aws

用户指南

AWS Payment Cryptography



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Payment Cryptography: 用户指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS 支付密码学？ 1
概念 2
行业术语 3
常见密钥类型 4
其他术语 6
相关服务 10
有关更多信息 11
端点 11
控制面板端点 11
数据面板端点 14
开始使用 17
先决条件 17
步骤 1：创建密钥 17
步骤 2：使用密钥生成 CVV2 值 18
步骤 3：验证已在步骤 2 中生成的值 19
第 4 步：进行阴性测试 20
第 5 步：（可选）清除 20
管理密钥 22
创建密钥 22
创建 2KEY TDES 基础派生密钥 23
为创建 2KEY TDES 密钥 CVV/CVV2 25
创建 HMAC 密钥 26
创建密 AES-256 钥 27
创建 PIN 加密密钥 (PEK) 28
创建非对称 (RSA) 密钥 29
创建 PIN 验证值 (PVV) 密钥 30
创建非对称 ECC 密钥 31
列出密钥 32
启用和禁用 密钥 33
开始密钥使用 33
停止密钥使用 35
复制密钥 37
Multi-Region 密钥复制的好处 37
Multi-Region 密钥复制的工作原理 37

限制和注意事项 37
启用 Multi-Region 密钥复制 38
禁用 Multi-Region 密钥复制 40
安全注意事项 40
最佳实践 41
定价 41
删除 密钥 41
关于等待期限 42
导入和导出密钥 45
导入密钥 47
导出密钥 70
高级主题 89
使用别名 107
关于别名 107
在应用程序中使用别名 110
相关 API 111
获取密钥 111
让公众 key/certificate 与密钥对相关联 113
标记密钥 113
关于中的标签 AWS Payment Cryptography 114
在控制台中查看密钥标签 115
使用 API 操作管理密钥标签 115
控制对标签的访问 118
使用标签控制对密钥的访问 122
了解密钥属性 125
对称密钥 125
非对称密钥 127
数据操作 129
加密、解密和重新加密数据 129
加密数据 130
解密数据 134
生成并验证卡数据 137
生成卡数据 137
验证卡数据 139
生成、转换和验证 PIN 数据 141
转换 PIN 数据 142

生成 PIN 数据 144
验证 PIN 数据 148
生成并验证 ARQC 152
生成身份验证请求 (ARQC) 密码 153
验证身份验证请求 (ARQC) 密码 155
生成并验证 MAC 158
生成 MAC 160
验证 MAC 163
特定数据操作的密钥类型 165
GenerateCardData 166
VerifyCardData 167
GeneratePinData （用于 VISA/ABA 计划） 168
GeneratePinData （适用于 IBM3624） 169
VerifyPinData （用于 VISA/ABA 计划） 170
VerifyPinData （适用于 IBM3624） 170
解密数据 171
加密数据 172
转换 PIN 数据 173
Generate/Verify MAC 174
GenerateMacEmvPinChange 176
VerifyAuthRequestCryptogram 177
Import/Export 钥匙 177
未使用的密钥类型 178
常见使用案例 179
发行人和发行人处理器 179
常规函数 179
网络特定功能 209
收购和付款促进者 240
使用动态按键 241
特定区域的功能 243
AS2805 243
初始密钥 (KEK) 交换 244
KEK 的验证 246
创建和传输工作密钥 249
导出工作密钥 250
大头针翻译 251

Mac 生成和验证 252
安全性 254
数据保护 254
保护密钥材料 255
数据加密 256
静态加密 256
传输中加密 256
互联网络流量隐私 256
恢复能力 257
区域隔离 257
Multi-tenant 设计 258
基础结构安全性 258
物理主机的隔离 258
使用亚马逊 VPC 和 AWS PrivateLink 259
注意事项 AWS 支付加密 VPC 终端节点 259
为创建 VPC 终端节点 AWS Payment Cryptography 260
连接到 VPC 端点 261
控制对 VPC 端点的访问 261
在策略语句中使用 VPC 端点 264
记录您的 VPC 端点 267
混合后量子 TLS 270
关于后量子 TLS 271
使用方法 271
关于 PQC 271
使用方法 271
安全最佳实践 272
合规性验证 273
服务的合规性 273
PIN 合规 274
常见话题 274
评估范围 276
交易处理操作 277
P2PE 合规 281
Identity and access management 282
受众 282
使用身份进行身份验证 282

AWS 账户 根用户 283
IAM 用户和群组 283
IAM 角色 283
使用策略管理访问 283
Identity-based 政策 284
Resource-based 政策 284
访问控制列表（ACL） 284
其他策略类型 284
多个策略类型 285
操作方法 AWS 支付密码学与 IAM 配合使用 285
AWS 支付密码学政策 Identity-based 285
授权基于 AWS 支付密码学标签 287
Identity-based 策略示例 287
策略最佳实践 288
使用控制台 288
允许用户查看他们自己的权限 289
能够访问的各个方面 AWS Payment Cryptography 290
使用指定的密钥调用 API 的能力 290
明确拒绝资源的能力 291
Resource-based 政策 292
注意事项 293
管理基于资源的策略 294
Resource-based 策略示例 295
Multi-party 批准 297
概述 297
受保护的操作 298
先决条件 298
启用和禁用 MPA 298
开始使用 299
示例：导入启用了 MPA 的根证书 299
AWS CloudTrail 记录 MPA 事件 300
检查请求状态并处理失败 303
问题排查 305
监控 306
CloudTrail 日志 306
AWS 中的支付密码学信息 CloudTrail 307

控制飞机事件 CloudTrail 307
中的数据事件 CloudTrail 308
了解 AWS 支付密码学控制平面日志文件条目 309
了解 AWS 支付密码学数据平面日志文件条目 312
加密详细信息 315
设计目标 316
基本原理 316
加密基元 317
熵和随机数生成 317
对称密钥操作 317
非对称密钥操作 318
密钥存储 318
使用对称密钥导入密钥 318
使用非对称密钥导入密钥 318
密钥导出 319
每笔交易派生唯一密钥 (DUKPT) 协议 319
密钥层次结构 319
内部操作 321
HSM 保护 322
通用密钥管理 324
客户密钥的管理 327
通信安全 328
日志记录和监控 329
客户操作 329
生成密钥 330
导入密钥 330
导出密钥 331
删除 密钥 331
轮换 密钥 331
配额 332
文档历史记录 333
..... cccxxxv

什么是 AWS 支付密码学？

AWS Payment Cryptography 是一项托管 AWS 服务，可根据支付卡行业 (PCI) 标准提供对支付处理中使用的加密功能和密钥管理的访问权限，而无需您购买专用的支付 HSM 实例。 AWS Payment Cryptography 为执行支付功能的客户（例如收单机构、支付促进机构、网络、交换机、处理器和银行）提供了将其支付加密操作迁移到更靠近云端应用程序的地方，并最大限度地减少对包含专用支付的辅助数据中心或托管设施的依赖。 HSMs

该服务旨在满足适用的行业规则，包括 PCI PIN、PCI P2PE 和 PCI DSS，并且该服务利用经过 PCI PTS HSM V3 和 FIPS 140-2 Level 3 认证的硬件。它旨在支持低延迟、高水平的正常运行时间和弹性。 AWS Payment Cryptography 具有完全的弹性 HSMs，消除了内部部署的许多操作要求，例如需要配置硬件、安全地管理密钥材料以及在安全的设施中维护紧急备份。 AWS Payment Cryptography 还为您提供了以电子方式与合作伙伴共享密钥的选项，无需共享纸质明文组件。

您可以使用 AWS Payment Cryptography 控制面板 API 来创建和管理密钥。

您可以使用 AWS Payment Cryptography 数据面板 API，以便使用加密密钥进行与支付相关的交易处理和相关的加密操作。

AWS 支付密码学提供了可用于管理密钥的重要功能：

- 创建和管理对称和非对称 AWS 支付加密密钥，包括 TDES、AES 和 RSA 密钥，并指定其预期用途，例如生成 CVV 或 DUKPT 密钥派生。
- 在硬件安全模块 (HSMs) 的保护下，自动安全地存储您的 AWS 支付密码学密钥，同时在用例之间强制执行密钥分离。
- 创建、删除、列出和更新别名，这些别名是 “友好名称”，可用于访问或控制对您的 P AWS ayment Cryptography 密钥的访问权限。
- 标记您的 AWS 支付密码学密钥以进行识别、分组、自动化、访问控制和成本跟踪。
- 使用密钥加密密钥 (KEK) 在 P AWS ayment Cryptography 和 HSM（或第三方）之间导入和导出对称密钥，符合 TR-31（可互操作的安全密钥交换密钥区块规范）。
- 使用非对称密钥对在 AWS 支付密码学和其他系统之间导入和导出对称密钥加密密钥 (KEK)，然后使用电子手段，例如 TR-34（使用非对称技术分发对称密钥的方法）。

您可以在加密操作中使用您的 AWS 支付加密密钥，例如：

- 使用对称或非对称 AWS 支付加密密钥对数据进行加密、解密和重新加密。

- 根据 PCI PIN 规则，在加密密钥之间安全地转换敏感数据（例如持卡人密码），而不会暴露明文。
- 生成或验证持卡人数据，例如 CVV 或 ARQC。 CVV2
- 生成并验证持卡人 pin 码。
- 生成或验证 MAC 签名。

概念

了解 AWS 支付密码学中使用的基本术语和概念，以及如何使用它们来帮助您保护数据。

别名

与 AWS 支付加密密钥关联的用户友好名称。在许多 AWS 支付密码学 API 操作中，别名可以与密钥 ARN 互换使用。别名允许轮换或以其他方式更改密钥，而不会影响您的应用程序代码。别名是最多 256 个字符的字符串。它可以唯一标识账户和区域内关联的 AWS 支付密码密钥。在 AWS 支付密码学中，别名始终以开头。`alias/`

别名的格式如下：

```
alias/<alias-name>
```

例如：

```
alias/sampleAlias2
```

密钥 ARN

密钥 ARN 是 AWS Payment Cryptography 中密钥条目的 Amazon 资源名称（ARN））。它是 AWS 支付密码学密钥的唯一且完全限定的标识符。密钥 ARN 包括 AWS 账户、区域和随机生成的 ID。ARN 与密钥材料无关，也并非源自密钥材料。由于它们是在创建或导入操作期间自动分配的，因此这些值不具备幂等性。多次导入同一个密钥将导致多个密钥 ARNs 具有自己的生命周期。

密钥 ARN 的格式如下：

```
arn:<partition>:payment-cryptography:<region>:<account-id>:alias/<alias-name>
```

以下是示例密钥 ARN：

```
arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h
```

密钥标识符

密钥标识符是对密钥的引用，其中一个（或多个）是 AWS 支付密码学操作的典型输入。有效的密钥标识符可以是 Key Arn 或密钥别名。

AWS 付款密码学密钥

AWS 支付密码学密钥（密钥）用于所有加密功能。密钥可以由您使用 create key 命令直接生成，也可以通过调用密钥导入添加到系统中。可以通过查看属性来确定密钥的来源 KeyOrigin。 AWS Payment Cryptography 还支持加密操作期间使用的派生密钥或中间密钥，例如 DUKPT 使用的密钥。

这些密钥在创建时定义了不可变和可变属性。算法、长度和用法等属性是在创建时定义的，无法更改。其他内容（例如生效日期或到期日期）可以修改。有关AWS 支付密码学密钥属性的完整列表，请参阅 AWS 支付密码学 API 参考。

AWS 支付密码学密钥的密钥类型主要由 ANSI X9 TR 31 定义，这些密钥类型仅限于 PCI PIN v3.1 要求 19 中规定的预期用途。

按照 PCI PIN v3.1 第 18-3 条要求的规定进行存储、与其他账户共享或导出时，属性将使用密钥块绑定到密钥。

密钥在 AWS 支付加密平台中使用称为密钥 Amazon 资源名称 (ARN) 的唯一值进行识别。

> **Note**
>
> 密钥ARN是在最初创建密钥或将密钥导入 AWS 支付加密服务时生成的。因此，如果使用导入密钥功能多次添加相同的密钥材料，则相同的密钥材料将位于多个密钥 ARNS 下，但每个密钥都有不同的密钥生命周期。

行业术语

主题

- 常见密钥类型
- 其他术语

常见密钥类型

AWS 付款密码学密钥

AWS 付款加密密钥存在于单个 AWS 区域密钥中。它由存储在 AWS 支付加密服务中的关键元数据和材料组成。密钥可以作为密 TR-31 钥块从外部来源导入，也可以由 AWS 支付加密服务生成。

AWK

收单机构工作密钥 (AWK) 是通常用于在 acquirer/acquirer 处理器和网络（例如Visa或 Mastercard）之间交换数据的密钥。从历史上看，AWK 利用 3DES 进行加密，将表示为 TR31_P0_PIN_ENCRYPTION_KEY。

BDK

基础派生密钥 (BDK) 是用于派生后续密钥的工作密钥，通常用作 PCI PIN 和 PCI P2PE DUKPT 流程的一部分。它表示为 TR31_B0_BASE_DERIVATION_KEY。

CMK

卡片主密钥 (CMK) 是一个或多个卡牌专用密钥，通常源自发行人主密钥 P AN 和 PSN，通常是 3D ES 密钥。在个性化过程中，这些密钥存储在 EMV 芯片上。CMK 的示例包括 AC、SMI 和 SMC 密钥。

CMK-AC

应用程序密码 (AC) 密钥用作 EMV 交易的一部分，用于生成交易密码，是一种卡片主密钥。

CMK-SMI

安全消息完整性 (SMI) 密钥用作 EMV 的一部分，用于验证使用 MAC 发送到卡的有效负载（例如 pin 码更新脚本）的完整性。它是一种卡片主密钥。

CMK-SMC

安全消息保密性 (SMC) 密钥用作 EMV 的一部分，用于加密发送到卡片的数据，例如密码更新。它是一种卡片主密钥。

CVK

卡片验证密钥 (CVK) 是用于使用定义的算法生成 CVV、CVV2 和类似值以及验证输入的密钥。它表示为 TR31_C0_CARD_VERIFICATION_KEY。

IMK

发行方主密钥 (IMK) 是用作 EMV 芯片卡个性化一部分的主密钥。通常会有 3 个 IMK，分别用于 AC（密码）、SMI（脚本主密钥 integrity/signature）和 SMC（脚本主密钥）密钥。 confidentiality/encryption

IK

初始密钥 (IK) 是 DUKPT 过程中使用的第一个密钥，源自基本派生密钥 (BDK)。此密钥上不会处理任何交易，但它用于派生未来将用于交易的密钥。中定义了创建 IK 的派生方法。 X9.24-1:2017使用 TDES BDK 时， X9.24-1:2009 是适用的标准，IK 被初始密码加密密钥 (IPEK) 所取代。

IPEK

初始 PIN 加密密钥 (IPEK) 是 DUKPT 流程中使用的初始密钥，源自基本派生密钥 (BDK)。此密钥上不会处理任何交易，但它用于派生未来将用于交易的密钥。IPEK 用词不当，因为这个密钥也可以用来派生数据加密和 Mac 密钥。中定义了创建 IPEK 的推导方法。 X9.24-1:2009使用 AES BDK 时， X9.24-1:2017 是适用的标准，IPEK 被初始密钥 (IK) 所取代。

IWK

发行者工作密钥 (IWK) 是通常用于在 issuer/issuer 处理器和网络（例如Visa或 Mastercard）之间交换数据的密钥。从历史上看，IWK 利用 3DES 进行加密，表示为 TR31_P0_PIN_ENCRYPTION_KEY。

KBPK

密钥块保护密钥 (KBPK) 是一种对称密钥，用于保护密钥块，从而 wrap/encrypt 保护其他密钥。KBPK与KEK类似，但是K E K直接保护密钥材料， TR-31 而在类似的方案中，KBPK仅间接保护工作密钥。使用时 TR-31，TR31_K1_KEY_BLOCK_PROTECTION_KEY_KEY_PROTECTION_KEY 是正确的密钥类型，尽管出于历史目的，可以互换支持 TR31_K0_ KEY_NECRYPTION_KEY。

KEK

密钥加密密钥 (KEK) 是用于加密其他密钥以进行传输或存储的密钥。根据标准，用于保护其他密钥的密钥通常为 TR31_K0_KE KeyUsage Y_ENCRYPTION_KE Y。TR-31

PEK

PIN 加密密钥 (PEK) 是一种工作密钥，用于加密 PIN 以进行存储或在双方之间传输。IWK 和 AWK 是 pin 加密密钥具体用途的两个示例。这些密钥表示为 TR31_P0_PIN_ENCRYPTION_KEY。

PGK

PGK（Pin 生成密钥）是 P in 验证密钥的另一个名称。它实际上并不用于生成引脚（默认情况下，引脚是加密随机数），而是用于生成验证值，例如PVV。

PRK

主区域密钥是已启用复制功能的给定支付加密密钥的权威复制源。PRK 是指密钥复制配置中的来源支付密码学 Multi-Region 密钥角色。在付款加密密钥上启用复制后，该特定密钥复制配置被称为 PRK。

PVK

PIN 验证密钥 (PVK) 是一种工作密钥，用于生成 PVV 等 PIN 验证值。两种最常见的类型是用于生成 IBM3624 偏移值的 TR31_V1_IBM3624_PIN_VERIFICATION_KEY 和用于验证值的 TR31_V2_VISA_PIN_VERIFICATION_KEY。 Visa/ABA 这也可以称为 Pin 生成密钥。

RRK

副本区域密钥是从 PRK 安全复制到配置副本的复制密钥材料和元数据 AWS 区域。RRK 是支付加密密钥的只读副本。RRK 是指特定密钥在密 Multi-Region 钥复制配置中扮演的角色。任何关键元数据更改（包括复制设置）都必须应用于 PRK。

其他术语

ARQC

授权请求密码 (ARQC) 是由 EMV 标准芯片卡（或等效的非接触式实现）在交易时生成的密码。通常，ARQC 由芯片卡生成，并在交易时转发给发卡机构或其代理进行验证。

CVV

信用卡验证值是一种静态的秘密值，传统上嵌入在磁条上，用于验证交易的真实性。该算法还用于其他用途，例如 iCvV、CAVV、CVV2。对于其他用例，它可能不会以这种方式嵌入。

CVV2

信用卡验证值 2 是一种静态的秘密值，传统上印在支付卡的正面（或背面），用于验证不在卡上的付款（例如通过电话或在线）的真实性。它使用与 CVV 相同的算法，但服务代码设置为 000。

iCvV

iCvV 是一个 CVV2-like 值，但在 EMV（芯片）卡上嵌入了 track2 的等效数据。该值是使用服务代码 999 计算得出的，它不同 CVV1/CVV2 于防止窃取的信息被用于创建不同类型的新付款凭证。例如，如果获得了芯片交易数据，则无法使用这些数据生成磁条 (CVV1) 或用于在线购买 (CVV2)。

