfyp**** 111760096384**** 11176
0096384**** - kst-phzz6108e50c15333w**** - 37 - -
```

当您启用安全审计后，专属KMS会将审计日志以小时为单位投递到您指定的OSS存储空间，以应对监管要求和业务需求。审计启用安全审计前请确保您已经创建OSS存储空间。具体操作，请参见创建存储空间。

> **说明** 启用安全审计后，将在1小时内生成并投递审计日志。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**详情**。
5. 在**专属KMS实例详情**面板，打开**安全审计**开关。
6. 在**配置安全审计**对话框，选择**日志存储位置**。
7. 单击**确定**。

   成功启用安全审计后，安全审计状态由**未开启**变为**已开启**。您也可以按需修改安全审计配置或者禁用安全审计。

3.4.2. 管理密钥

当您为专属KMS标准版实例创建密钥后，可以根据实际需求禁用密钥、开启删除保护或计划删除密钥。

禁用密钥

密钥创建完成后，默认为启用状态。您可以禁用密钥，被禁用的密钥无法用于加密解密、签名验签。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，单击目标密钥**操作**列的**禁用**。
6. 在**禁用密钥**对话框，单击**确定**。

   成功禁用密钥后，密钥状态由**启用中**变为**已禁用**。您也可以单击**启用**，再次启用密钥。

开启删除保护

为密钥开启删除保护后，将无法通过控制台或API删除该密钥，从而避免误删除密钥。

> **说明** **待删除**状态的密钥无法开启删除保护。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，单击目标密钥名称。
6. 在**密钥详情**区域，单击**开启删除保护**。
7. 在**确认开启**对话框，单击**确定**。

   成功开启删除保护后，删除保护状态由**未开启**变为**已开启**。您也可以单击**关闭删除保护**，为密钥关闭删除保护，以便删除该密钥。

计划删除密钥

密钥删除后将无法恢复，使用该密钥加密的内容及产生的数据密钥也将无法解密。因此，KMS只提供计划删除密钥的方式，而不提供直接删除的方式。如果需要删除密钥，推荐您使用禁用密钥功能。

> **说明** 请确保密钥已经关闭删除保护。

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，选择专属KMS标准版实例所在的地域。

   支持专属KMS标准版的地域为：华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国香港、马来西亚（吉隆坡）和新加坡。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，在目标密钥**操作**列选择 > **计划删除密钥**。

6. 在**计划删除密钥**对话框，填写预删除周期。系统会在预删除周期后删除密钥。

   预删除周期取值为7~366天，默认值为366天。
7. 单击**确定**。

   成功计划删除密钥后，密钥状态变为**待删除**。处于**待删除**状态的密钥无法用于加密解密、签名验签。您也可以选择 > **取消删除密钥**撤销删除密钥的申请。

3.4.3. 导入对称密钥材料

当您为专属KMS标准版实例创建对称密钥材料来源为外部的密钥时，专属KMS不会为您创建的用户主密钥（CMK）在密码机集群生成对称密钥材料，此时您可以将自己的对称密钥材料导入到CMK中。本文为您介绍如何导入外部对称密钥材料。

背景信息

您可以调用DescribeKey接口判断对称密钥材料来源。当Origin为*EXTERNAL*时，说明对称密钥材料由外部导入，称为**外部密钥**。

当您选择对称密钥材料来源为外部，使用您自己导入的对称密钥材料时，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对称密钥材料。

- 由于对称密钥材料将导入到您的密码机集群中，且暂不支持调用DeleteKeyMaterial接口删除对称密钥材料。因此您需要调用ScheduleKeyDeletion接口，等待您指定的计划删除时间后，密码机集群中的对称密钥材料随着CMK一起被删除。
- 每个CMK只能拥有一个对称密钥材料。一个对称密钥材料导入CMK后，CMK将与该对称密钥材料绑定，后续将无法导入其他对称密钥材料。
- 支持导入128位、192位或256位对称密钥作为对称密钥材料。

步骤一：创建外部密钥

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，选择专属KMS标准版实例所在的地域。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，单击**创建密钥**。
6. 在**创建密钥**对话框，选择**密钥类型**。

   支持的密钥类型为和Aliyun_SM4、Aliyun_AES_128、Aliyun_AES_192、Aliyun_AES_256。
7. 设置**别名**和**描述**。
8. 单击**高级选项**，选择**密钥材料来源**为**外部**。
9. 勾选**我了解使用外部密钥材料的方法和意义**，然后单击**确定**。

步骤二：获取导入对称密钥材料参数

导入对称密钥材料参数包括一个用于加密对称密钥材料的公钥，以及一个导入令牌。

1. 在**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。
2. 在**密钥材料**区域，单击**获取导入参数**。
3. 在**获取导入密钥材料的参数**对话框，选择**公钥类型**、**加密算法**，然后单击**下一步**。

   > **说明**
   > - 当**公钥类型**取值为RSA_2048时，您可以选择加密算法RSAES_PKCS1_V1_5（默认值）或RSAES_OAEP_SHA_256，本文以RSAES_PKCS1_V1_5为例为您介绍。
   > - 当**公钥类型**取值为EC_SM2时，您可以选择加密算法SM2PKE。

4. 下载加密公钥和**导入令牌**，然后单击**关闭**。

步骤三：加密对称密钥材料

以通过OPENSSL加密一个2048比特的RSA公钥为例：使用的加密算法需要与获取导入对称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。

1. 创建一个对称密钥材料，使用OPENSSL产生一个32字节的随机数进行演示。
2. 根据指定的加密算法（以RSAES_PKCS1_V1_5为例）加密对称密钥材料。
3. 将加密后的对称密钥材料进行Base64编码，保存为文本文件。

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

步骤四：导入对称密钥材料

导入令牌与加密对称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对称密钥材料。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。

1. 在**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。
2. 在**对称密钥材料**区域，单击**导入密钥材料**。
3. 在**导入打包后的密钥材料**对话框，上传**打包后的密钥材料**和**导入令牌**，并单击**确定**。
   - **打包后的密钥材料**：上传步骤三：加密对称密钥材料生成的对称密钥材料文本文件。
   - **导入令牌**：上传步骤二：获取导入对称密钥材料参数获取的导入令牌文本文件。

执行结果

导入对称密钥材料成功后，密钥状态从**待导入**更新为**启用中**。

3.4.4. 导入非对称密钥材料

您可以从源环境（通常为线下的密钥管理设施KMI，或者线下的硬件安全模块HSM）将非对称密钥导入专属KMS标准版实例。本文为你介绍如何在源环境生成非对称密钥材料，随后上传到专属KMS标准版实例以便导入非对称密钥材料。

背景信息

您可以调用DescribeKey接口判断非对称密钥材料来源。当Origin为*EXTERNAL*时，说明非对称密钥材料由外部导入，称为**外部密钥**。

当您创建外部密钥，使用您自己导入的非对称密钥材料时，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非对称密钥材料。
- 由于非对称密钥材料将导入到您的密码机集群中，且暂不支持调用DeleteKeyMaterial接口删除非对称密钥材料。因此您需要调用ScheduleKeyDeletion接口，等待7天到30天后，密码机集群中的非对称密钥材料随着CMK一起被删除。
- 每个CMK只能拥有一个非对称密钥材料。一个非对称密钥材料导入CMK后，CMK将与该非对称密钥材料绑定，后续将无法导入其他非对称密钥材料。

导入非对称密钥材料时需使用密钥如下表所示。

密钥	描述	提供者	标记说明
TAK（Target Asymmetric Key）	待导入的目标非对称密钥。	源环境的系统或者工具（如线下的密钥管理设施或者HSM）。	• TAKpub：公钥部分。 • TAKpriv：私钥部分。
IWK（Import Wrapping Key）	用于导入TAK的加密公钥。	阿里云专属KMS标准版实例。	• IWKpub：公钥部分。 • IWKpriv：私钥部分。
ESK（Ephemeral Symmetric Key）	一个瞬时存在的对称密钥，用于直接加密TAKpriv。	源环境的系统或者工具，在完成对TAK的导出操作后立即销毁。	ESK：对称算法的瞬时密钥。

创建外部密钥

导入非对称密钥材料前，您需要先在专属KMS中创建一个密钥材料来源为外部的非对称密钥。

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，选择专属KMS标准版实例所在的地域。

3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS标准版实例**操作**列的**管理**。
5. 在**用户主密钥**区域，单击**创建密钥**。
6. 在**创建密钥**对话框，选择**密钥类型**。

   支持的非对称密钥类型为：RSA_2048、RSA_3072、RSA_4096、EC_P256、EC_P256K和EC_SM2。
7. 设置**别名**和**描述**。
8. 单击**高级选项**，选择**密钥材料来源**为**外部**。
9. 勾选**我了解使用外部密钥材料的方法和意义**，然后单击**确定**。

创建外部密钥成功后，您可以根据密钥类型选择导入SM2、RSA或ECC密钥材料。

导入SM2密钥材料

当您创建了密钥类型为EC_SM2的外部密钥后，可通过以下方式导入SM2密钥材料。

1. 从专属KMS获取导入非对称密钥材料的参数。

   导入非对称密钥材料的参数包括一个用于加密非对称密钥材料的公钥（IWKpub），以及一个导入令牌。

   i. 在**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。
   ii. 在**密钥材料**区域，单击**获取导入参数**。
   iii. 在**获取导入密钥材料的参数**对话框，选择**公钥类型**为**EC_SM2**、**加密算法**为**SM2PKE_SM4_ECB**，然后单击**下一步**。
   iv. 下载加密公钥和**导入令牌**，然后单击**关闭**。
2. 在源环境处理待导入的非对称密钥材料。

   步骤概览：

   i. 生成一个SM4类型的瞬时密钥ESK。
   ii. 使用步骤1中获取的加密公钥（IWKpub）加密ESK，得到瞬时对称密钥密文Cipher(ESK)，加密算法需要采用SM2公钥加密。
   iii. 使用ESK加密导出TAKpriv的椭圆曲线参数D（32字节），获取目标非对称密钥的私钥密文Cipher(TAKpriv)。其中加密模式为ECB，填充模式为NoPadding。
   iv. 组装结果数据，获取加密密钥材料，组装格式为Cipher(ESK) || Cipher(TAKpriv)。

   假设您在源环境中使用GMSSL产生待导入的非对称密钥材料，本示例对上述步骤概览进行详细讲解。

i. 创建一个SM2类型的目标非对称密钥私钥，获取私钥的椭圆曲线的参数D。

```
#生成SM2私钥。
gmssl ecparam -name sm2p256v1 -genkey -outform DER -out TakSm2Priv.bin
#查看私钥的D值。
gmssl asn1parse -inform DER -in TakSm2Priv.bin
#执行结果示例。
0:d=0  hl=2 l=   8 prim: OBJECT            :sm2p256v1
  10:d=0  hl=2 l= 119 cons: SEQUENCE
  12:d=1  hl=2 l=   1 prim: INTEGER           :01
  15:d=1  hl=2 l=  32 prim: OCTET STRING    [HEXDUMP]:CB55E5ECC20BF7E9249ADEDE990E
F141D14252E024734EB058A6B9F103F12A04
  49:d=1  hl=2 l=  10 cons: cont [ 0 ]
  51:d=2  hl=2 l=   8 prim: OBJECT            :sm2p256v1
  61:d=1  hl=2 l=  68 cons: cont [ 1 ]
  63:d=2  hl=2 l=  66 prim: BIT STRING
#将D值写入文件。
echo 'CB55E5ECC20BF7E9249ADEDE990EF141D14252E024734EB058A6B9F103F12A04' | xxd -r -p
> TakSm2D.bin
```

ii. 创建一个SM4类型的瞬时对称密钥ESK。

```
gmssl rand -out EskSm4.bin 16
```

iii. 使用步骤中获取的加密公钥加密ESK，得到瞬时对称密钥密文Cipher(ESK)，加密算法需要采用SM2公钥加密。

```
gmssl sm2utl -encrypt -in EskSm4.bin -pubin -inkey PublicKey.pem -out CipherEsk.bin
```

iv. 使用ESK加密私钥的椭圆曲线参数D，获取目标非对称密钥密文的私钥Cipher(TAKpriv)。其中加密模式为ECB，填充模式为NoPadding。

```
xxd -l 16  -c 16 -ps EskSm4.bin | xargs -I {} openssl enc -sms4-ecb -e  -K {} -in
TakSm2D.bin -nosalt -nopad -out CipherTakPriv.bin
```

v. 组装结果数据，获取加密密钥材料，组装格式为Cipher(ESK) || Cipher(TAKpriv)。

```
cat CipherEsk.bin CipherTakPriv.bin > EncryptedKeyMaterial.bin
```

3. 导入对称密钥材料。

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**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。

ii. 在**对称密钥材料**区域，单击**导入密钥材料**。

iii. 在**导入打包后的密钥材料**对话框，上传**打包后的密钥材料**和**导入令牌**。

- **打包后的密钥材料**：上传步骤2生成的对称密钥材料文本文件。
- **导入令牌**：上传步骤1获取的导入令牌文本文件。

iv. 单击**确定**。

导入对称密钥材料成功后，密钥状态从**待导入**更新为**启用中**。

导入RSA或ECC密钥材料

当您创建了密钥类型为RSA_2048、RSA_3072、RSA_4096、EC_P256或者EC_P256K的外部密钥后，可以导入RSA或ECC密钥材料，以下以RSA_2048为例为您进行介绍。

1. 从专属KMS获取导入非对称密钥材料的参数。

   导入非对称密钥材料的参数包括一个用于加密非对称密钥材料的公钥（IWKpub），以及一个导入令牌。

   i. 在**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。

   ii. 在**密钥材料**区域，单击**获取导入参数**。

   iii. 在**获取导入密钥材料的参数**对话框，选择**公钥类型**为**RSA_2048**、**加密算法**为**RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_PAD**，然后单击**下一步**。

   iv. 下载加密公钥和**导入令牌**，然后单击**关闭**。

2. 在源环境处理待导入的非对称密钥材料。

   步骤概览：

   i. 在源环境中，生成一个AES_256类型的瞬时对称密钥ESK。

   ii. 使用步骤1中获取的加密公钥（IWKpub）加密导出ESK，得到瞬时对称密钥密文Cipher(ESK)。加密时采用RSAES_OAEP标准加密，其中MGF1和哈希算法为SHA256。

   iii. 使用ESK加密导出TAKpriv，获取目标非对称密钥的私钥密文Cipher(TAKpriv)。加密时采用ECB模式进行加密，填充模式采用PKCS#7 Padding。

   > **说明** TAKpriv格式需要遵循：RSA私钥根据RFC3447进行编码，ECC私钥根据RFC5915进行编码。随后根据RFC5208包装为PKCS#8格式。

   iv. 组装结果数据，获取加密密钥材料，组装格式为Cipher(ESK) || Cipher(TAKpriv)。

   假设您在源环境中使用OPENSSL产生待导入的非对称密钥材料，本示例对上述步骤概览进行详细讲解。

   i. 创建一个AES_256类型的瞬时对称密钥ESK。

   ```
   openssl rand -out EskAes256.bin 32
   ```

   ii. 创建一个RSA_2048类型的目标非对称密钥私钥，并且将私钥转为PKCS#8格式。

   ```
   openssl genrsa -out TakPrivPkcs1.pem 2048
   openssl pkcs8 -topk8 -inform PEM -in TakPrivPkcs1.pem -outform der -nocrypt -out Ta
   kPrivPkcs8.bin
   ```

   iii. 使用加密公钥加密ESK，获取瞬时对称密钥密文Cipher(ESK)。加密时采用RSAES OAEP标准加密，其中MGF1和哈希算法为SHA256。

   ```
   openssl pkeyutl -encrypt -pubin -inkey PublicKey.pem  -in EskAes256.bin  -pkeyopt
   rsa_padding_mode:oaep -pkeyopt rsa_oaep_md:sha256 -pkeyopt rsa_mgf1_md:sha256 -out
   CipherEsk.bin
   ```

   iv. 使用ESK加密PKCS#8格式RSA_2048私钥TAKpriv，获取目标非对称密钥的私钥密文Cipher(TAKpriv)，加密模式为ECB，填充模式为PKCS#7 Padding。

   ```
   xxd -l 32  -c 32 -ps EskAes256.bin | xargs -I {} openssl enc  -aes-256-ecb -e  -K {
   } -in
   TakPrivPkcs8.bin -nosalt -out CipherTakPriv.bin
   ```

v. 组装结果数据，得到加密密钥材料，组装格式为Cipher(ESK) || Cipher(TAKpriv)。

```
cat CipherEsk.bin CipherTakPriv.bin > EncryptedKeyMaterial.bin
openssl enc -e -base64 -A -in EncryptedKeyMaterial.bin -out EncryptedKeyMaterial_base64.txt
```

3. 导入非对称密钥材料。

导入非对称密钥材料时，可以导入从未导入过密钥材料的外部密钥。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**用户主密钥**区域，单击目标密钥ID，进入密钥管理页面。

ii. 在**对称密钥材料**区域，单击**导入密钥材料**。

iii. 在**导入打包后的密钥材料**对话框，上传**打包后的密钥材料**和**导入令牌**。

- **打包后的密钥材料**：上传步骤2生成的对称密钥材料文本文件。
- **导入令牌**：上传步骤1获取的导入令牌文本文件。

iv. 单击**确定**。

导入对称密钥材料成功后，密钥状态从**待导入**更新为**启用中**。

代码示例

通过KMS Java SDK在专属KMS中创建并导入SM2、ECC和RSA密钥的代码示例如下：

```
import java.security.InvalidAlgorithmParameterException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.interfaces.ECPrivateKey;
import java.security.spec.ECGenParameterSpec;
import java.security.spec.MGF1ParameterSpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Random;
import javax.crypto.Cipher;
import javax.crypto.spec.OAEPParameterSpec;
import javax.crypto.spec.PSource.PSpecified;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import com.aliyuncs.AcsRequest;
import com.aliyuncs.AcsResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.http.ProtocolType;
import com.aliyuncs.kms.model.v20160120.CreateKeyRequest;
import com.aliyuncs.kms.model.v20160120.CreateKeyResponse;
import com.aliyuncs.kms.model.v20160120.GetParametersForImportRequest;
```

```
import com.aliyuncs.kms.model.v20160120.GetParametersForImportResponse;
import com.aliyuncs.kms.model.v20160120.ImportKeyMaterialRequest;
import com.aliyuncs.kms.model.v20160120.ImportKeyMaterialResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import org.apache.commons.lang3.tuple.Pair;
import org.bouncycastle.asn1.gm.GMNamedCurves;
import org.bouncycastle.asn1.x9.X9ECParameters;
import org.bouncycastle.crypto.engines.SM2Engine;
import org.bouncycastle.crypto.params.ECDomainParameters;
import org.bouncycastle.crypto.params.ECPublicKeyParameters;
import org.bouncycastle.crypto.params.ParametersWithRandom;
import org.bouncycastle.jcajce.provider.asymmetric.ec.BCECPublicKey;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
public class BringYourOwnAsymmetricKeySample {
    static String regionId = "cn-hangzhou";
    static String accessKeyId = "*** Provide your AccessKeyId ***";
    static String accessKeySecret = "*** Provide your AccessKeySecret ***";
    static String dedicatedKmsInstanceId = "*** Provide your DedicatedKmsInstanceId ***";
    DefaultAcsClient kmsClient;
    private final String SM2PKE_SM4_ECB = "SM2PKE_SM4_ECB";
    private final String RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_PAD = "RSAES_OAEP_SHA_256_AES
_256_ECB_PKCS7_PAD";
    private static Provider BC = new BouncyCastleProvider();
    private static X9ECParameters x9ECParameters = GMNamedCurves.getByName("sm2p256v1");
    private static ECDomainParameters ecDomainParameters = new ECDomainParameters(x9ECParam
eters.getCurve(), x9ECParameters.getG(), x9ECParameters.getN());
    static {
        java.security.Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastlePr
ovider());
    }
    public static void main(String[] args) {
        //初始化KMS SDK。
        DefaultAcsClient client = getClientForPublicEndpoint(regionId, accessKeyId, accessK
eySecret);
        BringYourOwnAsymmetricKeySample sample = new BringYourOwnAsymmetricKeySample(client
);
        //创建并导入EC_SM2类型的外部密钥。
        sample.doByok("EC_SM2", "EC_SM2", sample.SM2PKE_SM4_ECB, "SM4");
        //创建并导入EC_P256类型的外部密钥。
        sample.doByok("EC_P256", "RSA_2048", sample.RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_PA
D, "AES_256");
        //创建并导入RSA类型的外部密钥。
        sample.doByok("RSA_2048", "RSA_2048", sample.RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_P
AD, "AES_256");
    }
    public static DefaultAcsClient getClientForPublicEndpoint(String regionId, String acces
sKeyId, String accessKeySecret) {
        /**
         * Construct an Aliyun Client:
         * Set RegionId, AccessKeyId and AccessKeySecret
         */
        IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyId, accessKey
Secret);
```

```
        DefaultAcsClient client = new DefaultAcsClient(profile);
        return client;
    }
    public BringYourOwnAsymmetricKeySample(DefaultAcsClient kmsClient) {
        this.kmsClient = kmsClient;
    }
    public void doByok(String targetKeySpec, String wrappingKeySpec, String wrappingAlgorit
hm, String ephemeralKeySpec) {
        try {
            //创建ECC外部密钥。
            CreateKeyResponse.KeyMetadata keyMetadata = this.createExternalKeyInDkms(dedica
tedKmsInstanceId, targetKeySpec, "SIGN/VERIFY");
            String keyId = keyMetadata.getKeyId();
            //获取导入密钥材料。
            GetParametersForImportResponse parametersForImportResponse = this.getParameters
ForImport(keyId, wrappingKeySpec,
                wrappingAlgorithm);
            String importToken = parametersForImportResponse.getImportToken();
            String publicKeyBase64 = parametersForImportResponse.getPublicKey();
            //生成瞬时对称密钥。
            byte[] ephemeralSymmetricKeyPlaintext = this.generateEphemeralSymmetricKey(ephe
meralKeySpec);
            //生成目标非对称密钥。
            byte[] targetAsymmetricKeyPlaintext = this.generateTargetAsymmetricKey(targetKe
ySpec);
            //使用加密公钥加密瞬时对称密钥。
            byte[] ephemeralSymmetricKeyCipher = this.encryptEphemeralSymmetricKey(publicKe
yBase64,
                wrappingAlgorithm, ephemeralSymmetricKeyPlaintext);
            //使用瞬时对称密钥加密目标非对称密钥。
            byte[] targetAsymmetricKeyCipher = this.encryptTargetAsymmetricKey(ephemeralSym
metricKeyPlaintext, targetAsymmetricKeyPlaintext,
                wrappingAlgorithm);
            //生成密钥材料。
            byte[] encryptedKeyMaterial = new byte[ephemeralSymmetricKeyCipher.length + tar
getAsymmetricKeyCipher.length];
            System.arraycopy(ephemeralSymmetricKeyCipher, 0, encryptedKeyMaterial, 0, ephem
eralSymmetricKeyCipher.length);
            System.arraycopy(targetAsymmetricKeyCipher, 0, encryptedKeyMaterial, ephemeralS
ymmetricKeyCipher.length, targetAsymmetricKeyCipher.length);
            String encryptedKeyMaterialBase64 =  DatatypeConverter.printBase64Binary(encryp
tedKeyMaterial);
            //导入密钥材料到专属KMS。
            this.importKeyMaterial(keyId, encryptedKeyMaterialBase64, importToken, 0L);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private GetParametersForImportResponse getParametersForImport(String keyId, String keyS
pec, String algorithm) throws Exception {
        GetParametersForImportRequest request = new GetParametersForImportRequest();
        request.setAcceptFormat(FormatType.JSON);
        request.setMethod(MethodType.POST);
        request.setProtocol(ProtocolType.HTTPS);
        request.setKeyId(keyId);
```

```
        request.setKeyId(keyId);
        request.setWrappingKeySpec(keySpec);
        request.setWrappingAlgorithm(algorithm);
        GetParametersForImportResponse resp;
        try {
            resp = this.getAcsResponseWithRetry(request);
        } catch (Exception e) {
            throw e;
        }
        return resp;
    }
    private CreateKeyResponse.KeyMetadata createExternalKeyInDkms(String dedicatedKmsInstan
ce, String keySpec, String keyUsage) throws Exception {
        CreateKeyRequest request = new CreateKeyRequest();
        request.setOrigin("EXTERNAL"); //创建外部密钥。
        request.setKeyStoreId(dedicatedKmsInstance);
        request.setKeySpec(keySpec);
        request.setKeyUsage(keyUsage);
        request.setProtocol(ProtocolType.HTTPS);
        request.setAcceptFormat(FormatType.JSON);
        request.setMethod(MethodType.POST);
        CreateKeyResponse.KeyMetadata ret = null;
        String requestId = null;
        try {
            CreateKeyResponse response = getAcsResponseWithRetry(request);
            ret = response.getKeyMetadata();
            requestId = response.getRequestId();
        } catch (Exception e) {
            throw e;
        }
        return Pair.of(ret, requestId).getKey();
    }
    private <T extends AcsResponse> T getAcsResponseWithRetry(AcsRequest<T> request) throws
ServerException,
        ClientException {
        String expStr = "Retry Max Times";
        for (int i = 0; i < 3; i++) {
            try {
                T resp = this.kmsClient.getAcsResponse(request);
                if (resp == null) {
                    throw new ClientException("Get a null response");
                }
                return resp;
            } catch (ServerException e) {//这里会生成NUllPointer。
                throw e;
            } catch (ClientException e) {
                expStr = e.toString();
                if (expStr.contains("SDK.ServerUnreachable")) {//need retry
                    continue;
                }
                throw e;
            }
        }
        throw new ClientException(expStr);
    }
```

```
    private byte[] generateEphemeralSymmetricKey(String ephemeralSymmetricKeySpec) throws E
xception {
        int ephemeralSymmetricKeyLength = 32; //瞬时对称密钥是AES_256时，长度为32比特。
        if ("SM4".equals(ephemeralSymmetricKeySpec)) {
            ephemeralSymmetricKeyLength = 16;
        }
        byte[] key = new byte[32];
        new Random().nextBytes(key);
        return key;
    }
    private byte[] generateTargetAsymmetricKey(String keySpec) throws Exception {
        PrivateKey privateKey = null;
        //生成SM2密钥，并获取私钥的D值。
        if ("EC_SM2".equals(keySpec)) {
            ECPrivateKey ecPrivateKey = (ECPrivateKey)generateSm2KeyPair().getPrivate();
            byte[] dT = ecPrivateKey.getS().toByteArray();
            byte[] d = new  byte[32];
            if (dT.length == 33) {
                System.arraycopy(dT, 1, d, 0, 32);
            }
            return dT.length == 32 ? dT : d;
        }
        //生成RSA或者ECC私钥。
        if (keySpec.contains("RSA")) {
            String[] keySpecAttrs = keySpec.split("_");
            int bits = Integer.parseInt(keySpecAttrs[keySpecAttrs.length - 1]);
            privateKey = generateRsaKeyPair(bits).getPrivate();
        } else if  (keySpec.contains("EC")) {
            if (keySpec.contains("P256K")) {
                //生成EC_P256K私钥。
                privateKey  = generateEccKeyPair("secp256k1").getPrivate();
            } else {
                //生成EC_P256私钥。
                privateKey=   generateEccKeyPair("secp256r1").getPrivate();
            }
        }
        if (privateKey != null) {
            //返回PKCS#8格式的私钥。
            return  privateKey.getEncoded();
        }
        return null;
    }
    private  KeyPair generateEccKeyPair(String keySpec)
        throws NoSuchAlgorithmException, InvalidAlgorithmParameterException {
        ECGenParameterSpec ecSpec = new ECGenParameterSpec(keySpec);
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("EC");
        keyPairGenerator.initialize(ecSpec, new SecureRandom());
        return keyPairGenerator.generateKeyPair();
    }
    private  KeyPair generateRsaKeyPair(int length) throws Exception {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(length);
        return keyGen.genKeyPair();
    }
```

```
    private KeyPair generateSm2KeyPair() throws Exception {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC", "BC");
        keyGen.initialize(new ECGenParameterSpec("sm2p256v1"), new SecureRandom());
        return keyGen.genKeyPair();
    }
    private byte[] encryptEphemeralSymmetricKey (String publicKeyBase64, String wrappingAlg
orithm, byte[] ephemeralSymmetricKeyPlaintext) throws Exception {
        PublicKey publickey = null;
        byte[] enchbk = null;
        if ("RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_PAD".equals(wrappingAlgorithm)) {
            publickey = parseDerPublicKey("RSA", publicKeyBase64);
            Cipher oaepFromAlgo = Cipher.getInstance("RSA/ECB/OAEPWithSHA-1AndMGF1Padding")
;
            OAEPParameterSpec oaepParams = new OAEPParameterSpec("SHA-256", "MGF1", new MGF
1ParameterSpec("SHA-256"), PSpecified.DEFAULT);
            oaepFromAlgo.init(Cipher.ENCRYPT_MODE, publickey, oaepParams);
            enchbk = oaepFromAlgo.doFinal(ephemeralSymmetricKeyPlaintext);
        } else if ("SM2PKE_SM4_ECB".equals(wrappingAlgorithm)) {
            publickey = parseDerPublicKey("EC", publicKeyBase64, BC);
            BCECPublicKey localECPublicKey = (BCECPublicKey) publickey;
            ECPublicKeyParameters ecPublicKeyParameters = new ECPublicKeyParameters(localEC
PublicKey.getQ(), ecDomainParameters);
            SM2Engine sm2Engine = new SM2Engine(SM2Engine.Mode.C1C3C2);
            sm2Engine.init(true, new ParametersWithRandom(ecPublicKeyParameters));
            enchbk = sm2Engine.processBlock(ephemeralSymmetricKeyPlaintext, 0, ephemeralSym
metricKeyPlaintext.length);
        } else {
            throw new Exception("Invalid wrappingAlgorithm");
        }
        return enchbk;
    }
    private PublicKey parseDerPublicKey(String keyType, String pemKey) throws Exception {
        byte[] derKey = DatatypeConverter.parseBase64Binary(pemKey);
        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(derKey);
        return KeyFactory.getInstance(keyType).generatePublic(keySpec);
    }
    private PublicKey parseDerPublicKey(String keyType, String pemKey, Provider provider) t
hrows Exception {
        byte[] derKey = DatatypeConverter.parseBase64Binary(pemKey);
        X509EncodedKeySpec keySpec = new X509EncodedKeySpec(derKey);
        return KeyFactory.getInstance(keyType, provider).generatePublic(keySpec);
    }
    private byte[] encryptTargetAsymmetricKey (byte[] secretKey, byte[] targetAsymmetricKey
Plaintext, String wrappingAlgorithm)
        throws Exception {
        if ("RSAES_OAEP_SHA_256_AES_256_ECB_PKCS7_PAD".equals(wrappingAlgorithm)) {
            SecretKeySpec secretKeySpec = new SecretKeySpec(secretKey, "AES");
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
            cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);
            return cipher.doFinal(targetAsymmetricKeyPlaintext);
        } else if ("SM2PKE_SM4_ECB".equals(wrappingAlgorithm)) {
            SecretKeySpec secretKeySpec = new SecretKeySpec(secretKey, "SM4");
            Cipher cipher = Cipher.getInstance("SM4/ECB/NoPadding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);
```

```
            return cipher.doFinal(targetAsymmetricKeyPlaintext);
        }
        throw new Exception("Invalid WrappingAlgorithm");
    }
    private boolean importKeyMaterial(
        String keyId,
        String material,
        String token,
        Long expire
    ) throws Exception {
        ImportKeyMaterialRequest req = newImportKeyMaterialRequest(
            keyId, material, token, expire);
        try {
            ImportKeyMaterialResponse resp = this.getAcsResponseWithRetry(req);
        } catch (Exception e) {
            throw e;
        }
        return true;
    }
    private ImportKeyMaterialRequest newImportKeyMaterialRequest(
        String keyId,
        String material,
        String token,
        Long expire
    ) {
        ImportKeyMaterialRequest request = new ImportKeyMaterialRequest();
        request.setAcceptFormat(FormatType.JSON);
        request.setMethod(MethodType.POST);
        request.setProtocol(ProtocolType.HTTPS);
        request.setEncryptedKeyMaterial(material);
        request.setImportToken(token);
        request.setKeyId(keyId);
        request.setKeyMaterialExpireUnix(expire);
        return request;
    }
}
```

3.4.5. 管理应用接入点

当您为专属KMS实例创建应用接入点后，应用接入点权限策略作用域会显示专属KMS。您可以根据需要更新应用接入点、删除应用接入点或者删除Client Key。

创建应用接入点

当专属KMS实例处于**已启用**状态时，您可以为实例快速创建应用接入点（AAP）和应用身份凭证（Client Key），以便应用程序正常访问专属KMS。

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，选择专属KMS实例所在的地域。
3. 在左侧导航栏，单击**专属KMS**。
4. 在**专属KMS**页面，找到目标专属KMS实例，在**操作**列单击**详情**。

5. 在**应用接入指南**区域，单击**快速创建应用接入点**。
6. 在**快速配置应用身份凭证和权限**面板，设置应用接入点信息。
   i. 输入**应用接入点名称**。
   ii. 设置**访问控制策略**。
   - **允许访问资源**：默认填写 `Key/*` ，表示允许访问当前专属KMS实例的全部密钥。
   - **允许网络来源**：允许访问的网络类型和IP地址。您可以设置私网IP地址或者私网网段，多个IP地址之间用半角逗号（,）分隔。

   iii. 单击**创建**。
7. 在**应用身份凭证**对话框，获取**凭证口令**和**应用身份凭证内容**（Client Key）。
   - **凭证口令**：单击**复制口令**，获取凭证口令。
   - **应用身份凭证内容**：单击**下载应用身份凭证**，保存应用身份凭证信息。
     应用身份凭证信息包含凭证ID（KeyId）和凭证内容（PrivateKeyData），示例如下：

     ```
     {
       "KeyId": "KAAP.71be72c8-73b9-44e0-bb75-81ee51b4****",
       "PrivateKeyData": "MIIJwwIBAz****ICNXX/pOw=="
     }
     ```

     说明 专属KMS不会保存Client Key的凭证内容，因此您只能在创建Client Key时获取到加密的PKCS12文件（凭证内容），请妥善保管。
8. 单击**关闭**。

   应用接入点创建成功后，您可以在左侧导航栏单击**应用管理**查看应用接入点信息，包括认证方式、权限策略、网络控制规则、Client Key等。
9. 在**应用接入指南**区域，单击**获取实例CA证书**下方的**下载**，下载.pem格式的CA证书文件。

更新应用接入点

您可以根据需要更新应用接入点的权限策略，更新应用使用专属KMS实例的权限，从而实现不同应用拥有不同实例访问权限的目的。

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 找到目标应用接入点名称，然后单击右上角的**更新**。
5. 在**更新应用接入点**对话框，更新权限策略。
   i. 单击**可选策略**右侧的 ＋ 图标。

ii. 在**创建权限策略**对话框，配置以下参数，然后单击**创建**。

配置项	说明
权限策略名称	权限策略的名称。
作用域	权限策略的适用范围。 选择专属KMS实例的服务ID。
RBAC权限	权限管理模板，表示权限策略对具体资源的操作。 选择CryptoServiceKeyUser。
允许访问资源	权限策略被授权的具体对象。可以通过以下两种方法设置： ■ 方法一：在**可选资源**区域，选择已有资源，然后单击 < 图标。 ■ 方法二：在**已选资源区域**，单击 + 图标，然后手动输入资源，最后单击**添加**。 **说明** 资源支持通配符（*）作为后缀。
网络控制规则	权限策略允许访问的网络类型和IP地址。 您可以在**可选规则**区域，选择已有规则，或者按照以下步骤创建并添加新规则。 a. 单击 + 图标。 b. 在**创建网络访问规则**对话框，设置以下参数： ■ **名称**：网络访问规则的名称。 ■ **网络类型**：应用访问KMS的网络类型。 选择Private，适用于应用程序访问部署到VPC内的专属服务。 ■ **描述信息**：网络访问规则的详细信息。 ■ **允许私网地址**：允许应用程序访问的网络地址。 您可以设置私网IP地址或者网段，多个IP地址之间用半角逗号（,）间隔。 c. 单击**创建**。 d. 选择已有规则，然后单击 < 图标。

iii. 选择已有策略，然后单击 < 图标。

6. 输入描述信息，然后单击**更新**。

删除应用接入点

应用接入点是使用专属KMS实例需要的凭证，删除后您可能无法正常使用专属KMS实例，请谨慎操作。删除应用接入点将同步删除应用接入点绑定的所有Client Key。

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，选择应用接入点所在的地域。

3. 在左侧导航栏，单击**应用管理**。
4. 单击目标应用接入点**操作**列的**删除**。
5. 在**删除应用接入点**对话框，单击**确定**。

删除Client Key

Client Key是应用接入点用于身份认证的凭证，其凭证内容需要在首次创建时妥善保存。如果您忘记了Client Key的凭证内容，可以删除该Client Key，然后创建一个新的Client Key。

Client Key删除方法如下：

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 单击应用接入点名称。
5. 在**Client Key**区域，单击目标Client Key**操作**列的**删除**。
6. 在**删除Client Key**对话框，单击**确定**。

3.4.6. 升级实例

购买专属KMS实例后，如果实例规格无法满足您的实际业务需要，您可以在密钥管理服务控制台升级实例规格。本文介绍如何升级实例规格。

背景信息

升级实例表示提升当前专属KMS实例的规格配置，您需要补齐实例的规格差价。

> **说明** 目前仅支持升级**专有网络数量**规格。同一地域的多个VPC中使用同一个专属KMS实例时，如果选择关联的VPC数量超过了关联资源限制，您需要升级**专有网络数量**。

操作步骤

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，选择专属KMS实例所在的地域。
3. 在左侧导航栏，单击**专属KMS**。
4. 单击目标专属KMS实例**操作**列的**详情**。
5. 在**专属KMS实例详情**页签，单击**服务规格**后的**升配**。
6. 在**变配**页签选择**专有网路数量**，勾选**服务协议**。
7. 单击**立即购买**并完成**支付**。

3.5. 专属KMS SDK

3.5.1. 专属KMS Java SDK

专属KMS SDK for Java帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS标准版实例并正常连接密码机，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 已经获取专属KMS标准版实例VPC地址，并确保可以通过以下方式访问专属KMS标准版实例VPC地址：
  - 在激活密码机实例集群时设置的VPC中访问专属KMS标准版实例VPC地址。
  - 本地设备所在网络可以正常解析并访问专属KMS标准版实例VPC地址。

  具体操作，请参见查询专属KMS标准版实例。

安装SDK

在项目中添加alibabacloud-dkms-gcs-sdk的依赖，可从Maven仓库中自动下载发布的Java安装包。

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>alibabacloud-dkms-gcs-sdk</artifactId>
  <version>x.x.x</version>
</dependency>
```

> **说明** 专属KMS SDK的最新版本，请参见专属KMS SDK for Java。

初始化SDK

您可以初始化一个专属KMS标准版实例的Java客户端，用于调用专属KMS标准版实例管理的密钥等资源。使用Java SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。
   为保障生产环境通信安全，需要配置Java可信证书。具体配置过程如下：
   i. 将CA证书文件拆分为2个证书文件。
      - 文件1：`rootca.pem`

        ```
        -----BEGIN CERTIFICATE-----
        <Root CA Certificate BASE64 Content>
        -----END CERTIFICATE-----
        ```

      - 文件2：`subca.pem`

        ```
        -----BEGIN CERTIFICATE-----
        <Sub CA Certificate BASE64 Content>
        -----END CERTIFICATE-----
        ```

   ii. 使用keytool工具将拆分后的CA证书导入至`JAVA_HOME/lib/security/cacerts`。
      - 导入文件1（`rootca.pem`）

        ```
        keytool -importcert -alias PrivateKmsCA_RootCA -keystore cacerts -storepass changeit -file rootca.pem
        ```

      - 导入文件2（`subca.pem`）

        ```
        keytool -importcert -alias PrivateKmsCA_SubCA -keystore cacerts -storepass changeit -file subca.pem
        ```

iii. 验证代码。

```
URL serviceUrl = new URL("https://<service_id>.cryptoservice.kms.aliyuncs.com");
serviceUrl.openConnection().connect();
```

> **说明** 如果没有 `javax.net.ssl.SSLHandshakeException` 的异常提示，则配置正确。

开发环境可使用 `RuntimeOptions` 设置临时忽略可信证书的验证。示例代码如下：

```
import com.aliyun.dkms.gcs.openapi.util.models.RuntimeOptions;
RuntimeOptions runtimeOptions = new RuntimeOptions();
runtimeOptions.setIgnoreSSL(true);
...
client.encryptWithOptions(encryptRequest, runtimeOptions);
```

2. 创建专属KMS标准版Client。
   创建专属KMS标准版Client时，需要指定实例的Endpoint。EndPoint为专属KMS标准版实例服务地址去掉**http://**。关于专属KMS标准版实例服务地址的更多信息，请参见查询专属KMS标准版实例。

```
import com.aliyun.dkms.gcs.openapi.models.Config;
import com.aliyun.dkms.gcs.sdk.Client;
//连接协议，固定为HTTPS。
String protocol = "https";
//Endpoint，专属KMS标准版实例服务地址去掉http://。
String endpoint = "<service_id>.cryptoservice.kms.aliyuncs.com";
//专属KMS标准版实例Client Key。
String clientKey = "<your client key>";
//专属KMS标准版实例Client Key解密口令。
String clientKeyPass = "<your client key password>";
Client client = new Client(new Config()
                           .setProtocol(protocol)
                           .setEndpoint(endpoint)
                           .setClientKeyContent(clientKey)
                           .setPassword(clientKeyPass));
```

代码示例

- 使用专属KMS标准版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS标准版实例加密密钥的ID或别名（Alias）。
String cipherKeyId = "<your cipher key id>";
//待加密数据。
byte[] originData = <your origin data to encrypt>;
EncryptRequest encryptRequest = new EncryptRequest();
encryptRequest.setKeyId(cipherKeyId);
encryptRequest.setPlaintext(originData);
EncryptResponse encryptResponse = client.encrypt(encryptRequest);
//加密数据。
byte[] cipherData = encryptResponse.getCiphertextBlob();
//Cipher初始向量，用于解密数据。
byte[] iv = encryptResponse.getIv();
//请求ID。
String requestId = encryptResponse.getRequestId();
```

- 使用专属KMS标准版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS标准版实例解密密钥的ID或别名（Alias）。
String cipherKeyId = "<your cipher key id>";
//待解密数据。
byte[] cipherData = <your cipher data to decrypt>;
//Cipher初始向量，必须与加密时一致。
byte[] iv = <IV value>;
DecryptRequest decryptRequest = new DecryptRequest();
        decryptRequest.setKeyId(cipherKeyId);
        decryptRequest.setCiphertextBlob(cipherData);
        decryptRequest.setIv(iv);
DecryptResponse decryptResponse = client.decrypt(decryptRequest);
//原始数据。
byte[] originData = decryptResponse.getPlaintext();
//请求ID。
String requestId = decryptResponse.getRequestId();
```

- 使用专属KMS标准版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS标准版实例签名密钥的ID或别名（Alias）。
String signerKeyId = "<the signer key id>";
//待签名数据。
byte[] message = <the data to sign>;
SignRequest signRequest = new SignRequest();
signRequest.setKeyId(signKeyId);
signRequest.setMessage(message);
signRequest.setMessageType("RAW");
SignResponse signResponse = client.sign(signRequest);
//签名值。
byte[] signature = signResponse.getSignature();
//请求ID。
String requestId = signResponse.getRequestId();
```

- 使用专属KMS标准版Client调用Verify接口验证数字签名
  详细代码示例，请参见原始代码。

```
import com.aliyun.dkms.gcs.sdk.Client;
import com.aliyun.dkms.gcs.sdk.models.*;
//专属KMS标准版实例签名密钥的ID或别名（Alias）。
String signerKeyId = "<the signer key id>";
//待验证签名的数据。
byte[] message = <the data to sign>;
VerifyRequest verifyRequest = new VerifyRequest();
verifyRequest.setKeyId(signerKeyId);
verifyRequest.setMessage(message);
verifyRequest.setMessageType("RAW");
verifyRequest.setSignature(signature);
VerifyResponse verifyResponse = client.verify(verifyRequest);
//验签结果。
boolean valid = verifyResponse.getValue();
//请求ID。
String requestId = verifyResponse.getRequestId();
```

3.5.2. 专属KMS PHP SDK

专属KMS SDK for PHP帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS标准版实例并正常连接密码机，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 已经获取专属KMS标准版实例VPC地址，并确保可以通过以下方式访问专属KMS标准版实例VPC地址：
  - 在激活密码机实例集群时设置的VPC中访问专属KMS标准版实例VPC地址。
  - 本地设备所在网络可以正常解析并访问专属KMS标准版实例VPC地址。

  具体操作，请参见查询专属KMS标准版实例。

安装SDK

- **方式一（推荐）：通过Composer来管理项目依赖**

  i. 在终端中切换到项目目录，直接执行以下代码安装AlibabaCloud DKMS-GCS SDK for PHP作为依赖项。

  ```
  composer require alibabacloud/dkms-gcs-sdk
  ```

  ii. 在composer.json中添加以下内容，声明对AlibabaCloud DKMS-GCS SDK for PHP的依赖。

  ```
  "require": {
    "alibabacloud/dkms-gcs-sdk": "^0.2.1"
    }
  ```

  iii. 在终端中切换到项目目录下，执行以下代码安装依赖。

  ```
  composer install
  ```

  iv. 使用Composer安装完成后，在PHP代码中引入依赖。

  ```
  require_once __DIR__ . '/vendor/autoload.php';
  ```

- **方式二：直接下载SDK源码**

  访问开源代码仓库下载SDK源码，在PHP代码中引入SDK目录下的autoload.php文件。

  ```
  require_once '/path/to/dkms-gcs-sdk/autoload.php';
  ```

初始化SDK

您可以初始化一个专属KMS标准版实例的PHP客户端，用于调用专属KMS标准版实例管理的密钥等资源。使用PHP SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。
   为保障生产环境通信安全，需要配置Java可信证书。
   设置 `RuntimeOptions` 的 `verify` 字段。示例代码如下：

   ```
   <?php
   use AlibabaCloud\Dkms\Gcs\OpenApi\Util\Models\RuntimeOptions;
   $runtimeOptions = new RuntimeOptions();
   $runtimeOptions->verify = 'path/to/caCert.pem';
   ...
   $encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
   ```

   开发环境可使用 `RuntimeOptions` 的 `ignoreSSL` 设置临时忽略可信证书的验证。示例代码如下：

   ```
   <?php
   use AlibabaCloud\Dkms\Gcs\OpenApi\Util\Models\RuntimeOptions;
   $runtimeOptions = new RuntimeOptions();
   $runtimeOptions->ignoreSSL = true;
   ...
   $encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
   ```

2. 创建专属KMS标准版Client。
   创建专属KMS标准版Client时，需要指定实例的Endpoint。EndPoint为专属KMS标准版实例VPC地址去掉HTTPS。关于专属KMS标准版实例VPC地址的更多信息，请参见查询专属KMS标准版实例。

```
<?php
use AlibabaCloud\Dkms\Gcs\Sdk\Client as AlibabaCloudDkmsGcsSdkClient;
use AlibabaCloud\Dkms\Gcs\OpenApi\Models\Config as AlibabaCloudDkmsGcsOpenApiConfig;
$config = new AlibabaCloudDkmsGcsOpenApiConfig();
//连接协议，固定为HTTPS。
$config->protocol = 'https';
//专属KMS标准版实例Client Key。
$config->clientKeyContent = '<your client key content>';
//专属KMS标准版实例Client Key解密口令。
$config->password = '<your client key password>';
//Endpoint，专属KMS标准版实例的服务地址去掉"HTTPS://"。
//示例：<service_id>.cryptoservice.kms.aliyuncs.com;
$config->endpoint = '<your dkms instance service address>';
$client = new AlibabaCloudDkmsGcsSdkClient($config);
```

代码示例

- 使用专属KMS标准版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\EncryptRequest;
$encryptRequest = new EncryptRequest();
//专属KMS标准版实例加密密钥的ID或别名（Alias）。
$encryptRequest->keyId = '<your cipher key id>';
//待加密数据。
$encryptRequest->plaintext = \AlibabaCloud\Dkms\Gcs\OpenApi\Util\Utils::toBytes('encrypt
plaintext');
$encryptResponse = $client->encryptWithOptions($encryptRequest, $runtimeOptions);
//密文。
$ciphertextBlob = $encryptResponse->ciphertextBlob;
//Cipher初始向量，用于解密数据。
$iv = $encryptResponse->iv;
//请求ID。
$requestId = $encryptResponse->requestId;
```

- 使用专属KMS标准版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\DecryptRequest;
$decryptRequest = new DecryptRequest();
//专属KMS标准版实例解密密钥的ID或别名（Alias）。
$decryptRequest->keyId = '<your cipher key id>';
//待解密数据，加密返回的密文。
$decryptRequest->ciphertextBlob = null;
//Cipher初始向量，必须与加密时一致。
$decryptRequest->iv = null;
$decryptResponse = $client->decryptWithOptions($decryptRequest, $runtimeOptions);
//原始明文数据。
$plaintext = $decryptResponse->plaintext;
//请求ID。
$requestId = $decryptResponse->requestId;
```

- 使用专属KMS标准版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\SignRequest;
$signRequest = new SignRequest();
//专属KMS标准版实例签名密钥的ID或别名（Alias）。
$signRequest->keyId = 'your cipher key id';
//待签名数据。
$signRequest->message = <the data to sign>;
//待签名数据类型，RAW-原始数据，DIGEST-原始数据摘要。
$signRequest->messageType = 'RAW';
$signResponse = $client->signWithOptions($signRequest, $runtimeOptions);
//签名值。
$signature = $signResponse->signature;
//请求ID。
$requestId = $signResponse->requestId;
```

- 使用专属KMS标准版Client调用Verify接口验证数字签名
  详细代码示例，请参见原始代码。

```
<?php
require __DIR__ . '/vendor/autoload.php';
use AlibabaCloud\Dkms\Gcs\Sdk\Models\VerifyRequest;
$verifyRequest = new VerifyRequest();
//专属KMS标准版实例签名密钥的ID或别名（Alias）。
$verifyRequest->keyId = 'your cipher key id';
//待验证签名的数据。
$verifyRequest->message = <the data to sign>;
//待签名数据类型，RAW表示原始数据，DIGEST表示原始数据摘要。
$verifyRequest->messageType = 'RAW';
//待验证签名值。
$verifyRequest->signature = null;
$verifyResponse = $client->verifyWithOptions($verifyRequest, $runtimeOptions);
//验签结果。
$value = $verifyResponse->value;
//请求ID。
$requestId = $verifyResponse->requestId;
```

3.5.3. 专属KMS Go SDK

专属KMS SDK for Go帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

您可以访问开源代码仓库，查看SDK源码及代码示例。同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS标准版实例并正常连接密码机，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 已经获取专属KMS标准版实例VPC地址，并确保可以通过以下方式访问专属KMS标准版实例VPC地址：
  - 在激活密码机实例集群时设置的VPC中访问专属KMS标准版实例VPC地址。
  - 本地设备所在网络可以正常解析并访问专属KMS标准版实例VPC地址。

  具体操作，请参见查询专属KMS标准版实例。

安装SDK

- 方式一：使用 `go mod` 管理您的依赖。
  在 `go mod` 中添加以下内容安装依赖包。

```
require (
    github.com/aliyun/alibabacloud-dkms-gcs-go-sdk v0.2.0
)
```

- 方式二：使用 `go get` 获取远程代码包。

```
$ go get -u github.com/aliyun/alibabacloud-dkms-gcs-go-sdk
```

初始化SDK

您可以初始化一个专属KMS标准版实例的Go客户端，用于调用专属KMS标准版实例管理的密钥等资源。使用Go SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。

为保障生产环境通信安全，需要配置可信证书。
将CA证书内容设定为 `RuntimeOptions` 的verify字段，示例代码如下：

```
import (
    dedicatedkmsopenapiutil "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi-uti
l"
    "github.com/alibabacloud-go/tea/tea"
    "io/ioutil"
)
// 验证服务端证书
ca, err := ioutil.ReadFile("path/to/caCert.pem")
if err != nil {
    panic(err)
}
runtimeOptions := &dedicatedkmsopenapiutil.RuntimeOptions{
    Verify: tea.String(string(ca)),
}
...
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
```

开发环境可使用 `RuntimeOptions` 的ignoreSSL字段临时忽略可信证书的验证。示例代码如下：

```
import (
    dedicatedkmsopenapiutil "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi-uti
l"
    "github.com/alibabacloud-go/tea/tea"
)
// 忽略服务端证书
runtimeOptions := &dedicatedkmsopenapiutil.RuntimeOptions{
    IgnoreSSL: tea.Bool(true),
}
...
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
```

2. 创建专属KMS标准版Client。
创建专属KMS标准版Client时，需要指定标准版实例的Endpoint。EndPoint为专属KMS标准版实例服务地址去掉HTTPS。关于专属KMS标准版实例服务地址的更多信息，请参见查询专属KMS标准版实例。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
  "github.com/alibabacloud-go/tea/tea"
)
config := &dedicatedkmsopenapi.Config{
  // 连接协议，固定为HTTPS
  Protocol: tea.String("https"),
  // 专属KMS标准版实例Client Key
    ClientKeyContent: tea.String("<your client key content>"),
  // 专属KMS标准版实例Client Key解密口令
    Password: tea.String("<your client key password>"),
  // Endpoint，专属KMS标准版实例的服务地址去掉'HTTPS://'
    Endpoint: tea.String("<service_id>.cryptoservice.kms.aliyuncs.com"),
}
client, err := dedicatedkmssdk.NewClient(config)
```

代码示例

- 使用专属KMS标准版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 待加密数据。
plaintext := []byte("encrypt plaintext")
// 专属KMS标准版实例加密密钥的ID或别名（Alias）。
keyId := "<your cipher key id>"
encryptRequest := &dedicatedkmssdk.EncryptRequest{
    KeyId:     tea.String(keyId),
    Plaintext: plaintext,
}
encryptResponse, err := client.EncryptWithOptions(encryptRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 密文。
cipher := encryptResponse.CiphertextBlob
// Cipher初始向量，用于解密数据。
iv := encryptResponse.Iv
// 请求ID。
requestId := tea.StringValue(encryptResponse.RequestId)
```