它使用一把???钥匙

DUKPT

派生的每笔交易唯一密钥 (DUKPT) 是一种密钥管理标准，通常用于定义在物理上使用一次性加密密钥。 POS/POI从历史上看，DUKPT 利用 3DES 进行加密。DUKPT 的行业标准在 ANSI 中定义。 X9.24-3-2017

ECC

ECC（椭圆曲线密码学）是一种公钥密码系统，它使用椭圆曲线的数学来创建加密密钥。ECC 提供的安全级别与 RSA 等传统方法相同，但密钥长度要短得多，从而以更有效的方式提供同等的安全性。这与 RSA 不是切实可行的解决方案（RSA 密钥长度 > 4096 位）的用例尤其相关。 AWS 支付密码学支持 NIST 定义的曲线，用于 ECDH 操作。

ECDH

ECDH（Elliptic Curve Diffie-Hellman）是一种密钥协议协议，允许双方建立共享密钥（例如KE K或PEK）。在ECDH中，甲方和乙方各有自己的公私密钥对，彼此交换公钥（以 AWS 支付密码学证书的形式）以及密钥派生元数据（派生方法、哈希类型和共享信息）。双方将自己的私钥乘以对方的公钥，由于椭圆曲线的属性，双方都能够推导（生成）生成的密钥。

EMV

EMV（最初是Europay、Mastercard、Visa）是一个与支付利益相关者合作创建可互操作的支付标准和技术的技术机构。一个示例标准是针对 chip/contactless 卡及其与之交互的支付终端，包括使用的加密技术。EMV 密钥派生是指根据一组初始密钥为每张支付卡生成唯一密钥的方法，例如 IMK

HSM

硬件安全模块 (HSM) 是一种物理设备，用于保护加密操作（例如加密、解密和数字签名）以及用于这些操作的底层密钥。

KCAAS

密钥托管人即服务 (KCAAS) 提供与密钥管理相关的各种服务。对于支付密钥，他们通常可以将纸质密钥组件转换为 AWS 付款密码学支持的电子表格，或者将受电子保护的密钥转换为某些供应商可能需要的纸质组件。他们还可能为丢失会对您的持续运营造成不利影响的密钥提供密钥托管服务。KCAAS供应商能够以符合PCI DSS、PCI PIN和PCI P2PE标准的方式帮助客户减轻在 AWS 支付密码学等安全服务之外管理密钥材料的运营负担。 AWS Payment Cryptography 物理密钥交换作为内置 KCAAS 功能提供，用于将纸质关键组件转换为电子格式。

KCV

密钥校验值 (KCV) 是指各种校验和方法，主要用于在无需访问实际密钥材料的情况下比较彼此的密钥。KCV 也被用于完整性验证（尤其是在交换密钥时），尽管此角色现在已作为密钥块格式的一部分包括在内，例如 TR-31。对于 TDES 密钥，KCV 是通过使用要检查的密钥对每个值为零的 8 个字节进行加密，并保留加密结果的 3 个最高阶字节来计算的。对于 AES 密钥，KCV 使用 CMAC 算法计算，其中输入数据为 16 个字节的零，并保留加密结果的 3 个最高位字节。对于 HMAC 密钥，KCV 是使用在零长度消息上创建密钥时选择的哈希值计算的，取最左边的 3 个字节。

KDH

密钥分发主机 (KDH) 是在密钥交换过程中发送密钥的设备或系统，例如TR-34。从 AWS Payment Cryptography 发送密钥时，它被视为 KDH。

KIF

密钥注入工具 (KIF) 是一种安全设施，用于初始化支付终端，包括向支付终端加载加密密钥。

KRD

密钥接收设备 (KRD) 是在密钥交换过程中接收密钥的设备，例如TR-34。向 AWS 支付密码学发送密钥时，它被视为 KRD。

KSN

密钥序列号 (KSN) 是一个用作 DUKPT encryption/decryption 输入的值，用于为每笔交易创建唯一的加密密钥。KSN 通常由一个 BDK 标识符、一个半唯一的终端 ID 以及一个交易计数器组成，该计数器在给定支付终端上处理的每次转换时递增。尽管密钥集ID和终端ID之间的边界对 AWS 支付密码学的功能没有影响，但对于TDES来说，10字节的KSN通常由24位的密钥集ID组成，19位用于终端ID，21位用于交易计数器。 X9.24对于 AES，12 字节的 KSN 通常由 BDK ID 的 32 位、派生标识符 (ID) 的 32 位和事务计数器的 32 位组成。

mPoC

mPoC（商业硬件上的移动销售点）是一项 PCI 标准，它满足了解决方案的安全要求，这些解决方案使商家能够使用智能手机或其他商用现货 (COTS) 移动设备接受持卡人 PIN 或非接触式支付。

PAN

主账号 (PAN) 是信用卡或借记卡等账户的唯一标识符。长度通常为 13-19 位数字。前 6-8 位数字标识网络和发卡行。

PIN 码块

在处理或传输过程中包含 PIN 码以及其他数据元素的数据块。PIN 码块格式标准化了 PIN 码块的内容以及如何处理它以检索 PIN 码。大多数 PIN 块由 PIN 和 PIN 长度组成，通常包含部分或全部

PAN。 AWS 支付密码学支持 ISO 9564-1 格式 0、1、3 和 4。AES 密钥需要格式 4。在验证或转换 PIN 码时，需要指定传入或传出数据的 PIN 码块。

POI

互动点 (POI) 也经常与销售点 (POS) 匿名使用，是持卡人与之交互以出示其支付凭证的硬件设备。POI 的一个示例是商家位置的实物终端。有关经过认证的 PCI PTS POI 终端列表，请访问 PCI 网站。

PSN

PAN 序列号 (PSN) 是一个数值，用于区分使用相同 PAN 发行的多张卡。

公有密钥

使用非对称密码（RSA、ECC）时，公钥是公私密钥对的公共组成部分。加密详细信息介绍公有密钥可以共享并分发给需要为公有-私有密钥对拥有者加密数据的实体。对于数字签名操作，公有密钥用于验证签名。

私有密钥

使用非对称密码（RSA、ECC）时，私钥是公私密钥对的私有组件。私有密钥用于解密数据或创建数字签名。与对称 AWS 支付密码学密钥类似，私钥由 HSM 安全创建。这些密钥仅在 HSM 的易失存储器中解密，并且仅在处理加密请求所需的时间内解密。

PVV

PIN 验证值 (PVV) 是一种加密输出，可用于在不存储实际引脚的情况下验证引脚。尽管这是一个通用术语，但在 AWS 支付密码学的背景下，PVV是指Visa或ABA PVV方法。这个 PVV 是一个四位数的数字，其输入是卡号、平移序列号、平移本身和 PIN 验证密钥。在验证阶段，Payment Cryptography在内部使用交易数据重新创建PVV，并将其与 AWS AWS 支付密码学客户存储的值再次进行比较。这样，它在概念上类似于加密哈希或 MAC。

RSA Wrap/Unwrap

RSA wrap 使用非对称密钥来封装对称密钥（例如 TDES 密钥），以便传输到另一个系统。只有具有匹配私钥的系统才能解密有效负载并加载对称密钥。相反，RSA 解包将安全地解密使用 RSA 加密的密钥，然后将该密钥加载到支付密码中。 AWS RSA wrap 是一种交换密钥的低级方法，它不以密钥块格式传输密钥，也不使用发送方的有效载荷签名。应考虑使用其他控制措施来确定天意和关键属性不会发生变化。

TR-34 也在内部使用 RSA，但它是一种单独的格式，不可互操作。

服务代码

服务代码是编码在支付卡磁条和芯片上的三位数值，用于定义银行卡的属性，例如可以在哪里使用信用卡以及是否需要额外授权。计算验证值时，服务代码用作输入。不同的服务代码值会生成不同类型的验证值，例如 CVV（使用卡磁条中的服务代码）、iCVV（芯片卡的服务代码 999）和 CVV 2（服务代码 000）。

TR-31

TR-31 （正式定义为 ANSI X9 TR 31）是一种密钥块格式，由美国国家标准协会 (ANSI) 定义，支持在与密钥数据本身相同的数据结构中定义关键属性。 TR-31 密钥块格式定义了一组与密钥关联的按键属性，以便将它们组合在一起。 AWS Payment Cryptography 尽可能使用 TR-31 标准化术语来确保正确的密钥分离和密钥用途。 TR-31 已被 ANSI 取代。 X9.143-2022

TR-34

TR-34 是 ANSI 的实现 X9.24-2 ，它描述了一种使用非对称技术（例如 RSA）安全分发对称密钥（例如 3DES 和 AES）的协议。 AWS 支付密码学使用允许安全导入和导出密钥 TR-34 的方法。

X9.143

X9.143 是一种密钥块格式，由美国国家标准协会 (ANSI) 定义，用于支持在同一数据结构中保护密钥和关键属性。密钥块格式定义了一组与密钥关联的按键属性，以便将它们组合在一起。 AWS Payment Cryptography 尽可能使用 X9.143 标准化术语来确保正确的密钥分离和密钥用途。 X9.143 取代了之前的TR-31提案，尽管在大多数情况下，它们是向后和向前兼容的，而且术语通常可以互换使用。

相关服务

AWS Key Management Service

AWS 密钥管理服务 (AWS KMS) 是一项托管服务，可让您轻松创建和控制用于保护数据的加密密钥。 AWS KMS 使用硬件安全模块 (HSMs) 来保护和验证您的 AWS KMS 密钥。

AWS CloudHSM

AWS CloudHSM 在 AWS 云中为客户提供专用的通用型 HSM 实例。 AWS CloudHSM 可以提供各种加密功能，例如创建密钥、数据签名或加密和解密数据。

有关更多信息

- 要了解 AWS 支付密码学中使用的术语和概念，请参阅AWS 支付密码学概念。
- 有关 AWS 支付密码控制面板 API 的信息，请参阅AWS 支付密码控制面板 API 参考。
- 有关 AWS 支付密码学数据面板 API 的信息，请参阅AWS 支付加密数据平面 API 参考。
- 有关支付密码学如何使用密码学和保护 AWSAWS 支付密码学密钥的详细技术信息，请参阅加密详细信息。

的终端节点 AWS Payment Cryptography

要以编程方式连接 AWS Payment Cryptography，您可以使用终端节点，即服务入口点的 URL。 AWS 软件开发工具包和命令行工具会 AWS 区域 根据请求的区域上下文自动使用服务的默认终端节点，因此通常无需显式设置这些值。必要时，您可以为 API 请求指定不同的终端节点。

控制面板端点

区域名称	区域	端点	协议	
美国东部（俄亥俄州）	us-east-2	controlplane.payment-cryptography.us-east-2.amazonaws.com controlplane.payment-cryptography.us-east-2.api.aws	HTTPS HTTPS	
美国东部（弗吉尼亚州北部）	us-east-1	controlplane.payment-cryptography.us-east-1.amazonaws.com controlplane.payment-cryptography.us-east-1.api.aws	HTTPS HTTPS	
美国西部（俄勒冈州）	us-west-2	controlplane.payment-cryptography.us-west-2.amazonaws.com controlplane.payment-cryptography.us-west-2.api.aws	HTTPS HTTPS	
非洲（开普敦）	af-south-1	controlplane.payment-cryptography.af-south-1.amazonaws.com	HTTPS	

区域名称	区域	端点	协议
		controlplane.payment-cryptography.af-south-1.api.aws	HTTPS
亚太地区（海得拉巴）	ap-south-2	controlplane.payment-cryptography.ap-south-2.amazonaws.com controlplane.payment-cryptography.ap-south-2.api.aws	HTTPS HTTPS
亚太地区（孟买）	ap-south-1	controlplane.payment-cryptography.ap-south-1.amazonaws.com controlplane.payment-cryptography.ap-south-1.api.aws	HTTPS HTTPS
亚太地区（大阪）	ap-northeast-3	controlplane.payment-cryptography.ap-northeast-3.amazonaws.com controlplane.payment-cryptography.ap-northeast-3.api.aws	HTTPS HTTPS
亚太地区（新加坡）	ap-southeast-1	controlplane.payment-cryptography.ap-southeast-1.amazonaws.com controlplane.payment-cryptography.ap-southeast-1.api.aws	HTTPS HTTPS
亚太地区（悉尼）	ap-southeast-2	controlplane.payment-cryptography.ap-southeast-2.amazonaws.com controlplane.payment-cryptography.ap-southeast-2.api.aws	HTTPS HTTPS
亚太地区（东京）	ap-northeast-1	controlplane.payment-cryptography.ap-northeast-1.amazonaws.com controlplane.payment-cryptography.ap-northeast-1.api.aws	HTTPS HTTPS

区域名称	区域	端点	协议
加拿大（中部）	ca-central-1	controlplane.payment-cryptography.ca-central-1.amazonaws.com controlplane.payment-cryptography.ca-central-1.api.aws	HTTPS HTTPS
欧洲地区（法兰克福）	eu-central-1	controlplane.payment-cryptography.eu-central-1.amazonaws.com controlplane.payment-cryptography.eu-central-1.api.aws	HTTPS HTTPS
欧洲地区（爱尔兰）	eu-west-1	controlplane.payment-cryptography.eu-west-1.amazonaws.com controlplane.payment-cryptography.eu-west-1.api.aws	HTTPS HTTPS
欧洲地区（伦敦）	eu-west-2	controlplane.payment-cryptography.eu-west-2.amazonaws.com controlplane.payment-cryptography.eu-west-2.api.aws	HTTPS HTTPS
欧洲地区（巴黎）	eu-west-3	controlplane.payment-cryptography.eu-west-3.amazonaws.com controlplane.payment-cryptography.eu-west-3.api.aws	HTTPS HTTPS
南美洲（圣保罗）	sa-east-1	controlplane.payment-cryptography.sa-east-1.amazonaws.com controlplane.payment-cryptography.sa-east-1.api.aws	HTTPS HTTPS

数据面板端点

区域名称	区域	端点	协议
美国东部（俄亥俄州）	us-east-2	dataplane.payment-cryptography.us-east-2.amazonaws.com dataplane.payment-cryptography.us-east-2.api.aws	HTTPS HTTPS
美国东部（弗吉尼亚州北部）	us-east-1	dataplane.payment-cryptography.us-east-1.amazonaws.com dataplane.payment-cryptography.us-east-1.api.aws	HTTPS HTTPS
美国西部（俄勒冈州）	us-west-2	dataplane.payment-cryptography.us-west-2.amazonaws.com dataplane.payment-cryptography.us-west-2.api.aws	HTTPS HTTPS
非洲（开普敦）	af-south-1	dataplane.payment-cryptography.af-south-1.amazonaws.com dataplane.payment-cryptography.af-south-1.api.aws	HTTPS HTTPS
亚太地区（海得拉巴）	ap-south-2	dataplane.payment-cryptography.ap-south-2.amazonaws.com dataplane.payment-cryptography.ap-south-2.api.aws	HTTPS HTTPS
亚太地区（孟买）	ap-south-1	dataplane.payment-cryptography.ap-south-1.amazonaws.com dataplane.payment-cryptography.ap-south-1.api.aws	HTTPS HTTPS

区域名称	区域	端点	协议
亚太地区（大阪）	ap-northeast-3	dataplane.payment-cryptography.ap-northeast-3.amazonaws.com dataplane.payment-cryptography.ap-northeast-3.api.aws	HTTPS HTTPS
亚太地区（新加坡）	ap-southeast-1	dataplane.payment-cryptography.ap-southeast-1.amazonaws.com dataplane.payment-cryptography.ap-southeast-1.api.aws	HTTPS HTTPS
亚太地区（悉尼）	ap-southeast-2	dataplane.payment-cryptography.ap-southeast-2.amazonaws.com dataplane.payment-cryptography.ap-southeast-2.api.aws	HTTPS HTTPS
亚太地区（东京）	ap-northeast-1	dataplane.payment-cryptography.ap-northeast-1.amazonaws.com dataplane.payment-cryptography.ap-northeast-1.api.aws	HTTPS HTTPS
加拿大（中部）	ca-central-1	dataplane.payment-cryptography.ca-central-1.amazonaws.com dataplane.payment-cryptography.ca-central-1.api.aws	HTTPS HTTPS
欧洲地区（法兰克福）	eu-central-1	dataplane.payment-cryptography.eu-central-1.amazonaws.com dataplane.payment-cryptography.eu-central-1.api.aws	HTTPS HTTPS

区域名称	区域	端点	协议
欧洲地区（爱尔兰）	eu-west-1	dataplane.payment-cryptography.eu-west-1.amazonaws.com dataplane.payment-cryptography.eu-west-1.api.aws	HTTPS HTTPS
欧洲地区（伦敦）	eu-west-2	dataplane.payment-cryptography.eu-west-2.amazonaws.com dataplane.payment-cryptography.eu-west-2.api.aws	HTTPS HTTPS
欧洲地区（巴黎）	eu-west-3	dataplane.payment-cryptography.eu-west-3.amazonaws.com dataplane.payment-cryptography.eu-west-3.api.aws	HTTPS HTTPS
南美洲（圣保罗）	sa-east-1	dataplane.payment-cryptography.sa-east-1.amazonaws.com dataplane.payment-cryptography.sa-east-1.api.aws	HTTPS HTTPS

AWS 支付密码学入门

要开始使用 AWS 支付加密，您首先需要创建密钥，然后在各种加密操作中使用它们。以下教程提供了生成用于 generating/verifying CVV2 值的密钥的简单用例。要尝试其他示例并探索 AWS 中的部署模式，请尝试以下AWS 支付密码学研讨会或浏览我们的示例项目 GitHub

本教程将引导您创建单个密钥并使用该密钥执行加密操作。之后，如果您不再需要密钥，则可以将其删除，从而完成密钥的生命周期。

> **Warning**
>
> 本用户指南中的示例可能使用示例值。我们强烈建议不要在生产环境中使用样本值，例如密钥序列号。

主题

- 先决条件
- 步骤 1：创建密钥
- 步骤 2：使用密钥生成 CVV2 值
- 步骤 3：验证已在步骤 2 中生成的值
- 第 4 步：进行阴性测试
- 第 5 步：（可选）清除

先决条件

在您开始之前，请确保：

- 您有权访问该服务。有关更多信息，请参阅 IAM policy。
- 您已安装 AWS CLI。您也可以使用AWS SDKs或AWS APIs访问 AWS 支付密码学，但本教程中的说明使用。 AWS CLI

步骤 1：创建密钥

第一步是创建一个密钥。在本教程中，您将创建一个用于生成和验证 CVV/ 值的 CVK 双长度 3DES (2KEY TDES) 密钥。CVV2

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
        "KeyAttributes": {
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "CADDA1",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

请注意代表密钥的 KeyArn，例如：arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi。您需要在下一步中执行该操作。