- 使用专属KMS标准版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS标准版实例解密密钥的ID或别名（Alias）。
keyId := "<your cipher key id>"
// 待解密数据，加密返回的密文。
ciphertextBlob := []byte("<your cipher data to decrypt>")
// Cipher初始向量，必须与加密时一致。
iv := []byte("<IV value>")
decryptRequest := &dedicatedkmssdk.DecryptRequest{
    KeyId:          tea.String(keyId),
    CiphertextBlob: ciphertextBlob,
    Iv:             iv,
}
decryptResponse, err := client.DecryptWithOptions(decryptRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 原始明文数据。
plaintext := decryptResponse.Plaintext
// 请求ID。
requestId := tea.StringValue(decryptResponse.RequestId)
```

- 使用专属KMS标准版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS实例签名密钥的ID或别名（Alias）。
signerKeyId := "<the signer key id>"
// 待签名数据。
message := []byte("<the data to sign>")
// 待签名数据类型，RAW-原始数据，DIGEST-摘要。
messageType := "RAW"
signRequest := &dedicatedkmssdk.SignRequest{
    KeyId:       tea.String(signerKeyId),
    Message:     message,
    MessageType: tea.String(messageType),
}
signResponse, err := client.SignWithOptions(signRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 签名值。
signature := signResponse.Signature
// 请求ID。
requestId := tea.StringValue(signResponse.RequestId)
```

- 使用专属KMS标准版Client调用Verify接口验证数字签名

详细代码示例，请参见原始代码。

```
import (
    dedicatedkmsopenapi "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/openapi"
    dedicatedkmssdk "github.com/aliyun/alibabacloud-dkms-gcs-go-sdk/sdk"
    "github.com/alibabacloud-go/tea/tea"
)
// 专属KMS标准版实例签名密钥的ID或别名（Alias）。
signerKeyId := "<the signer key id>"
// 待验证签名的数据。
message := []byte("<the data to sign>")
// 待签名数据类型，RAW-原始数据，DIGEST-摘要。
messageType := "RAW"
// 待验证签名值。
signature := []byte("<the signature>")
verifyRequest := &dedicatedkmssdk.VerifyRequest{
    KeyId:       tea.String(signerKeyId),
    Message:     message,
    MessageType: tea.String(messageType),
    Signature:   signature,
}
verifyResponse, err := client.VerifyWithOptions(verifyRequest, runtimeOptions)
if err != nil {
    panic(err)
}
// 验签结果。
value := tea.BoolValue(verifyResponse.Value)
// 请求ID。
requestId := tea.StringValue(verifyResponse.RequestId)
```

3.5.4. 专属KMS Python SDK

专属KMS SDK for Python帮助您通过简单的编程访问专属KMS的API，实现加密解密、签名验签的业务诉求。

背景信息

- 如果您使用Python3访问专属KMS SDK，可以查看Python3开源代码仓库。
- 如果您使用Python2访问专属KMS SDK，可以查看Python2开源代码仓库。

同时也欢迎您提出宝贵意见，或者提供代码示例。

前提条件

- 您已经启用专属KMS标准版实例并正常连接密码机，为实例创建密钥及应用接入点，并保存了Client Key及CA证书。具体操作，请参见快速入门。
- 已经获取专属KMS标准版实例VPC地址，并确保可以通过以下方式访问专属KMS标准版实例VPC地址：
  - 在激活密码机实例集群时设置的VPC中访问专属KMS标准版实例VPC地址。
  - 本地设备所在网络可以正常解析并访问专属KMS标准版实例VPC地址。

  具体操作，请参见查询专属KMS标准版实例。

安装SDK

- 如果您通过Python3使用专属KMS SDK安装alibabacloud-dkms-gcs模块，安装命令如下：

```
pip install alibabacloud-dkms-gcs
```

- 如果您通过Python2使用专属KMS SDK安装alibabacloud-dkms-gcs-python2模块，安装命令如下：

```
pip install alibabacloud-dkms-gcs-python2
```

初始化SDK

您可以初始化一个专属KMS标准版实例的Python客户端，用于调用专属KMS标准版实例管理的密钥等资源。使用Python SDK发起专属KMS API请求，您需要初始化一个Client实例，并根据需要修改Config的默认配置项。

1. 配置CA证书。
   为保障生产环境通信安全，需要配置可信证书。
   将CA证书路径设定为 `RuntimeOptions` 的verify字段，示例代码如下：

```
# -*- coding: utf-8 -*-
from openapi_util.models import RuntimeOptions
runtime_options = RuntimeOptions()
# 忽略SSL验证
# runtime_options.ignore_ssl = True
# ca证书路径
runtime_options.verify = "<your-ca-certificate-file-path>"
...
response = client.encrypt_with_options(request, runtime_options)
```

2. 创建专属KMS标准版Client。
   创建专属KMS标准版Client时，需要指定实例的Endpoint。EndPoint为专属KMS标准版实例服务地址去掉HTTPS。关于专属KMS标准版实例服务地址的更多信息，请参见查询专属KMS标准版实例。

```
# -*- coding: utf-8 -*-
from openapi.models import Config
from sdk.client import Client
config = Config()
# 连接协议，固定为HTTPS
config.protocol = "https"
# 专属KMS标准版实例Client Key
config.client_key_content = "<your-client-key-content>"
# 专属KMS标准版实例Client Key解密口令
config.password = "<your-password>"
# Endpoint，专属KMS标准版实例服务地址去掉HTTPS
config.endpoint = "<your-endpoint>"
client = Client(config)
```

代码示例

- 使用专属KMS标准版Client调用Encrypt接口加密数据
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import EncryptRequest
request = EncryptRequest()
# 待加密数据
request.plaintext = "<your-plaintext>".encode("utf-8")
# 专属KMS标准版实例加密密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
encrypt_response = client.encrypt_with_options(request, runtime_options)
# 加密数据
ciphertext_blob = encrypt_response.ciphertext_blob
# cipher初始向量，用于解密数据
iv = encrypt_response.iv
# 请求ID
request_id = encrypt_response.request_id
```

- 使用专属KMS标准版Client调用Decrypt接口解密密文
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import DecryptRequest
request = DecryptRequest()
# 待解密数据
request.ciphertext_blob = "<your-ciphertext-blob>"
# 专属KMS标准版实例解密密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
# Cipher初始向量，必须与加密时一致
request.iv = "<your-iv>"
decrypt_response = client.decrypt_with_options(request, runtime_options)
// 原始明文数据
plaintext = decrypt_response.plaintext;
// 请求ID
request_id = decrypt_response.request_id;
```

- 使用专属KMS标准版Client调用Sign接口进行数字签名
  详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import SignRequest
request = SignRequest()
# 专属KMS标准版实例签名密钥的ID或别名（Alias）
request.key_id = "<your-key-id>"
# 待签名数据
request.message = "<your_raw_message>"
# RAW-原始消息，DIGEST-摘要
request.message_type = "RAW"
# 签名算法
request.algorithm = "<your-algorithm>"
sign_response = client.sign_with_options(request, runtime_options)
# 签名值
signature = sign_response.signature
# 请求ID
request_id = sign_response.request_id
```

- 使用专属KMS标准版Client调用Verify接口验证数字签名

详细代码示例，请参见原始代码。

```
# -*- coding: utf-8 -*-
from sdk.models import SignRequest
request = VerifyRequest()
# 专属KMS标准版实例签名密钥的ID或别名（Alias）。
request.key_id = "<your-key-id>"
# 待验证签名的数据
request.message = "<your_raw_message>"
# RAW-原始消息，DIGEST-摘要
request.message_type = "RAW"
# 签名算法
request.algorithm = "<your-algorithm>"
# 签名值
request.signature = "<your-signature>"
verify_response = client.verify_with_options(request, runtime_options)
# 验签结果
valid = verify_response.valid
# 请求ID
request_id = verify_response.request_id
```

4.凭据管家

4.1. 凭据管家概述

KMS凭据管家（Secrets Manager）为您提供凭据的全生命周期管理和安全便捷的应用接入方式，帮助您规避在代码中硬编码凭据带来的敏感信息泄露风险。

为什么使用凭据管家

数据库账号口令、服务器账号口令、SSH Key、访问密钥等凭据的泄露，是当今数据安全面临的主要威胁之一。为了降低数据泄露的安全风险，执行有效的凭据保护和定期轮转非常关键。企业在实施凭据保护和轮转策略时，面临以下挑战：

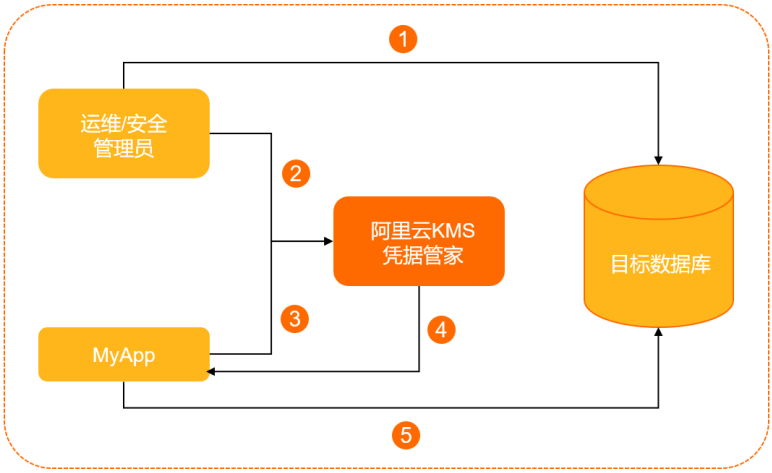
- 为了保护凭据，需要对其进行加密。应用部署的流程变长，带来很高的研发运维成本，也很难强制执行。
- 缺乏凭据自动轮转的软件设施，而人工轮转依赖安全、运维、研发等多个角色相互配合，流程制定和实施难度高且容易出错。
- 缺乏针对凭据泄露事件的快速应急响应能力，处理凭据泄露事件时容易造成系统故障。
- 缺乏对各类云资源所需凭据的中心化管理手段，无法实现规模化管理，管理成本高。

使用KMS凭据管家（Secrets Manager）可以为您带来以下安全优势：

- 通过托管和加密凭据，防止凭据硬编码带来的凭据泄露以及高价值资产泄露，提升数据安全性。
- 提供安全便捷的客户端接入方式，应用程序以无代码或者低代码的方式，动态使用凭据。
- 具备应急处置能力，当您通过人工一次性轮转的方式更新凭据时，应用程序不受影响。
- 支持高频次轮转的全动态凭据，缩小凭据的有效时间窗口，进一步避免凭据泄露带来的安全风险。
- 支持通过API、阿里云ROS或Terraform等运维编排工具，满足中心化、规模化的安全管理需求。

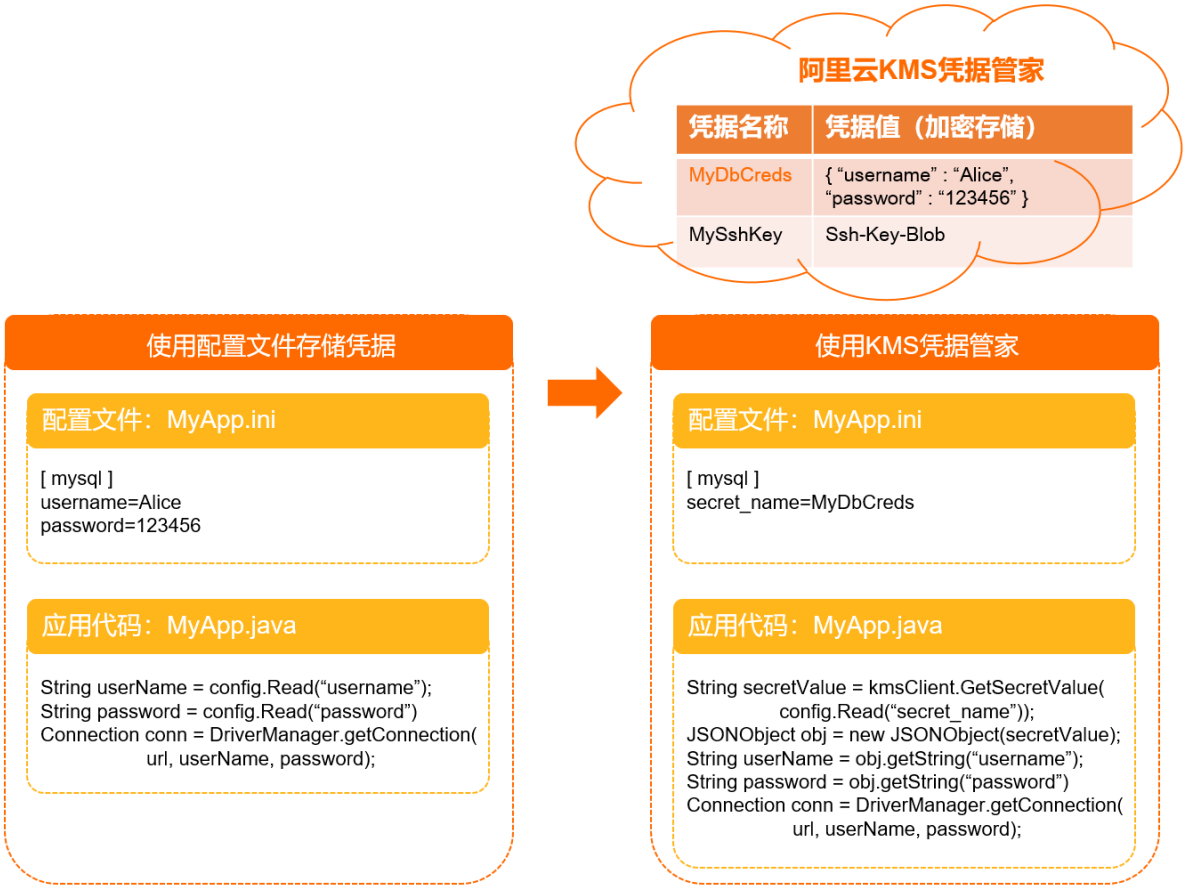
应用场景

以数据库用户名和口令为例，为您介绍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。



功能特性

- 加密保护凭据：凭据管家使用KMS中的用户主密钥（CMK）对凭据进行加密保护。您既可以指定特定的CMK，也可以依赖凭据管家为您自动生成的CMK。凭据管家会为每个阿里云账号在每个地域分配一个独立的CMK用作默认系统加密。
- 动态使用凭据：在程序中通过凭据管家的客户端工具，动态读取凭据，使用实时的最新凭据。
- 自动轮转凭据：凭据管家对特定类型凭据支持“开箱即用“的自动轮转，您也可以通过函数计算编码的方式，自定义周期性轮转凭据。
- 控制访问和审计使用：您可以通过访问控制RAM控制对凭据的访问，也可以通过操作审计Actiontrail审计对凭据的轮转、读取等操作。

使用凭据管家

凭据管家主要用于安全管理、应用研发和部署。

- 安全管理
  基于实际使用场景选择合适的凭据类型，创建、管理和轮转凭据。
  - RDS数据库账号口令：动态RDS凭据概述。
  - RAM用户访问密钥：动态RAM凭据概述。
  - ECS用户密码和公私钥：动态ECS凭据概述。
  - 通用凭据：通用凭据概述。
- 应用研发和部署

您可以使用KMS SDK、凭据管家客户端、Kubernetes插件等多种方式接入凭据管家，通过凭据的名称，在应用中动态读取凭据的敏感值。更多信息，请参见应用程序接入凭据管家。

4.2. 管理应用接入点

您可以创建应用接入点AAP（Application Access Point），控制应用程序如何使用凭据。

支持的地域

华东1（杭州）、华东2（上海）、华北2（北京）、华北3（张家口）、华南1（深圳）、新加坡、澳大利亚（悉尼）、马来西亚（吉隆坡）、印度尼西亚（雅加达）、美国（弗吉尼亚）、杭州金融云、上海金融云、深圳金融云和北京政务云。

创建应用接入点

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，选择应用接入点所在的地域。
3. 在左侧导航栏，单击**应用管理**。
4. 单击**创建应用接入点**。
5. 在**创建应用接入点**对话框，设置基本信息。
   i. 输入**名称**和**描述信息**。

   > **说明** 应用接入点名称在阿里云账号下的当前地域内唯一。

   ii. 在**认证方式**区域，选择认证方式。

认证方式	说明	示例
RAMRole	如果您为应用程序的运行环境（例如：ECS实例、ACK集群、函数计算）绑定了RAM角色，可以使用RAMRole的认证方式。此时需指定以下信息： ■ **委托信任**：对应用程序进行身份认证时，KMS会校验RAM角色的委托信任规则。您可以指定绑定的RAM角色类型，自动配置委托信任规则。 取值： ■ ECS实例角色：应用部署在ECS实例。 ■ ACK Worker角色：应用部署在ACK集群。 ■ 函数计算角色：应用部署在函数计算。 ■ **角色名称**：绑定的RAM角色名称。	■ **委托信任**：ECS实例角色。 ■ **角色名称**：ECSRole。
Client Key	您可以使用Client Key的认证方式，为AAP绑定客户端证书。AAP使用证书的公钥，对应用程序进行身份认证。 选择该方式时，您需要在AAP创建完成后，绑定Client Key。具体操作，请参见为AAP绑定Client Key。	-

   iii. 单击**下一步**。
6. 设置权限策略。

i. 单击**可选策略**右侧的 ＋ 图标。

ii. 在**创建权限策略**对话框，设置以下参数，然后单击**创建**。

参数名称	参数说明	示例
权限策略名称	权限策略的名称。	RAMPolicy
作用域	权限策略的适用范围。 取值： ■ 共享KMS：权限策略适用于KMS。 ■ 专属KMS实例ID：权限策略适用于指定的专属KMS实例。	共享KMS
RBAC权限	权限管理模板，表示权限策略对具体资源的操作。 取值： ■ SecretUser：对KMS进行凭据相关操作，可操作的接口为GetSecretValue。 ■ CryptoServiceKeyUser：对专属KMS实例进行密码运算操作。	SecretUser
允许访问资源	权限策略被授权的具体对象。可以通过以下两种方法设置： ■ 方法一：在**可选资源**区域，选择已有资源，然后单击 < 图标。 ■ 方法二：在**已选资源区域**，单击 ＋ 图标，然后手动输入资源，最后单击**添加**。 **说明** 资源支持通配符（*）作为后缀。	secret/dataKey****

参数名称	参数说明	示例
网络控制规则	权限策略允许访问的网络类型和IP地址。 您可以在**可选规则**区域，选择已有规则，或者按照以下步骤创建并添加新规则。 a. 单击 ＋ 图标。 b. 在**创建网络访问规则**对话框，设置以下参数： ■ **名称**：网络访问规则的名称。 ■ **网络类型**：应用访问KMS的网络类型。 取值： ■ Public：适用于应用程序访问KMS的公网Endpoint。 ■ VPC：适用于应用程序访问KMS的VPC Endpoint。 ■ Private：适用于应用程序访问部署到VPC内的专属服务。 ■ **描述信息**：网络访问规则的详细信息。 ■ 允许地址：允许应用程序访问的网络地址。 取值： ■ 当网络类型为Public时：公网IP地址。 ■ 当网络类型为VPC时：VPC ID，以及VPC内的IP地址或者网段。 ■ 当网络类型为Private时：私网IP地址或者网段。 **说明** 多个IP地址间用半角逗号（,）分隔。 c. 单击**创建**。 d. 选择已有规则，然后单击 ＜ 图标。	■ **名称**：Network。 ■ **网络类型**：VPC。 ■ **描述信息**：访问指定的VPC。 ■ **VPC ID**：vpc-bp1drih00fwsrgz2p****。 ■ **来源IP**：192.168.0.0/16。

iii. 选择已有策略，然后单击 ＜ 图标。

iv. 单击**下一步**。

7. 检查应用接入点信息，然后单击**创建**。

为AAP绑定Client Key

认证方式为Client Key的AAP创建完成后，您可以为其绑定用于身份认证的Client Key。

1. 单击应用接入点名称。
2. 在**Client Key**区域，单击**创建Client Key**。
3. 在**创建Client Key**对话框，设置以下参数。

认证方式	说明	示例
Client Key加密口令	在您使用Client Key访问KMS时需要使用该口令对Client Key私钥文件进行解密，请妥善保管。	Test****
有效期	有效期时间范围外使用Client Key会访问失败。	2022年4月3日-2027年3月4日

4. 单击**确定**。
5. 在**创建成功**对话框，获取**口令**和**Client Key**。
   - **口令**：单击**Client Key解密口令**右侧的**复制**，获取口令。
   - **Client Key**：单击**下载Client Key**，获取Client Key的凭证信息。
     凭证信息包含Key ID和私钥（PrivateKeyData），示例如下：

```
{
  "KeyId": "KAAP.71be72c8-73b9-44e0-bb75-81ee51b4****",
  "PrivateKeyData": "MIIJwwIBAz****ICNXX/pOw=="
}
```

> **说明** KMS不会保存Client Key的私钥，因此您只能在创建Client Key时获取到加密的PKCS12文件（私钥文件），请妥善保管。

4.3. 应用程序接入凭据管家

应用程序可以采用多种方式接入凭据管家，动态使用凭据。

接入方式

应用程序可以使用多种开发工具接入凭据管家，不同开发工具说明及其应用场景如下：

开发工具	说明	应用场景
KMS SDK	KMS SDK帮助您构造HTTPS请求，更好地使用KMS API。	• 获取凭据值频率较低 • 需要对凭据进行创建、删除、增加新版本的凭据值等操作
凭据管家客户端	凭据管家客户端（SecretsManager Client）支持简单地配置客户端缓存频率，并定时刷新存储在凭据管家的凭据。	• 在客户端周期性或者频繁获取凭据值 • 需要根据凭据值进行相关操作
凭据管家JDBC客户端	凭据管家JDBC客户端（SecretsManager JDBC）支持在JDBC连接中简单地使用托管在凭据管家的凭据。	使用动态RDS凭据通过Java程序访问数据库
多种阿里云SDK的托管凭据插件	多种阿里云SDK的托管凭据插件是一个帮助您更有效通过动态RAM凭据快速使用阿里云服务的插件。	使用动态RAM凭据获取阿里云访问凭据访问阿里云服务

开发工具	说明	应用场景
凭据管家Kubernetes插件	凭据管家Kubernetes插件是一种以无代码方式快速集成凭据管家能力的插件。	以无代码的方式周期性更新配置

使用KMS SDK

以下以Java语言为例，为您介绍如何在应用程序中使用动态RDS凭据。如果您使用的不是动态RDS凭据而是其他类型的凭据，也可以使用此方法。

1. 获取密钥管理服务Java SDK的依赖声明。

   需要获取的版本请参见SDK概览。示例如下：

```
<dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-core</artifactId>
   <version>4.5.16</version>
 </dependency>
 <dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-kms</artifactId>
   <version>2.12.0</version>
 </dependency>
 <dependency>
   <groupId>com.alibaba</groupId>
   <artifactId>fastjson</artifactId>
   <version>1.2.9</version>
 </dependency>
 <dependency>
   <groupId>org.apache.commons</groupId>
   <artifactId>commons-lang3</artifactId>
   <version>3.4</version>
 </dependency>
```

2. 应用程序从凭据管家获取数据库账号和口令，建立数据库连接。

   示例代码如下：

```
package com.aliyun.kms.samples;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.http.ProtocolType;
import com.aliyuncs.kms.model.v20160120.GetSecretValueRequest;
import com.aliyuncs.kms.model.v20160120.GetSecretValueResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import org.apache.commons.lang3.tuple.Pair;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class RdsSecretSampleCode {
```

```
    private static final String MYSQL_JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String MSSQL_JDBC_DRIVER = "com.microsoft.sqlserver.jdbc.SQLSe
rverDriver";
    private static KmsClient kmsClient;
    static {
        kmsClient = KmsClient.getKMSClient("<regionId>", "<accessKeyId>", "<accessKeySe
cret>");
    }
    static class KmsClient {
        private DefaultAcsClient acsClient;
        private KmsClient(DefaultAcsClient acsClient) {
            this.acsClient = acsClient;
        }
        private static KmsClient getKMSClient(String regionId, String accessKeyId, Stri
ng accessKeySecret) {
            IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyId, a
ccessKeySecret);
            DefaultAcsClient client = new DefaultAcsClient(profile);
            return new KmsClient(client);
        }
    }
    // 通过凭据信息获取MySQL链接。
    public static Connection getMySQLConnectionBySecret(String secretName, String jdbcU
rl) throws ClassNotFoundException, SQLException, ClientException {
        Class.forName(MYSQL_JDBC_DRIVER);
        Pair<String, String> userAndPasswordPair = getUserAndPasswordPair(secretName);
        return DriverManager.getConnection(jdbcUrl, userAndPasswordPair.getKey(), userA
ndPasswordPair.getValue());
    }
    // 通过凭据信息获取MSSQL链接。
    public static Connection getMSSQLConnectionBySecret(String secretName, String jdbcU
rl) throws ClassNotFoundException, SQLException, ClientException {
        Class.forName(MSSQL_JDBC_DRIVER);
        Pair<String, String> userAndPasswordPair = getUserAndPasswordPair(secretName);
        return DriverManager.getConnection(jdbcUrl, userAndPasswordPair.getKey(), userA
ndPasswordPair.getValue());
    }
    // 通过凭据信息获取指定的数据库账号和口令。
    private static Pair<String, String> getUserAndPasswordPair(String secretName) throw
s ClientException {
        final GetSecretValueRequest request = new GetSecretValueRequest();
        request.setProtocol(ProtocolType.HTTPS);
        request.setAcceptFormat(FormatType.JSON);
        request.setMethod(MethodType.POST);
        request.setSecretName(secretName);
        GetSecretValueResponse response = kmsClient.acsClient.getAcsResponse(request);
        JSONObject secretDataJSON = JSON.parseObject(response.getSecretData());
        return Pair.of(secretDataJSON.getString("AccountName"), secretDataJSON.getStrin
g("AccountPassword"));
    }
}
```

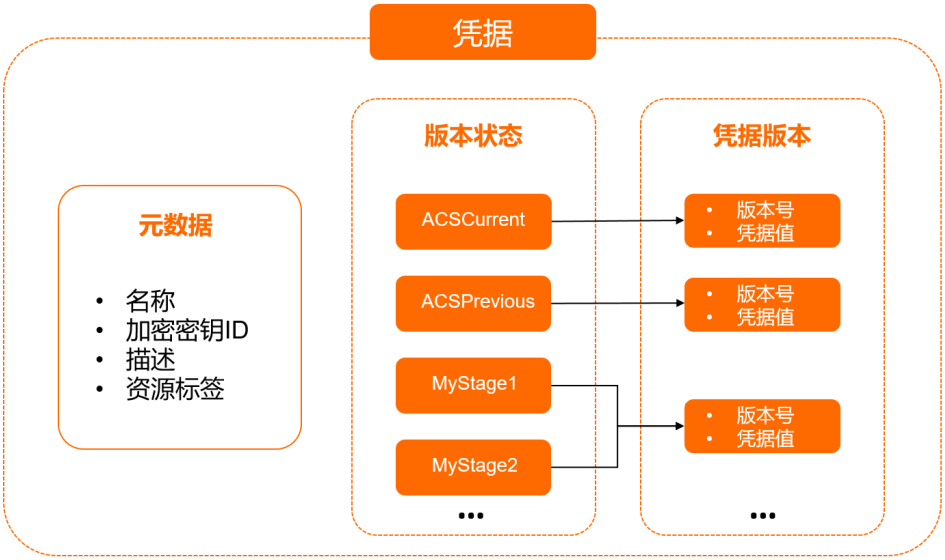
4.4. 通用凭据

4.4.1. 通用凭据概述

通用凭据（Generic Secret）是凭据管家支持的基础类型凭据。您可以使用通用凭据存储账号口令、访问密钥、OAuth密钥和Token、API Key等任意的敏感数据。通用凭据支持多个版本，方便您更新凭据的值。

了解通用凭据

通用凭据包含元数据、凭据版本（Secret Version）和版本状态（Version Stage）。



组成部分	说明
元数据	凭据的元数据包含以下部分： • 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。用于您指定自选密钥CMK标识符。 • 其他数据。例如：描述信息、资源标签等。
凭据版本	您存入到凭据中的敏感数据（凭据值）被存储为一个凭据版本。您可以通过凭据名称加上版本号读取任意版本的凭据值。每个版本号所标识的凭据版本只能被写入一次，不能被修改。
版本状态	凭据的版本可以通过版本状态来引用。凭据管家内建了**ACSCurrent**和**ACSPrevious**两个版本状态。默认情况下，PutSecretValue将新写入的版本设置为**ACSCurrent**，GetSecretValue读取**ACSCurrent**引用的版本。您也可以自定义版本状态。 **说明** 版本状态类似指针，遵循以下原则： • 每个版本状态只能引用一个版本，而每个版本可以被零个到多个版本状态引用。 • 当凭据的版本数超过上限时，最早的可回收版本（没有关联版本状态的版本）会被删除。

使用通用凭据

您可以通过以下方式使用通用凭据：

- 管理通用凭据
- 轮转通用凭据
- 在应用程序中使用通用凭据

4.4.2. 管理通用凭据

本文以命令行工具（CLI）为例，为您介绍如何管理通用凭据（Generic Secrets）。您也可以通过阿里云控制台或API管理通用凭据。

创建通用凭据

- 示例1：创建通用凭据时不指定加密密钥
  执行以下命令调用CreateSecret接口创建通用凭据，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返回以下结果：

```
{
  "Arn": "acs:kms:cn-shanghai:111760096384****:secret/db_cred",
  "RequestId": "ef0e4234-085c-4676-9ab6-159f2338aaf0",
  "SecretName": "db_cred",
  "SecretType": "Generic",
  "VersionId": "v1"
}
```

- 示例2：创建通用凭据时指定加密密钥
  执行以下命令调用CreateSecret接口创建通用凭据，KMS凭据管家使用指定的用户主密钥（CMK）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。
  > - 如果创建通用凭据时指定加密密钥，CreateSecret的调用者还需要具备指定CMK的kms:GenerateDataKey权限。

  KMS返回以下结果：

```
{
  "Arn": "acs:kms:cn-shanghai:111760096384****:secret/ssh_key",
  "RequestId": "ef0e4234-085c-4676-9ab6-159f2338aaf0",
  "SecretName": "ssh_key",
  "SecretType": "Generic",
  "VersionId": "v1"
}
```

查询通用凭据列表

执行以下命令调用ListSecrets接口查询通用凭据列表。

```
aliyun kms ListSecrets
```

KMS返回以下结果：

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

查询通用凭据的凭据值

执行以下命令调用GetSecretValue接口查询通用凭据的凭据值。

```
aliyun kms GetSecretValue --SecretName ssh_key
```

KMS返回以下结果：

```
{
  "CreateTime": "2021-07-08T05:51:50Z",
  "RequestId": "1415f5c7-ecb2-495e-8051-4cd466022c1f",
  "SecretData": "{\"test\":\"test\"}",
  "SecretDataType": "text",
  "SecretName": "ssh_key",
  "SecretType": "Generic",
  "VersionId": "v1",
  "VersionStages": {
    "VersionStage": [
      "ACSCurrent"
    ]
  }
}
```

查询通用凭据元数据信息

执行以下命令调用DescribeSecret接口查询通用凭据元数据信息。

```
aliyun kms DescribeSecret --SecretName ssh_key
```

KMS返回以下结果：

```
{
    "Arn": "acs:kms:cn-shanghai:111760096384****:secret/ssh_key",
    "SecretName": "ssh_key",
    "SecretType": "Generic",
    "EncryptionKeyId": "Example-CMK-Id",
    "Description": "",
    "CreateTime": "2020-01-22T03:57:09Z",
    "UpdateTime": "2020-01-22T03:57:09Z",
    "RequestId": "ca61398f-e61e-4552-aa7e-957955f6125s"
}
```

删除通用凭据

- 执行以下命令调用DeleteSecret接口删除通用凭据。
  - 默认情况下，删除的凭据可以在30天内恢复。

    ```
    aliyun kms DeleteSecret --SecretName ssh_key
    ```

    KMS返回以下结果：

    ```
    {
        "SecretName": "ssh_key",
        "RequestId": "3e54b02b-6461-46bb-afd5-dbd29d96eead",
        "PlannedDeleteTime": "2020-02-21T04:24:04.58616562Z"
    }
    ```

  - 删除凭据时，可以指定恢复窗口为7天。

    ```
    aliyun kms DeleteSecret --SecretName ssh_key --RecoveryWindowInDays 7
    ```

    KMS返回以下结果：

    ```
    {
        "SecretName": "ssh_key",
        "RequestId": "95ec4f18-8f97-4fd5-b7c6-1588979dse4s",
        "PlannedDeleteTime": "2020-01-29T04:25:14.165242211Z"
    }
    ```

- 删除凭据时，如果指定强制删除，则不允许恢复。

```
aliyun kms DeleteSecret --SecretName ssh_key --ForceDeleteWithoutRecovery true
```

KMS返回以下结果：

```
{
    "SecretName": "ssh_key",
    "RequestId": "75efc9c3-8e21-4e38-b6e4-486886be1546",
    "PlannedDeleteTime": "2020-01-22T12:28:22.006884739+08:00"
}
```

- 执行以下命令调用RestoreSecret接口，在恢复窗口期内恢复凭据。

```
aliyun kms RestoreSecret --SecretName ssh_key
```

KMS返回以下结果：

```
{
    "RequestId": "12770cee-92af-42f5-88e0-cbaa7e0c1254",
    "SecretName": "ssh_key"
}
```

轮转通用凭据

如果您需要对凭据进行轮转，请参见轮转通用凭据。

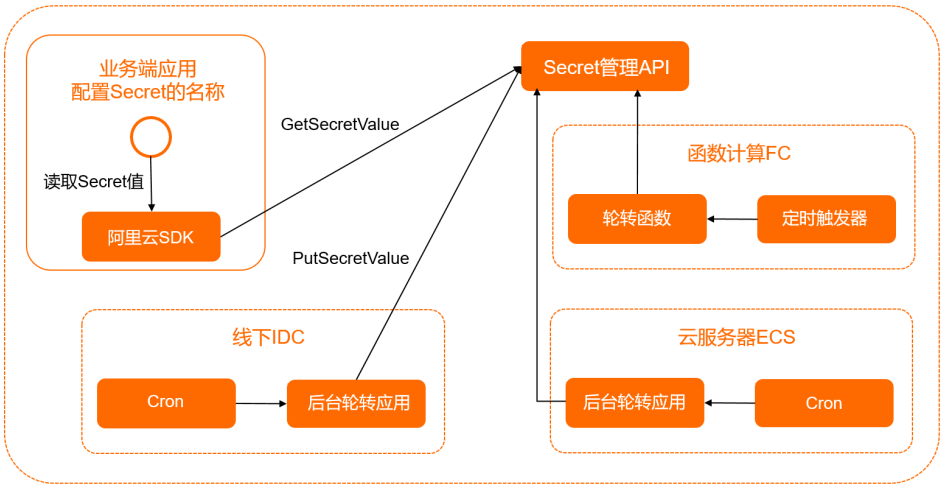
4.4.3. 轮转通用凭据

本文为您介绍轮转通用凭据的典型场景和操作。

典型场景

凭据消费者和凭据产生者可以通过凭据管家，实现凭据相关功能。

- 凭据管家：托管凭据。例如：Secret管理API对凭据进行托管。
- 凭据消费者：需要使用凭据的应用。例如：业务端应用调用GetSecretValue接口获取最新的凭据值。
- 凭据产生者：运维系统或管理员。例如：线下IDC、云服务器ECS和函数计算FC调用CreateSecret接口创建凭据，调用PutSecretValue接口轮转凭据。



前提条件

轮转通用凭据前，请确保您已经完成以下操作：

- 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**。
- 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**版本的凭据值。

通过一个API轮转通用凭据

如果目标系统中有已生效的凭据，当您需要将该凭据存入凭据管家时，可以调用一个API轮转通用凭据。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. 您可以使用CLI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
```

3. 调用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。

通过多个API轮转通用凭据

大多情况下，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。然后将新的凭据值注册到目标系统，将KMS凭据的待定版本轮转为当前版本。

您可以使用CLI轮转托管在凭据管家的数据库用户口令，具体操作如下：

1. 调用GetRandomPassword接口，使用轮转程序为新的口令产生一个随机的字符串。

```
aliyun kms GetRandomPassword --ExcludePunctuation true
```

KMS返回以下结果：

```
{
    "RequestId": "e36ca295-6e47-4dfb-9df1-48d19df41545",
    "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。
执行以下命令，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. 执行以下命令，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
```

5. 调用GetSecretValue接口返回被标记为**ACSCurrent**状态的版本的凭据值，即为最新的账号口令。

> **说明** 您也可以直接使用RDS凭据，由凭据管家帮您完成自动轮转的任务。更多信息，请参见动态RDS凭据概述。

4.5. 动态RDS凭据

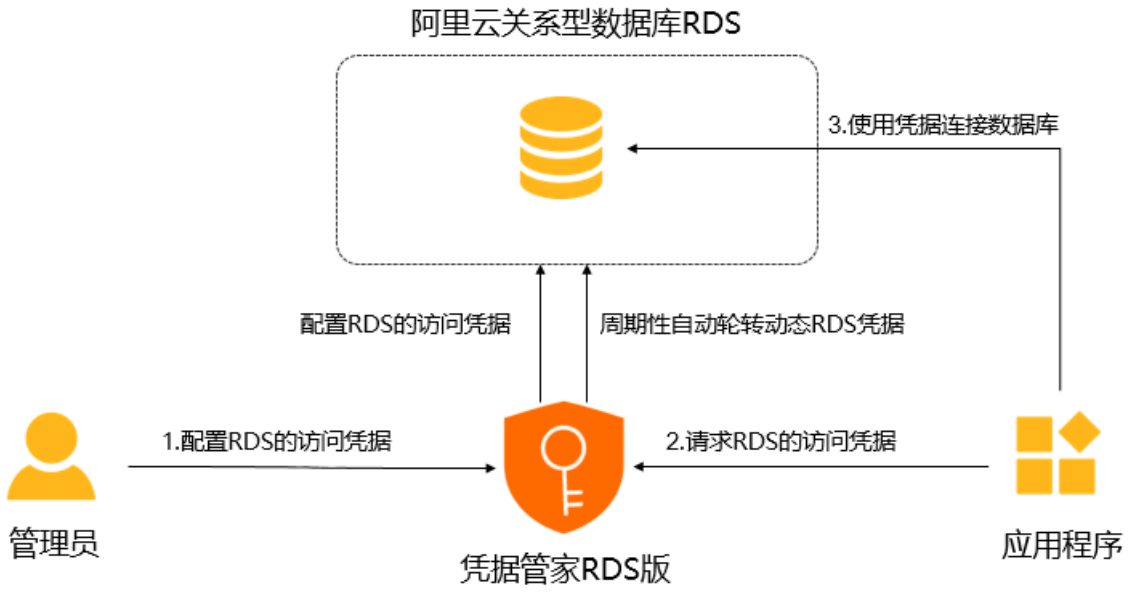
4.5.1. 动态RDS凭据概述

针对数据库的攻击是数据安全面临的主要威胁之一。针对阿里云关系型数据库RDS，凭据管家支持配置动态RDS凭据，对凭据进行全自动的定期轮换，降低业务数据面临的安全威胁。

产品架构

使用动态RDS凭据，应用程序将无需配置静态数据库账号口令。管理员在凭据管家创建全托管RDS凭据，设置自动轮转周期之后，应用程序调用GetSecretValue接口获取仅在下次轮转前有效的账号口令，用于访问RDS托管数据库。

RDS凭据轮转成功后，凭据关联的RDS实例的账号和口令将同步发生更新。请您不要删除凭据关联的RDS实例，避免凭据轮转失败。



使用限制

动态RDS凭据支持特定的RDS数据库：RDS MySQL、RDS MariaDB TX、RDS SQL Server（2017集群版除外）和RDS PostgreSQL。

使用动态RDS凭据

1. 创建动态RDS凭据
2. 监控动态RDS凭据轮转

3. 应用程序接入凭据管家

4.5.2. 管理动态RDS凭据

您可以创建动态RDS凭据，对RDS凭据进行全自动的定期轮换，从而降低RDS凭据泄露的安全风险。本文为您介绍如何通过KMS控制台创建、删除或还原动态RDS凭据。

前提条件

- 请确保您已经创建阿里云RDS实例。具体操作，请参见创建RDS MySQL实例。
- 当RAM用户或RAM角色管理凭据时，您需要将系统策略AliyunKMSSecretAdminAccess授予该RAM用户或RAM角色，使其拥有以下权限：
  - 使用凭据管家相关功能的权限。
  - 查询RDS实例、管理账号等相关功能的权限。
  - 创建托管RDS凭据使用的服务关联角色的权限。

创建动态RDS凭据

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，选择凭据所在的地域。
3. 在左侧导航栏，单击**凭据**。
4. 单击**创建凭据**。
5. 在**创建凭据**对话框，配置以下参数，然后单击**下一步**。
   - **选择凭据类型**：选择**托管RDS凭据**。
   - **凭据名称**：输入凭据名称。
   - **选择RDS实例**：选择阿里云账号下已有的RDS实例。
   - **设置凭据值**：选择托管方式，并设置凭据值。
     - **双账号托管**（推荐）：适用于程序化访问数据库场景。托管两个相同权限的账号，保证口令重置切换的瞬间，程序访问不被中断。
       - 单击**一键创建和授权**页签，配置账号名、选择数据库并指定权限。

         > **说明** 一键创建和授权不会立即为您配置新的账号，而是在您审核确认凭据信息之后进行配置。

       - 单击**导入已有账号**页签，选择用户名、配置口令。

         > **说明** 建议您将口令配置为创建RDS实例用户账号时对应的密码。如果导入的账号和口令不匹配，您可以在凭据首次轮转之后，获取正确的账号和口令。

     - **单账号托管**：适用于高权限账号或者人工运维账号托管场景。口令重置切换的瞬间，凭据的当前版本可能暂时无法使用。
       - 单击**一键创建和授权**页签，配置账号名、选择账号类型。
         您可以选择**普通账号**和**高权限账号**两种账号类型。当您选择**普通账号**时，还需选择数据库并指定权限。
       - 单击**导入已有账号**页签，选择用户名、配置口令。
   - **描述信息**：输入凭据的描述信息。

6. 在**创建凭据**对话框，选中**开启自动轮转**，配置**轮转周期**，然后单击**下一步**。

   > **说明** 如果无需自动轮转RDS凭据，请选择**关闭自动轮转**。

7. 在**创建凭据**对话框，审核凭据配置信息，然后单击**确定**。
8. 在**创建成功**对话框，单击**关闭**。

删除动态RDS凭据

删除RDS凭据前，请确认该RDS凭据已不被使用。

您可以选择计划删除凭据和立即删除凭据两种方式，删除不需要的动态RDS凭据。

1. 在目标RDS凭据右侧的**操作**列，选择**更多 > 计划删除凭据**。
2. 在**删除凭据**对话框，选择凭据删除方式，然后单击**确定**。
   - 选择**计划删除凭据**，然后设置**预删除周期（7~30天）**。系统将在预删除周期后删除凭据。
     在预删除周期内，您可以还原凭据，取消删除操作。具体操作，请参见还原动态RDS凭据。
   - 选择**立即删除凭据**，系统将立即删除凭据。

还原动态RDS凭据

当您选择了计划删除凭据的方式删除动态RDS凭据时，在预删除周期内，可以还原动态RDS凭据，取消删除操作。还原动态RDS凭据后，即可正常使用动态RDS凭据。

1. 在目标RDS凭据右侧的**操作**列，选择**更多 > 还原凭据**。
2. 在**还原凭据**对话框，单击**确定**。

4.5.3. 监控动态RDS凭据轮转

KMS凭据管家支持向云监控投递动态RDS凭据轮转事件，您可以通过云监控的控制台查询轮转事件、创建事件报警，从而实现事件报警、异常事件自动化处理等需求。

查询轮转事件

1. 登录云监控控制台。
2. 在左侧导航栏，单击**事件监控**。
3. 在**事件监控**页面，单击**事件查询**页签。
4. 从下拉列表中选择**密钥管理服务**，然后设置事件类型、事件名称和查询时间段。
5. 在事件列表中找到目标事件，然后单击右侧**操作**列的**查看详情**查询事件详细信息。

创建事件报警

您可以创建事件报警，及时了解动态RDS凭据轮转情况，并自动处理异常事件。例如：您可以监控动态RDS凭据轮转失败的事件，通过接入函数计算，自动修复故障。

1. 登录云监控控制台。
2. 在左侧导航栏，单击**事件监控**。
3. 在**事件监控**页面，单击**报警规则**页签，然后单击**创建事件报警**。
4. 在**创建/修改事件报警**面板，设置以下参数。

参数	说明
报警规则名称	报警规则的名称。例如：secrets_rotation_failed或secrets_rotation_success。
事件类型	选择**系统事件**。
产品类型	选择**密钥管理服务**。
事件类型	报警事件的类型。取值： ○ **异常**：仅对动态RDS凭据轮转失败事件进行报警。 ○ **Notification**：仅对动态RDS凭据轮转成功事件进行报警。 ○ **全部类型**：对所有动态RDS凭据轮转事件进行报警。
事件等级	您需要订阅的事件等级。取值： ○ **严重**：当动态RDS凭据轮转失败时选择该等级。 ○ **信息**：当动态RDS凭据轮转成功时选择该等级。

参数	说明
事件名称	选择您需要消费的事件名称。取值： ○ **托管凭据轮转失败**：仅对动态RDS凭据轮转失败事件进行报警。 ○ **托管凭据轮转成功**：仅对动态RDS凭据轮转成功事件进行报警。 ○ **全部事件**：对所有动态RDS凭据轮转事件进行报警。 **说明** 不建议您选择**全部事件**，建议您按照事件对业务的影响程度创建不同等级的事件通知。
资源范围	选择**全部资源**。此时任何资源发生相关事件，都会按照配置发送事件通知。
报警通知	○ **联系人组** 默认设置为**云账号报警联系人**。 ○ **通知方式** 取值： ■ **Warning（短信+邮箱+钉钉机器人）** ■ **Info（邮箱+钉钉机器人）**
消息处理方式	您可以配置**消息服务队列**、**函数计算**、**URL回调**（GET或POST）或**日志服务**，实现事件的自动化处理。

5. 单击**确定**。

报警通知内容

报警通知的格式为 `<资源类型>:<对资源执行的操作>:<执行结果>` 。当您为动态RDS凭据轮转事件设置报警后，系统会根据轮转情况为您发送事件报警通知。

- `Secret:RotateSecret:Failure` ：动态RDS凭据轮转失败事件。
  您可以从事件的 `content` 属性中查看动态RDS凭据轮转事件的相关信息，包括凭据关联的RDS实例ID（`RotationEntityArn` 字段），失败的原因（ `failureInfo` 字段）等。示例代码如下：

```
{
    "product": "KMS",
    "eventTime": "20180816T135935.689+0800",
    "level": "CRITICAL",
    "name": "Secret:RotateSecret:Failure",
    "regionId": "cn-hangzhou",
    "resourceId": " acs:kms:cn-hangzhou:123456789:secret/secretId",
    "status": "Failed",
    "content": {
        "eventId": "eventId",
        "secretName": "SecretName",
        "secretType": "Rds",
        "RotationEntityArn": "acs:rds:$regionId:$accountId:dbinstance/$dbinstanceid",
        "rotationStatus": "Invalid",
        "rotationSubType": "SingleUser",
        "failureInfo": {
            "errorCode": "Kms:ErrorCode",
            "errorMessage": "errorMessage"
        },
        "failureTime": "2012-03-12T05:55:36Z"
    },
    "ver": "1.0"
}
```

- `Secret:RotateSecret:Success`：动态RDS凭据轮转成功事件。
  示例代码如下：

```
{
    "product":"KMS",
    "instanceName":"secretId",
    "level":"INFO",
    "name":"Secret:RotateSecret:Success",
    "regionId":"cn-hangzhou",
    "resourceId":" acs:kms:cn-hangzhou:123456789:secret/secretId",
    "status":"Normal",
       "content":{
      "eventId": "eventId",
      "secretName": "SecretName",
      "secretType": "Rds",
      "RotationEntityArn": "acs:rds:$regionId:$accountId:dbinstance/$dbinstanceid",
      "rotationStatus": "Enabled",
      "secretSubType": "SingleUser",
      "successTime": "2012-03-12T05:55:36Z"
    },
    "ver":"1.0"
}
```

4.5.4. 动态RDS凭据服务关联角色

当您使用KMS的动态RDS凭据功能时，需要通过服务关联角色，访问阿里云关系型数据库RDS（Relational Database Service）相关资源。本文为您介绍动态RDS凭据服务关联角色（AliyunServiceRoleForKMSSecretsManagerForRDS）的权限策略、创建及删除操作。

权限说明

角色名称：AliyunServiceRoleForKMSSecretsManagerForRDS。

权限策略：AliyunServiceRolePolicyForKMSSecretsManagerForRDS。

权限说明：凭据管家使用该角色为您管理动态RDS凭据，完成RDS账号口令的轮转等任务。

```
{
  "Statement": [
    {
      "Action": [
        "rds:ResetAccountPassword",
        "rds:DescribeDBInstanceAttribute",
        "rds:DescribeAccounts"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "secretsmanager-rds.kms.aliyuncs.com"
        }
      }
    }
  ],
  "Version": "1"
}
```

创建服务关联角色

在创建动态RDS凭据的过程中，如果之前没有创建过服务关联角色，会自动进行创建。

删除服务关联角色

删除服务关联角色前，您需要先删除当前阿里云账号下的动态RDS凭据，然后在RAM控制台删除服务关联角色。具体操作，请参见删除RAM角色。

4.6. 动态RAM凭据

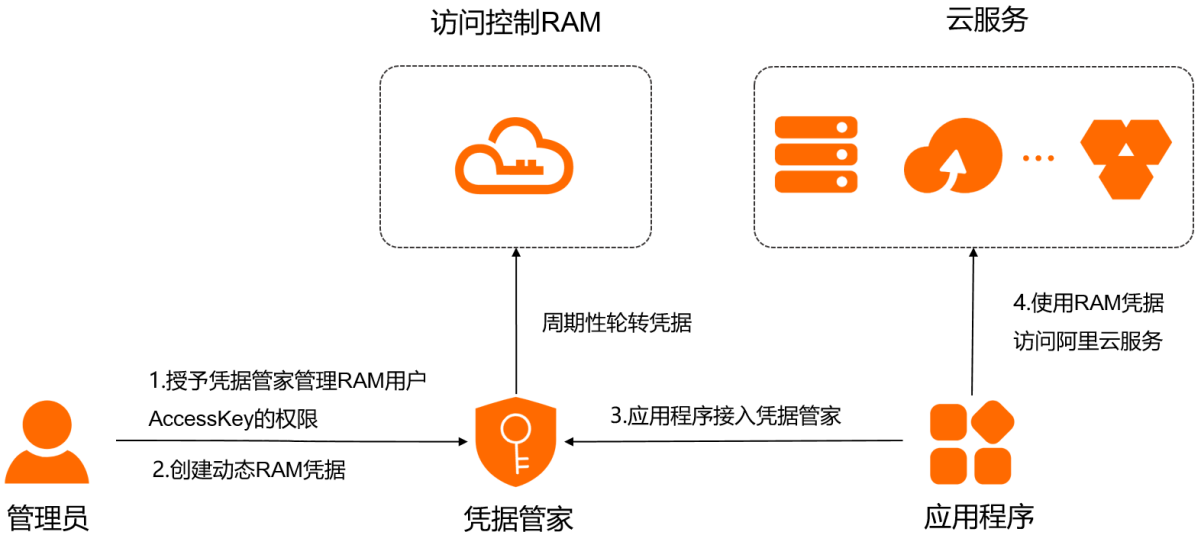
4.6.1. 动态RAM凭据概述

RAM凭据是指RAM用户的访问密钥（AccessKey），包括AccessKey ID和AccessKey Secret，用于RAM用户在调用阿里云API时完成身份验证。凭据管家可以对托管的RAM凭据进行全自动的定期轮换，将静态的RAM凭据动态化，从而降低RAM凭据泄漏的风险。除定期轮转外，凭据管家还支持立即轮转，在RAM凭据泄漏情况下快速更换AccessKey。

使用动态RAM凭据

使用动态RAM凭据，您无需在应用程序中配置AccessKey。管理员在凭据管家创建动态RAM凭据，设置自动轮转周期之后，应用程序调用GetSecretValue接口获取有效的AccessKey，用于调用阿里云API。

RAM凭据轮转成功后，凭据关联的RAM用户的AccessKey将同步发生更新。请您不要删除凭据关联的RAM用户，避免凭据轮转失败。



动态RAM凭据操作流程如下：

1. 授予凭据管家管理RAM用户AccessKey的权限。
2. 创建动态RAM凭据。
3. 应用程序接入凭据管家。
4. 使用RAM凭据访问阿里云服务。

使用限制

当前凭据管家只支持托管RAM用户的AccessKey，不支持托管阿里云账号的AccessKey。

4.6.2. 授予凭据管家管理RAM用户AccessKey的权限

当您使用凭据管家托管RAM凭据时，需要通过RAM服务角色为凭据管家授权，使凭据管家拥有管理RAM用户AccessKey的权限。本文为您介绍如何授予凭据管家管理RAM用户AccessKey的权限。