步骤 2：使用密钥生成 CVV2 值

在此步骤中，您将使用步骤 1 中的密钥 CVV2 为给定的PAN到期日期生成一个。

```
$  aws payment-cryptography-data generate-card-validation-data \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --primary-account-number=171234567890123 \
    --generation-attributes CardVerificationValue2={CardExpiryDate=0123}
```

```
{
    "CardDataGenerationKeyCheckValue": "CADDA1",
    "CardDataGenerationKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/tqv5yij6wtxx64pi",
    "CardDataType": "CARD_VERIFICATION_VALUE_2",
    "CardDataValue": "144"
}
```

注意 cardDataValue，在本例中为 3 位数字 144。您需要在下一步中执行该操作。

步骤 3：验证已在步骤 2 中生成的值

在此示例中，您将使用在步骤 1 中创建的密钥验证 CVV2 来自步骤 2 的。

运行以下命令来验证 CVV2。

```
$  aws payment-cryptography-data  verify-card-validation-data \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --primary-account-number=171234567890123 \
    --verification-attributes CardVerificationValue2={CardExpiryDate=0123} \
    --validation-data 144
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "CADDA1"
}
```

该服务返回 200 的 HTTP 响应，表示它已验证 CVV2。

第 4 步：进行阴性测试

在此步骤中，您将创建一个阴性测试，其中 CVV2 不正确且无法验证。您尝试 CVV2 使用在步骤 1 中创建的密钥来验证不正确的密钥。例如，如果持卡人在结账时输入了错误 CVV2 的内容，则这是预期的操作。

```
$  aws payment-cryptography-data  verify-card-validation-data \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --primary-account-number=171234567890123 \
    --verification-attributes CardVerificationValue2={CardExpiryDate=0123} \
    --validation-data 999
```

```
Card validation data verification failed.
```

该服务返回 400 的 HTTP 响应，消息为“信用卡验证数据验证失败”，原因为 INVALID_VALIDATION_DATA。

第 5 步：（可选）清除

现在，您可以删除已在步骤 1 中创建的密钥。为最大限度地减少不可恢复的更改，默认密钥删除期为七天。

```
$ aws payment-cryptography delete-key \
    --key-identifier=arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi
```

```
{
    "Key": {
        "CreateTimestamp": "2022-10-27T08:27:51.795000-07:00",
        "DeletePendingTimestamp": "2022-11-03T13:37:12.114000-07:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_3KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
```

```
                "Decrypt": true,
                "DeriveKey": false,
                "Encrypt": true,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": true,
                "Verify": false,
                "Wrap": true
            },
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY"
        },
        "KeyCheckValue": "CADDA1",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "DELETE_PENDING",
        "UsageStartTimestamp": "2022-10-27T08:27:51.753000-07:00"
    }
}
```

请注意输出中的两个字段。默认情况下，deletePendingTimestamp 设置为未来的七天。keyState 设置为 DELETE_PENDING。您可以在预定删除时间之前的任何时间通过调用 restore-key 取消此删除。

管理密钥

要开始使用 AWS 支付加密技术，请创建 AWS 支付加密密钥。

本节介绍如何在整个生命周期中创建和管理各种 AWS 支付加密密钥类型。您将学习如何创建、查看和编辑密钥，以及如何标记密钥、创建密钥别名以及启用或禁用密钥。

AWS 支付密码学密钥是一种区域资源。如果您打算在多个分区和账户中使用给定密钥 AWS 区域，则可以启用 Multi-Region 密钥复制，将密钥材料和元数据安全地复制到 AWS 区域 您在同一个 AWS 分区和账户中指定的密钥中。密钥复制中的源 Multi-Region 密钥被称为主区域密钥 (PRK)，它仍然是所有密钥管理活动的权威来源。复制的密钥称为副本区域密钥 (RRK)，这是 PRK 的只读副本。您应该考虑将 Multi-Region 密钥与密钥一起使用，以实现可用性、灾难恢复和低延迟方面的设计目标。

主题

- 创建密钥
- 列出密钥
- 启用和禁用 密钥
- 复制 AWS 支付加密密钥
- 删除 密钥
- 导入和导出密钥
- 使用别名
- 获取密钥
- 标记密钥
- 了解的关键属性 AWS 支付加密密钥

创建密钥

您可以使用 CreateKey API 操作创建 AWS 支付加密密钥。创建密钥时，您可以指定密钥算法、密钥用法、允许的操作以及密钥是否可导出等属性。创建 AWS 支付加密密钥后，您无法更改这些属性。

> **Note**
>
> 如果您启用了 Multi-Region 密钥复制功能， AWS 账户 并且您创建了支付加密密钥，则该密钥将自动成为主区域密钥 (PRK)。即使您没有在CreateKey命令中指定`--replication-regions`参数，也会复制 PRK。有关更多信息，请参阅 Multi-Region 密钥复制的工作原理。

示例

- 创建 2KEY TDES 基础派生密钥
- 为创建 2KEY TDES 密钥 CVV/CVV2
- 创建 HMAC 密钥
- 创建密 AES-256 钥
- 创建 PIN 加密密钥 (PEK)
- 创建非对称 (RSA) 密钥
- 创建 PIN 验证值 (PVV) 密钥
- 创建非对称 ECC 密钥

创建 2KEY TDES 基础派生密钥

> **Note**
>
> 尽管 KeyUsage = TR31_B0_BASE_DERIVATION_KEY 支持 3KEY TDES，但目前没有任何操作支持这种密钥类型。ANSI X9.24 标准 (DUKPT) 仅定义了对 TDES_2KEY 和 AES 密钥类型的支持。

Example

此命令为 DUKPT 创建 2KEY TDES 派生密钥，该密钥将从美国西部（俄勒冈）复制到美国东部（俄亥俄州）地区。响应包括请求参数、用于后续调用的亚马逊资源名称 (ARN) 和密钥校验值 (KCV)。

```
$ aws payment-cryptography create-key --exportable --key-attributes \
     "KeyUsage=TR31_B0_BASE_DERIVATION_KEY, \
     KeyClass=SYMMETRIC_KEY,KeyAlgorithm=TDES_2KEY, \
     KeyModesOfUse={DeriveKey=true}" \
     --replication-regions us-east-2 --region us-west-2
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2022-10-26T16:04:11.642000-07:00",
        "Enabled": true,
        "Exportable": true,
```

```
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_2KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": true,
                "Encrypt": false,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": false,
                "Wrap": false
            },
            "KeyUsage": "TR31_B0_BASE_DERIVATION_KEY"
        },
        "KeyCheckValue": "FE23D3",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2022-10-26T16:04:11.559000-07:00"
}
```

为创建 2KEY TDES 密钥 CVV/CVV2

Example

此命令创建 2KEY TDES 密钥，用于生成和验证 CVV/CVV2值。响应包括请求参数、用于后续调用的亚马逊资源名称 (ARN) 和密钥校验值 (KCV)。

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY, \
    KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY, \
    KeyModesOfUse='{Generate=true,Verify=true}'
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2022-10-26T16:04:11.642000-07:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_2KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": false,
                "Encrypt": false,
                "Generate": true,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": true,
                "Wrap": false
            },
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY"
        },
        "KeyCheckValue": "AEA5CD",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2022-10-26T16:04:11.559000-07:00"
    }
}
```

创建 HMAC 密钥

Example

HMAC 密钥用于生成或验证哈希消息身份验证码 (HMAC)。对于 HMAC 密钥，哈希类型是在创建密钥时分配的（例如 HMAC_SHA224 和 HMAC_SHA512），无法修改。

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=HMAC_SHA512,KeyUsage=TR31_M7_HMAC_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse='{Gene
 = true,Verify = true}'
```

输出示例：

```
{
 "Key": {
 "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/qnobl5lghrzunce6",
 "KeyAttributes": {
 "KeyUsage": "TR31_M7_HMAC_KEY",
 "KeyClass": "SYMMETRIC_KEY",
 "KeyAlgorithm": "HMAC_SHA512",
 "KeyModesOfUse": {
 "Encrypt": false,
 "Decrypt": false,
 "Wrap": false,
 "Unwrap": false,
 "Generate": true,
 "Sign": false,
 "Verify": true,
 "DeriveKey": false,
 "NoRestrictions": false
 }
 },
 "KeyCheckValue": "2976E7",
 "KeyCheckValueAlgorithm": "HMAC",
 "Enabled": true,
 "Exportable": true,
 "KeyState": "CREATE_COMPLETE",
 "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
 "CreateTimestamp": "2025-07-30T10:06:12.142000-07:00",
 "UsageStartTimestamp": "2025-07-30T10:06:12.128000-07:00"
 }
}
```

创建密 AES-256 钥

Example

此命令为数据加密和解密创建 AES-256 对称密钥。AES 密钥为敏感数据提供高度加密，通常用于支付处理以加密持卡人数据和其他敏感信息，但是 TDES 更常用于 EMV 等发卡机构用例。

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=AES_256,KeyUsage=TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY,KeyClass=SYMMETRIC_KEY,Key
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2025-02-02T10:15:30.142000-08:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyAlgorithm": "AES_256",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": true,
                "DeriveKey": false,
                "Encrypt": true,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": true,
                "Verify": false,
                "Wrap": true
            },
            "KeyUsage": "TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY"
        },
        "KeyCheckValue": "2976F5",
        "KeyCheckValueAlgorithm": "CMAC",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2025-02-02T10:15:30.128000-08:00"
    }
}
```

创建 PIN 加密密钥 (PEK)

Example

此命令创建用于加密 PIN 值的 3KEY TDES 密钥，尽管根据您的互操作性需求，密码也可以是 AES。在验证期间（例如在交易中），您可以使用此密钥安全地存储 PIN 或解密 PIN。响应包括请求参数、后续调用的 ARN 和 KCV。

```
$ aws payment-cryptography create-key --exportable --key-attributes \
    KeyAlgorithm=TDES_3KEY,KeyUsage=TR31_P0_PIN_ENCRYPTION_KEY, \

 KeyClass=SYMMETRIC_KEY,KeyModesOfUse='{Encrypt=true,Decrypt=true,Wrap=true,Unwrap=true}'
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2022-10-27T08:27:51.795000-07:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_3KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": true,
                "DeriveKey": false,
                "Encrypt": true,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": true,
                "Verify": false,
                "Wrap": true
            },
            "KeyUsage": "TR31_P0_PIN_ENCRYPTION_KEY"
        },
        "KeyCheckValue": "7CC9E2",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2022-10-27T08:27:51.753000-07:00"
    }
}
```

创建非对称 (RSA) 密钥

Example

此命令生成一个新的非对称 RSA 2048 位密钥对。它创建了一个新的私钥及其匹配的公钥。您可以使用 get PublicCertificate API 检索公钥。

```
$ aws payment-cryptography create-key --exportable \
    --key-attributes
 KeyAlgorithm=RSA_2048,KeyUsage=TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION, \
    KeyClass=ASYMMETRIC_KEY_PAIR,KeyModesOfUse='{Encrypt=true,
 Decrypt=True,Wrap=True,Unwrap=True}'
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2022-11-15T11:15:42.358000-08:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
nsq2i3mbg6sn775f",
        "KeyAttributes": {
            "KeyAlgorithm": "RSA_2048",
            "KeyClass": "ASYMMETRIC_KEY_PAIR",
            "KeyModesOfUse": {
                "Decrypt": true,
                "DeriveKey": false,
                "Encrypt": true,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": true,
                "Verify": false,
                "Wrap": true
            },
            "KeyUsage": "TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION"
        },
        "KeyCheckValue": "40AD487F",
        "KeyCheckValueAlgorithm": "SHA-1",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2022-11-15T11:15:42.182000-08:00"
    }
}
```

创建 PIN 验证值 (PVV) 密钥

Example

此命令创建一个 3KEY TDES 密钥来生成 PVV 值。您可以使用此密钥生成 PVV，该PVV可以与随后计算出的 PVV 进行比较。响应包括请求参数、后续调用的 ARN 和 KCV。

```
$ aws payment-cryptography create-key --exportable \
    --key-attributes KeyAlgorithm=TDES_3KEY,KeyUsage=TR31_V2_VISA_PIN_VERIFICATION_KEY,
 \
    KeyClass=SYMMETRIC_KEY,KeyModesOfUse='{Generate=true,Verify=true}'
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2022-10-27T10:22:59.668000-07:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
        "KeyAttributes": {
            "KeyAlgorithm": "TDES_3KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": false,
                "Encrypt": false,
                "Generate": true,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": true,
                "Wrap": false
            },
            "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY"
        },
        "KeyCheckValue": "7F2363",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2022-10-27T10:22:59.614000-07:00"
    }
}
```

创建非对称 ECC 密钥

Example

此命令生成 ECC 密钥对，用于在双方之间建立 ECDH（椭圆曲线 Diffie-Hellman）密钥协议。使用 ECDH，各方生成自己的 ECC 密钥对，密钥用途 K3 和使用模式 X，然后交换公钥。然后，双方使用其私钥和收到的公钥来建立共享的派生密钥。

为了保持支付中加密密钥的一次性使用原则，我们建议不要将ECC密钥对重复用于多种用途，例如ECDH密钥派生和签名。

```
$ aws payment-cryptography create-key --exportable \
    --key-attributes
 KeyAlgorithm=ECC_NIST_P256,KeyUsage=TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT, \
    KeyClass=ASYMMETRIC_KEY_PAIR,KeyModesOfUse='{DeriveKey=true}'
```

输出示例：

```
{
    "Key": {
        "CreateTimestamp": "2024-10-17T01:31:55.908000+00:00",
        "Enabled": true,
        "Exportable": true,
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
wc3rjsssguhxtilv",
        "KeyAttributes": {
            "KeyAlgorithm": "ECC_NIST_P256",
            "KeyClass": "ASYMMETRIC_KEY_PAIR",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": true,
                "Encrypt": false,
                "Generate": false,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": false,
                "Wrap": false
            },
            "KeyUsage": "TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT"
        },
        "KeyCheckValue": "7E34F19F",
        "KeyCheckValueAlgorithm": "SHA-1",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "UsageStartTimestamp": "2024-10-17T01:31:55.866000+00:00"
    }
}
```

列出密钥

使用该ListKeys操作获取您的账户和区域中可供您访问的密钥列表。

Example

```
$ aws payment-cryptography list-keys
```

输出示例：

```
{
    "Keys": [
        {
            "CreateTimestamp": "2022-10-12T10:58:28.920000-07:00",
            "Enabled": false,
            "Exportable": true,
            "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
            "KeyAttributes": {
                "KeyAlgorithm": "TDES_3KEY",
                "KeyClass": "SYMMETRIC_KEY",
                "KeyModesOfUse": {
                    "Decrypt": true,
                    "DeriveKey": false,
                    "Encrypt": true,
                    "Generate": false,
                    "NoRestrictions": false,
                    "Sign": false,
                    "Unwrap": true,
                    "Verify": false,
                    "Wrap": true
                },
                "KeyUsage": "TR31_P1_PIN_GENERATION_KEY"
            },
            "KeyCheckValue": "7F2363",
            "KeyCheckValueAlgorithm": "ANSI_X9_24",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
            "KeyState": "CREATE_COMPLETE",
            "UsageStopTimestamp": "2022-10-27T14:19:42.488000-07:00"
        }
    ]
}
```

启用和禁用 密钥

您可以禁用和重新启用 AWS 支付加密密钥。密钥在创建完成后默认处于启用状态。如果禁用密钥，则在重新启用该密钥之前，该密钥不能用于任何加密操作。 Start/stop 使用命令立即生效，因此建议您在进行此类更改之前查看使用情况。您也可以使用可选 `timestamp` 参数，将更改（开始或停止使用）设置为在将来生效。

由于它是临时性的，很容易撤消，因此禁用 AWS 支付加密密钥是比删除 AWS 支付加密密钥更安全的替代方法，这种操作具有破坏性和不可逆性。如果您正在考虑删除 AWS 支付加密密钥，请先将其禁用，并确保将来无需使用该密钥来加密或解密数据。

主题

- 开始密钥使用
- 停止密钥使用

开始密钥使用

必须启用密钥使用才能使用密钥进行加密操作。如果密钥未启用，则可以通过此操作使其可用。该字段`UsageStartTimestamp`将表示密钥何时处于 became/will 活动状态。对于已启用的令牌，这将是过去，如果令牌待激活，则是将来。

Example

在此示例中，要求启用密钥以使用密钥。响应中包含关键信息，并且启用标志已转换为 true。这也将反映在列表密钥响应对象中。

```
$ aws payment-cryptography start-key-usage --key-identifier "arn:aws:payment-
cryptography:us-east-2:111122223333:key/alsuwfxug3pgy6xh"
```

```
{
      "Key": {
      "CreateTimestamp": "2022-10-12T10:58:28.920000-07:00",
      "Enabled": true,
      "Exportable": true,
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwfxug3pgy6xh",
      "KeyAttributes": {
          "KeyAlgorithm": "TDES_3KEY",
          "KeyClass": "SYMMETRIC_KEY",
          "KeyModesOfUse": {
              "Decrypt": true,
              "DeriveKey": false,
              "Encrypt": true,
              "Generate": false,
              "NoRestrictions": false,
              "Sign": false,
              "Unwrap": true,
              "Verify": false,
              "Wrap": true
          },
          "KeyUsage": "TR31_P1_PIN_GENERATION_KEY"
      },
      "KeyCheckValue": "369D",
      "KeyCheckValueAlgorithm": "ANSI_X9_24",
      "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
      "KeyState": "CREATE_COMPLETE",
      "UsageStartTimestamp": "2022-10-27T14:09:59.468000-07:00"
  }
}
```

停止密钥使用

如果您不再打算使用密钥，则可以停止使用密钥以防止进一步的加密操作。此操作不是永久性的，因此您可以通过开始密钥使用来撤消该操作。您也可以将密钥设置为将来禁用。该字段`UsageStopTimestamp`将表示密钥何时被 became/will 禁用。

Example

在此示例中，要求在将来停止使用密钥。执行后，除非通过开始密钥使用重新启用，否则该密钥不能用于加密操作。响应包含密钥信息，并且启用标志已转换为 false。这也将反映在列表密钥响应对象中。

```
$ aws payment-cryptography stop-key-usage --key-identifier "arn:aws:payment-
cryptography:us-east-2:111122223333:key/alsuwfxug3pgy6xh"
```