步骤一：创建自定义策略

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**权限管理 > 权限策略**。
3. 在**权限策略**页面，单击**创建权限策略**。
4. 选择**脚本编辑**并输入以下脚本。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ram:ListAccessKeys",
                "ram:CreateAccessKey",
                "ram:DeleteAccessKey",
                "ram:UpdateAccessKey"
            ],
            "Resource": "*"
        }
    ],
    "Version": "1"
}
```

5. 单击**下一步：编辑基本信息**，设置名称为AliyunKMSManagedRAMCrendentialsRolePolicy。
6. 单击**确定**。

步骤二：创建RAM角色并授权

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 角色**。
3. 创建RAM角色。
   i. 在**角色**页面，单击**创建角色**。
   ii. 在**创建角色**面板，选择可信实体类型为**阿里云服务**，然后单击**下一步**。
   iii. 选择角色类型为**普通服务角色**。
   iv. 设置**角色名称**为AliyunKMSManagedRAMCrendentialsRole。
   v. 选择受信服务为**密钥管理服务**。
   vi. 单击**完成**。
4. 为RAM角色授权。
   i. 单击**为角色授权**，被授权主体会自动填入。
   ii. 在**添加权限**面板，选择**自定义策略**，然后选中权限策略AliyunKMSManagedRAMCrendentialsRolePolicy。
   iii. 单击**确定**。
   iv. 单击**完成**。

4.6.3. 管理动态RAM凭据

您可以创建动态RAM凭据，对RAM凭据进行全自动的定期轮换，从而降低RAM凭据泄露的安全风险。本文为您介绍如何通过KMS控制台创建、删除或还原动态RAM凭据。

前提条件

- 阿里云账号和具有相关权限的RAM用户或RAM角色都可以管理动态RAM凭据。
  当RAM用户或RAM角色管理动态RAM凭据时，您需要将系统策略AliyunKMSSecretAdminAccess和AliyunRAMFullAccess授予该RAM用户或RAM角色。
- 管理动态RAM凭据前，您需要通过RAM服务角色授予凭据管家管理RAM用户AccessKey的权限。具体操

作，请参见授予凭据管家管理RAM用户AccessKey的权限。

- 请为待托管RAM凭据的RAM用户创建AccessKey。具体操作，请参见为RAM用户创建访问密钥。

创建动态RAM凭据

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RAM凭据托管的地域。
3. 在左侧导航栏，单击**凭据**。
4. 单击**创建凭据**。
5. 在**创建凭据**对话框，配置以下参数，然后单击**下一步**。
   - **选择凭据类型**：选择**托管RAM凭据**。
   - **选择RAM用户**：选择您要托管凭据的RAM用户，所选RAM用户需要至少有一个AccessKey。
   - **设置凭据值**：在AccessKey ID右侧输入对应的AccessKey Secret。

     > **说明** 建议您输入正确的AccessKey Secret。如果AccessKey Secret不正确，您将在RAM凭据首次轮转后获取一组新的AccessKey ID和AccessKey Secret。

   - **描述信息**：输入RAM凭据的描述信息。
6. 在**创建凭据**对话框，选中**开启自动轮转**，配置**轮转周期**，然后单击**下一步**。

   > **说明** 如果无需自动轮转RAM凭据，请选择**关闭自动轮转**。

7. 在**创建凭据**对话框，审核凭据配置信息，然后单击**确定**。
8. 在**创建成功**对话框，单击**关闭**。

   您也可以单击**查看凭据详情**查看凭据信息。

删除动态RAM凭据

删除RAM凭据前，请确认该RAM凭据已不被使用。

您可以选择计划删除凭据和立即删除凭据两种方式，删除不需要的动态RAM凭据。删除动态RAM凭据不会删除RAM用户的AccessKey。

1. 在左侧导航栏，单击**凭据**。
2. 在目标RAM凭据右侧的**操作**列，选择**更多 > 计划删除凭据**。
3. 在**删除凭据**对话框，选择凭据删除方式，然后单击**确定**。
   - 选择**计划删除凭据**，然后设置**预删除周期（7~30天）**。系统将在预删除周期后删除凭据。
     在预删除周期内，您可以还原凭据，取消删除操作。具体操作，请参见还原动态RAM凭据。
   - 选择**立即删除凭据**，系统将立即删除凭据。

还原动态RAM凭据

当您选择了计划删除凭据的方式删除动态RAM凭据时，在预删除周期内，可以还原动态RAM凭据，取消删除操作。还原动态RAM凭据后，即可正常使用动态RAM凭据。

1. 在左侧导航栏，单击**凭据**。
2. 在目标RAM凭据右侧的**操作**列，选择**更多 > 还原凭据**。
3. 在**还原凭据**对话框，单击**确定**。

4.7. 动态ECS凭据

4.7.1. 动态ECS凭据概述

ECS凭据是指您登录ECS实例时用于身份验证的口令和公私钥。凭据管家可以对托管的ECS凭据进行定期或人工轮转，为ECS实例更换高强度口令和公私钥，从而降低ECS凭据泄露的风险。

产品优势

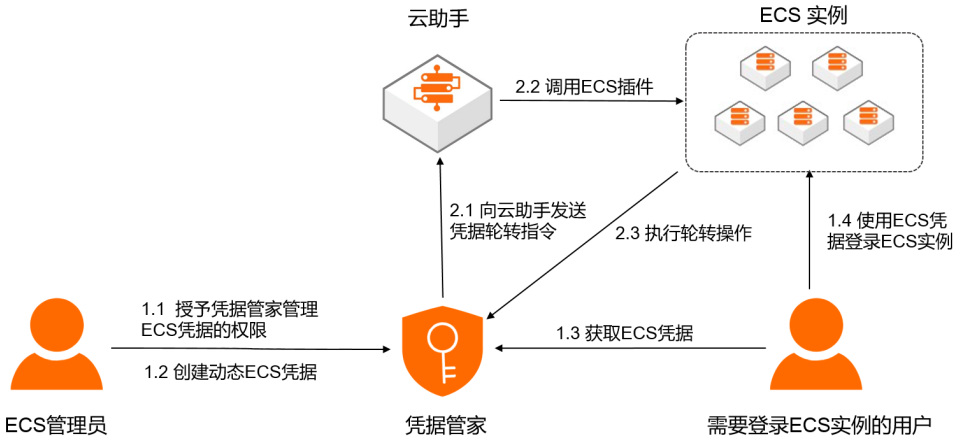
凭据管家通过以下技术手段，帮助您实现对ECS凭据的**集中安全管理**。

- **加密保护**：基于专有硬件的加密保护，保障凭据存储的安全。
- **权限管控**：基于RAM的细粒度权限管控，有效降低凭据泄露风险。
- **人工轮转**：当ECS凭据泄露情况下可以立即更换凭据，快速阻断入侵威胁。
- **定期轮转**：周期性为ECS实例更换高强度口令和公私钥，即使未察觉凭据泄露也能保障ECS实例的安全。

使用动态ECS凭据

使用动态ECS凭据，您需要授予凭据管家管理ECS实例口令和公私钥的相关权限，然后将ECS实例登录凭据托管到凭据管家。当您需要登录ECS实例时，从凭据管家获取实例对应的凭据，然后使用凭据登录ECS实例。

轮转ECS凭据时，您无需进行任何操作。当自动或立即轮转触发后，凭据管家会向云助手发起凭据轮转指令，云助手调用ECS实例上的插件完成凭据轮转。轮转成功后即可使用新凭据登录ECS实例。



动态ECS凭据操作流程如下：

1. 创建动态ECS凭据。
2. 监控动态ECS凭据轮转。
3. 使用ECS凭据登录ECS实例。

使用限制

Linux系统支持轮转口令和公私钥（SSH Key），Windows系统仅支持轮转口令。

4.7.2. 管理动态ECS凭据

您可以创建动态ECS凭据，对ECS凭据进行定期或人工轮转，从而降低ECS凭据泄露的安全风险。本文为您介绍如何通过KMS控制台创建、轮转、删除或还原动态ECS凭据。

前提条件

- 请确保您已经创建阿里云ECS实例。具体操作，请参见创建ECS实例。

- 阿里云账号和具有相关权限的RAM用户或RAM角色都可以管理动态ECS凭据。
  当RAM用户或RAM角色管理凭据时，需要将系统策略AliyunKMSSecretAdminAccess授予该RAM用户或RAM角色，使其拥有以下权限：
  - 使用凭据管家相关功能的权限。
  - 查询ECS实例的权限。
  - 创建动态ECS凭据使用的服务关联角色的权限。

  具体操作，请参见为RAM用户授权和为RAM角色授权。

创建动态ECS凭据

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，选择凭据所在的地域。

   > **说明** 凭据所在的地域需要跟待托管的ECS实例所在地域相同。

3. 在左侧导航栏，单击**凭据**。
4. 单击**创建凭据**。
5. 在**创建凭据**对话框，配置以下参数，然后单击**下一步**。
   - **选择凭据类型**：选择**托管ECS凭据**。
   - **凭据名称**：输入凭据名称。
   - **托管实例**：选择阿里云账号下已有的ECS实例。
   - **托管用户**：填写ECS实例上已有的用户名称，例如：root（Linux系统）或Administrator（Windows系统）。
   - **设置凭据值**：选择**口令**或**密钥对**，填入对应的初始值。

     > **说明** 如果初始值不正确，您将在ECS凭据首次轮转后获取到正确的口令或密钥对。

   - **描述信息**：输入ECS凭据的描述信息。
6. 在**创建凭据**对话框，选中**开启自动轮转**，配置**轮转周期**，然后单击**下一步**。

   > **说明** 如果无需自动轮转ECS凭据，请选择**关闭自动轮转**。

7. 在**创建凭据**对话框，审核凭据配置信息，单击**确定**。
   创建成功后，您可以在凭据列表中查看**凭据类型**为**托管ECS凭据**的动态ECS凭据。

轮转动态ECS凭据

当ECS凭据泄露时，您可以通过控制台立即轮转功能快速轮转动态ECS凭据，阻断入侵威胁。

1. 单击目标ECS凭据名称，然后单击**立即轮转**。
2. 在**提示**对话框，选择**是否使用自定义凭据**。
   - 打开开关：使用自定义凭据并指定新凭据值。
   - 关闭开关：KMS将自动创建32位的随机口令或RSA2048公私钥对。
3. 单击**确认轮转**。
4. 在**已触发轮转**对话框，单击**关闭**。

删除动态ECS凭据

删除ECS凭据前，请确保该ECS凭据已不被使用。

您可以选择计划删除凭据和立即删除凭据两种方式，删除不需要的动态ECS凭据。删除动态ECS凭据不会影响ECS实例上已配置的口令或公私钥。

1. 在目标ECS凭据右侧的**操作**列，选择**更多 > 计划删除凭据**。
2. 在**删除凭据**对话框，选择凭据删除方式，然后单击**确定**。
   - 选择**计划删除凭据**，然后设置**预删除周期（7~30天）**。系统将在预删除周期后删除凭据。
     在预删除周期内，您可以还原凭据，取消删除操作。具体操作，请参见还原动态ECS凭据。
   - 选择**立即删除凭据**，系统将立即删除凭据。

还原动态ECS凭据

当您选择了计划删除凭据的方式删除动态ECS凭据时，在预删除周期内，可以还原动态ECS凭据，取消删除操作。还原动态ECS凭据后，即可正常使用动态ECS凭据。

1. 在目标ECS凭据右侧的**操作**列，选择**更多 > 还原凭据**。
2. 在**还原凭据**对话框，单击**确定**。

4.7.3. 监控动态ECS凭据轮转

KMS凭据管家支持向云监控投递动态ECS凭据轮转事件，您可以通过云监控的控制台查询轮转事件、创建事件报警，从而实现事件报警、异常事件自动化处理等需求。

查询轮转事件

1. 登录云监控控制台。
2. 在左侧导航栏，单击**事件监控**。
3. 在**事件监控**页面，单击**事件查询**页签。
4. 从下拉列表中选择**密钥管理服务**，然后设置事件类型、事件名称和查询时间段。
5. 在事件列表中找到目标事件，然后单击右侧**操作**列的**查看详情**查询事件详细信息。

创建事件报警

您可以创建事件报警，及时了解动态ECS凭据轮转情况，并自动处理异常事件。例如：您可以监控动态ECS凭据轮转失败的事件，通过接入函数计算，自动修复故障。

1. 在**事件监控**页面，单击**报警规则**页签，然后单击**创建事件报警**。
2. 在**创建/修改事件报警**面板，设置以下参数。

参数	说明
报警规则名称	报警规则的名称。例如：secrets_rotation_failed或secrets_rotation_success。
事件类型	选择**系统事件**。
产品类型	选择**密钥管理服务**。
事件类型	报警事件的类型。取值： ○ **异常**：仅对动态ECS凭据轮转失败事件进行报警。 ○ **Notification**：仅对动态ECS凭据轮转成功事件进行报警。 ○ **全部类型**：对所有动态ECS凭据轮转事件进行报警。
事件等级	您需要订阅的事件等级。取值： ○ **严重**：当动态ECS凭据轮转失败时选择该等级。 ○ **信息**：当动态ECS凭据轮转成功时选择该等级。
事件名称	选择您需要消费的事件名称。取值： ○ **托管凭据轮转失败**：仅对动态ECS凭据轮转失败事件进行报警。 ○ **托管凭据轮转成功**：仅对动态ECS凭据轮转成功事件进行报警。 ○ **全部事件**：对所有动态ECS凭据轮转事件进行报警。 **说明** 不建议您选择**全部事件**，建议您按照事件对业务的影响程度创建不同等级的事件通知。
资源范围	选择**全部资源**。此时任何资源发生相关事件，都会按照配置发送事件通知。
报警通知	○ **联系人组** 默认设置为**云账号报警联系人**。 ○ **通知方式** 取值： ■ **Warning（短信+邮箱+钉钉机器人）** ■ **Info（邮箱+钉钉机器人）**
消息处理方式	您可以配置**消息服务队列**、**函数计算**、**URL回调**（GET或POST）或**日志服务**，实现事件的自动化处理。

3. 单击**确定**。

报警通知内容

报警通知的格式为 `<资源类型>:<对资源执行的操作>:<执行结果>` 。当您为动态ECS凭据轮转事件创建报警后，系统会根据轮转情况为您发送事件报警通知。

- `Secret:RotateSecret:Failure` ：动态ECS凭据轮转失败事件。
  您可以从事件的 `content` 属性中查看失败原因（failureInfo字段）等。示例代码如下：

```
{
    "product": "KMS",
    "eventTime": "20180816T135935.689+0800",
    "level": "CRITICAL",
    "name": "Secret:RotateSecret:Failure",
    "regionId": "cn-hangzhou",
    "resourceId": "acs:kms:cn-hangzhou:188989715694****:secret/secretName",
    "status": "Failed",
    "content": {
        "eventId": "eventId",
        "secretName": "SecretName",
        "secretType": "ECS",
        "RotationEntityArn": "acs:kms:cn-hangzhou:188989715694****:secret/secretName",
        "rotationStatus": "Invalid",
        "rotationSubType": "Password",
        "failureInfo": {
            "errorCode": "Kms:ErrorCode",
            "errorMessage": "errorMessage"
        },
        "failureTime": "2012-03-12T05:55:36Z"
    },
    "ver": "1.0"
}
```

- `Secret:RotateSecret:Success` ：动态ECS凭据轮转成功事件。
  示例代码如下：

```
{
    "product":"KMS",
    "instanceName":"secretId",
    "level":"INFO",
    "name":"Secret:RotateSecret:Success",
    "regionId":"cn-hangzhou",
    "resourceId":"acs:kms:cn-hangzhou:188989715694****:secret/secretName",
    "status":"Normal",
       "content":{
      "eventId": "eventId",
      "secretName": "SecretName",
      "secretType": "ECS",
      "RotationEntityArn": "acs:kms:cn-hangzhou:188989715694****:secret/secretName",
      "rotationStatus": "Enabled",
      "secretSubType": "Password",
      "successTime": "2012-03-12T05:55:36Z"
    },
    "ver":"1.0"
}
```

4.7.4. 动态ECS凭据服务关联角色

当您使用KMS的动态ECS凭据功能时，需要通过服务关联角色授予凭据管家访问ECS、云助手等相关资源的权限。本文为您介绍动态ECS凭据服务关联角色（AliyunServiceRoleForKMSSecretsManagerForECS）的权限策略、创建及删除操作。

权限说明

角色名称：AliyunServiceRoleForKMSSecretsManagerForECS

权限策略：AliyunServiceRolePolicyForKMSSecretsManagerForECS

权限说明：凭据管家使用该角色为您管理动态ECS凭据，完成ECS口令、公私钥的轮转任务。

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeInstances",
        "ecs:InvokeCommand",
        "ecs:DescribeInvocationResults"
      ],
      "Resource": [
        "acs:ecs:*:*:instance/*",
        "acs:ecs:*:*:command/cmd-ACS-KMS-RotateECSSecret*"
      ]
    },
    {
      "Action": [
        "kms:ListSecretVersionIds",
        "kms:GetSecretValue",
        "kms:DescribeSecret",
        "kms:PutSecretValue",
        "kms:UpdateSecretVersionStage"
      ],
      "Resource": "acs:kms:*:*:secret/acs/ecs/*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "secretsmanager-ecs.kms.aliyuncs.com"
        }
      }
    }
  ]
}
```

创建服务关联角色

在创建动态ECS凭据的过程中，如果之前没有创建过服务关联角色，会自动进行创建。

删除服务关联角色

删除服务关联角色前，您需要先删除当前阿里云账号下的动态ECS凭据，然后在RAM控制台删除服务关联角色。具体操作，请参见删除RAM角色。

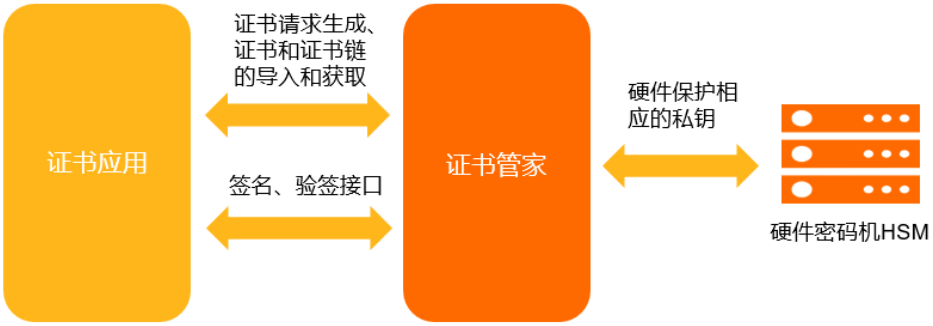
5.证书管家

5.1. 证书管家概述

证书管家为您提供高可用、高安全的密钥和证书托管能力，以及签名验签能力。

产品架构

证书应用（使用数字证书的应用程序）可以通过证书管家生成证书请求，导入或导出数字证书及其证书链。证书管家后端通过硬件密码机HSM（Hardware Security Module）保护用户数字证书密钥的安全。



产品优势

- 密钥安全存储
  证书管家使用托管密码机保障证书密钥的产生、存储安全。关于托管密码机的更多信息，请参见托管密码机概述。
- 生命周期管理
  支持管理密钥和证书，可以生成证书请求、导入证书和证书链、检查证书链签名有效性，并检查证书有效性。
- 多种公钥算法
  支持RSA_2048、EC_P256和EC_SM2等多种公钥算法，支持证书格式X.509 v3，符合PKI/CA相关标准。
- API便于集成
  支持多个API接口，帮助您在开发环境高效集成证书服务，快速进行产品部署，为您提供快速开发上线证书相关功能的能力。

5.2. 证书管家快速入门

您可以使用证书管家托管密钥和证书，并进行签名验签。本文为您介绍如何创建、下载、导入和使用证书。

步骤一：创建并下载证书

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，选择证书所在的地域。
3. 在左侧导航栏，单击**证书**。
4. 单击**创建证书**。
5. 在**创建证书**对话框，设置以下参数。

参数	说明
名称	证书使用主体名称。

参数	说明
国家/地区	使用ISO 3166-1的二位国家代码。例如：CN代表中国。
省/市	省、直辖市、自治区或特别行政区名称。
城市	城市名称。
公司名称	企业、单位、组织或机构的法定名称。 单击右侧加号，可以添加多个公司名称。
部门名称	部门名称。 单击右侧加号，可以添加多个部门名称。
邮箱	证书持有者或管理者邮箱。
主体别名	当证书为DV证书时，可使用主体别名生成多域名证书请求。 单击右侧加号，输入主体别名，然后单击 。
密钥类型	取值： ○ RSA_2048 ○ EC_SM2 ○ EC_P256 **说明** 请根据证书应用系统密钥算法支持情况选择。EC_P256安全性较高，RSA_2048兼容性较好，但部分应用系统将在2030年12月31日停止支持RSA_2048密钥。
私钥可否导出	证书私钥是否需要导出使用。取值： ○ 是：证书私钥需要导出使用。 ○ 否：证书私钥不需要导出使用。建议选择否，以便使用更高安全级别的密钥保护。

6. 单击**创建证书**。
7. 在**创建证书成功**对话框，单击**下载证书请求**。
8. 单击**确定**。

步骤二：获取CA颁发的证书

将步骤一下载的.csr格式的证书请求文件提交给CA机构，获取正式的证书和证书链。

步骤三：导入证书

1. 在证书列表页面，找到目标证书，在**操作**列选择**更多 > 导入证书**。
2. 在**导入证书**对话框，输入或上传步骤二获取的证书和证书链。
3. 单击**确定**。

导入证书成功后，证书状态为**启用中**，您可以使用证书进行密钥管理、签名验签等操作。

步骤四：证书签名

- 方法一：调用CertificatePrivateKeySign接口使用指定证书生成数字签名。
- 方法二：通过KMS SDK使用指定证书生成数字签名。关于KMS SDK的更多信息，请参见SDK概览。Java代码示例如下：

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.kms.model.v20160120.CertificatePrivateKeySignRequest;
import com.aliyuncs.kms.model.v20160120.CertificatePrivateKeySignResponse;
import org.apache.commons.codec.binary.Base64;
/**
 *  @param client 表示Alibaba Cloud SDK Client，详见Alibaba Cloud SDK for Java文档。
 *  @param certId 表示证书ID，指定要使用的证书。
 *  @param sigAlg 表示数字签名算法，详见KMS CertificatePrivateKeySign接口文档。
 *  @param message 表示待签名内容，需要小于等于4KB。
 */
public byte[] doSignByCertificate(DefaultAcsClient client, String certId, String sigAlg,
byte[] message) throws ClientException {
    String msgB64 = Base64.encodeBase64String(message); // 对待签名内容进行Base64编码。
    CertificatePrivateKeySignRequest request = new CertificatePrivateKeySignRequest();
    request.setCertificateId(certId);
    request.setAlgorithm(sigAlg);
    request.setMessage(msgB64);
    CertificatePrivateKeySignResponse response = client.getAcsResponse(request);
    String sigB64 = response.getSignatureValue();
    return Base64.decodeBase64(sigB64); // 对返回数据进行Base64解码获取签名值数据。
}
```

步骤五：证书验签

- 方法一：调用CertificatePublicKeyVerify接口使用指定证书验证数字签名。
- 方法二：通过KMS SDK使用指定证书验证数字签名。关于KMS SDK的更多信息，请参见SDK概览。Java代码示例如下：

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.kms.model.v20160120.CertificatePublicKeyVerifyRequest;
import com.aliyuncs.kms.model.v20160120.CertificatePublicKeyVerifyResponse;
import org.apache.commons.codec.binary.Base64;
/**
 *  @param client 表示Alibaba Cloud SDK Client，详见Alibaba Cloud SDK for Java文档。
 *  @param certId 表示证书ID，指定要使用的数字证书。
 *  @param sigAlg 表示数字签名算法，详见KMS CertificatePrivateKeySign接口参考。
 *  @param message 表示待验证内容，需要小于等于4KB。
 *  @param signature 表示待验证内容的数字签名。
 */
public Boolean doVerifyByCertificate(DefaultAcsClient client, String certId, String sigAl
g, byte[] message, byte[] signature) throws ClientException {
    String msgB64 = Base64.encodeBase64String(message); // 对待验证内容进行Base64编码。
    String sigB64 = Base64.encodeBase64String(signature); // 对签名值进行Base64编码。
    CertificatePublicKeyVerifyRequest request = new CertificatePublicKeyVerifyRequest();
    request.setCertificateId(certId);
    request.setAlgorithm(sigAlg);
    request.setMessage(msgB64);
    request.setSignatureValue(sigB64);
    CertificatePublicKeyVerifyResponse response = client.getAcsResponse(request);
    return response.getSignatureValid();
}
```

5.3. 导入密钥和证书

当您需要将其他证书应用系统的密钥迁移并托管至证书管家时，需要先从其他证书应用系统导出PFX/PKCS12格式的密钥文件，然后将其导入到密钥管理服务控制台。

操作步骤

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，选择证书所在的地域。
3. 在左侧导航栏，单击**证书**。
4. 单击**导入密钥及证书**。
5. 在**导入密钥和证书**对话框，设置**PKCS12文件保护口令**，输入或上传PKCS12格式文件。
6. 单击**确定**。

> **说明** 导入的密钥及证书支持再次导出。如果您需要使用更高安全级别的密钥安全等级，请在创建证书时将**私钥可否导出**设置为否。具体操作，请参见创建并下载证书。

5.4. 禁用、吊销或删除证书

您可以使用证书管家托管密钥和证书，并进行签名验签。当您不再使用该证书时，可以禁用、吊销或删除证书。

前提条件

请确保您的证书处于已启用状态。

禁用证书

当您暂时无需使用证书时，可以禁用证书。禁用后的证书信息将保留，后续您可以根据需求再次启用证书。

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，选择证书所在的地域。
3. 在左侧导航栏，单击**证书**。
4. 单击目标证书右侧**操作**列的**禁用**。
5. 在**禁用证书**对话框，单击**确定**。

> **说明** 您也可以单击**操作**列的**启用**，再次启用证书。

吊销证书

当CA机构作废了证书时，您可以将密钥管理服务控制台中证书的状态设置为**已吊销**。吊销后的证书信息将保留，后续您仅可以查看证书信息，但不可以启用或禁用证书。

1. 单击目标证书右侧**操作**列的**已吊销**。
2. 在**吊销证书**对话框，单击**确定**。

删除证书

当您无需使用证书管家管理证书时，可以删除证书。删除证书前，请确保证书没有用于签名验签。

1. 在目标证书右侧**操作**列选择**更多 > 删除证书**。
2. 在**删除证书**对话框，单击**确定**。

6.云产品与KMS的集成

6.1. 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

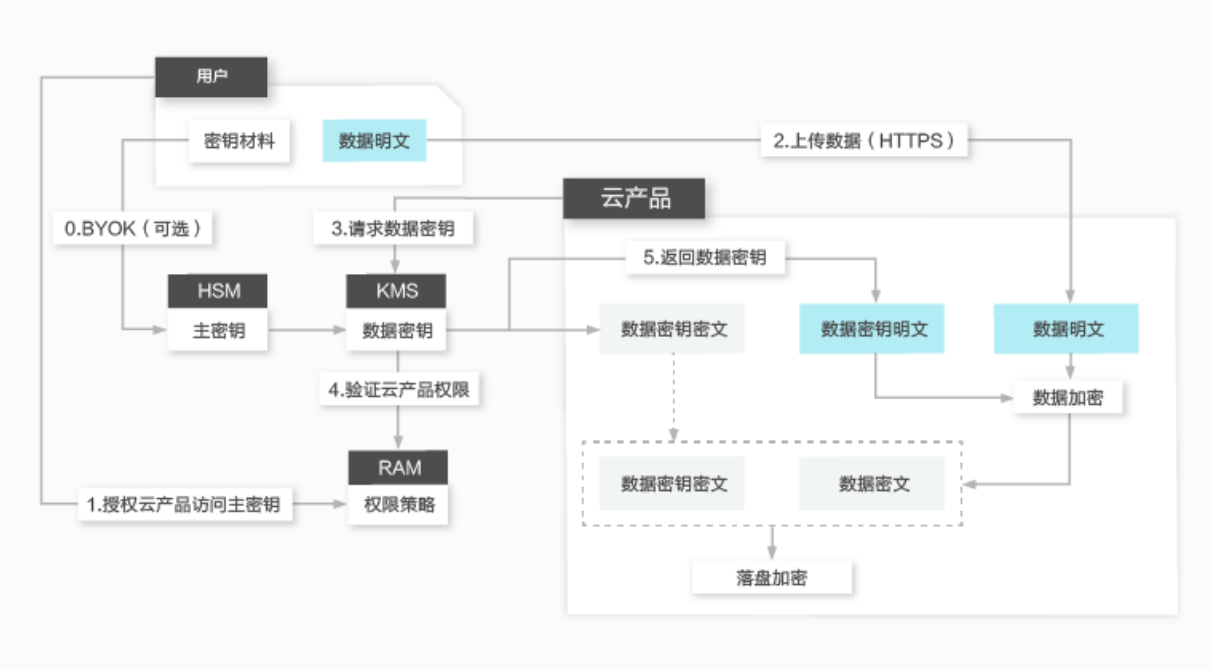
您可以根据数据保护需求，选择托管在KMS中的不同类型密钥用于服务端加密：

- 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。

6.2. 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）对不同场景和类型的数据进行加密保护。

工作负载数据加密

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加密用户数据，也支持使用用户自选密钥加密用户数据。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