```
{
  "Key": {
      "CreateTimestamp": "2022-10-12T10:58:28.920000-07:00",
      "Enabled": false,
      "Exportable": true,
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwfxug3pgy6xh",
      "KeyAttributes": {
          "KeyAlgorithm": "TDES_3KEY",
          "KeyClass": "SYMMETRIC_KEY",
          "KeyModesOfUse": {
              "Decrypt": true,
              "DeriveKey": false,
              "Encrypt": true,
              "Generate": false,
              "NoRestrictions": false,
              "Sign": false,
              "Unwrap": true,
              "Verify": false,
              "Wrap": true
          },
          "KeyUsage": "TR31_P1_PIN_GENERATION_KEY"
      },
      "KeyCheckValue": "369D",
      "KeyCheckValueAlgorithm": "ANSI_X9_24",
      "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
      "KeyState": "CREATE_COMPLETE",
      "UsageStopTimestamp": "2022-10-27T14:09:59.468000-07:00"
  }
}
```

复制 AWS 支付加密密钥

AWS Payment Cryptography 支持密 Multi-Region 钥复制，允许您将任何给定 AWS 支付加密密钥中的密钥材料和元数据安全地分发到同一 AWS 分区和账户 AWS 区域 中的一个或多个支付加密密钥。

源密钥被称为主区域密钥 (PRK)，仍然是所有密钥管理活动的权威来源，而 PRK 和副本区域密钥 (RRK) 均可用于各自的加密操作。 AWS 区域

Multi-Region 密钥复制的好处

以下概述了 Multi-Region 密钥复制的一些好处。

- 更轻松地设置高可用应用程序 -Paym AWS ent Cryptography 可为您处理密钥分配，因此您可以在多个密钥中使用一个密钥，而 AWS 区域 无需创建给定密钥的解耦副本。
- 高可用性和低延迟密钥 -通过密 Multi-Region 钥复制，您可以多次访问密钥，从而 AWS 区域 使密钥高度可用，从而降低延迟。
- 密钥材料耐久性 -副本区域密钥是完整的密钥副本，可以在加密操作中独立于其主区域密钥使用。在 PRK 发生灾难性数据丢失时，RRK 可提供耐用的副本。

Multi-Region 密钥复制的工作原理

启用 Multi-Region 密钥复制后，Paym AWS ent Cryptography 服务使用安全的密钥分发机制将密钥材料和元数据复制到 AWS 区域 您指定的副本。对主区域密钥元数据（例如密钥属性、状态和启用）的更改会自动复制到副本区域密钥中。

限制和注意事项

以下是一些 Multi-Region 关键的复制限制和注意事项。

- 付款加密密钥必须设置为可导出才能使用 Multi-Region 密钥复制。
- 您必须为一个 AWS 区域 或特定的支付加密密钥启用此功能。
  - 如果为启用此功能 AWS 区域，则启用后创建的所有 AWS 支付加密密钥都将复制到指定的。AWS 区域在此区域创建的密钥将成为主区域密钥。该区域中的现有密钥不会自动复制。您可以在 Multi-Region 密钥级别为现有 AWS 区域 密钥启用密钥复制。
  - 每个 AWS 区域 都可以有唯一的 Multi-Region 密钥复制设置。
  - 密钥的 Multi-Region 复制设置优先于 AWS 区域 Multi-Region 密钥复制设置。
- 无法将副本区域密钥配置为复制到其他区域密钥 AWS 区域。

- Multi-Region 密钥复制可用于对称支付加密密钥，例如三重数据加密标准 (3DES)、高级加密标准 (AES) 和 Hash-based消息身份验证码 (HMAC)，以及非对称支付加密密钥，例如 RSA 和椭圆曲线加密 (ECC)。
- 副本区域密钥是只读密钥。对主区域密钥的所有更改都将应用于副本区域密钥。
- 主区域密钥的更改最终与副本区域密钥一致。
- 支付加密密钥只能使用相同的 AWS 分区和账户进行复制。
- 副本区域密钥计入您的 AWS 账户 等级 AWS 支付密码限制。
- 主区域密钥和副本区域密钥使用相同的密钥标识符，这允许您在 IAM 策略中按相同的 ARN 引用这两个密钥。
- 您必须拥有副本 AWS 区域 的`CreateKey`权限才能成功复制。

启用 Multi-Region 密钥复制

您可以通过两种方式为 AWS 支付加密 Multi-Region 密钥启用密钥复制。

1. AWS 区域: 启用 Multi-Region 密钥复制 AWS 区域 后，将应用于在该密钥中创建的所有新密钥。此方法为所有密钥提供一致的复制。
2. 特定 AWS 支付加密密钥：您可以管理单个密 Multi-Region 钥的密钥复制，从而实现更精细的控制。

启用 Multi-Region 密钥复制后，您的支付加密密钥将复制到 AWS 区域 您指定的密钥。

> **Important**
>
> Multi-Region 密钥复制无法暂停。启用复制后，您的密钥将自动复制到 AWS 区域 您指定的密钥。 Multi-Region 可以禁用特定密钥 AWS 区域 或支付加密密钥的密钥复制。要删除副本区域密钥，必须将其 AWS 区域 作为复制区域从主区域密钥中删除。
> 或者，您可以在 PRK 上调用 `StopKeyUsage` API 或 stop-key-usage CLI 命令来停止使用 PRK 和所有关联的 RRK。您将无法在加密操作中使用这些密钥。使用 `StopKeyUsage` API 或 stop-key-usage CLI 命令不会停止正在进行的为你的 PRK 启用的 Multi-Region 密钥复制。

您可以通过调用 `GetDefaultKeyReplicationRegions` API 或 C get-default-key-replication-regions LI 命令 AWS 区域 来检查 Multi-Region 特定中 AWS 支付加密密钥的密钥复制设置。您调用此 API 操作或命令的密钥将成为您的 PRK 。 AWS 区域

使用以下步骤启用 Multi-Region 密钥复制。

For AWS 区域

- 使用以下命令 AWS 区域 为您指定的 Multi-Region 密钥启用密钥复制。在此示例中，在美国东部（俄亥俄州）和美国西部（俄勒冈）启用了 Multi-Region 密钥复制。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography enable-default-key-replication-regions \
    --replication-regions us-east-2 us-west-2
```

> **Note**
>
> 为启用 Multi-Region 密钥复制 AWS 区域 不会更改任何现有 AWS 支付加密密钥的复制配置。您可以在密钥级别为现有密钥启用此功能。只有为启用密 Multi-Region 钥复制后创建的密钥才 AWS 区域 会使用区域复制设置。

For specific AWS Payment Cryptography keys

- 使用以下命令为特定支付加密 Multi-Region 密钥启用密钥复制。在本示例中，在美国东部（俄亥俄州）启用了 Multi-Region 密钥复制。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography add-key-replication-regions \
    --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h \
    --replication-regions us-east-2
```

或者，您可以通过在创建密钥请求 AWS 区域 中添加复制来创建启用此功能的新付款加密密钥。

> **Note**
>
> 密钥复制设置优先于 AWS 区域 复制设置。

禁用 Multi-Region 密钥复制

如果要禁用 Multi-Region 密钥复制，则可以调用disable-default-key-replication或 remove-key-replication-regions CLI 命令，具体取决于 Multi-Region 密钥复制的启用方式。您需要指定密钥的 ARN 并禁用 Multi-Region 密钥复制。 AWS 区域

注意事项

复制区域密钥的删除最终是一致的。

您可以通过调用 `GetDefaultKeyReplicationRegions` API 或 C get-default-key-replication-regions LI 命令 AWS 区域 来检查 Multi-Region 特定中 AWS 支付加密密钥的密钥复制设置。

使用以下步骤禁用 Multi-Region 密钥复制。

For AWS 区域

- 使用以下命令禁用 AWS 区域 您指定的 Multi-Region 密钥复制。在此示例中，美国东部（俄亥俄州）禁用了 Multi-Region 密钥复制。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography disable-default-key-replication-regions \
    --replication-regions us-east-2
```

For specific AWS Payment Cryptography keys

- 使用以下命令禁用特定支付加密 Multi-Region 密钥的密钥复制。在此示例中，美国东部（俄亥俄州）的 Multi-Region 密钥复制处于禁用状态。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography remove-key-replication-regions \
    --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h \
    --replication-regions us-east-2
```

安全注意事项

以下是对支付加密 Multi-Region 密钥使用密钥复制时的安全注意事项。有关更多信息，请参阅 [的安全最佳实践 AWS Payment Cryptography]。

- 限制共享密钥材料。
- 在创建 IAM 策略时，请遵循最低权限权限的主体。
- 您无法更改副本区域密钥，因为它是只读密钥。

最佳实践

以下是使用 AWS 付款加密 Multi-Region 密钥复制密钥时的一些最佳实践。

- 即使不是立即将 Multi-Region 密钥复制到指定密钥， AWS 区域 也要确保您的应用程序继续运行。如果您需要知道 Multi-Region 密钥复制何时完成，则可以使用 GetKey API 操作进行监控。您可以使用监控密钥复制事件AWS CloudTrail。
- 测试和实施自动部署流程，以防从一个区域故障转移 AWS 区域 到另一个区域。

定价

您需要为使用 AWS 支付加密技术创建的副本区域密钥付费。这些密钥按次收费 AWS 区域。有关最新的支付加密定价信息，请参阅AWS 支付加密定价页面。

删除 密钥

删除 AWS 支付加密密钥会删除密钥材料和与该密钥相关的所有元数据，除非在 AWS 支付加密之外有密钥的副本，否则该密钥是不可逆的。删除密钥后，您不能再解密用该密钥加密的数据，这意味着该数据将无法恢复。只有当您确定不再需要使用密钥且其他任何当事方不在使用此密钥时，才能将其删除。如果您不确定，可以考虑停止使用密钥，而不是将其删除。如果以后需要再次使用已禁用的密钥，则可以重新启用该密钥，但是除非能够从其他来源重新导入已删除的 AWS 付款加密密钥，否则无法恢复该密钥。

在删除密钥之前，应确保不再需要该密钥。 AWS Payment Cryptography 不存储 CVV2 等加密操作的结果，也无法确定任何持久加密材料是否需要密钥。

AWS Payment Cryptography 永远不会删除属于活跃 AWS 账户的密钥，除非您明确安排删除密钥，并且强制等待期到期。

但是，您可以出于以下一个或多个原因选择删除 AWS 支付加密密钥：

- 完成不再需要的密钥的密钥生命周期

- 避免与维护未使用的 AWS 支付加密密钥相关的管理开销

> **Note**
>
> 如果您关闭或删除您的 AWS 账户，您的 AWS 支付加密密钥将无法访问。除了关闭账户之外，您无需安排删除 AWS 支付加密密钥。

AWS 当您计划删除支付加密密钥以及实际删除支付加密密钥时， AWS 付款加密会在您的AWS CloudTrail日志中 AWS 记录一个条目。

使用 Multi-Region 密钥复制时，删除作为主区域密钥 (PRK) 的支付加密密钥，副本区域密钥 (RRK) 也将自动删除。无法像 PRK 那样删除 RRK。如果您想删除 RRK，则需要修改 PR K 的复制区域。

关于等待期限

由于删除密钥是不可逆的，因此 AWS 支付密码学要求您将等待期设置为 3—180 天。默认的等待期限为七天。

但是，实际等待期限可能最多比您计划的时间长 24 小时。要获取删除 AWS 支付加密密钥的实际日期和时间，请使用这些 GetKey 操作。请务必记下时区。

在等待期间， AWS 支付加密密钥状态和密钥状态为 “待删除”。

> **Note**
>
> 待删除的 AWS 支付加密密钥不能用于任何加密操作。

等待期结束后， AWS 支付密码学会删除 AWS 支付加密密钥、其别名和所有相关的 AWS 支付加密元数据。

利用等待期确保您现在或将来不需要 AWS 支付加密密钥。如果您在等待期内发现确实需要密钥，可以在等待期限结束前取消密钥删除。等待期限结束后，将无法取消密钥删除， 将删除密钥。

Example

在此示例中，请求删除密钥。除了基本的密钥信息外，还有两个相关字段是密钥状态已更改为 DELETE_PENDING，删除PendingTimestamp 表示当前计划删除密钥的时间。

```
$ aws payment-cryptography delete-key \
                    --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```

```
  {
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_3KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "0A3674",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": false,
        "Exportable": true,
        "KeyState": "DELETE_PENDING",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T12:01:29.969000-07:00",
        "UsageStopTimestamp": "2023-06-05T14:31:13.399000-07:00",
        "DeletePendingTimestamp": "2023-06-12T14:58:32.865000-07:00"
    }
}
```

Example

在此示例中，待处理的删除被取消。成功完成后，密钥将不再按照之前的时间表被删除。响应包含基本的密钥信息；此外，两个相关字段已更改 - KeyState 和 deletePendingTimestamp。KeyState 返回到 CREATE_COMPLETE 的值，同时 DeletePendingTimestamp 被移除。

```
$ aws payment-cryptography restore-key --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_3KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "0A3674",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": false,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-08T12:01:29.969000-07:00",
        "UsageStopTimestamp": "2023-06-08T14:31:13.399000-07:00"
    }
}
```

导入和导出密钥

您可以从其他解决方案导入 AWS 支付加密密钥并将其导出到其他解决方案，例如 HSM。许多客户使用导入和导出功能与服务提供商交换密钥。我们设计了 AWS 支付密码学，以使用现代的电子方法进行密钥管理，帮助您保持合规性和控制力。我们建议使用基于标准的电子密钥交换而不是纸质密钥组件。如果您需要继续处理纸质密钥组件，直到所有合作伙伴都支持电子密钥交换，则可以使用物理密钥交换。

最低密钥优势及其对导入和导出功能的影响

PCI 需要特定的最低密钥强度才能进行加密操作、密钥存储和密钥传输。修订PCI标准后，这些要求可能会发生变化。规则规定，用于存储或传输的封装密钥的强度必须至少与受保护的密钥一样强。我们在导出期间自动执行此要求，并防止密钥受到较弱密钥的保护，如下表所示。

下表显示了支持的封装密钥、保护密钥和保护方法的组合。

	包装钥匙											
保护之钥	TDES	TDES	AES_	AES_	AES_	RSA_	RSA_	RSA_	ecc_	ecc_	ecc_	注意
TDES_2KE	TR-3	TR-3	TR-3	TR-3	TR-3	TR-3	TR-3	TR-3	ECDI	ECDI	ECDI	
TDES_3KE	✗ 不支持	TR-3	TR-3	TR-3	TR-3	TR-3	TR-3	TR-3	ECDI	ECDI	ECDI	
AES_128	✗ 不支持	✗ 不支持	TR-3	TR-3	TR-3	✗ 不支持	TR-3	TR-3	ECDI	ECDI	ECDI	
AES_192	✗ 不支持	✗ 不支持	✗ 不支持	TR-3	TR-3	✗ 不支持	✗ 不支持	✗ 不支持	✗ 不支持	ECDI	ECDI	
AES_256	✗ 不	✗ 不	✗ 不	✗ 不	TR-3	✗ 不	✗ 不	✗ 不	✗ 不	✗ 不	ECDI	

	包装钥匙											
保护之钥	TDES	TDES	AES_	AES_	AES_	RSA_	RSA_	RSA_	ecc_p	ecc_p	ecc_p	注意
	支持	支持	支持	支持		支持	支持	支持	支持	支持		

有关更多信息，请参阅附录 D-PCI HSM 标准中批准算法的最小和等效密钥大小和强度。

密钥加密密钥 (KEK) 交换

我们建议使用 ANSI X9.24 TR-34 标准。这种初始密钥类型可以称为密钥加密密钥 (KEK)、区域主密钥 (ZMK) 或区域控制主密钥 (ZCMK)。如果您的系统或合作伙伴 TR-34 尚不支持，则可以使用 RSA Wrap/Unwrap 。如果您的需求包括交换 AES-256密钥，则可以使用 ECDH 。

> **Note**
>
> 要导入自己的测试密钥或将密钥与现有 HSM 同步，请参阅上的 AWS 支付加密示例代码。GitHub

工作密钥 (WK) 交换

我们使用行业标准（ANSI X9.24 TR 31-2018 和 X9.143）来交换工作密钥。这要求你已经使用 RSA Wrap TR-34、ECDH或类似的方案交换了KEK。这种方法符合 PCI PIN 要求，即始终以加密方式将密钥材料绑定到其类型和用法。工作密钥包括收购方工作密钥、发行者工作密钥、BDK 和 IPEK。

主题

- 导入密钥
- 导出密钥
- 高级主题

导入密钥

> **Important**
>
> 示例需要最新版本的 AWS CLI V2。在开始之前，请确保您已升级到最新版本。

目录

- 导入密钥简介
- 导入对称密钥
  - 使用非对称技术导入密钥 () TR-34
  - 使用非对称技术 (ECDH) 导入密钥
  - 使用非对称技术导入密钥 (RSA Unwrap)
  - 使用预先建立的密钥交换密钥导入对称密钥 () TR-31
- 导入非对称（RSA、ECC）公钥
  - 导入 RSA 公钥
  - 导入 ECC 公钥

导入密钥简介

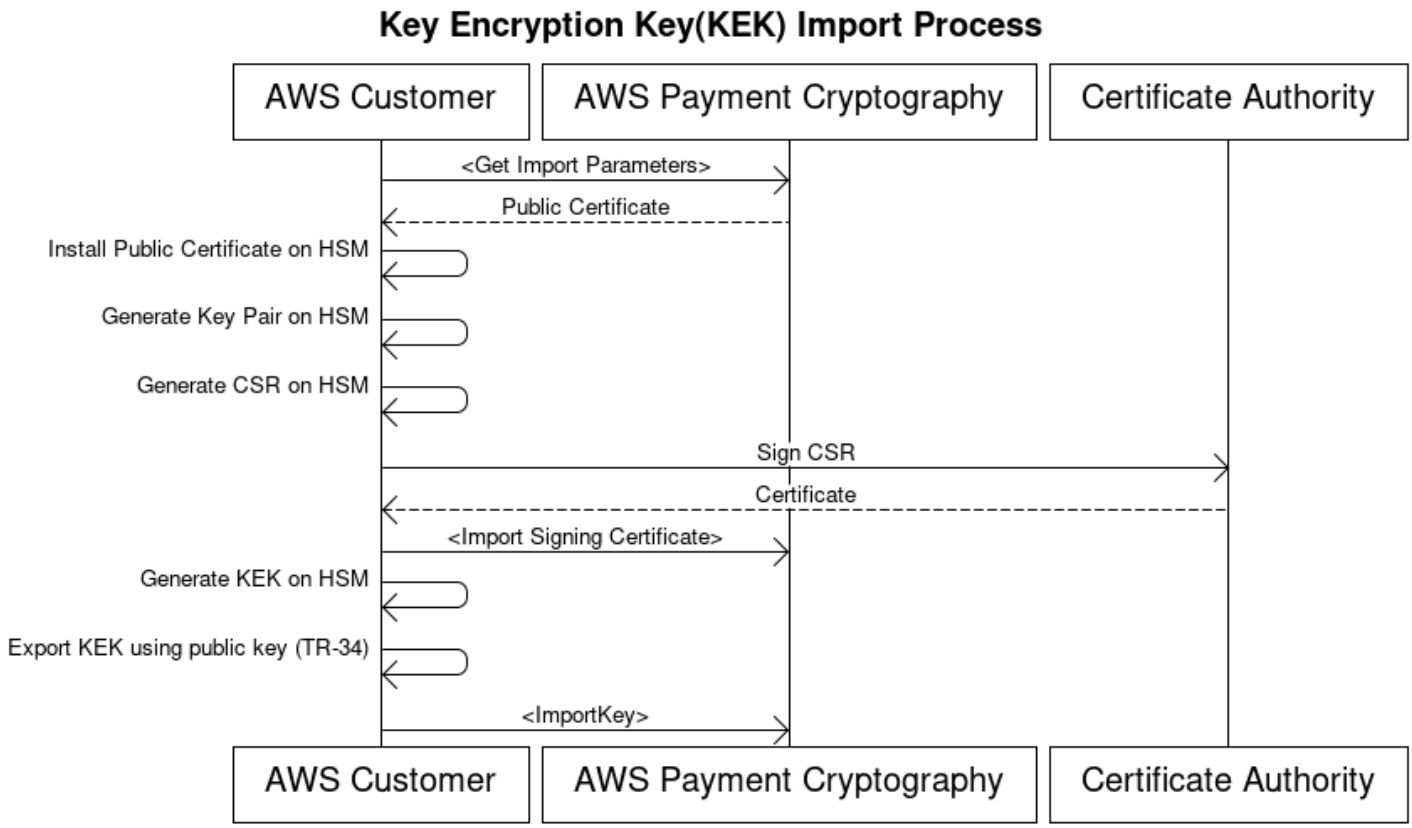
> **Note**
>
> 使用 TR-31 或 TR-34 密钥块导入密钥时 X9.143，Paym AWS ent Cryptography 通常会保留（但不使用）任何可选标头。HM（HMAC 哈希类型）标头用于加密操作。KP 标头（包装密钥的 KCV）特定于导入过程，不保留。

与交易对手交换密钥时，通常首先交换密钥交换密钥（KEK）。然后，该密钥将用于保护后续密钥。使用电子格式，可以使用非对称技术（例如 TR-34 ECDH或RSA包装）来交换KEK。后续密钥将使用对称密钥交换进行交换， TR-31例如。该KEK将持续很长时间，并且只能根据政策及其定义的加密期限每隔几年更新一次。

如果只交换一两个密钥，您也可以选择使用非对称技术直接交换该密钥，例如 BDK。 AWS 支付密码学支持两种密钥交换方法。

导入对称密钥

使用非对称技术导入密钥 () TR-34



TR-34 使用 RSA 非对称加密技术对称密钥进行加密和签名以进行交换。这样可以确保封装密钥的机密性（加密）和完整性（签名）。

要导入自己的密钥，请查看上的 AWS GitHub支付密码学示例项目。有关如何从其他平台进行 import/export 密钥的说明，示例代码可在这些平台上找到，GitHub或参阅这些平台的用户指南。

1. 调用 “初始化导入” 命令

   调用 `get-parameters-for-import` 以初始化导入过程。此 API 为密钥导入生成密钥对，对密钥进行签名，并返回证书和证书根目录。使用此密钥加密要导出的密钥。 TR-34用术语来说，这被称为 KRD 证书。这些证书采用 base64 编码，有效期短，仅用于此目的。保存该`ImportToken`值。

   ```
   $ aws payment-cryptography get-parameters-for-import \
       --key-material-type TR34_KEY_BLOCK \
       --wrapping-key-algorithm RSA_2048
   ```

```
{
    "ImportToken": "import-token-bwxli6ocftypneu5",
    "ParametersValidUntilTimestamp": 1698245002.065,
    "WrappingKeyCertificateChain": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0....",
    "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0....",
    "WrappingKeyAlgorithm": "RSA_2048"
}
```

2. 在密钥源系统上安装公共证书

   对于大多数 HSM，您需要安装、加载或信任步骤 1 中生成的公共证书才能使用它导出密钥。这可能包括整个证书链，也可以仅包括步骤 1 中的根证书，具体取决于 HSM。

3. 在源系统上生成密钥对并提供证书链给 AWS Payment Cryptography

   为确保传输的有效载荷的完整性，发送方（密钥分发主机或 KDH）对其进行签名。为此目的生成公钥并创建公钥证书 (X509) 以提供给 AWS 支付加密。

   从 HSM 传输密钥时，在该 HSM 上创建密钥对。HSM、第三方或诸如之类的服务 AWS 私有 CA 可以生成证书。

   使用带有 o KeyMaterialType f `RootCertificatePublicKey` 和 of 的`importKey`命令将根证书加载到 AWS 支付加密中
   KeyUsageType 。`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`

   对于中间证书，使用带有 o KeyMaterialType f `TrustedCertificatePublicKey` 和 of KeyUsageType 的`importKey`命令`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`。对多个中间证书重复此过程。使用链`KeyArn`中最后导入的证书作为后续导入命令的输入。

   > **Note**
   >
   > 不要导入叶子证书。在导入命令期间直接提供。

4. 从源系统导出密钥

   许多 HSM 和相关系统都支持使用该 TR-34 规范导出密钥。将步骤 1 中的公钥指定为 KRD（加密）证书，将步骤 3 中的密钥指定为 KDH（签名）证书。要导入到 AWS 支付加密系统，请将该格式指定为 TR-34.2012 非 CMS 两阶段格式，也可称为 TR-34 Diebold 格式。

5. 调用导入密钥

使用 o KeyMaterialType f TR34_KEY_BLOCK 调用 ImportKey API。使用步骤 3 中导入的最后一个 CA 的 KeyArn `certificate-authority-public-key-identifier`、步骤 4 中的封装密钥材料以及步骤 3 中的叶子证书。`key-material` `signing-key-certificate`包括步骤 1 中的导入令牌。

```
$ aws payment-cryptography import-key \
    --key-material='{"Tr34KeyBlock": { \
    "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/zabouwe3574jysdl", \
    "ImportToken": "import-token-bwxli6ocftypneu5", \
    "KeyBlockFormat": "X9_TR34_2012", \
    "SigningKeyCertificate":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2RENDQXFTZ0F3SUJ...", \
    "WrappedKeyBlock":
 "308205A106092A864886F70D010702A08205923082058E020101310D300B0609608648016503040201308203.
 \
    }'
```

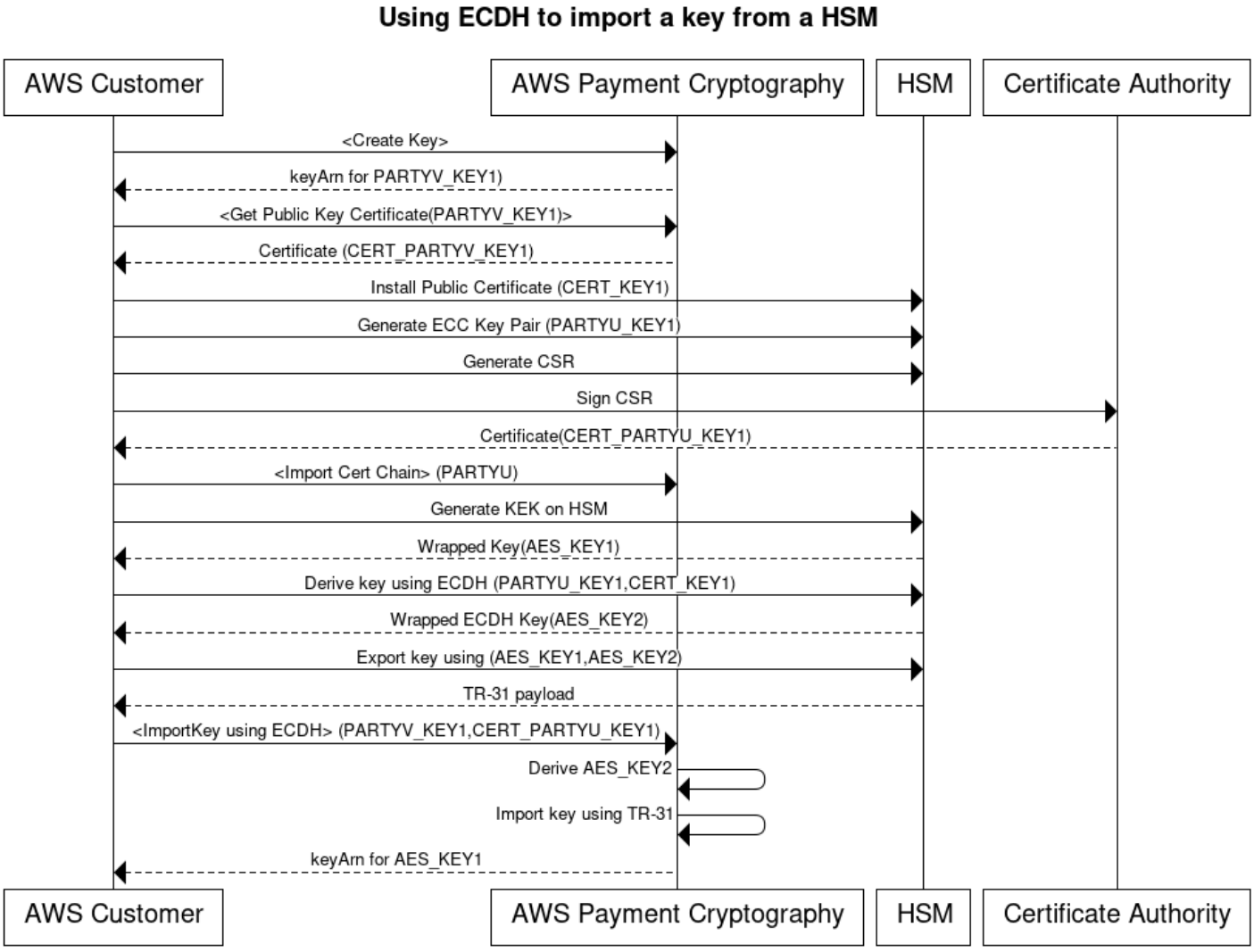
```
{
  "Key": {
    "CreateTimestamp": "2023-06-13T16:52:52.859000-04:00",
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
    "KeyAttributes": {
      "KeyAlgorithm": "TDES_3KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyModesOfUse": {
        "Decrypt": true,
        "DeriveKey": false,
        "Encrypt": true,
        "Generate": false,
        "NoRestrictions": false,
        "Sign": false,
        "Unwrap": true,
        "Verify": false,
        "Wrap": true
      },
      "KeyUsage": "TR31_K1_KEY_ENCRYPTION_KEY"
    },
    "KeyCheckValue": "CB94A2",
```

```
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "KeyOrigin": "EXTERNAL",
    "KeyState": "CREATE_COMPLETE",
    "UsageStartTimestamp": "2023-06-13T16:52:52.859000-04:00"
  }
}
```

6. 使用导入的密钥进行加密操作或后续导入

   如果导入 KeyUsage 的是 TR31_K0_KEY_ENCRYPTION_KEY，则可以使用此密钥进行后续密钥导入。 TR-31对于其他密钥类型（例如 TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY），您可以直接使用该密钥进行加密操作。

使用非对称技术 (ECDH) 导入密钥



椭圆曲线 Diffie-Hellman (ECDH) 使用 ECC 非对称加密技术在双方之间建立共享密钥，无需预先交换密钥。ECDH 密钥是短暂的，因此 AWS 支付密码学不存储它们。在此过程中，使用ECDH进行一次性KBPK/KEK计算。该派生密钥将立即用于包装您要传输的实际密钥，该密钥可能是另一个 KBPK、IPEK 密钥或其他密钥类型。

导入时，发送系统通常称为第 U 方（发起方），AWS 支付加密称为第 V 方（响应方）。

> **Note**
>
> 虽然 ECDH 可用于交换任何对称密钥类型，但它是唯一可以安全传输 AES-256 密钥的方法。

1. 生成 ECC 密钥对

   致电`create-key`为该过程创建 ECC 密钥对。此 API 为密钥导入或导出生成密钥对。创建时，指定使用此 ECC 密钥可以派生哪种密钥。使用 ECDH 交换（包装）其他密钥时，使用值。`TR31_K1_KEY_BLOCK_PROTECTION_KEY`

   > **Note**
   >
   > 尽管低级 ECDH 会生成可用于任何目的的派生密钥，但 AWS 支付密码学允许密钥仅用于单一派生密钥类型，从而限制了密钥意外重复用于多种用途。

   ```
   $ aws payment-cryptography create-key --exportable --key-attributes
    KeyAlgorithm=ECC_NIST_P256,KeyUsage=TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT,KeyClass=ASYM
    --derive-key-usage "TR31_K1_KEY_BLOCK_PROTECTION_KEY"
   ```

   ```
   {
                   "Key": {
                       "KeyArn": "arn:aws:payment-cryptography:us-
   east-2:111122223333:key/wc3rjsssguhxtilv",
                       "KeyAttributes": {
                           "KeyUsage": "TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT",
                           "KeyClass": "ASYMMETRIC_KEY_PAIR",
                           "KeyAlgorithm": "ECC_NIST_P256",
                           "KeyModesOfUse": {
                               "Encrypt": false,
                               "Decrypt": false,
                               "Wrap": false,
   ```

```
                            "Unwrap": false,
                            "Generate": false,
                            "Sign": false,
                            "Verify": false,
                            "DeriveKey": true,
                            "NoRestrictions": false
                        }
                    },
                    "KeyCheckValue": "2432827F",
                    "KeyCheckValueAlgorithm": "CMAC",
                    "Enabled": true,
                    "Exportable": true,
                    "KeyState": "CREATE_COMPLETE",
                    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                    "CreateTimestamp": "2025-03-28T22:03:41.087000-07:00",
                    "UsageStartTimestamp": "2025-03-28T22:03:41.068000-07:00"
                }
            }
```

2. 获取公钥证书

   get-public-key-certificate致电接收公钥作为 X.509 证书，该证书由您的账户 CA 签署，该证书专用于特定地区的 AWS 支付加密。

   Example

```
$ aws payment-cryptography get-public-key-certificate \
                --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/wc3rjsssguhxtilv
```

```
{
            "KeyCertificate": "LS0tLS1CRUdJTi...",
            "KeyCertificateChain": "LS0tLS1CRUdJT..."
      }
```

3. 在交易对手系统上安装公共证书（U 方）

   对于许多 HSM，您需要安装、加载或信任步骤 1 中生成的公共证书才能使用它导出密钥。这可能包括整个证书链，也可以仅包括步骤 1 中的根证书，具体取决于 HSM。有关更多信息，请查阅您的 HSM 文档。

4. 在源系统上生成 ECC 密钥对并提供证书链给 AWS Payment Cryptography

在 ECDH 中，各方生成一对密钥并商定一个通用密钥。为了使 AWS 支付密码学得出密钥，它需要交易对手的公钥格式的公钥。 X.509

从 HSM 传输密钥时，在该 HSM 上创建密钥对。对于支持密钥块的 HSM，密钥标头将类似于`D0144K3EX00E0000`。创建证书时，通常在 HSM 上生成 CSR，然后 HSM、第三方或服务等 AWS 私有 CA 可以生成证书。

使用带有 o KeyMaterialType f `RootCertificatePublicKey` 和 of 的`importKey`命令将根证书加载到 AWS 支付加密中
KeyUsageType 。`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`

对于中间证书，使用带有 o KeyMaterialType f `TrustedCertificatePublicKey` 和 of KeyUsageType 的`importKey`命令`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`。对多个中间证书重复此过程。使用链`KeyArn`中最后导入的证书作为后续导入命令的输入。

> **Note**
>
> 不要导入叶子证书。在导入命令期间直接提供。

5. 在 Party U HSM 上使用 ECDH 派生一次性密钥

   许多 HSM 和相关系统支持使用 ECDH 建立密钥。将步骤 1 中的公钥指定为公钥，将步骤 3 中的密钥指定为私钥。有关允许的选项，例如派生方法，请参阅 API 指南。

   > **Note**
   >
   > 诸如哈希类型之类的派生参数必须在两端完全匹配。否则，您将生成不同的密钥。

6. 从源系统导出密钥

   最后，使用标准 TR-31 命令将要传输的密钥导出到 AWS 支付加密系统。将 ECDH 派生密钥指定为 KBPK。要导出的密钥可以是任何 TDES 或 AES 密钥，只要包装密钥的强度至少与要导出的密钥一样强大。 TR-31

7. 调用导入密钥

   `import-key`使用 o KeyMaterialType f 调用 API `DiffieHellmanTr31KeyBlock`。使用步骤 3 中导入的最后一个 CA 的 KeyArn `certificate-authority-public-key-`

identifier、步骤 4 中的封装密钥材料以及步骤 3 中的叶子证书。key-material public-key-certificate包括步骤 1 中的私钥 ARN。

```
$ aws payment-cryptography import-key \
          --key-material='{
            "DiffieHellmanTr31KeyBlock": {
              "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-
cryptography:us-east-2:111122223333:key/swseahwtq2oj6zi5",
              "DerivationData": {
                "SharedInformation": "1234567890"
              },
              "DeriveKeyAlgorithm": "AES_256",
              "KeyDerivationFunction": "NIST_SP800",
              "KeyDerivationHashAlgorithm": "SHA_256",
              "PrivateKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/wc3rjsssguhxtilv",
              "PublicKeyCertificate":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN....",
              "WrappedKeyBlock":
 "D0112K1TB00E0000D603CCA8ACB71517906600FF8F0F195A38776A7190A0EF0024F088A5342DB98E2735084A7
            }
          }'
```

```
{
        "Key": {
          "CreateTimestamp": "2025-03-13T16:52:52.859000-04:00",
          "Enabled": true,
          "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
          "KeyAttributes": {
            "KeyAlgorithm": "TDES_3KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
              "Decrypt": true,
              "DeriveKey": false,
              "Encrypt": true,
              "Generate": false,
              "NoRestrictions": false,
              "Sign": false,
              "Unwrap": true,
              "Verify": false,
              "Wrap": true
```

```
            },
            "KeyUsage": "TR31_K1_KEY_ENCRYPTION_KEY"
          },
          "KeyCheckValue": "CB94A2",
          "KeyCheckValueAlgorithm": "ANSI_X9_24",
          "KeyOrigin": "EXTERNAL",
          "KeyState": "CREATE_COMPLETE",
          "UsageStartTimestamp": "2025-03-13T16:52:52.859000-04:00"
        }
      }
```

8. 使用导入的密钥进行加密操作或后续导入

   如果导入 KeyUsage 的是 TR31_K0_KEY_ENCRYPTION_KEY，则可以使用此密钥进行后续密钥导入。 TR-31对于其他密钥类型（例如 TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY），您可以直接使用该密钥进行加密操作。

使用非对称技术导入密钥 (RSA Unwrap)