服务名称	描述	相关文档
容器服务Kubernetes版ACK	容器服务中的以下两类工作负载数据支持基于KMS的服务端加密： • Kubernetes Secrets 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。 • 存储卷 存储卷可以是云盘、OSS、或者NAS卷。针对各类存储卷，可以采用特定存储类型的服务端KMS加密方式。例如：您可以创建一个加密云盘，随后挂载为Kubernetes存储卷。	使用阿里云KMS进行Secret的落盘加密
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。	无
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

持久化存储数据加密

服务名称	描述	相关文档
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
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。	服务器端加密
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

服务名称	描述	相关文档
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

数据库加密

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 • PostgreSQL：云盘加密
云数据库MongoDB	提供透明数据加密TDE功能，加密方式和RDS类似。	设置透明数据加密TDE
云数据库PolarDB	提供透明数据加密TDE功能，加密方式和RDS类似。	• PolarDB MySQL：设置透明数据加密TDE • PolarDB O引擎：设置透明数据加密TDE • PolarDB PostgreSQL：设置透明数据加密TDE
云数据库OceanBase	提供透明数据加密TDE功能，加密方式和RDS类似。	TDE 透明加密
云数据库Redis	提供透明数据加密TDE功能，加密方式和RDS类似。	开启透明数据加密TDE

日志数据加密

服务名称	描述	相关文档
操作审计 Actiontrail	创建单账号跟踪或者多账号跟踪时，如果选择投递到OSS，可以在操作审计控制台直接加密操作事件。	• 创建单账号跟踪 • 创建多账号跟踪
日志服务 SLS	日志服务支持通过KMS对数据进行加密存储，提供数据静态保护能力。	数据加密

大数据与人工智能

服务名称	描述	相关文档
大数据计算 MaxCompute	MaxCompute支持使用服务密钥或者自选KMS密钥进行数据加密。	数据加密
机器学习PAI	机器学习PAI产品架构中，计算引擎、容器服务、数据存储等各个数据流转环节，相应的云服务均可以配置服务端加密，保护数据的安全与隐私。	无

更多场景

服务名称	描述	相关文档
内容分发网络 CDN	当使用OSS Bucket作为源站时，可以使用基于OSS的服务端加密保护分发内容。请参考CDN文档，配置CDN访问加密Bucket。	OSS私有Bucket回源
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。	阿里云私有加密
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
云效代码管理 Codeup	云效代码管理Codeup使用KMS服务密钥对用户提交的源码进行加密，确保云端代码数据不泄露。在代码加密过程中，每个代码库都有对应的数据密钥（DK），并通过信封加密机制对代码数据进行加密，在获得用户密钥授权的前提下： • 当用户使用Git客户端或页面提交Git数据时，Codeup对收到的代码数据执行加密，并持久化存储加密后的数据。 • 当用户使用Git客户端或页面访问Git数据时，Codeup根据授权对保存的加密数据进行解密，并把原始数据返回给用户。 **说明** 启用Codeup代码加密服务，几乎不影响代码托管性能。	代码仓库加密

7.API参考

7.1. API概览

请根据使用场景选择对应的API。

场景大类	细分场景	描述	API选型	API区别
用户主密钥	用户主密钥管理	管理用户主密钥生命周期，查询用户主密钥相关信息。	KMS API	无
	密钥版本管理	对用户主密钥进行密钥轮转，查询密钥版本信息。		
	别名管理	管理别名的声明周期，查询别名相关信息。 别名是独立的对象，必须和唯一的用户主密钥进行绑定，从而可以在特定API中代替 `KeyId` 参数来指代用户主密钥。		
	密码运算	使用密钥进行密码运算，例如：数据的加密和解密。	• KMS API • 专属KMS API	• 当您使用专属KMS对数据进行密码运算时，需要选用专属KMS API，否则需选用KMS API。二者虽然功能相似，但是数据格式不同，因此不能混用。 • 专属KMS API支持通过专属KMS实例的VPC地址访问。共享KMS API支持通过共享KMS网关访问。
凭据管家	凭据管理	将敏感信息通过凭据进行托管和保护，同时提供凭据分发和轮转能力。	KMS API	无
证书管家	证书管理	对证书进行创建、删除、更新、查询、签名验签等操作。		
其他	标签管理	管理资源关联的标签生命周期，查询指定资源的标签信息。		
	公共接口	开通KMS、查询KMS状态和可用地域等。		

8.SDK参考

8.1. 概述

密钥管理服务KMS（Key Management Service）提供了多种类型的开发工具，您可以根据业务需要选择合适的开发工具。

开发工具	说明
KMS SDK	KMS SDK帮助您构造HTTPS请求，更好地使用KMS API。 KMS SDK特点如下： • 结合阿里云SDK核心库（SDK Core）使用，SDK接口和KMS API一一对应。 • 适用于KMS的所有业务，包括数据加密解密、数字签名验签和动态获取凭据等业务场景。
加密SDK	加密SDK（Encryption SDK）是一个建立在KMS API之上的客户端密码库，其特点如下： • 基于KMS API封装了业务逻辑、最佳实践和设计模式，方便开发者在业务系统中进行集成。 • 适用于数据加密解密和数字签名验签的业务场景。
凭据管家客户端	凭据管家客户端（SecretsManager Client）支持简单地配置客户端缓存频率，并定时刷新存储在凭据管家的凭据，其特点如下： • 基于KMS凭据管家API封装了业务逻辑、最佳实践和设计模式，方便开发者在业务系统中进行集成。 • 适用于动态使用凭据管家中凭据的业务场景，不再使用硬编码的方式。
凭据管家JDBC客户端	凭据管家JDBC客户端（SecretsManager JDBC）支持在JDBC连接中简单地使用托管在凭据管家的凭据，其特点如下： • 基于KMS凭据管家动态RDS凭据封装了业务逻辑、最佳实践和设计模式，方便开发者在业务系统中集成。 • 主要适用于数据库应用中动态使用托管在凭据管家中的动态RDS凭据，告别对数据库账号密码机密信息的硬编码。
多种阿里云SDK的托管凭据插件	多种阿里云SDK的托管凭据插件是一个帮助您更有效通过托管RAM凭据快速使用阿里云服务的插件，其特点如下： • 借助托管RAM凭据托管访问凭据和轮转能力，更易于开发者在业务系统中快速集成解决访问凭据问题。 • 基于应用接入点构建应用级别访问控制，确保访问凭据安全性。
专属KMS SDK	专属KMS SDK帮助您通过简单的编程快速访问专属KMS API，实现加密解密、签名验签等业务诉求，其特点如下： • 基于专属KMS API封装了业务逻辑、最佳实践和设计模式，方便开发者在业务系统中进行集成。 • 使用ClientKey，提供更安全的身份控制。

8.2. 加密SDK

8.2.1. 加密SDK概述

加密SDK（Encryption SDK）是一个客户端密码库，通过与密钥管理服务KMS（Key Management Service）结合使用，帮助您快速实现数据的加解密、签名验签功能。

功能特性

- 集成KMS托管保护密钥，满足安全与合规要求。
- 极简接口，支持一话一密（每个消息使用不同的数据密钥）、签名验签等密码运算功能。
- 可扩展设计模式，支持多话一密等定制化使用能力。

产品优势

- 封装最佳实践，助力极简代码开发。
  加密SDK为每个待加密的数据创建唯一的数据密钥（Data Key），遵循密码设计的最佳实践。每次数据加密使用一个数据密钥的最佳实践，也叫一话一密。
- 丰富的业务兼容能力。
  支持多种加密算法、工作模式、填充方式，满足不同的业务、迁移等需求。
- 数据加解密跨地域可用。
  加密SDK可配置不同地域的用户主密钥，一行代码加密数据后，可在不同地域解密使用，从而保证数据跨地域可用和灾备能力。

快速入门

以下为您提供不同语言的加密SDK快速入门：

- 加密SDK快速入门（Java）
- 加密SDK快速入门（Python）

8.2.2. 快速入门

8.2.2.1. 加密SDK快速入门（Java）

加密SDK（Encryption SDK）是一个客户端密码库，通过与密钥管理服务KMS（Key Management Service）结合使用，帮助您快速实现数据的加解密、签名验签功能。本文以Java语言为例，为您介绍如何快速使用加密SDK进行数据加解密。

背景信息

您可以访问alibabacloud-encryption-sdk-java，查看代码示例。

在本地安装加密SDK

1. 编译安装加密SDK。

```
git clone https://github.com/aliyun/alibabacloud-encryption-sdk-java.git
cd alibabacloud-encryption-sdk-java
mvn clean install -DskipTests
```

2. 在项目中添加依赖。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>alibabacloud-encryption-sdk-java</artifactId>
    <version>1.0.7</version>
</dependency>
```

通过Maven仓库安装加密SDK

在项目中添加alibabacloud-encryption-sdk-java的依赖，可从Maven仓库中自动下载发布的Java安装包。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>alibabacloud-encryption-sdk-java</artifactId>
    <version>1.0.x</version>
</dependency>
```

> **说明** 加密SDK当前版本为1.0.7。具体版本，请参见Alibabacloud Encryption SDK Java。

数据加解密示例

- 对字节数组类型的数据进行加解密

```
public class BasicEncryptionExample {
    private static final String ACCESS_KEY_ID = "<AccessKeyId>";
    private static final String ACCESS_KEY_SECRET = "<AccessKeySecret>";
    private static final String CMK_ARN = "acs:kms:RegionId:UserId:key/CmkId";
    private static final byte[] PLAIN_TEXT = "Hello World".getBytes(StandardCharsets.UTF_
8);
    public static void main(String[] args) {
        //1.创建访问aliyun配置。
        AliyunConfig config = new AliyunConfig();
        config.withAccessKey(ACCESS_KEY_ID, ACCESS_KEY_SECRET);
        //2.创建SDK，传入访问aliyun配置。
        AliyunCrypto aliyunSDK = new AliyunCrypto(config);
        //3.创建provider，用于提供数据密钥或签名。
        BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_ARN);
        //设置不同的算法（默认为AES_GCM_NOPADDING_256）。
        //provider.setAlgorithm(CryptoAlgorithm.SM4_GCM_NOPADDING_128);
        //4.加密上下文。
        Map<String, String> encryptionContext = new HashMap<>();
        encryptionContext.put("one", "one");
        encryptionContext.put("two", "two");
        //5.调用加密和解密接口。
        CryptoResult<byte[]> cipherResult = aliyunSDK.encrypt(provider, PLAIN_TEXT, encry
ptionContext);
        CryptoResult<byte[]> plainResult = aliyunSDK.decrypt(provider, cipherResult.getRe
sult());
        Assert.assertArrayEquals(PLAIN_TEXT, plainResult.getResult());
    }
}
```

> 说明
> - 本示例的完整代码请参见SimpleEncryptAndDecryptSample.java。
> - 关于如何获取AccessKey ID和AccessKey Secret，请参见获取AccessKey。

- 对字节流类型的数据进行加解密

```
public class FileStreamSample {
    private static final String FILE = "README.md";
    // accessKeyId accessKeySecret
    private static final String ACCESS_KEY_ID = "<AccessKeyId>";
    private static final String ACCESS_KEY_SECRET = "<AccessKeySecret>";
    // 日志系统。
    private static final Logger LOGGER = LoggerFactory.getLogger(FileStreamSample.class);
    // ARN格式的用户主密钥ID。
    private static final String CMK_ARN = "acs:kms:RegionId:UserId:key/CmkId";
    public static void main(String[] args) throws IOException {
        AliyunConfig config = new AliyunConfig();
        config.withAccessKey(ACCESS_KEY_ID, ACCESS_KEY_SECRET);
        encryptStream(config);
        decryptStream(config);
        Assert.assertEquals(getFileMD5(FILE), getFileMD5(FILE + ".decrypted"));
    }
    private static void encryptStream(AliyunConfig config) throws IOException {
        //1.创建SDK，传入访问aliyun配置。
        AliyunCrypto aliyunSDK = new AliyunCrypto(config);
        //2.构建加密上下文。
        final Map<String, String> encryptionContext = new HashMap<>();
        encryptionContext.put("this", "context");
        encryptionContext.put("can help you", "to confirm");
        encryptionContext.put("this data", "is your original data");
        //3.创建提供数据密钥的provider。
        BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_ARN);
        //4.创建输入输出流。
        FileInputStream inputStream = new FileInputStream(FILE);
        FileOutputStream outputStream = new FileOutputStream(FILE + ".encrypted");
        //5.调用加密接口。
        try {
            aliyunSDK.encrypt(provider, inputStream, outputStream, encryptionContext);
        } catch (InvalidAlgorithmException e) {
            System.out.println("Failed.");
            System.out.println("Error message: " + e.getMessage());
        }
    }
    private static void decryptStream(AliyunConfig config) throws IOException {
        //1.创建SDK，传入访问aliyun配置。
        AliyunCrypto aliyunSDK = new AliyunCrypto(config);
        //2.创建提供数据密钥的provider。
        BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_ARN);
        //3.创建输入输出流。
        FileInputStream inputStream = new FileInputStream(FILE + ".encrypted");
        FileOutputStream outputStream = new FileOutputStream(FILE + ".decrypted");
        //4.调用解密接口。
        try {
```

```
            try {
                aliyunSDK.decrypt(provider, inputStream, outputStream);
            } catch (InvalidAlgorithmException e) {
                System.out.println("Failed.");
                System.out.println("Error message: " + e.getMessage());
            }
        }
        private static String getFileMD5(String fileName) {
            File file = new File(fileName);
            if  (!file.isFile()) {
                return null;
            }
            MessageDigest digest;
            byte[] buffer = new byte[4096];
            try (FileInputStream in = new FileInputStream(file)){
                digest = MessageDigest.getInstance("MD5");
                int len;
                while  ((len = in.read(buffer)) != -1) {
                    digest.update(buffer,  0 , len);
                }
                return Hex.encodeHexString(digest.digest());
            }  catch  (Exception e) {
                e.printStackTrace();
            }
            return null;
        }
    }
```

8.2.2.2. 加密SDK快速入门（Python）

加密SDK（Encryption SDK）是一个客户端密码库，通过与密钥管理服务KMS（Key Management Service）结合使用，帮助您快速实现数据的加解密、签名验签功能。本文以Python语言为例，为您介绍如何快速使用加密SDK进行数据加解密。

背景信息

您可以访问alibabacloud-encryption-sdk-python，查看代码示例。

在本地安装加密SDK

1. 安装加密SDK。

```
git clone https://github.com/aliyun/alibabacloud-encryption-sdk-python.git
cd alibabacloud-encryption-sdk-python
python setup.py install
```

2. 验证加密SDK版本。

   i. 执行以下命令，进入Python语言环境。

```
python
```

ii. 执行以下命令，验证加密SDK版本。

```
import aliyun_encryption_sdk
aliyun_encryption_sdk.__version__
```

执行命令后，Python控制台显示版本号 `'0.1.1'` 。

对字节数组类型的数据进行加解密

```
# -*- coding: UTF-8 -*-
"""Example showing basic encryption and decryption."""
import base64
import os
from aliyun_encryption_sdk.cache.local import LocalDataKeyMaterialCache
from aliyun_encryption_sdk.ckm.cache import CachingCryptoKeyManager
from aliyun_encryption_sdk.client import AliyunCrypto
from aliyun_encryption_sdk.kms import AliyunConfig
from aliyun_encryption_sdk.provider.default import DefaultDataKeyProvider
def build_aliyun_crypto(cache=False):
    config = AliyunConfig(ACCESS_KEY_ID, ACCESS_KEY_SECRET)
    client = AliyunCrypto(config)
    if cache:
        client.crypto_key_manager = CachingCryptoKeyManager(LocalDataKeyMaterialCache(), 5)
    return client
def encrypt_sample():
    print("原文: " + PLAIN_TEXT)
    provider = DefaultDataKeyProvider(AES_KEY_ARN)
    client = build_aliyun_crypto(False)
    cipher_text, enc_material = client.encrypt(provider, PLAIN_TEXT.encode("utf-8"), ENCRYP
TION_CONTEXT)
    cipher_text_str = base64.standard_b64encode(cipher_text).decode("utf-8")
    print(u"加密密文: " + cipher_text_str)
    return cipher_text_str
def decrypt_sample(cipher_text):
    cipher_text_bytes = base64.standard_b64decode(cipher_text.encode("utf-8"))
    provider = DefaultDataKeyProvider(AES_KEY_ARN)
    client = build_aliyun_crypto(False)
    plain_text, dec_material = client.decrypt(provider, cipher_text_bytes)
    print(u"解密结果: " + bytes.decode(plain_text))
    return plain_text
if __name__ == '__main__':
    PLAIN_TEXT = "some plaintext"
    ACCESS_KEY_ID = os.getenv("ACCESS_KEY_ID")
    ACCESS_KEY_SECRET = os.getenv("ACCESS_KEY_SECRET")
    AES_KEY_ARN = os.getenv("AES_KEY_ARN")
    ENCRYPTION_CONTEXT = {
        "this": "context",
        "can help you": "to confirm",
        "this data": "is your original data"
    }
    cipherText = encrypt_sample()
    decrypt_sample(cipherText)
```

对字节流类型的数据进行加解密

```
# -*- coding: UTF-8 -*-
"""Example showing basic encryption and decryption."""
import os
from aliyun_encryption_sdk.cache.local import LocalDataKeyMaterialCache
from aliyun_encryption_sdk.ckm.cache import CachingCryptoKeyManager
from aliyun_encryption_sdk.client import AliyunCrypto
from aliyun_encryption_sdk.kms import AliyunConfig
from aliyun_encryption_sdk.provider.default import DefaultDataKeyProvider
def build_aliyun_crypto(cache=False):
    config = AliyunConfig(ACCESS_KEY, ACCESS_KEY_SECRET)
    client = AliyunCrypto(config)
    if cache:
        client.crypto_key_manager = CachingCryptoKeyManager(LocalDataKeyMaterialCache(), 5)
    return client
def file_stream_sample():
    origin_file_path = r"some_file"
    encrypted_file_path = r"enc_file"
    decrypted_file_path = r"dec_file"
    provider = DefaultDataKeyProvider(AES_KEY_ARN)
    client = build_aliyun_crypto()
    with open(origin_file_path, "rb") as f, open(encryped_file_path, "wb") as cipher_text:
        encrypted_stream, _ = client.encrypt_stream(provider, f)
        with encrypted_stream as stream:
            for content in stream:
                cipher_text.write(content)
    with open(encryped_file_path, "rb") as f, open(decrypted_file_path, "wb") as plain_text
:
        decrypted_stream, _ = client.decrypt_stream(provider, f)
        with decrypted_stream as stream:
            for content in stream:
                plain_text.write(content)
if __name__ == '__main__':
    ACCESS_KEY_ID = os.getenv("ACCESS_KEY_ID")
    ACCESS_KEY_SECRET = os.getenv("ACCESS_KEY_SECRET")
    AES_KEY_ARN = os.getenv("AES_KEY_ARN")
    file_stream_sample()
```

8.2.3. 加密场景

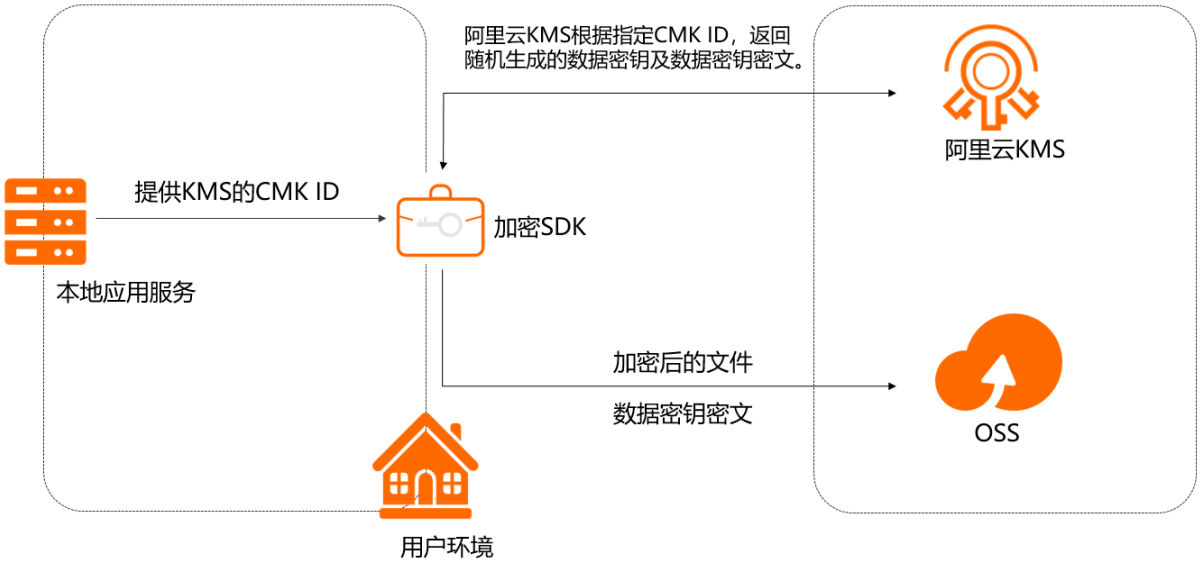
8.2.3.1. 对OSS进行客户端加解密

当您对对象存储OSS进行客户端加解密时，每个对象（Object）将使用一个数据密钥。本文为您介绍如何对OSS对象进行客户端加解密。

使用场景

当使用KMS托管用户主密钥用于客户端数据加密时，无需向加密客户端提供任何加密密钥，只需在上传对象时指定KMS用户主密钥资源名称（CMK ARN）即可。

加密机制



1. 获取加密密钥。
   通过使用CMK ARN，加密SDK（Encryption SDK）首先向KMS发送一个请求，申请一个用于加密Object的数据密钥（Data Key）。作为响应，KMS会返回一个随机生成的数据密钥明文（Data Key）以及一个数据密钥密文（Encrypted Data Key）。
2. 加密数据并上传至OSS。
   加密SDK接收到KMS返回的数据密钥明文以及数据密钥密文后，将使用数据密钥明文进行本地加密，然后将数据密钥密文封装到消息头并存储到对象元数据，最后将加密的对象上传到OSS。

解密机制

1. 下载Object。
   加密SDK从OSS服务端下载加密的Object以及作为对象元数据存储的消息头，并解析消息头得到数据密钥密文。
2. 解密Object。
   加密SDK从CMK ARN中解析出地域信息，将数据密钥密文发送至对应地域的KMS服务器。作为响应，KMS将使用指定的CMK对数据密钥密文进行解密，并且将数据密钥明文返回给本地加密客户端。加密SDK使用数据密钥明文对加密后的对象进行解密，得到原始对象。

> **说明** 使用GCM（Galois Counter Mode）模式对数据对象进行加密，解密时可以对数据进行整体解密，也可以对指定的分段数据进行解密。整体解密时会验证数据的完整性校验数据，校验通过后得到解密数据，分段解密时不对数据完整性进行校验，只对分段数据进行解密。

代码示例

您可以通过OSSEncryption获取SDK代码示例。

8.2.3.2. 使用多个地域的CMK加密和解密数据

阿里云在全球几十个地域，提供了密钥管理KMS、云数据库RDS和对象存储OSS等服务，这些服务在各个地域之间互相独立。使用加密SDK，您可以配置多个地域的KMS用户主密钥（CMK）来加密数据，构建跨地域的数据加密解密能力。

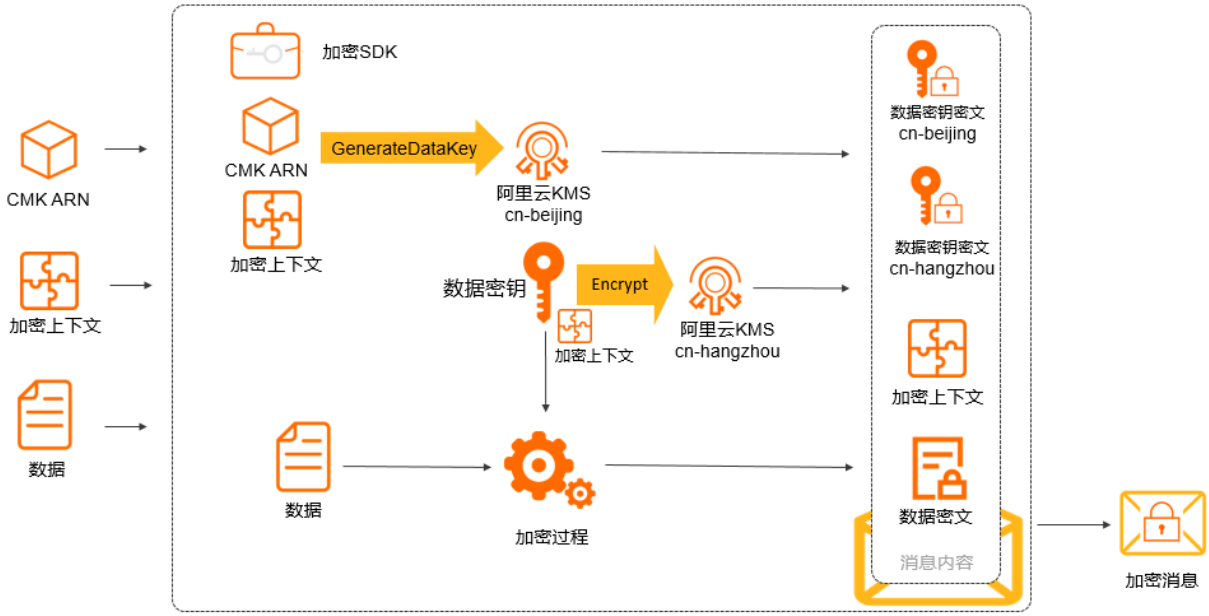
使用场景

在下列情形中，您可以通过加密SDK配置对应地域的KMS CMK，完成对敏感数据（例如：数据库列或字段、OSS对象等）的加密，使得加密数据具有对等地域的解密能力。

- 您使用了云数据库RDS的跨地域备份、同步等功能。
- 您使用了对象存储OSS的跨地域复制功能。
- 您通过其他方式对加密数据进行跨地域的复制。

加密数据

加密机制



1. 调用KMS北京地域的GenerateDataKey接口，得到数据密钥和数据密钥密文。
2. 调用KMS杭州地域的Encrypt接口，对数据密钥进行加密，得到数据密钥密文。
3. 根据加密算法及工作模式所需生成初始向量，使用数据密钥对您的数据进行加密，得到数据密文及认证信息。

   > 说明 仅GCM（Galois Counter Mode）模式有认证信息。

4. 将两个数据密钥密文、加密上下文、算法信息封装到消息头，生成消息头的认证信息。
5. 将初始向量、数据密文、认证信息（可选）封装到消息体。
6. 将消息头和消息体封装为加密消息。

加密代码示例

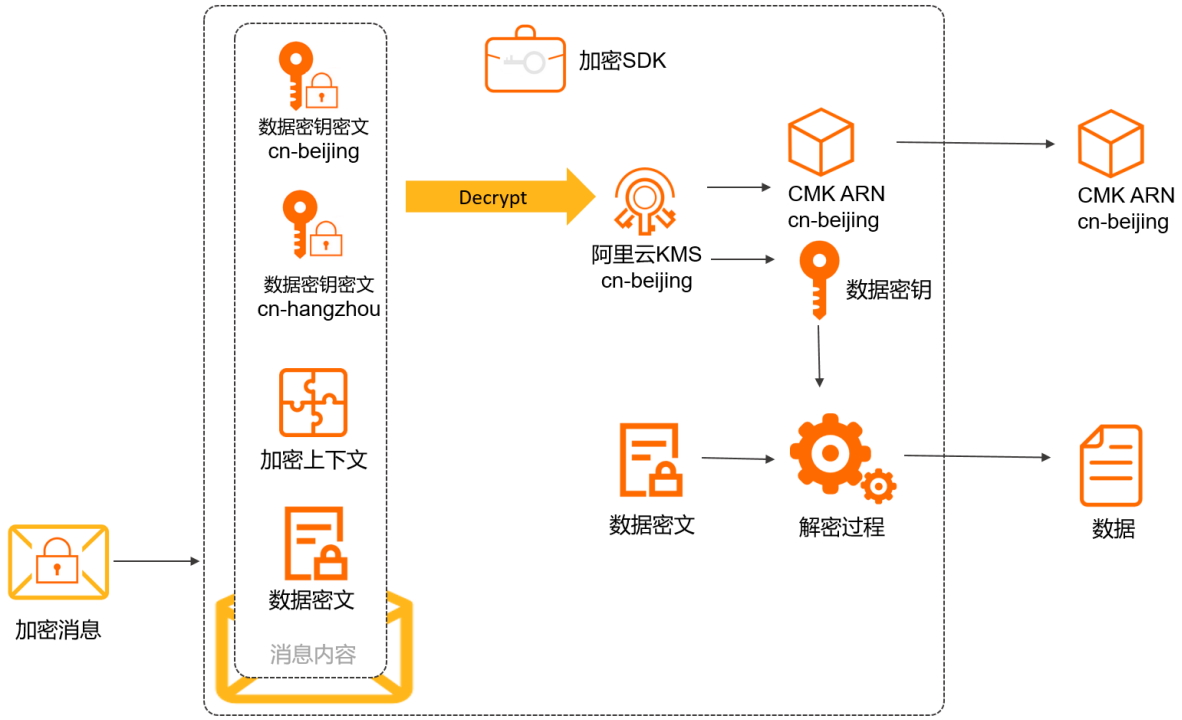
本示例使用了华北2（cn-beijing）和华东1（cn-hangzhou）两个地域，您需要在两个地域创建AES或SM4类型的CMK，并且获取其阿里云资源名称（ARN），配置到以下代码中。其中华北2（cn-beijing）为主地域，在该地域调用KMS的GenerateDataKey接口产生数据密钥，使用华东1（cn-hangzhou）的CMK加密数据密钥。代码示例如下：

```
public class MultiCMKEncryptionExample {
    private static final String ACCESS_KEY_ID = "<AccessKeyId>";
    private static final String ACCESS_KEY_SECRET = "<AccessKeySecret>";
    private static final String CMK_BEIJING_ARN = "acs:kms:cn-beijing:UserId:key/CmkId";
    private static final byte[] PLAIN_TEXT = "Hello World".getBytes(StandardCharsets.UTF_8)
;
    private static final String CMK_HANGZHOU_ARN = "acs:kms:cn-hangzhou:UserId:key/CmkId";
    private static final List<String> CMK_ARN_LIST;
    static {
        CMK_ARN_LIST = new ArrayList<>();
        CMK_ARN_LIST.add(CMK_HANGZHOU_ARN);
    }
    public static void main(String[] args) {
        //创建访问阿里云配置。
        AliyunConfig config = new AliyunConfig();
        config.withAccessKey(ACCESS_KEY_ID, ACCESS_KEY_SECRET);
        //创建SDK，传入访问阿里云配置。
        AliyunCrypto aliyunSDK = new AliyunCrypto(config);
        //创建provider，用于提供数据密钥或签名。
        BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_BEIJING_ARN);
        //设置不同的算法（可设置，默认为AES_GCM_NOPADDING_256）
        //provider.setAlgorithm(CryptoAlgorithm.SM4_GCM_NOPADDING_128);
        // *** 设置多CMK（可设置，默认为单CMK）****
        provider.setMultiCmkId(CMK_ARN_LIST);
        //加密上下文。
        Map<String, String> encryptionContext = new HashMap<>();
        encryptionContext.put("one", "one");
        encryptionContext.put("two", "two");
        //调用加密接口。
        CryptoResult<byte[]> cipherResult = aliyunSDK.encrypt(provider, PLAIN_TEXT, encrypt
ionContext);
        Assert.assertArrayEquals(PLAIN_TEXT, plainResult.getResult());
    }
}
```

> **说明** 本示例的完整代码请参见MultiCmkSample.java。

解密数据

解密机制



1. 根据指定的CMK ARN，从加密消息的消息头中查找匹配的数据密钥密文。
2. 调用KMS的Decrypt接口，得到数据密钥。
3. 根据数据密钥，计算消息头的认证信息，对消息头中的认证信息进行比对并处理异常情况。
4. 根据加密算法及工作模式，对数据密文进行解密，得到数据。

解密代码示例

```
public class MultiCMKDecryptionExample {
    private static final String ACCESS_KEY_ID = "<AccessKeyId>";
    private static final String ACCESS_KEY_SECRET = "<AccessKeySecret>";
    private static final String CIPHER_TEXT_BASE64 = "==base64 encode of ciphertext==";
    private static final String CMK_BEIJING_ARN = "acs:kms:cn-beijing:UserId:key/CmkId";
    private static final String CMK_HANGZHOU_ARN = "acs:kms:cn-hangzhou:UserId:key/CmkId";
    public static void main(String[] args) {
        //创建访问阿里云配置。
        AliyunConfig config = new AliyunConfig();
        config.withAccessKey(ACCESS_KEY_ID, ACCESS_KEY_SECRET);
        //创建SDK，传入访问阿里云配置。
        AliyunCrypto aliyunSDK = new AliyunCrypto(config);
        //创建provider，用于提供数据密钥。
        // 北京地域进解密数据密钥密文时，指定北京地域的CMK ARN。
        BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_BEIJING_ARN);
        // 杭州地域进解密数据密钥密文时，指定杭州地域的CMK ARN。
        //BaseDataKeyProvider provider = new DefaultDataKeyProvider(CMK_HANGZHOU_ARN);
        //设置不同的算法（可设置，默认为AES_GCM_NOPADDING_256）
        //provider.setAlgorithm(CryptoAlgorithm.SM4_GCM_NOPADDING_128);
        //调用解密接口。
        byte[] cipherTextBytes = Base64.decode(CIPHER_TEXT_BASE64);
        CryptoResult<byte[]> plainResult = aliyunSDK.decrypt(provider, cipherTextBytes);
    }
}
```

8.2.3.3. 数据签名验签

对于非对称密钥类型的用户主密钥（CMK），您可以使用私钥对消息或信息产生签名。由于私钥受到严格保护，因此只有受信者可以使用私钥来产生签名，签名后您可以使用公钥验证签名。
数据签名验签的优势如下：

- 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。

代码示例

```
/**
 *
 * 签名验签示例
 */
public class DigestMessageSignatureVerifySample {
    private static final String ASYM_CMK_ARN = "acs:kms:RegionId:UserId:key/CmkId";
    private static final String KEY_VERSION_ID = "<KEY_VERSION_ID>";
    // accessKeyId accessKeySecret
    private static final String ACCESS_KEY_ID = "<AccessKeyId>";
    private static final String ACCESS_KEY_SECRET = "<AccessKeySecret>";
    private static final byte[] MESSAGE_TEXT = "this is test.".getBytes();
    private static final byte[] DIGEST_HEX_TEXT = "FECC75FE2A23D8EAFBA452EE0B8B6B56BECCF522
78BF1398AADDEECFE0EA0FCE".getBytes();
    public static void main(String[] args) {
        // 创建访问阿里云配置。
        AliyunConfig config = new AliyunConfig();
        config.withAccessKey(ACCESS_KEY_ID, ACCESS_KEY_SECRET);
        // 创建SDK，传入访问阿里云配置。
        AliyunCrypto aliyunCrypto = new AliyunCrypto(config);
        // 设置签名验签Provider。
        SignatureProvider provider = new KmsAsymmetricKeyProvider(ASYM_CMK_ARN, KEY_VERSION
_ID, SignatureAlgorithm.RSA_PKCS1_SHA_256);
        // 使用原始消息。
        byte[] signature = aliyunCrypto.sign(provider, MESSAGE_TEXT, ContentType.MESSAGE).g
etResult();
        Boolean isOk = aliyunCrypto.verify(provider, MESSAGE_TEXT, signature, ContentType.M
ESSAGE).getResult();
        assertTrue(isOk);
        // 使用消息摘要。
        byte[] sha256Digest = hex2Bytes(DIGEST_HEX_TEXT);
        signature = aliyunCrypto.sign(provider, sha256Digest, ContentType.DIGEST).getResult
();
        isOk = aliyunCrypto.verify(provider, sha256Digest, signature, ContentType.DIGEST).g
etResult();
        assertTrue(isOk);
    }
}
```

> **说明** 本示例的完整代码请参见libabacloud-encryption-sdk-java-examples-signVerify。

8.2.3.4. 数据库敏感数据加密

数据库加密技术属于主动防御机制，可以防止明文存储引起的数据泄密、突破边界防护的外部黑客攻击以及来自内部高权限用户的数据窃取，从根本上解决数据库敏感数据泄漏问题。通过加密SDK在客户端加密的数据可以存储在关系数据库或非关系数据库中。本文为您介绍数据库敏感数据加密的使用场景、原理和示例。

使用场景

- 数据库敏感数据被拖库后，避免因明文存储导致的数据泄露。
  通常情况下，数据库中的数据是以明文形式进行存储和使用的，一旦数据文件（或备份文件）丢失，可能引发严重的数据泄露问题。而在拖库攻击中，明文存储的数据对于攻击者同样没有任何秘密可言。此时您需要对数据进行加密，避免数据泄露。

- 对高权限用户，数据库敏感数据加密可以防范内部窃取数据造成的数据泄露。
  数据库加密可以提供独立于数据库系统自身权限控制体系之外的增强权限控制的能力，由专用的加密系统为数据库中的敏感数据设置访问权限，从而有效限制数据库超级用户或其他高权限用户对敏感数据的访问行为，保障数据安全。

加密和解密原理

- 加密原理
  i. 创建密钥。
  加密SDK向密钥管理服务KMS（Key Management Service）发送调用GenerateDataKey接口请求，申请一个数据密钥（Data Key）。KMS返回数据密钥以及数据密钥密文（Encrypted Data Key）。
  ii. 加密并存储数据。
  a. 使用数据密钥对数据进行加密，得到加密结果，进行Base64编码。
  b. 将加密数据的Base64编码存储在数据库中。
- 解密原理
  检索并解密数据。
  i. 从数据库读取密文。
  ii. 将密文进行Base64解密，解析密文消息。加密SDK调用KMS的Decrypt接口将数据密钥进行解密，KMS返回数据密钥给本地加密客户端。
  iii. 解密数据。加密SDK（Encryption SDK）使用数据密钥对数据密文进行解密，得到原始数据。

示例

加密SDK用于面向应用的数据库加密，密钥由KMS产生和管理。

通过以下Spring JPA和Python示例代码，可以实现对User表中email字段的写入加密和读取解密。示例中每个字段使用一个数据密钥，解密时设置了密钥缓存，对同一字段进行多次查询时会检索缓存中可用的数据密钥。

为保证字段能存储加密后的字段，需要对被加密字段的长度进行扩容，扩容比例为3倍。

Spring JPA示例 Python示例

关于数据库敏感数据加密的更多示例，请参见Spring JPA示例和Python示例。

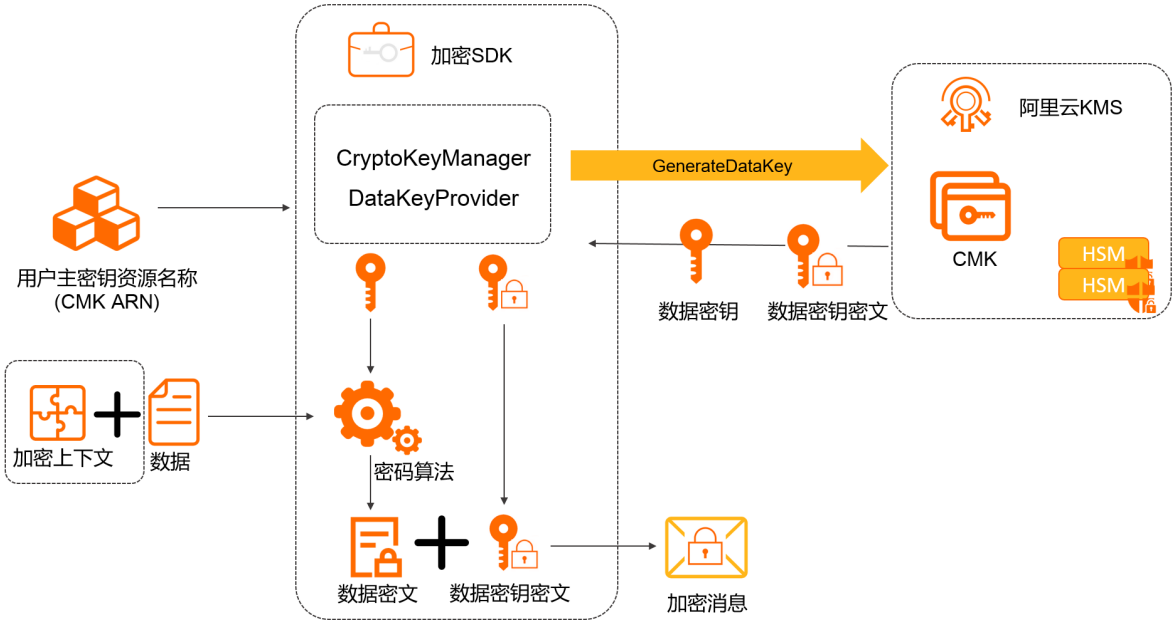
8.2.4. 原理与数据格式

8.2.4.1. 加密解密与签名验签的原理

加密SDK（Encryption SDK）支持加密解密、签名验签，了解其工作原理可以帮助您更好地使用加密SDK。

加密解密原理

- 加密原理
  加密SDK使用数字信封对数据进行加密。加密SDK支持通过用户主密钥（CMK）保护数据密钥（DataKey），使用数据密钥加密数据。关于信封加密的详情，请参见什么是信封加密。

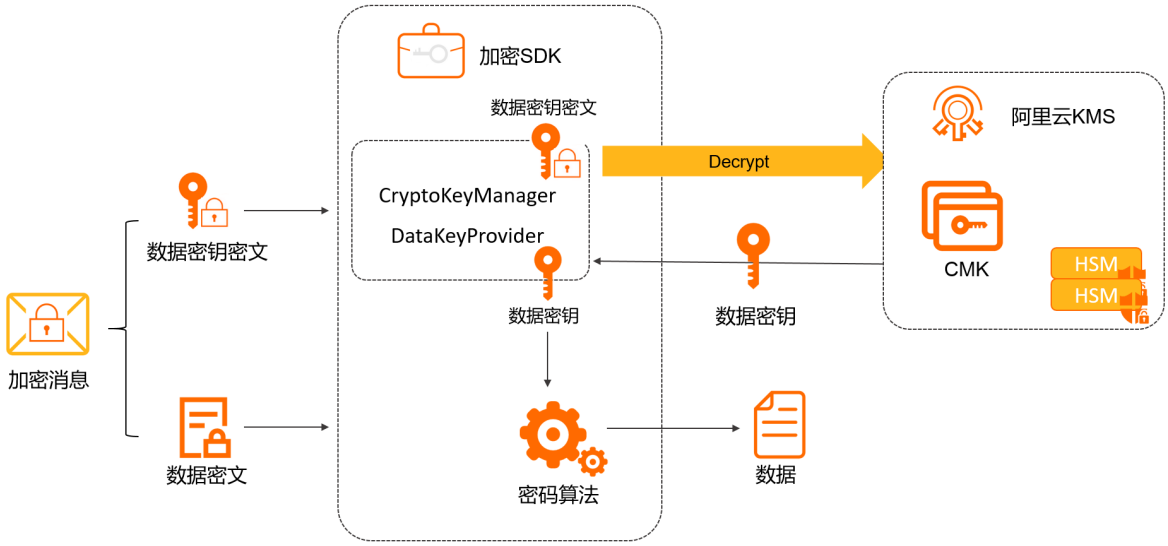


i. 调用GenerateDataKey接口获取数据密钥和数据密钥密文。

ii. 使用数据密钥对数据进行加密，获取数据密文。

iii. 将数据密钥密文和数据密文进行编码，获取加密消息。

- 解密原理



i. 调用Decrypt接口获取数据密钥。

ii. 使用数据密钥对数据密文进行解密，获取数据。

签名验签原理

加密SDK通过非对称类型用户密钥提供签名、验签功能，支持RSA、ECC和SM2非对称密钥算法。更多信息，请参见非对称密钥概述。

签名验签过程如下：

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，使用场景如下：

- 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。
- 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、有效期和签发者等信息。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。

> **说明** 用户主密钥、数据密钥、数据密钥密文、加密密文、加密上下文等概念，请参见基本概念。

8.2.4.2. 加密与签名数据的格式

了解加密与签名数据的格式，可以帮助您解读加密和签名信息。

加密数据格式

- 加密SDK（Encryption SDK）加密数据结果

加密数据结果	组成部分	说明
消息头	版本	当前版本值为1。
	算法	更多信息，请参见算法。
	数据密钥列表	数据密钥列表由1到多个数据密钥组成，数据密钥分为以下两部分： ○ 用户主密钥资源名称（Aliyun Resource Name ARN）：格式为 `acs:kms:RegionId:UserId:key/CmkId` ，包含地域信息、用户ID、用户主密钥ID。 ○ 数据密钥密文：数据密钥被指定CMK的主版本加密后的密文，调用GenerateDataKey返回的CipherBlob。
	加密上下文	加密上下文数据，作为对称加密算法的额外认证数据。
	头部认证初始向量	计算头部认证信息的初始向量值，为随机数。
	头部认证信息	通过GMAC计算头部认证信息，校验失败时返回加密消息格式异常。
消息体	初始向量	初始化向量IV（initialization vector）是一个固定长度的输入值。一般为随机数或伪随机数（pseudo random）。
	密文	数据加密的结果。
	认证数据	GCM（Galois Counter Mode）模式返回的认证数据，用于校验数据的完整性，完整性校验失败时返回解密失败。

消息头中的算法详情如下表所示：

算法号	算法信息	算法	工作模式	密钥长度（bit）	初始向量长度（Byte）
1	AES_GCM_NOPADDING_128	AES	GCM	128	12
2	AES_GCM_NOPADDING_256	AES	GCM	256	12
3	AES_CBC_NOPADDING_128	AES	CBC	128	16
4	AES_CBC_NOPADDING_256	AES	CBC	256	16
5	AES_CBC_PKCS5_128	AES	CBC	128	16
6	AES_CBC_PKCS5_256	AES	CBC	256	16
7	AES_CTR_NOPADDING_128	AES	CTR	128	16
8	AES_CTR_NOPADDING_256	AES	CTR	256	16
9	SM4_GCM_NOPADDING_128	SM4	GCM	128	16
10	SM4_CBC_NOPADDING_128	SM4	CBC	128	16
11	SM4_CBC_PKCS5_128	SM4	CBC	128	16
12	SM4_CTR_NOPADDING_128	SM4	CTR	128	16

说明 仅AES_GCM_NOPADDING_128、AES_GCM_NOPADDING_256和SM4_GCM_NOPADDING_128包含16字节的认证数据。

- 加密数据结果格式定义
  加密数据结果使用ASN.1进行编码，加密数据结果格式的ASN.1定义如下：

```
EncryptionMessage ::== SEQUENCE {
    encryptionHead        EncryptionHead          --消息头
    encryptionBody        EncryptionBody          --消息体
}
EncryptionHead ::== SEQUENCE {
    version               INTEGER                 --版本
    algorithm             INTEGER                 --算法
    encryptedDataKeys     SET EncryptedDataKey    --数据密钥列表
    encryptionContext     SET EncryptionContext   --加密上下文
    headerIv              OCTECT STRING           --头部认证初始向量
    headerAuthTag         OCTECT STRING           --头部认证信息
}
EncryptionBody ::== SEQUENCE{
    iv                    OCTECT STRING           --初始向量
    cipherText            OCTECT STRING           --密文
    authTag               OCTECT STRING           --GCM认证信息
}
EncryptedDataKey ::== SEQUENCE {
    cmkArn                OCTECT STRING           --KMS用户主密钥的ARN
    encryptedDataKey      OCTECT STRING           --数据密钥加密后的密文
}
EncryptionContext ::== SEQUENCE {
    key                   OCTECT STRING
    value                 OCTECT STRING
}
```

- 加密数据结果示例

```
SEQUENCE (2 elem)
  SEQUENCE (6 elem)
    INTEGER 1                                                          // 版本
    INTEGER 2                                                          // 算法
    SET (2 elem)                                                       // 数据密钥列表
      SEQUENCE (2 elem)
        OCTET STRING (77 byte) acs:kms:cn-beijing:1540355698xxxxx:key/2fad5f44-9573-4f28-
8956-xxxx…
        OCTET STRING (108 byte) 36613739356232362D626163642xxxx262642D383630612D323563313
839316131663…
      SEQUENCE (2 elem)
        OCTET STRING (77 byte) acs:kms:cn-hangzhou:1540355698xxxxx:key/f6d61352-82bb-450a
-b105-xxxx…
        OCTET STRING (108 byte) 62623630646439352D343165302xxxx237382D616233332D356262636
136643633643…
    SET (5 elem)                                                       // 加密上下文集合
      SEQUENCE (2 elem)
        OCTET STRING (11 byte) encryption
        OCTET STRING (7 byte) context
      SEQUENCE (2 elem)
        OCTET STRING (7 byte) is not
        OCTET STRING (6 byte) secret
      SEQUENCE (2 elem)
        OCTET STRING (9 byte) but adds
        OCTET STRING (15 byte) useful metadata
      SEQUENCE (2 elem)
        OCTET STRING (18 byte) that can help you
        OCTET STRING (17 byte) be confident that
      SEQUENCE (2 elem)
        OCTET STRING (26 byte) the data you are handling
        OCTET STRING (23 byte) is what you think it is
    OCTET STRING (12 byte) E66C1CE19C79F3FBCD62858D                    // 头部认证初始向量
    OCTET STRING (16 byte) CEEC46C65670E82CD78028AC0104D083            // 头部认证数据
  SEQUENCE (3 elem)                                                    // 加密消息
    OCTET STRING (12 byte) EF49E2CBB768A7AD0FB0FE20                    // 初始向量
    OCTET STRING (13 byte) 89A4AB43CD793F7711767C491A                  // 密文
    OCTET STRING (16 byte) 2E93DA019B7A6507155BA3AA252750E3            // 认证数据
```

- 加密数据结果长度
  - (108B+77B)*CMK数量

    > **说明** 108B表示CMK的ARN长度为108字节，77B表示GenerateDataKey返回的CipherBlob长度为77字节。

  - 加密上下文长度
  - ASN1编码30B

    > **说明** 30B表示ASN1编码的类型和长度为30字节。

  - 密文长度
  - 初始向量

- 认证信息

签名数据格式

加密SDK的签名运算调用阿里云密钥管理服务的非对称签名接口AsymmetricSign，返回签名值对应的二进制数据。

8.3. 凭据管家SDK

8.3.1. 凭据管家客户端

凭据管家客户端（SecretsManager Client）基于KMS凭据管家API封装了业务逻辑、最佳实践和设计模式，更易于开发者在业务系统中集成。主要适用于应用中动态使用托管在凭据管家中的凭据，告别对敏感信息的硬编码。

功能特性

凭据管家客户端具有以下功能特性：

- 支持开发者在应用中快速集成凭据管家能力，一行代码读取凭据信息。
- 封装凭据在应用中缓存和刷新的功能。
- 封装API错误的重试机制，智能处理服务端错误。
- 开放插件式设计模式，支持开发者自定义扩展缓存、错误重试等功能模块。

凭据管家Java客户端

安装SDK

凭据管家客户端支持Java语言，您可以访问SecretsManager Client for Java开源代码仓库了解更多代码信息。

您可以通过Maven在项目中使用凭据管家Java客户端，需要添加的依赖信息如下：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>alibabacloud-secretsmanager-client</artifactId>
    <version>x.x.x</version>
</dependency>
```

> **说明** 凭据管家Java客户端具体版本，请参见alibabacloud-secretsmanager-client-java。

示例代码

- 通过配置文件（secretsmanager.properties）构建客户端
  建议您采用基于Client Key的应用接入点，通过凭据管家Java SDK使用Client Key。关于如何创建Client Key，请参见为AAP绑定Client Key。
  凭据管家Java客户端在1.1.8及以上版本支持基于Client Key应用接入点访问凭据管家，需要配置以下配置文件：

```
## 配置访问方式。
credentials_type=client_key
## 读取Client Key的解密密码：支持从环境变量或者文件读取。
client_key_password_from_env_variable=#your client key private key password environment v
ariable name#
client_key_password_from_file_path=#your client key private key password file path#
## 获取Client Key的私钥文件。
client_key_private_key_path=#your client key private key file path#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
```

通过配置文件（secretsmanager.properties）构建客户端的示例代码如下：

```
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClient;
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClientBuilder;
import com.aliyuncs.kms.secretsmanager.client.exception.CacheSecretException;
import com.aliyuncs.kms.secretsmanager.client.model.SecretInfo;
public class CacheClientEnvironmentSample {
    public static void main(String[] args) {
        try {
            SecretCacheClient client = SecretCacheClientBuilder.newClient();
            SecretInfo secretInfo = client.getSecretInfo("#secretName#");
            System.out.println(secretInfo);
        } catch (CacheSecretException e) {
            e.printStackTrace();
        }
    }
}
```

- 通过指定参数（Access Key、Access Secret、RegionID等）构建客户端