概述： AWS 支付密码学在 TR-34 不可行的情况下支持 RSA wrap/unwrap 进行密钥交换。例如 TR-34，该技术使用RSA非对称加密来加密对称密钥以进行交换。但是，与之不同的是 TR-34，这种方法不要求发送方对有效载荷进行签名。此外，这种 RSA 封装技术无法在传输过程中保持密钥元数据的完整性，因为它不包括密钥块。

> **Note**
>
> 您可以使用 RSA 封装来导入或导出 TDES 和 AES-128 密钥。

1. 调用 “初始化导入” 命令

   调get-parameters-for-import用以 o KeyMaterialType f 初始化导入过程KEY_CRYPTOGRAM。RSA_2048用于交换 TDES 密钥WrappingKeyAlgorithm时使用。RSA_4096在交换 TDES RSA_3072 或 AES-128 密钥时使用或。此 API 为密钥导入生成密钥对，使用证书根对密钥进行签名，并返回证书和证书根。使用此密钥加密要导出的密钥。这些证书的有效期很短，仅用于此目的。

```
$ aws payment-cryptography get-parameters-for-import \
    --key-material-type KEY_CRYPTOGRAM \
    --wrapping-key-algorithm RSA_4096
```

```
{
  "ImportToken": "import-token-bwxli6ocftypneu5",
  "ParametersValidUntilTimestamp": 1698245002.065,
  "WrappingKeyCertificateChain": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0....",
  "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0....",
  "WrappingKeyAlgorithm": "RSA_4096"
}
```

2. 在密钥源系统上安装公共证书

   对于许多 HSM，您需要安装、加载或信任步骤 1 中生成的公共证书（and/or其根）才能使用它导出密钥。

3. 从源系统导出密钥

   许多 HSM 和相关系统都支持使用 RSA 封装导出密钥。将步骤 1 中的公钥指定为加密证书 (`WrappingKeyCertificate`)。如果您需要信任链，请使用步骤 1 中的。`WrappingKeyCertificateChain`从 HSM 导出密钥时，将格式指定为 RSA，填充模式 = PKCS #1 v2.2 OAEP（使用 SHA 256 或 SHA 512）。

4. Call 导入密钥

   import-key使用 o `KeyMaterialType` f 调用 API `KeyMaterial`。您需要步骤 1 `ImportToken` 中的内容和步骤 3 中的`key-material`（封装密钥材料）。提供密钥参数（例如密钥用法），因为 RSA 封装不使用密钥块。

```
$ cat import-key-cryptogram.json
```

```
{
  "KeyMaterial": {
    "KeyCryptogram": {
      "Exportable": true,
      "ImportToken": "import-token-bwxli6ocftypneu5",
      "KeyAttributes": {
        "KeyAlgorithm": "AES_128",
        "KeyClass": "SYMMETRIC_KEY",
        "KeyModesOfUse": {
          "Decrypt": true,
          "DeriveKey": false,
          "Encrypt": true,
          "Generate": false,
          "NoRestrictions": false,
```

```
          "Sign": false,
          "Unwrap": true,
          "Verify": false,
          "Wrap": true
        },
        "KeyUsage": "TR31_K0_KEY_ENCRYPTION_KEY"
      },
      "WrappedKeyCryptogram": "18874746731....",
      "WrappingSpec": "RSA_OAEP_SHA_256"
    }
  }
}
```

```
$ aws payment-cryptography import-key --cli-input-json file://import-key-
cryptogram.json
```

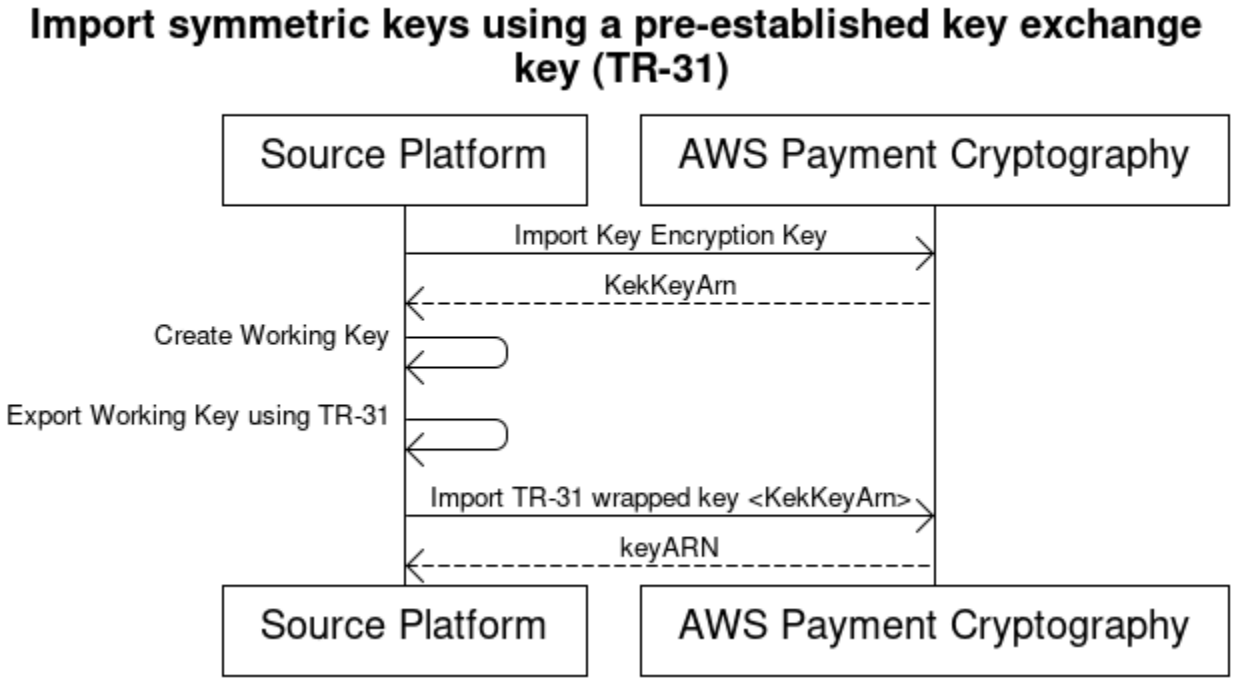
```
{
  "Key": {
    "KeyOrigin": "EXTERNAL",
    "Exportable": true,
    "KeyCheckValue": "DA1ACF",
    "UsageStartTimestamp": 1697643478.92,
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
    "CreateTimestamp": 1697643478.92,
    "KeyState": "CREATE_COMPLETE",
    "KeyAttributes": {
      "KeyAlgorithm": "AES_128",
      "KeyModesOfUse": {
        "Encrypt": true,
        "Unwrap": true,
        "Verify": false,
        "DeriveKey": false,
        "Decrypt": true,
        "NoRestrictions": false,
        "Sign": false,
        "Wrap": true,
        "Generate": false
      },
      "KeyUsage": "TR31_K0_KEY_ENCRYPTION_KEY",
      "KeyClass": "SYMMETRIC_KEY"
```

```
    },
    "KeyCheckValueAlgorithm": "CMAC"
  }
}
```

5. 使用导入的密钥进行加密操作或后续导入

   如果导入KeyUsage的
   是TR31_K0_KEY_ENCRYPTION_KEY或TR31_K1_KEY_BLOCK_PROTECTION_KEY，
   则可以使用此密钥进行后续密钥导入 TR-31。如果密钥类型是任何其他类型（例
   如TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY），则可以直接使用该密钥进行加密操作。

使用预先建立的密钥交换密钥导入对称密钥 () TR-31



交换多个密钥或支持密钥轮换时，合作伙伴通常首先交换初始密钥加密密钥 (KEK)。您可以使用TR-34或物理密钥交换等技术将 KEK 与 AWS 支付加密技术交换。

建立 KEK 后，您可以使用它来传输后续密钥（包括其他 KEK）。 AWS 支付加密支持使用 ANSI 进行这种密钥交换 TR-31，HSM 供应商广泛使用和支持 ANSI。

1. 导入密钥加密密钥 (KEK)

   确保你已经导入了 KEK 并有 KeyArn（或 KeyAlias）可用。

2. 在源平台上创建密钥

如果密钥不存在，请在源平台上创建。或者，您可以在 AWS 支付密码学上创建密钥并使用该export命令。

3. 从源平台导出密钥

   导出时，将导出格式指定为 TR-31。源平台将要求导出密钥和要使用的密钥加密密钥。

4. 导入到 AWS Payment Cryptography

   调用import-key命令时，使用您的密钥加密密钥的 KeyARN（或别名）。`WrappingKeyIdentifier`使用源平台的输出`WrappedKeyBlock`。

Example

```
$ aws payment-cryptography import-key \
    --key-material='{"Tr31KeyBlock": { \
    "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza", \
    "WrappedKeyBlock":
 "D0112B0AX00E00002E0A3D58252CB67564853373D1EBCC1E23B2ADE7B15E967CC27B85D5999EF58E11662991F
 \
    }'
```

```
{
  "Key": {
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
    "KeyAttributes": {
      "KeyUsage": "TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY",
      "KeyClass": "SYMMETRIC_KEY",
      "KeyAlgorithm": "AES_128",
      "KeyModesOfUse": {
        "Encrypt": true,
        "Decrypt": true,
        "Wrap": true,
        "Unwrap": true,
        "Generate": false,
        "Sign": false,
        "Verify": false,
        "DeriveKey": false,
        "NoRestrictions": false
      }
    },
    "KeyCheckValue": "0A3674",
    "KeyCheckValueAlgorithm": "CMAC",
    "Enabled": true,
    "Exportable": true,
    "KeyState": "CREATE_COMPLETE",
    "KeyOrigin": "EXTERNAL",
    "CreateTimestamp": "2023-06-02T07:38:14.913000-07:00",
    "UsageStartTimestamp": "2023-06-02T07:38:14.857000-07:00"
  }
}
```

导入非对称（RSA、ECC）公钥

所有导入的证书必须至少与链中颁发的（前身）证书一样强大。这意味着，RSA_2048 CA 只能用于保护 RSA_2048 分支证书，ECC 证书必须受另一份同等强度的 ECC 证书保护。ECC P384 证书只能由 P384 或 P521 CA 颁发。所有证书在导入时必须未过期。

导入 RSA 公钥

AWS 支付加密支持将公共 RSA 密钥作为 X.509 证书导入。要导入证书，请先导入其根证书。所有证书在导入时必须未过期。证书应采用 PEM 格式并采用 base64 编码。

1. 将根证书导入 AWS Payment Cryptography

   使用以下命令导入根证书：

Example

2. 将公钥证书导入 AWS Payment Cryptography

   现在，您可以导入公钥。由于 TR-34 和 ECDH 依赖于在运行时传递分支证书，因此此选项仅在使用来自其他系统的公钥加密数据时使用。 KeyUsage 将设置为 TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION。

Example

```
$ aws payment-cryptography import-key \
    --key-material='{"Tr31KeyBlock": { \
    "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza", \
    "WrappedKeyBlock":
 "D0112B0AX00E00002E0A3D58252CB67564853373D1EBCC1E23B2ADE7B15E967CC27B85D5999EF58E11662991F
 \
    }'
```

```
{
  "Key": {
    "CreateTimestamp": "2023-08-08T18:55:46.815000+00:00",
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/4kd6xud22e64wcbk",
    "KeyAttributes": {
      "KeyAlgorithm": "RSA_4096",
      "KeyClass": "PUBLIC_KEY",
      "KeyModesOfUse": {
        "Decrypt": false,
        "DeriveKey": false,
        "Encrypt": false,
        "Generate": false,
        "NoRestrictions": false,
        "Sign": false,
        "Unwrap": false,
        "Verify": true,
        "Wrap": false
      },
      "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"
    },
    "KeyOrigin": "EXTERNAL",
    "KeyState": "CREATE_COMPLETE",
    "UsageStartTimestamp": "2023-08-08T18:55:46.815000+00:00"
  }
}
```

导入 ECC 公钥

AWS 支付加密支持将公共 ECC 密钥作为 X.509 证书导入。要导入证书，请先导入其根 CA 证书和任何中间证书。所有证书在导入时必须未过期。证书应采用 PEM 格式并采用 base64 编码。

1. 将 ECC 根证书导入 AWS Payment Cryptography

   使用以下命令导入根证书：

Example

2. 将中间证书导入 AWS Payment Cryptography

   使用以下命令导入中间证书：

Example

3. 将公钥证书（Leaf）导入到 AWS Payment Cryptography

   尽管您可以导入 Leaf ECC 证书，但目前 AWS 支付加密中除了存储之外没有为其定义的函数。这是因为在使用 ECDH 函数时，叶子证书是在运行时传递的。

导出密钥

目录

- 导出对称密钥
  - 使用非对称技术导出密钥 () TR-34
  - 使用非对称技术 (ECDH) 导出密钥
  - 使用非对称技术导出密钥 (RSA Wrap)
  - 使用预先建立的密钥交换密钥导出对称密钥 () TR-31
- 导出 DUKPT 初始密钥 () IPEK/IK
- 指定要导出的键块标题
  - 常用标题
- 导出非对称 (RSA) 密钥

导出对称密钥

> **Important**
>
> 在开始 AWS CLI 之前，请确保您拥有最新版本的。要升级，请参阅安装 AWS CLI。

使用非对称技术导出密钥 () TR-34

TR-34 使用 RSA 非对称加密技术对称密钥进行加密和签名以进行交换。加密保护机密性，而签名则确保完整性。当您导出密钥时，P AWS ayment Cryptography 充当密钥分发主机 (KDH)，您的目标系统将成为密钥接收设备 (KRD)。

> **Note**
>
> 如果您的 HSM 支持 TR-34 导出但不支持 TR-34 导入，我们建议您首先使用在 HSM 和 AWS 支付加密之间建立共享 KEK。 TR-34然后，您可以使用 TR-31转移剩余的密钥。

1. 初始化导出过程

   运行get-parameters-for-export为密钥导出生成密钥对。我们使用这个密钥对来签署 TR-34 有效载荷。 TR-34 用术语来说，这是 KDH 签名证书。这些证书的有效期很短，仅在中ParametersValidUntilTimestamp指定的期限内有效。

   > **Note**
   >
   > 所有证书均采用 base64 编码。

   Example

   ```
   $ aws payment-cryptography get-parameters-for-export \
       --signing-key-algorithm RSA_2048 \
       --key-material-type TR34_KEY_BLOCK
   ```

   ```
   {
     "SigningKeyCertificate":
    "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2RENDQXFTZ0F3SUJ...",
     "SigningKeyCertificateChain": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS....",
     "SigningKeyAlgorithm": "RSA_2048",
     "ExportToken": "export-token-au7pvkbsq4mbup6i",
     "ParametersValidUntilTimestamp": "2023-06-13T15:40:24.036000-07:00"
   }
   ```

2. 导入 AWS 向您的收款系统发送付款加密证书

   将步骤 1 中的证书链导入您的接收系统。

3. 设置接收系统的证书

   为了保护传输的有效载荷，发送方 (KDH) 对其进行加密。您的接收系统（通常是您的 HSM 或合作伙伴的 HSM）需要生成公钥并创建 X.509 公钥证书。您可以使用 AWS 私有 CA 生成证书，但可以使用任何证书颁发机构。

   获得证书后，使用ImportKey命令将根证书导入 AWS 支付加密。将 KeyMaterialType 设置为 RootCertificatePublicKey，将 KeyUsageType 设置为 TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE。

我们TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE之所以使用，KeyUsageType是因为这是签署叶子证书的根密钥。您无需将分 AWS 支证书导入 Payment Cryptography，您可以将它们内联传递。

> **Note**
>
> 如果您之前导入了根证书，请跳过此步骤。对于中间证书，请使用TrustedCertificatePublicKey。

4. 导出你的密钥

使用KeyMaterialType设置为调ExportKey用 API TR34_KEY_BLOCK。你需要提供：

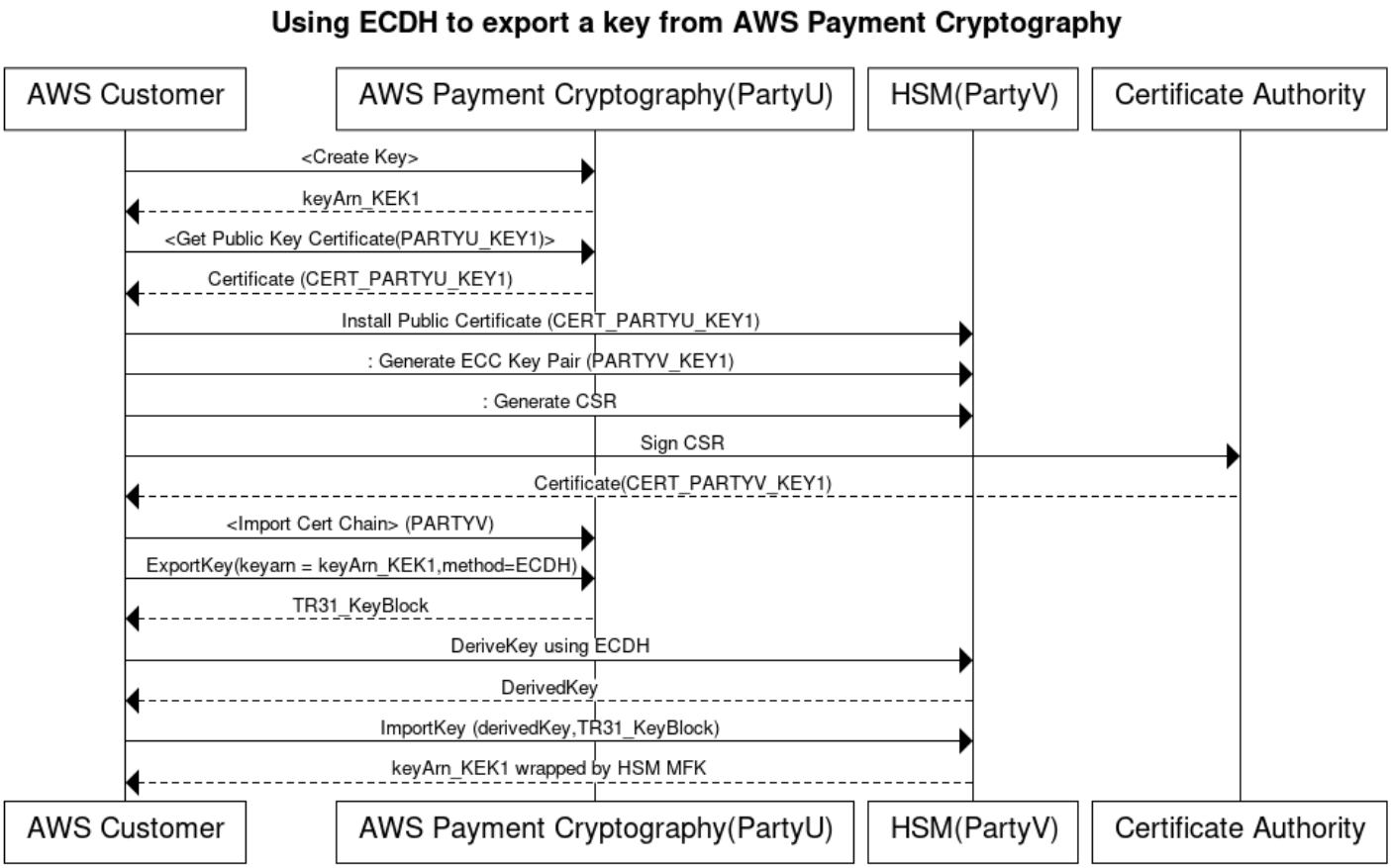
- 步骤 3 中根 CA 的 keyArn 作为 CertificateAuthorityPublicKeyIdentifier
- 步骤 3 中的叶子证书为 WrappingKeyCertificate
- 要导出的密钥的 KeyArn（或别名）作为 --export-key-identifier
- 步骤 1 中的导出令牌