```
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClient;
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClientBuilder;
import com.aliyuncs.kms.secretsmanager.client.exception.CacheSecretException;
import com.aliyuncs.kms.secretsmanager.client.model.SecretInfo;
import com.aliyuncs.kms.secretsmanager.client.service.BaseSecretManagerClientBuilder;
import com.aliyuncs.kms.secretsmanager.client.utils.CredentialsProviderUtils;
public class CacheClientSimpleParametersSample {
    public static void main(String[] args) {
        try {
            SecretCacheClient client = SecretCacheClientBuilder.newCacheClientBuilder(
                    BaseSecretManagerClientBuilder.standard().withCredentialsProvider(Cre
dentialsProviderUtils
                            .withAccessKey("#accessKeyId#", "#accessKeySecret#")).withReg
ion("#regionId#").build()).build();
            SecretInfo secretInfo = client.getSecretInfo("#secretName#");
            System.out.println(secretInfo);
        } catch (CacheSecretException e) {
            e.printStackTrace();
        }
    }
}
```

- 通过自定义参数构建客户端

```
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClient;
import com.aliyuncs.kms.secretsmanager.client.SecretCacheClientBuilder;
import com.aliyuncs.kms.secretsmanager.client.cache.FileCacheSecretStoreStrategy;
import com.aliyuncs.kms.secretsmanager.client.exception.CacheSecretException;
import com.aliyuncs.kms.secretsmanager.client.model.SecretInfo;
import com.aliyuncs.kms.secretsmanager.client.service.BaseSecretManagerClientBuilder;
import com.aliyuncs.kms.secretsmanager.client.service.DefaultRefreshSecretStrategy;
import com.aliyuncs.kms.secretsmanager.client.service.FullJitterBackoffStrategy;
import com.aliyuncs.kms.secretsmanager.client.utils.CredentialsProviderUtils;
public class CacheClientDetailParametersSample {
    public static void main(String[] args) {
        try {
            SecretCacheClient client = SecretCacheClientBuilder.newCacheClientBuilder(Bas
eSecretManagerClientBuilder.standard()
                    .withCredentialsProvider(CredentialsProviderUtils.withAccessKey("#acc
essKeyId#", "#accessKeySecret#"))
                    .withRegion("#regionId#")
                    .withBackoffStrategy(new FullJitterBackoffStrategy(3, 2000, 10000)).b
uild())
                    .withCacheSecretStrategy(new FileCacheSecretStoreStrategy("#cacheSecr
etPath#", true, "#salt#"))
                    .withRefreshSecretStrategy(new DefaultRefreshSecretStrategy("#ttlName
#"))
                    .withCacheStage("#stage#")
                    .withSecretTTL("#secretName#", 1 * 60 * 1000l)
                    .withSecretTTL("#secretName1#", 2 * 60 * 1000l).build();
            SecretInfo secretInfo = client.getSecretInfo("#secretName#");
            System.out.println(secretInfo);
        } catch (CacheSecretException e) {
            e.printStackTrace();
        }
    }
}
```

凭据管家Python客户端

安装SDK

凭据管家客户端支持Python语言，您可以访问SecretsManager Client for Python开源代码仓库了解更多代码信息。

您可以执行如下pip安装命令，在项目中使用凭据管家Python客户端。

```
pip install aliyun-secret-manager-client
```

> **说明** 凭据管家Python客户端具体版本，请参见aliyun-secretsmanager-client-python。

示例代码

- 通过系统环境变量构建客户端

```
from alibaba_cloud_secretsmanager_client.secret_manager_cache_client_builder import Secre
tManagerCacheClientBuilder
if __name__ == '__main__':
    secret_cache_client = SecretManagerCacheClientBuilder.new_client()
    secret_info = secret_cache_client.get_secret_info("#secretName#")
    print(secret_info.__dict__)
```

- 通过指定参数（Access Key、Access Secret、RegionID等）构建客户端

```
from alibaba_cloud_secretsmanager_client.secret_manager_cache_client_builder import Secre
tManagerCacheClientBuilder
from alibaba_cloud_secretsmanager_client.service.default_secret_manager_client_builder im
port DefaultSecretManagerClientBuilder
if __name__ == '__main__':
    secret_cache_client = SecretManagerCacheClientBuilder.new_cache_client_builder(Defaul
tSecretManagerClientBuilder.standard() \
        .with_access_key("#accessKeyId#", "#accessKeySecret#") \
        .with_region("#regionId#").build()) \
    .build();
    secret_info = secret_cache_client.get_secret_info("#secretName#")
    print(secret_info.__dict__)
```

- 通过自定义参数构建客户端

```
from alibaba_cloud_secretsmanager_client.secret_manager_cache_client_builder import Secre
tManagerCacheClientBuilder
from alibaba_cloud_secretsmanager_client.cache.file_cache_secret_store_strategy import Fi
leCacheSecretStoreStrategy
from alibaba_cloud_secretsmanager_client.service.default_secret_manager_client_builder im
port DefaultSecretManagerClientBuilder
from alibaba_cloud_secretsmanager_client.service.default_refresh_secret_strategy import D
efaultRefreshSecretStrategy
from alibaba_cloud_secretsmanager_client.service.full_jitter_back_off_strategy import Ful
lJitterBackoffStrategy
if __name__ == '__main__':
    secret_cache_client = SecretManagerCacheClientBuilder \
    .new_cache_client_builder(DefaultSecretManagerClientBuilder.standard().with_access_ke
y("#accessKeyId#", "#accessKeySecret#") \
         .with_back_off_strategy(FullJitterBackoffStrategy(3, 2000, 10000)) \
         .with_region("#regionId#").build()) \
     .with_cache_secret_strategy(FileCacheSecretStoreStrategy("#cacheSecretPath#", True,"
#salt#")) \
     .with_refresh_secret_strategy(DefaultRefreshSecretStrategy("#ttlName#")) \
     .with_cache_stage("#stage#") \
     .with_secret_ttl("#secretName#", 1 * 60 * 10001) \
     .build()
    secret_info = secret_cache_client.get_secret_info("#secretName#")
    print(secret_info.__dict__)
```

凭据管家Go客户端

安装SDK

凭据管家客户端支持Go语言，您可以访问SecretsManager Client for Go开源代码仓库了解更多代码信息。

您可以执行如下安装命令，在项目中使用凭据管家Go客户端。

```
go get -u github.com/aliyun/aliyun-secretsmanager-client-go
```

> **说明** 凭据管家Go客户端具体版本，请参见aliyun-secretsmanager-client-go。

示例代码

- 通过配置文件（secretsmanager.properties）构建客户端
  建议您采用基于Client Key的应用接入点，通过凭据管家Go SDK使用Client Key。关于如何创建Client Key，请参见为AAP绑定Client Key。
  凭据管家Go客户端在v1.0.1及以上版本支持基于Client Key应用接入点访问凭据管家，需要配置以下配置文件：

```
## 配置访问方式。
credentials_type=client_key
## 读取Client Key的解密密码：支持从环境变量或者文件读取。
client_key_password_from_env_variable=#your client key private key password environment variable name#
client_key_password_from_file_path=#your client key private key password file path#
## 获取Client Key的私钥文件。
client_key_private_key_path=#your client key private key file path#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
```

  通过配置文件（secretsmanager.properties）构建客户端的示例代码如下：

```
package main
import (
    "fmt"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk"
)
func main() {
    client, err := sdk.NewClient()
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    secretInfo, err := client.GetSecretInfo("#secretName#")
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    fmt.Printf("SecretValue:%s\n",secretInfo.SecretValue)
}
```

- 通过指定参数（Access Key、Access Secret、RegionID等）构建客户端

```
package main
import (
    "fmt"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk/service"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk"
)
func main() {
    client, err := sdk.NewSecretCacheClientBuilder(service.NewDefaultSecretManagerClientB
uilder().Standard().WithAccessKey("#accessKeyId#", "#accessKeySecret#").WithRegion("#regi
onId#").Build()).Build()
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    secretInfo, err := client.GetSecretInfo("#secretName#")
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    fmt.Printf("SecretValue:%s\n",secretInfo.SecretValue)
}
```

- 通过自定义参数构建客户端

```
package main
import (
    "fmt"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk/service"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk"
    "github.com/aliyun/aliyun-secretsmanager-client-go/sdk/cache"
)
func main() {
    client, err := sdk.NewSecretCacheClientBuilder(
        service.NewDefaultSecretManagerClientBuilder().Standard().WithAccessKey("#accessK
eyId#", "#accessKeySecret#").WithRegion("#regionId#").WithBackoffStrategy(&service.FullJi
tterBackoffStrategy{RetryMaxAttempts: 3, RetryInitialIntervalMills: 2000, Capacity: 10000
}).Build()).WithCacheSecretStrategy(cache.NewFileCacheSecretStoreStrategy("#cacheSecretPa
th#", true, "#salt#")).WithRefreshSecretStrategy(service.NewDefaultRefreshSecretStrategy(
"#jsonTTLPropertyName#")).WithCacheStage("ACSCurrent").WithSecretTTL("#secretName#", 1*60
*1000).Build()
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    secretInfo, err := client.GetSecretInfo("#secretName#")
    if err != nil {
        // Handle exceptions
        panic(err)
    }
    fmt.Printf("SecretValue:%s\n",secretInfo.SecretValue)
}
```

8.3.2. 凭据管家JDBC客户端

凭据管家JDBC客户端（SecretsManager JDBC）基于KMS凭据管家动态RDS凭据，封装了业务逻辑、最佳实践和设计模式，更易于开发者在业务系统中集成。凭据管家JDBC客户端主要用于在数据库应用中使用动态RDS凭据，告别对数据库账号密码机密信息的硬编码。

功能特性

- 提供通用的Java数据库连接JDBC（Java Database Connectivity）驱动，支持简单的数据库连接。
- 通过c3p0和DBCP数据库连接池连接数据库。
- 支持使用AccessKey、STS和ECS实例RAM角色等多种访问方式获取动态RDS凭据。
- 支持用户自定义的凭据刷新频率。

使用限制

- 仅支持动态RDS凭据，建议您使用双账号托管的动态RDS凭据。关于如何创建动态RDS凭据，请参见创建动态RDS凭据。
- 仅支持Java 1.8及以上版本。
- 仅支持MySQL、SQL Server、PostgreSQL和MariaDB四种数据库类型。

安装SDK

凭据管家JDBC客户端支持Java语言，您可以访问凭据管家JDBC开源代码仓库了解更多安装信息。

您可以通过Maven的方式安装Java SDK，需要添加的依赖信息如下：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-secretsmanager-jdbc</artifactId>
    <version>x.x.x</version>
</dependency>
```

> **说明** 凭据管家JDBC客户端具体版本，请参见aliyun-secretsmanager-jdbc release。

配置访问方式

阿里云凭据管家JDBC客户端启动时需要通过解析配置文件（secretsmanager.properties）加载访问方式，不同访问方式的配置文件示例如下：

- 通过AAP Client Key访问（推荐）
  关于如何创建Client Key，请参见为AAP绑定Client Key。

  > **说明** 凭据管家JDBC客户端在1.0.7及以上版本支持基于Client Key应用接入点访问凭据管家。

```
## 配置访问方式。
credentials_type=client_key
## 读取Client Key的解密密码：支持从环境变量或者文件读取。
client_key_password_from_env_variable=#your client key private key password environment v
ariable name#
client_key_password_from_file_path=#your client key private key password file path#
## 读取Client Key的私钥文件。
client_key_private_key_path=#your client key private key file path#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
```

- 通过AccessKey访问

```
## 配置访问方式。
credentials_type=ak
## 配置AccessKey ID。
credentials_access_key_id=#credentials_access_key_id#
## 配置AccessKey Secret。
credentials_access_secret=#credentials_access_secret#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
## 用户自定义的刷新频率。默认为6小时，最小值为5分钟，单位为毫秒。
refresh_secret_ttl=21600000
```

> 说明 关于如何获取AccessKey，请参见获取AccessKey。

- 通过STS访问

```
## 配置访问方式。
credentials_type=sts
## 配置AccessKey ID。
credentials_access_key_id=#credentials_access_key_id#
## 配置AccessKey Secret。
credentials_access_secret=#credentials_access_secret#
## 配置访问凭据Session名称。
credentials_role_session_name=#credentials_role_session_name#
## 配置RAM角色ARN。
credentials_role_arn=#credentials_role_arn#
## 配置访问凭据Policy。
credentials_policy=#credentials_policy#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
## 用户自定义的刷新频率。默认为6小时，最小值为5分钟，单位为毫秒。
refresh_secret_ttl=21600000
```

> 说明 关于如何获取AccessKey，请参见获取AccessKey。

- 通过ECS实例RAM角色访问

```
## 配置访问方式。
credentials_type=ecs_ram_role
## 配置ECS实例RAM角色名称。
credentials_role_name=#credentials_role_name#
## 配置关联的KMS地域。
cache_client_region_id=[{"regionId":"#regionId#"}]
## 用户自定义的刷新频率。默认为6小时，最小值为5分钟，单位为毫秒。
refresh_secret_ttl=21600000
```

> **说明** 关于如何创建并授权ECS实例RAM角色，请参见从ECS实例安全访问KMS。

示例代码

- **使用JDBC方式访问数据库**
  使用JDBC方式访问MySQL示例代码如下：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class SecretManagerJDBCSample {
    public static void main(String[] args) throws Exception {
        Class.forName("com.aliyun.kms.secretsmanager.MysqlSecretsManagerSimpleDriver");
        Connection connect = null;
        try {
            connect = DriverManager.getConnection("secrets-manager:mysql://<your-mysql-ip
>:<your-mysql-port>/<your-database-name>", "#your-mysql-secret-name#","");
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- **使用数据库连接池c3p0访问数据库**
  配置c3p0配置文件（c3p0.properties）示例代码如下：

```
c3p0.user=#your-mysql-secret-name#
c3p0.driverClass=com.aliyun.kms.secretsmanager.MysqlSecretsManagerSimpleDriver
c3p0.jdbcUrl=secrets-manager:mysql://<your-mysql-ip>:<your-mysql-port>/<your-database-nam
e>
```

- **使用数据库开源框架访问数据库**
  配置Spring配置文件示例代码如下：

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" >
      <property name="driverClass" value="com.aliyun.kms.secretsmanager.MysqlSecretsManag
erSimpleDriver" />
      <property name="user" value="#your-mysql-secret-name#" />
      <property name="jdbcUrl" value="secrets-manager:mysql://<your-mysql-ip>:<your-mysql
-port>/<your-database-name>" />
      <property name="maxPoolSize" value="500" />
      <property name="minPoolSize" value="5" />
      <property name="initialPoolSize" value="20" />
  </bean>
  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate" >
      <property name="dataSource" ref="dataSource" />
  </bean>
```

8.3.3. 多种阿里云SDK的托管凭据插件

您在使用阿里云产品的SDK时，如果引入匹配此SDK的托管凭据插件，则可以通过托管RAM凭据的名称使用RAM用户的AccessKey，而无需关心AccessKey是什么，或者它是否被轮换。

工作原理

应用程序使用托管凭据插件后，只需要引用托管RAM凭据（RAM用户的AccessKey）的凭据名称。插件将根据凭据名称从凭据管家获取凭据的值（AccessKey）并缓存在应用程序的内存中。插件也会以指定的间隔（可配置）从凭据管家重新获取凭据的值并刷新本地的缓存。

应用程序使用阿里云SDK时，将通过插件缓存的AccessKey对云产品发起请求。

在某些情况下，缓存的RAM凭据不再有效，这通常发生在管理员在凭据管家中人工轮转凭据，以响应安全事件时。使用无效RAM凭据调用阿里云服务通常会导致应用程序产生异常。如果异常中包含的错误代码为 `InvalidAccessKeyId.NotFound` 或 `InvalidAccessKeyId` ，则托管凭据插件将立即刷新缓存的RAM凭据并重试失败的请求。

如果使用过期AccessKey调用某些云服务API返回的错误代码和上述所列错误码不同，应用开发人员可以修改默认的错误重试行为。更多信息，请参见示例二：修改默认过期处理程序。

支持插件的阿里云SDK

阿里云提供了多种不同的SDK，针对不同SDK的托管凭据插件需要分别开发。下表列出了可以使用托管凭据插件的阿里云SDK。

阿里云SDK名称	支持插件的SDK版本	对应SDK的托管凭据插件
阿里云Java SDK	4.3.2~4.5.17	阿里云Java SDK托管凭据插件
OSS Java SDK	2.1.0~3.10.2	OSS Java SDK托管凭据插件
消息队列商业版TCP协议Java SDK	1.8.5.Final~1.8.7.4.Final	消息队列商业版TCP协议Java SDK托管凭据插件

安装托管凭据插件

针对阿里云的各类Java SDK的托管凭据插件，您可以访问多种阿里云SDK的托管凭据插件开源代码仓库了解更多安装信息。更多信息，请参见多种阿里云SDK的托管凭据插件开源代码仓库。

您可以通过Maven的方式在项目中按需引入阿里云SDK托管凭据Java插件。以下以OSS插件导入为例，为您介绍需要添加的依赖信息。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-sdk-oss-managed-credentials-provider</artifactId>
    <version>x.x.x</version>
</dependency>
```

> **说明** 多种阿里云SDK的托管凭据插件具体版本，请参见aliyun-sdk-managed-credentials-providers-java release。

示例一：在阿里云SDK中使用托管RAM凭据

1. 为云服务的SDK配置托管凭据插件。
   您可以通过配置文件（managed_credentials_providers.properties）指定从凭据管家获取托管凭据的方式，以配置应用接入点的Client Key为例进行介绍。关于如何创建Client Key，请参见为AAP绑定Client Key。

   ```
   ## 配置访问方式。
   credentials_type=client_key
   ## 读取Client Key的解密密码：支持从环境变量或者文件读取。
   client_key_password_from_env_variable=#your client key private key password environment variable name#
   client_key_password_from_file_path=#your client key private key password file path#
   ## 读取Client Key的私钥文件。
   client_key_private_key_path=#your client key private key file path#
   ## 配置关联的KMS地域。
   cache_client_region_id=[{"regionId":"#regionId#"}]
   ```

2. 使用托管RAM凭据访问云服务。
   - 方法一：编码方式（以托管RAM凭据访问OSS为例）

     ```
     import com.aliyun.kms.secretsmanager.plugin.oss.ProxyOSSClientBuilder;
     import com.aliyun.oss.OSS;
     import com.aliyun.oss.model.Bucket;
     import java.util.List;
     public class OssPluginSample {
         public static void main(String[] args) throws Exception {
             String secretName = "******";
             String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
             // 获取OSS客户端。
             OSS ossClient = new ProxyOSSClientBuilder().build(endpoint, secretName);
             List<Bucket> buckets = ossClient.listBuckets();
             for (Bucket bucket : buckets) {
                 if (bucket != null) {
                     // do something with bucket
                 }
             }
             // 通过以下方法关闭客户端来释放插件关联的资源。
              ossClient.shutdown();
         }
     }
     ```

   - 方法二：Spring Bean方式（以快速集成OSS SDK为例）

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean name="proxyOSSClientBuilder" class="com.aliyun.kms.secretsmanager.plugin.os
s.ProxyOSSClientBuilder" scope="singleton" />
</beans>
```

示例二：修改默认过期处理程序

托管凭据插件默认实现AKExpireHandler的接口调用，用来判断对云服务的调用是否使用了无效的凭据。使用无效凭据时，插件将重新从凭据管家获取最新的凭据，然后重试SDK对云服务的调用过程。

AKExpireHandler的接口定义如下：

```
package com.aliyun.kms.secretsmanager.plugin.common;
public interface AKExpireHandler<TException> {
  /**
   * 判断异常是否由AccessKey过期引起。
   *
   * @param e
   * @return
   */
  boolean judgeAKExpire(TException e);
}
```

您可以重新实现AKExpireHandler的接口调用，判断云服务返回的错误码是否由使用无效AccessKey引起，示例如下：

```
import com.aliyun.kms.secretsmanager.plugin.sdkcore.ProxyAcsClient;
import com.aliyun.kms.secretsmanager.plugin.common.AKExpireHandler;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import java.util.HashSet;
import java.util.Set;
public class SdkRetrySample {
  public static void main(String[]args) throws Exception{
    String region="cn-hangzhou";
    String secretName="******";
    // 获取阿里云SDK客户端，并对特定的错误码进行重新获取凭据。
    IAcsClient client = new ProxyAcsClient(
        region, secretName, new CustomHandler());
    // 业务代码。
    invoke(client,region);
    // 通过以下方法关闭客户端来释放插件关联的资源。
    client.shutdown();
  }
}
class CustomHandler implements AKExpireHandler<ClientException> {
  private Set<String> errorCodeSet;
  public CustomerHandler() {
      errorCodeSet = new HashSet<String>();
      // 自行添加错误码，触发客户端重新从凭据管家获取托管的RAM凭据。
      errorCodeSet.add("InvalidAccessKeyId.NotFound");
      errorCodeSet.add("InvalidAccessKeyId");
  }
  @Override
  public boolean judgeAKExpire(ClientException e) {
      return errorCodeSet.contains(e.getErrCode());
  }
}
```

8.3.4. 凭据管家Kubernetes插件

凭据管家Kubernetes插件支持配置KMS凭据管家存储凭据，您可以在插件中配置凭据管家中的凭据名称，插件将定期从凭据管家中读取最新的凭据值，缓存在Kubernetes集群中，从而可以按照使用Kubernetes原生Secret的方式，使用托管在凭据管家中的动态Secret。

如果您使用自建Kubernetes或者阿里云容器服务Kubernetes版ACK（Alibaba Cloud Container Service for Kubernetes），可以通过以下方式无代码集成KMS凭据管家：

- 通过阿里云容器服务Kubernetes版ACK安装插件
  i. 登录容器服务管理控制台。
  ii. 在左侧导航栏中，选择**市场 > 应用目录**。
  iii. 在**应用目录**页面，单击**阿里云应用**页签。
  iv. 搜索**ack-secret-manager**。
  v. 单击**ack-secret-manager**软件，安装插件。

> **说明** 您也可以访问ack-secret-manager安装插件。

- 访问kubernetes-external-secrets安装插件

> **说明** 为了进一步保护从凭据管家读取后缓存在Kubernetes集群中的Secret（同时也加密保护Kubernetes集群中的其他静态Secret，这些静态Secret通常为系统Secret），您可以对Kubernetes集群Secret进行一键加密。具体操作，请参见使用KMS一键加密Kubernetes集群Secret。

8.4. 最佳实践

8.4.1. 从ECS实例安全访问KMS

您可以为VPC网络下的ECS实例创建RAM服务角色，使ECS实例内的应用程序可以使用STS临时凭证或者通过SDK访问KMS。

步骤一：创建实例RAM角色并授权

1. 创建实例RAM角色。

   在OpenAPI开发者门户中调用RAM的CreateRole接口，创建实例RAM角色（EcsRamRoleTest）。请求参数设置如下：

   - RoleName：填写实例RAM角色名称（EcsRamRoleTest）。
   - AssumeRolePolicyDocument：填写如下策略内容，表示允许ECS扮演该角色。

   ```
   {
       "Statement": [
           {
               "Action": "sts:AssumeRole",
               "Effect": "Allow",
               "Principal": {
                   "Service": [
                       "ecs.aliyuncs.com"
                   ]
               }
           }
       ],
       "Version": "1"
   }
   ```

2. 授予实例RAM角色访问KMS的权限。

   在OpenAPI开发者门户中调用RAM的AttachPolicyToRole接口，为实例RAM角色（EcsRamRoleTest）添加AliyunKMSFullAccess系统权限。请求参数设置如下：

   - PolicyType：填写System，表示系统策略。
   - PolicyName：填写KMS系统策略名称（AliyunKMSFullAccess）。
   - RoleName：填写实例RAM角色名称（EcsRamRoleTest）。

步骤二：授予实例RAM角色

您可以通过如下两种方式为ECS实例授予RAM角色：

- 为已有ECS实例授予实例RAM角色
  在OpenAPI开发者门户中调用ECS的AttachInstanceRamRole接口，为已有的VPC类型ECS实例授予实例RAM角色。请求参数设置如下：

- RegionId：选择实例所在的地域ID。
- RamRoleName：填写实例RAM角色名称（EcsRamRoleTest）。
- InstanceIds：填写VPC类型ECS实例ID（["i-bXXXXXXXX"]）。

- 创建ECS实例时指定实例RAM角色
  i. 创建实例。
  在OpenAPI开发者门户中调用ECS的CreateInstance接口，创建ECS实例。请求参数设置如下：
  - RegionId：选择实例所在地域（cn-hangzhou）。
  - ImageId：填写实例的镜像（centos_7_03_64_40G_alibase_20170503.vhd）。
  - InstanceType：填写实例的规格（ecs.g6.large）。
  - VSwitchId：填写实例所在的VPC交换机的ID。

    > **说明** 实例RAM角色目前只支持VPC类型ECS实例，该参数必填。

  - RamRoleName：填写实例RAM角色名称（EcsRamRoleTest）。

  您也可以授权RAM用户使用实例RAM角色。具体操作，请参见授权RAM用户使用实例RAM角色。
  ii. 设置密码并启动实例。
  iii. 使用API或在控制台设置ECS实例访问公网的权限。

步骤三（可选）：获取临时授权Token

您可以获得实例RAM角色的临时授权Token，该临时授权Token可以执行实例RAM角色的权限和资源，并且自动周期性地更新。操作步骤如下：

1. 检索名为EcsRamRoleTest的实例RAM角色的临时授权Token。
   - Linux实例：执行命令 `curl http://100.100.100.200/latest/meta-data/ram/security-credentials/EcsRamRoleTest` 。
   - Windows实例：具体操作，请参见实例元数据。
2. 获得临时授权Token。

   返回示例如下：

```
{
   "AccessKeyId" : "STS.J8XXXXXXXXX4",
   "AccessKeySecret" : "9PjfXXXXXXXXBf2XAW",
   "Expiration" : "2017-06-09T09:17:19Z",
   "SecurityToken" : "CAIXXXXXXXXXXXwmBkleCTkyI+",
   "LastUpdated" : "2017-06-09T03:17:18Z",
   "Code" : "Success"
}
```

步骤四：使用SDK访问KMS

通过如下三种方式使用SDK访问KMS：

KMS SDK | 加密SDK | 凭据管家客户端

```
package com.aliyuncs.kms.examples;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.auth.AlibabaCloudCredentialsProvider;
import com.aliyuncs.auth.InstanceProfileCredentialsProvider;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.kms.model.v20160120.*;
import com.aliyuncs.profile.DefaultProfile;

public class RamRoleTest {
    public static void main(final String[] args) throws Exception {
        String regionId = "<region-id>";
        DefaultProfile profile = DefaultProfile.getProfile(regionId);

        // 设置RAM角色。
        String roleName = "<your-ecs-ram-role-name>"; // 本示例使用"EcsRamRoleTest"。

        // 设置ECS实例RAM角色的凭证Provider。
        AlibabaCloudCredentialsProvider provider = new
InstanceProfileCredentialsProvider(roleName);

        IAcsClient kmsClient = new DefaultAcsClient(profile, provider);

        EncryptRequest request = new EncryptRequest();

        // 指定用于加密Hello world的用户主密钥别名或者用户主密钥ID。
        request.setKeyId("alias/Apollo/SalaryEncryptionKey");
        request.setPlaintext("Hello world");

        try {
            EncryptResponse response = client.getAcsResponse(request);
            System.out.println(new Gson().toJson(response));
        } catch (ServerException e) {
            e.printStackTrace();
        } catch (ClientException e) {
            System.out.println("ErrCode:" + e.getErrCode());
            System.out.println("ErrMsg:" + e.getErrMsg());
            System.out.println("RequestId:" + e.getRequestId());
        }
    }
}
```