Example

```
$ aws payment-cryptography export-key \
    --export-key-identifier "example-export-key" \
    --key-material '{"Tr34KeyBlock": { \
    "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/4kd6xud22e64wcbk", \
    "ExportToken": "export-token-au7pvkbsq4mbup6i", \
    "KeyBlockFormat": "X9_TR34_2012", \
    "WrappingKeyCertificate":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2RENDQXFXZ0F3SUJBZ0lSQ..."} \
    }'
```

```
{
  "WrappedKey": {
    "KeyMaterial": "308205A106092A864886F70D010702A08205923082058...",
    "WrappedKeyMaterialFormat": "TR34_KEY_BLOCK"
  }
}
```

使用非对称技术 (ECDH) 导出密钥



椭圆曲线 Diffie-Hellman (ECDH) 使用 ECC 非对称加密技术在双方之间建立共享密钥，无需预先交换密钥。ECDH 密钥是临时的，因此 AWS 支付密码学不存储它们。在此过程中，使用ECDH 进行一次性KBPK/KEK计算。该派生密钥将立即用于包装您要传输的密钥，该密钥可以是另一个 KBPK、BDK、IPEK 密钥或其他密钥类型。

导出时， AWS 支付加密被称为 U 方（发起方），接收系统称为第 V 方（响应方）。

> **Note**
>
> ECDH 可用于交换任何对称密钥类型，但如果尚未建立 KEK，则它是唯一可用于传输 AES-256 密钥的方法。

1. 生成 ECC 密钥对

致电create-key为该过程创建 ECC 密钥对。此 API 为密钥导入或导出生成密钥对。创建时，指定使用此 ECC 密钥可以派生哪种密钥。使用 ECDH 交换（包装）其他密钥时，使用值。TR31_K1_KEY_BLOCK_PROTECTION_KEY

> **Note**
>
> 尽管低级 ECDH 会生成可用于任何目的的派生密钥，但 AWS 支付密码学允许密钥仅用于单一派生密钥类型，从而限制了密钥意外重复用于多种用途。

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=ECC_NIST_P256,KeyUsage=TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT,KeyClass=ASYM
 --derive-key-usage "TR31_K1_KEY_BLOCK_PROTECTION_KEY"
```

```
{
        "Key": {
            "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
wc3rjsssguhxtilv",
            "KeyAttributes": {
                "KeyUsage": "TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT",
                "KeyClass": "ASYMMETRIC_KEY_PAIR",
                "KeyAlgorithm": "ECC_NIST_P256",
                "KeyModesOfUse": {
                    "Encrypt": false,
                    "Decrypt": false,
                    "Wrap": false,
                    "Unwrap": false,
                    "Generate": false,
                    "Sign": false,
                    "Verify": false,
                    "DeriveKey": true,
                    "NoRestrictions": false
                }
            },
            "KeyCheckValue": "2432827F",
            "KeyCheckValueAlgorithm": "CMAC",
            "Enabled": true,
            "Exportable": true,
            "KeyState": "CREATE_COMPLETE",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
```

```
            "CreateTimestamp": "2025-03-28T22:03:41.087000-07:00",
            "UsageStartTimestamp": "2025-03-28T22:03:41.068000-07:00"
        }
    }
```

2. 获取公钥证书

   get-public-key-certificate致电接收公钥作为 X.509证书，该证书由您的账户 CA 签署，该证书专用于特定地区的 AWS 支付加密。

   Example

```
$ aws payment-cryptography get-public-key-certificate \
          --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/wc3rjsssguhxtilv
```

```
{
      "KeyCertificate": "LS0tLS1CRUdJTi...",
      "KeyCertificateChain": "LS0tLS1CRUdJT..."
    }
```

3. 在交易对手系统上安装公共证书（第五方）

   对于许多 HSM，您需要安装、加载或信任步骤 1 中生成的公共证书才能建立密钥。这可能包括整个证书链，也可以仅包括根证书，具体取决于 HSM。有关具体说明，请查阅您的 HSM 文档。

4. 在源系统上生成 ECC 密钥对并提供证书链给 AWS Payment Cryptography

   在 ECDH 中，各方生成一对密钥并商定一个通用密钥。为了使 AWS 支付密码学得出密钥，它需要交易对手的公钥格式的公钥。 X.509

   从 HSM 传输密钥时，在该 HSM 上创建密钥对。对于支持密钥块的 HSM，密钥标头将类似于D0144K3EX00E0000。创建证书时，通常在 HSM 上生成 CSR，然后 HSM、第三方或服务等 AWS 私有 CA 可以生成证书。

   使用带有 o KeyMaterialType f RootCertificatePublicKey 和 of 的importKey命令将根证书加载到 AWS 支付加密中
   KeyUsageType 。TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE

对于中间证书，使用带有 o KeyMaterialType f `TrustedCertificatePublicKey` 和 of KeyUsageType 的`importKey`命令`TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE`。对多个中间证书重复此过程。使用链`KeyArn`中最后导入的证书作为后续导出命令的输入。

> **Note**
>
> 不要导入叶子证书。在导出命令期间直接提供。

5. 从中派生密钥和导出密钥 AWS Payment Cryptography

导出时，该服务使用 ECDH 派生密钥，然后立即将其用作 KBPK 来包装要导出的密钥。 TR-31 要导出的密钥可以是任何 TDES 或 AES 密钥，只要包装密钥的强度至少与要导出的密钥一样强大。 TR-31

```
$ aws payment-cryptography export-key \
            --export-key-identifier arn:aws:payment-cryptography:us-
west-2:529027455495:key/e3a65davqhbpjm4h \
            --key-material='{
              "DiffieHellmanTr31KeyBlock": {
                "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-
cryptography:us-east-2:111122223333:key/swseahwtq2oj6zi5",
                "DerivationData": {
                  "SharedInformation": "ADEF567890"
                },
                "DeriveKeyAlgorithm": "AES_256",
                "KeyDerivationFunction": "NIST_SP800",
                "KeyDerivationHashAlgorithm": "SHA_256",
                "PrivateKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/wc3rjsssguhxtilv",
                "PublicKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FUR..."
              }
            }'
```

```
{
            "WrappedKey": {
                "WrappedKeyMaterialFormat": "TR31_KEY_BLOCK",
                "KeyMaterial":
 "D0112K1TB00E00007012724C0FAAF64DA50E2FF4F9A94DF50441143294E0E995DB2171554223EAA56D078C4CF
                "KeyCheckValue": "E421AD",
                "KeyCheckValueAlgorithm": "ANSI_X9_24"
```

```
            }
        }
```

6. 在 Party V HSM 上使用 ECDH 派生一次性密钥

   许多 HSM 和相关系统支持使用 ECDH 建立密钥。将步骤 1 中的公钥指定为公钥，将步骤 3 中的密钥指定为私钥。有关允许的选项，例如派生方法，请参阅 API 指南。

   > **Note**
   >
   > 诸如哈希类型之类的派生参数必须在两端完全匹配。否则，您将生成不同的密钥。

7. 将密钥导入目标系统

   最后，使用标准 TR-31 命令从 AWS 支付密码学中导入密钥。将 ECDH 派生密钥指定为 KBPK，并使用之前从支付加密中 AWS 导出的 TR-31 密钥块。

使用非对称技术导出密钥 (RSA Wrap)

当 TR-34 不可用时，您可以使用 RSA wrap/unwrap 进行密钥交换。例如 TR-34，此方法使用 RSA 非对称加密来加密对称密钥。但是，RSA 封装不包括：

- 由发送方签名有效载荷
- 在传输期间保持密钥元数据完整性的密钥块

> **Note**
>
> 您可以使用 RSA 封装来导出 TDES 和密钥。 AES-128

1. 在接收系统上创建 RSA 密钥和证书

   创建或识别用于接收封装密钥的 RSA 密钥。我们要求密钥采用 X.509 证书格式。确保证书由根证书签名，您可以将其导入 AWS 支付加密中。

2. 将根公共证书导入到 AWS Payment Cryptography

   import-key与导入证书的`--key-material`选项一起使用

```
$ aws payment-cryptography import-key \
```

```
    --key-material='{"RootCertificatePublicKey": { \
    "KeyAttributes": { \
    "KeyAlgorithm": "RSA_4096", \
    "KeyClass": "PUBLIC_KEY", \
    "KeyModesOfUse": {"Verify": true}, \
    "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"}, \
    "PublicKeyCertificate": "LS0tLS1CRUdJTiBDRV..."} \
    }'
```

```
{
  "Key": {
    "CreateTimestamp": "2023-09-14T10:50:32.365000-07:00",
    "Enabled": true,
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
nsq2i3mbg6sn775f",
    "KeyAttributes": {
      "KeyAlgorithm": "RSA_4096",
      "KeyClass": "PUBLIC_KEY",
      "KeyModesOfUse": {
        "Decrypt": false,
        "DeriveKey": false,
        "Encrypt": false,
        "Generate": false,
        "NoRestrictions": false,
        "Sign": false,
        "Unwrap": false,
        "Verify": true,
        "Wrap": false
      },
      "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"
    },
    "KeyOrigin": "EXTERNAL",
    "KeyState": "CREATE_COMPLETE",
    "UsageStartTimestamp": "2023-09-14T10:50:32.365000-07:00"
  }
}
```

3. 导出你的密钥

   告诉 P AWS ayment Cryptography 使用您的分支证书导出您的密钥。你需要指定：

   - 您在步骤 2 中导入的根证书的 ARN
   - 出口叶子证书

- 要导出的对称密钥

输出是对称密钥的十六进制编码二进制包装（加密）版本。

Example示例-导出密钥

```
$ cat export-key.json
```

```
{
  "ExportKeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
  "KeyMaterial": {
    "KeyCryptogram": {
      "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/zabouwe3574jysdl",
      "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDEXAMPLE...",
      "WrappingSpec": "RSA_OAEP_SHA_256"
    }
  }
}
```

```
$ aws payment-cryptography export-key \
    --cli-input-json file://export-key.json
```

```
{
  "WrappedKey": {
    "KeyMaterial":
 "18874746731E9E1C4562E4116D1C2477063FCB08454D757D81854AEAEE0A52B1F9D303FA29C02DC82AE778535
    "WrappedKeyMaterialFormat": "KEY_CRYPTOGRAM"
  }
}
```

4. 将密钥导入您的接收系统

许多 HSM 和相关系统支持使用 RSA unwrap（包括 AWS 支付加密）导入密钥。导入时，请指定：

- 步骤 1 中的公钥作为加密证书
- 格式为 RSA

- 填充模式为 PKCS #1 v2.2 OAEP（带有 SHA 256）

> **Note**
>
> 我们以 HexBinary 格式输出封装的密钥。如果您的系统需要不同的二进制表示形式，例如 base64，则可能需要转换格式。

使用预先建立的密钥交换密钥导出对称密钥 () TR-31

交换多个密钥或支持密钥轮换时，合作伙伴通常首先交换初始密钥加密密钥 (KEK)。您可以使用TR-34或物理密钥交换等技术将 KEK 与 AWS 支付加密技术交换。建立 KEK 后，您可以使用它来传输后续密钥，包括其他 KEK。我们使用受到 HSM 供应商广泛支持的 ANSI TR-31 来支持这种密钥交换。

1. 设置您的密钥加密密钥 (KEK)

   确保你已经交换了 KEK 并有 KeyArn（或 KeyAlias）可用。

2. 开启您的密钥 AWS Payment Cryptography

   如果密钥尚不存在，请创建密钥。或者，你可以在其他系统上创建密钥并使用导入命令。

3. 从中导出您的密钥 AWS Payment Cryptography

   以 TR-31 格式导出时，请指定要导出的密钥和要使用的打包密钥。

Example示例 — 使用 TR31 密钥块导出密钥

```
$ aws payment-cryptography export-key \
    --key-material='{"Tr31KeyBlock": \
    { "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza" }}' \
    --export-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp
```

```
{
  "WrappedKey": {
    "KeyCheckValue": "73C263",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "KeyMaterial":
 "D0144K0AB00E0000A24D3ACF3005F30A6E31D533E07F2E1B17A2A003B338B1E79E5B3AD4FBF7850FACF9A3784
    "WrappedKeyMaterialFormat": "TR31_KEY_BLOCK"
  }
}
```

4. 将密钥导入您的系统

   使用系统的导入密钥实现来导入密钥。

导出 DUKPT 初始密钥 () IPEK/IK

使用 DUKPT 时，您可以为一组终端生成单个基本派生密钥 (BDK)。终端无法直接访问 BDK。相反，每个终端都会收到一个唯一的初始终端密钥，称为 IPEK 或初始密钥 (IK)。每个 IPEK 都是使用唯一的密钥序列号 (KSN) 从 BDK 派生而来的。

KSN 结构因加密类型而异：

- 对于 TDES：10 字节的 KSN 包括：
  - 密钥集 ID 为 24 位
  - 终端 ID 为 19 位
  - 21 位用于交易计数器
- 对于 AES：12 字节 KSN 包括：
  - BDK ID 为 32 位
  - 派生标识符 (ID) 为 32 位

- 32 位用于交易计数器

我们提供了一种生成和导出这些初始密钥的机制。您可以使用 TR-31、 TR-34或 RSA 封装方法导出生成的密钥。请注意，IPEK 密钥不能永久保存，也不能用于支付加密的后续 AWS 操作。

我们不强制在 KSN 的前两部分之间进行拆分。如果要将派生标识符与 BDK 一起存储，则可以使用 AWS 标签。

> **Note**
>
> KSN 的计数器部分（AES DUKPT 为 32 位）不用于推导。 IPEK/IK例
> 如，12345678901234560001 和 12345678901234569999 的输入将生成相同的 IPEK。

```
$ aws payment-cryptography export-key \
    --key-material='{"Tr31KeyBlock": { \
    "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza"}} ' \
    --export-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --export-attributes 'ExportDukptInitialKey={KeySerialNumber=12345678901234560001}'
```

```
{
"WrappedKey": {
    "KeyCheckValue": "73C263",
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "KeyMaterial":
 "B0096B1TX00S000038A8A06588B9011F0D5EEF1CCAECFA6962647A89195B7A98BDA65DDE7C57FEA507559AF2A5D60
    "WrappedKeyMaterialFormat": "TR31_KEY_BLOCK"
}
}
```

指定要导出的键块标题

以 ASC 或格式导出时，您可以修改 TR-31 或 TR-34添加密钥块信息。下表描述了 TR-31 键块格式以及在导出期间可以修改的元素。

密钥方块属性	用途	你能在导出期间修改吗？	注意
版本 ID	定义用于保护密钥材料的方法。该标准包括： • 版本 A 和 C（密钥变体-已弃用） • 版本 B（使用 TDES 推导） • 版本 D（使用 AES 派生密钥）	否	我们将版本 B 用于 TDES 封装密钥，使用版本 D 进行 AES 封装密钥。我们仅支持版本 A 和 C 用于导入操作。
密钥块长度	指定剩余消息的长度	否	我们会自动计算这个值。在解密有效载荷之前，长度可能看起来不正确，因为我们可以根据规范的要求添加密钥填充。
密钥用法	定义密钥的允许用途，例如： • C0（信用卡验证） • B0（基本派生密钥）	否	
算法	指定底层密钥的算法。我们支持： • T (TDES) • H (HMAC) • （年龄）	否	我们按原样出口这个值。
密钥用法	定义允许的操作，例如：	是*	

密钥方块属性	用途	你能在导出期间修改吗？	注意
	• 生成并验证 (C) • Encrypt/Decrypt/Wrap/Unwrap (B)		
密钥版本	表示密钥的版本号 replacement/rotation。如果未指定，则默认为 00。	是-可以追加	
密钥可导出性	控制是否可以导出密钥： • N-没有可导出性 • E-根据 X9.24 （键块）导出 • S-以密钥块或非密钥块格式导出	是*	
可选密钥块	是-可以追加	可选密钥块是以加密方式绑定到密钥 name/value 的成对密钥。例如，DUKPT 密钥的 KeySet ID。我们会根据您的 name/value 配对输入自动计算区块数量、每个区块的长度和填充区块 (PB)。	

*修改值时，您的新值必须比 AWS 支付加密中的当前值更严格。例如：

- 如果当前的密钥使用模式为 Generate=True、Verify=True，则可以将其更改为 Generate=True、Verify=False
- 如果密钥已设置为不可导出，则无法将其更改为可导出

当您导出密钥时，我们会自动应用正在导出的密钥的当前值。但是，在发送到接收系统之前，您可能需要修改或附加这些值。以下是一些常见场景：

- 将密钥导出到支付终端时，将其可导出性设置为，`Not Exportable`因为终端通常只导入密钥，不应将其导出。
- 当您需要将关联的密钥元数据传递给接收系统时，使用 TR-31可选标头以加密方式将元数据绑定到密钥，而不是创建自定义负载。
- 使用该`KeyVersion`字段设置密钥版本以跟踪密钥轮换。

TR-31/X9.143 定义了常用标头，但您可以使用其他标头，只要它们满足 AWS 支付加密参数并且您的接收系统可以接受它们。有关导出期间的密钥区块标头的更多信息，请参阅 API 指南中的密钥区块标头。

以下是导出具有以下规格的 BDK 密钥（例如，到 KIF）的示例：

- 密钥版本：02
- KeyExportability: 不可导出
- KeySetID：00ABCDEFAB（00 表示 TDES 密钥，ABCDEFABCD 是初始密钥）

由于我们没有指定密钥的使用模式，因此该密钥继承了 arn: aws: payment-cryptography: us-east-2:111122223333: (= true) 的使用模式。key/5rplquuwozodpwsp DeriveKey

> **Note**
>
> 即使在本示例中将可导出性设置为不可导出，KI F 仍然可以：
>
> - 派生密钥，例如在 DUKPT 中IPEK/IK使用的密钥
> - 导出这些派生密钥以安装在设备上
>
> 这是标准特别允许的。

```
$ aws payment-cryptography export-key \
    --key-material='{"Tr31KeyBlock": { \
    "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza", \
```

```
    "KeyBlockHeaders": { \
    "KeyModesOfUse": { \
    "Derive": true}, \
    "KeyExportability": "NON_EXPORTABLE", \
    "KeyVersion": "02", \
    "OptionalBlocks": { \
    "BI": "00ABCDEFABCD"}}} \
    }' \
    --export-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp
```

```
{
"WrappedKey": {
    "WrappedKeyMaterialFormat": "TR31_KEY_BLOCK",
    "KeyMaterial": "EXAMPLE_KEY_MATERIAL_TR31",
    "KeyCheckValue": "A4C9B3",
    "KeyCheckValueAlgorithm": "ANSI_X9_24"
    }
}
```

常用标题

X9.143 为常见用例定义某些标头。除了 HM（HMAC 哈希）标头外， AWS 支付加密不解析或使用这些标头。

标头名称	用途	典型验证	注意
BI	DUKPT 的基本派生密钥标识符	2 个十六进制字符（TDES 为 00，AES 为 11），然后 TDES KSI 为 10 个十六进制字符，BDK ID (AES DUKPT) 为 8 个十六进制字符。	包含（BDK ID，对于 AES DUKPT）或密钥集标识符（KSI，对于 TDES DUKPT）。可以在交换 BDK ID 或 KSI 时使用，但不需要交换 IK 和 KS 方块中包含的其他数据。通常，在传输到 KIF 时使用 BI，而在向终端本身注入时使用 IK 或 KS。

标头名称	用途	典型验证	注意
HM	指定 HMAC 操作的哈希类型	• 10 — SHA-1 • 20 — SHA-224 • 21 — SHA-256 • 22 — SHA-384 • 23 — SHA-512 • 24 — SHA-512/224 • 25 — SHA-512/256 • 30 — SHA3-224 • 31 — SHA3-256 • 32 — SHA3-384 • 33 — SHA3-512 • 40 — SHAKE128 • 41 — SHAKE256	该服务在导出时自动填充此字段，并将在导入时对其进行解析。可以导入服务不支持的哈希类型，例如 SHAKE128，但可能无法用于加密函数。
IK	AES DUKPT 的初始密钥序列号	16 十六进制字符	该值用于实例化接收设备上初始 DUKPT 密钥的使用，它标识源自 BDK 的初始密钥。该字段通常包含推导数据，但不包含计数器。使用 KS 进行 TDES DUKPT。
KS	TDES DUKPT 的初始密钥序列号	20 十六进制字符	该值用于实例化接收设备上初始 DUKPT 密钥的使用，它标识源自 BDK 的初始密钥。该字段通常包含推导数据 + 归零计数器值。使用 IK 进行 AES DUKPT。

标头名称	用途	典型验证	注意
KP	KBPK 的密钥校验值	2 个十六进制字符代表 KCV 方法（0 表示 X9.24 方法，01 表示 CMAC 方法）。其次是 KCV 值，通常为 6 个十六进制字符。例如，010FA329 表示使用 01 (CMAC) 方法计算的 0FA329 的 KCV。	根据 ANSI X9.24-1-2017 附录 A 计算；KCV 用于密钥验证，而不是作为完整性机制。
PB	填充块	随机可打印 ASCII 字符	该服务在导出时自动填充此字段，以确保可选标头是加密块长度的倍数

导出非对称 (RSA) 密钥

要以证书形式导出公钥，请使用get-public-key-certificate命令。此命令返回：

- 该证书
- 根证书

两个证书均采用 base64 编码。

> **Note**
>
> 此操作不是阶级——即使使用相同的底层密钥，后续调用也可能会生成不同的证书。

Example

```
$ aws payment-cryptography get-public-key-certificate \
     --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/5dza7xqd6soanjtb
```

```
{
"KeyCertificate": "LS0tLS1CRUdJTi...",
"KeyCertificateChain": "LS0tLS1CRUdJT..."
}
```

高级主题

本节介绍高级密钥交换场景和配置。

主题

- 自带证书颁发机构 (BYOCA)
- 物理密钥交换

自带证书颁发机构 (BYOCA)

默认情况下，当服务内创建的非对称（RSA、ECC）密钥需要公钥证书时，这些证书由 AWS 支付加密和账户唯一证书颁发机构 (CA) 颁发。这旨在使其易于使用， X.509 无需识别或设置 CA 或管理证书签名请求 (CSR)。

AWS 支付密码学还允许您在出于政策或合规性原因需要时使用自己的 CA。

主题

- 概述
- BYOCA 导出工作流程
- BYOCA 导入工作流程
- 附加说明

概述

使用 BYOCA，您可以使用自己的 CA 进行 TR-34 import/export、RSA Unwrap 和 ECDH 密钥传输。当您需要在整个组织中建立一致的证书链或合作伙伴需要特定的 CA 证书时，请使用 BYOCA。以下示例显示了 TR-34 密钥导出和 TR-34 密钥导入的 BYOCA 工作流程。

与标准 TR-34 流程有三个主要区别：

1. 您可以使用创建 RSA 密钥。 CreateKey 之前，`GetParametersForExport`或者为你`GetParametersForImport`创建了它。
2. 该 GetCertificateSigningRequest API 创建 CSR。然后，您的外部 CA 可以对其进行签名。
3. ExportKey和 ImportKey API 在通话时接受证书。该令牌现在是可选的。

> ⚠ 重要注意事项
>
> - 这些示例使用 RSA-2048 密钥并封装密 TDES-2KEY 钥。导出时 AES-128，请确保所有密钥均为 RSA-3072 或 RSA-4096。
> - 最常见的错误是`SigningKeyIdentifier`和表示的密钥`SigningKeyCertificate`不匹配。

BYOCA 导出工作流程

以下步骤演示了完整的 BYOCA TR-34 导出工作流程。

Steps

- 步骤 1：创建 RSA 密钥
- 步骤 2：生成证书签名请求
- 第 3 步：查看 CSR（可选）
- 第 4 步：使用证书颁发机构签署 CSR
- 第 5 步：导入 CA 证书
- 第 6 步：获取 KRD 加密证书
- 第 7 步：使用 BYOCA 导出密钥

步骤 1：创建 RSA 密钥

首先，创建一个最终成为 KDH 签名证书的 RSA 密钥对。您可以添加标签来识别密钥的用途。

Example创建用于签名的 RSA 密钥

```
$ aws payment-cryptography create-key --exportable \
    --key-attributes
 KeyAlgorithm=RSA_2048,KeyUsage=TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE,KeyClass=ASYMMETRI
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
xgmq6fs6uow736uc",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "ASYMMETRIC_KEY_PAIR",
            "KeyAlgorithm": "RSA_2048",
            "KeyModesOfUse": {
                "Sign": true
            }
        },
        "KeyCheckValue": "41E3723C",
        "KeyCheckValueAlgorithm": "SHA_1",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY"
    }
}
```

请记下，KeyArn因为在下一步中将需要它。

步骤 2：生成证书签名请求

生成证书签名请求 (CSR)，由您的外部 CA 使用 GetCertificateSigningRequest API 签署。输出是一个 base64 编码的 PEM 文件。如果您使用 base64 对内容进行解码并保存它们，则将获得 PEM 格式的有效 CSR。

Example生成 CSR

```
$ aws payment-cryptography get-certificate-signing-request \
```

```
    --key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
xgmq6fs6uow736uc \
    --signing-algorithm SHA512 \
    --certificate-subject '{
        "CommonName": "MyCertificateAWSUSEAST",
        "Organization": "Amazon",
        "OrganizationUnit": "PaymentCryptography",
        "Country": "US",
        "StateOrProvince": "Virginia",
        "City": "Arlington"
    }'
```

```
{
    "CertificateSigningRequest": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..."
}
```

该CertificateSigningRequest字段包含整个 PEM 文件，采用 base64 编码。PEM 文件包含-----BEGIN CERTIFICATE REQUEST-----和标-----END CERTIFICATE REQUEST-----记。将此值发送到您的 CA 进行签名。

第 3 步：查看 CSR（可选）

您可以选择使用 OpenSSL 来查看 CSR 内容并确保其有效且符合预期。

Example使用 OpenSSL 审查 CSR

```
$ echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d | openssl req -
text
```

第 4 步：使用证书颁发机构签署 CSR

生成 CSR 后，您需要让证书颁发机构 (CA) 对其进行签名。在生产环境中，您通常会使用 AWS 私有 CA 组织已建立的 CA 基础架构。出于测试目的，您可以使用 OpenSSL 创建自签名证书。

使用 AWS 私有 CA

要使用签署 CSR AWS 私有 CA，请先解码 base64 编码的 CSR 并将其保存到文件中，然后使用 API。 IssueCertificate

Example使用以下方式签署 CSR AWS 私有 CA

```
$ echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d > csr.pem
```

```
$ aws acm-pca issue-certificate \
    --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012 \
    --csr fileb://csr.pem \
    --signing-algorithm SHA256WITHRSA \
    --validity Value=365,Type=DAYS
```

```
{
    "CertificateArn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012/certificate/abcdef1234567890"
}
```

然后检索已签名的证书：

Example检索已签名的证书

```
$ aws acm-pca get-certificate \
    --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012 \
    --certificate-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012/certificate/abcdef1234567890
```

```
{
    "Certificate": "-----BEGIN CERTIFICATE-----\nMIID...\n-----END CERTIFICATE-----",
    "CertificateChain": "-----BEGIN CERTIFICATE-----\nMIID...\n-----END
 CERTIFICATE-----"
}
```

保存证书内容以供导出步骤使用。在将其提供给 API 时，您需要对其进行 base64 编码。`ExportKey`

使用 OpenSSL 进行测试

出于测试目的，您可以使用 OpenSSL 创建自签名的 CA 并对 CSR 进行签名。首先，创建 CA 私钥和自签名证书：

Example使用 OpenSSL 创建测试 CA

```
$ # Generate CA private key
openssl genrsa -out ca-key.pem 4096

$ # Create self-signed CA certificate
```

```
openssl req -new -x509 -days 3650 -key ca-key.pem -out ca-cert.pem \
    -subj "/C=US/ST=Virginia/L=Arlington/O=TestOrg/CN=Test CA"
```

然后对上一步中的 CSR 进行解码，并使用您的测试 CA 对其进行签名：

Example使用 OpenSSL 签署 CSR

```
$ # Decode the base64-encoded CSR
echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d > csr.pem

$ # Sign the CSR with the CA
openssl x509 -req -in csr.pem -CA ca-cert.pem -CAkey ca-key.pem \
    -CAcreateserial -out signed-cert.pem -days 365 -sha512
```

```
Certificate request self-signature ok
subject=C=US, ST=Virginia, L=Arlington, O=Amazon, OU=PaymentCryptography,
 CN=MyCertificateAWSUSEAST
```

已签名的证书现已生效`signed-cert.pem`。将此证书提供给 API 时，您需要对该证书进行 base64 编码：`ExportKey`

Example Base64 对签名证书进行编码

```
$ cat signed-cert.pem | base64 -w 0
```

第 5 步：导入 CA 证书

要防止使用任意证书，必须首先信任正在使用的任何 CA。使用 ImportKey API 导入外部 CA 的根证书。如果使用中间 CA，请`import-key`再次调用，但`TrustedPublicKey`改为指定`RootCertificatePublicKey`并指定根 CA ARN。

Example导入根 CA 证书

```
$ aws payment-cryptography import-key --key-material='{
    "RootCertificatePublicKey": {
        "KeyAttributes": {
            "KeyAlgorithm": "RSA_4096",
            "KeyClass": "PUBLIC_KEY",
            "KeyModesOfUse": {
                "Verify": true
            },
```

```
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"
        },
        "PublicKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t..."
    }
}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
xivpaqy7qbbm7cdw",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096",
            "KeyModesOfUse": {
                "Verify": true
            }
        },
        "Enabled": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "EXTERNAL"
    }
}
```

记下导出步骤中使用的 KeyArn CA。

第 6 步：获取 KRD 加密证书

在此示例中，我们将导回 AWS 支付加密系统，因此我们调用该服务以使用 API 接收 KRD 公钥证书。GetParametersForImport 在实际场景中，这将由其他系统提供，例如HSM、ATM、支付终端或支付终端管理系统。

Example获取导入的参数

```
$ aws payment-cryptography get-parameters-for-import \
    --key-material-type "TR34_KEY_BLOCK" \
    --wrapping-key-algorithm RSA_2048
```

```
{
    "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
    "WrappingKeyCertificateChain": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
    "WrappingKeyAlgorithm": "RSA_2048",
```

```
    "ImportToken": "import-token-v2rxpl6drxeptn7w",
    "ParametersValidUntilTimestamp": "2025-11-01T18:45:31.271000-07:00"
}
```

第 7 步：使用 BYOCA 导出密钥

最后，使用 ExportKey API 将密钥 TR-34 与您自己的 CA-signed 证书一起导出。提供由您的外部 CA 签名的签名证书。

Example TR-34 使用 BYOCA 导出

```
$ aws payment-cryptography export-key \
    --export-key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
iox73p5f4c4yjiod \
    --key-material '{
        "Tr34KeyBlock": {
            "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-
cryptography:us-east-1:111122223333:key/j625deyfqlwctu57",
            "SigningKeyIdentifier": "arn:aws:payment-cryptography:us-
east-1:111122223333:key/xgmq6fs6uow736uc",
            "SigningKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
            "KeyBlockFormat": "X9_TR34_2012",
            "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t..."
        }
    }'
```

```
{
    "WrappedKey": {
        "WrappedKeyMaterialFormat": "TR34_KEY_BLOCK",
        "KeyMaterial": "3082055A06092A864886F70D010702A082054B30820547...",
        "KeyCheckValue": "3DCA31",
        "KeyCheckValueAlgorithm": "ANSI_X9_24"
    }
}
```

现在，接收系统可以使用标准导入流程 TR-34 导入导出的密钥块。

BYOCA 导入工作流程

以下步骤将引导您完成 BYOCA 密 TR-34 钥导入工作流程。使用此工作流程使用您自己的 CA-signed KRD（密钥接收设备）加密证书接收 AWS 付款加密密钥。通过 BYOCA 导入，您可以使用已经管理的 CA 来控制信任链。

Steps

- 步骤 1：创建 RSA 密钥
- 步骤 2：生成证书签名请求
- 第 3 步：查看 CSR（可选）
- 第 4 步：使用证书颁发机构签署 CSR
- 第 5 步：导入 CA 证书
- 步骤 6：从发送系统获取 KDH 签名证书
- 第 7 步：使用 BYOCA 导入密钥

步骤 1：创建 RSA 密钥

创建 RSA 密钥对用作 KRD 加密密钥。此密钥会在导入期间解封和解密传入的密钥。 TR-34 密钥不可导出，因为它作为接收密钥留在服务中。

Example创建 KRD 加密密钥

```
$ aws payment-cryptography create-key \
    --key-attributes
 KeyAlgorithm=RSA_2048,KeyUsage=TR31_K2_TR34_ASYMMETRIC_KEY,KeyClass=ASYMMETRIC_KEY_PAIR,KeyMod
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
bm7t4qv8hzk24jce",
        "KeyAttributes": {
            "KeyUsage": "TR31_K2_TR34_ASYMMETRIC_KEY",
            "KeyClass": "ASYMMETRIC_KEY_PAIR",
            "KeyAlgorithm": "RSA_2048",
            "KeyModesOfUse": {
                "Unwrap": true,
                "Decrypt": true
            }
        },
        "KeyCheckValue": "7FA29C1E",
        "KeyCheckValueAlgorithm": "SHA_1",
        "Enabled": true,
        "Exportable": false,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY"
```

```
    }
}
```

记下 KeyArn 的值。您在下一个步骤中需要用到它。

步骤 2：生成证书签名请求

使用 GetCertificateSigningRequest API 为 KRD 加密密钥生成证书签名请求 (CSR)。输出是一个 base64 编码的 PEM 文件。如果您使用 base64 对内容进行解码并保存它们，则将获得 PEM 格式的有效 CSR。

Example为 KRD 密钥生成 CSR

```
$ aws payment-cryptography get-certificate-signing-request \
    --key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
bm7t4qv8hzk24jce \
    --signing-algorithm SHA512 \
    --certificate-subject '{
        "CommonName": "MyImportKRDCertAWSUSEAST",
        "Organization": "Amazon",
        "OrganizationUnit": "PaymentCryptography",
        "Country": "US",
        "StateOrProvince": "Virginia",
        "City": "Arlington"
    }'
```

```
{
    "CertificateSigningRequest": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..."
}
```

该CertificateSigningRequest字段包含整个 PEM 文件，采用 base64 编码。PEM 文件包含-----BEGIN CERTIFICATE REQUEST-----和标-----END CERTIFICATE REQUEST-----记。将此值发送到您的 CA 进行签名。

第 3 步：查看 CSR（可选）

您可以选择使用 OpenSSL 来查看 CSR 内容并确保其有效且符合预期。

Example使用 OpenSSL 审查 CSR

```
$ echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d | openssl req -
text
```

第 4 步：使用证书颁发机构签署 CSR

生成 CSR 后，请由证书颁发机构 (CA) 对其进行签名。在生产环境中，您通常会使用 AWS 私有 CA 组织已建立的 CA 基础架构。出于测试目的，您可以使用 OpenSSL 创建自签名证书。

使用 AWS 私有 CA

要使用签署 CSR AWS 私有 CA，请先解码 base64 编码的 CSR 并将其保存到文件中，然后使用 API。IssueCertificate

Example使用以下方式签署 CSR AWS 私有 CA

```
$ echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d > krd-csr.pem

$ aws acm-pca issue-certificate \
    --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012 \
    --csr fileb://krd-csr.pem \
    --signing-algorithm SHA256WITHRSA \
    --validity Value=365,Type=DAYS
```

```
{
    "CertificateArn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012/certificate/fedcba0987654321"
}
```

然后检索已签名的证书：

Example检索已签名的证书

```
$ aws acm-pca get-certificate \
    --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012 \
    --certificate-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/12345678-1234-1234-1234-123456789012/certificate/fedcba0987654321
```

```
{
    "Certificate": "-----BEGIN CERTIFICATE-----\nMIID...\n-----END CERTIFICATE-----",
    "CertificateChain": "-----BEGIN CERTIFICATE-----\nMIID...\n-----END
 CERTIFICATE-----"
```

```
}
```

保存证书内容以供导入步骤使用。 Base64-encode 将其提供给 ImportKey API 之前的证书。

使用 OpenSSL 进行测试

出于测试目的，您可以使用 OpenSSL 创建自签名的 CA 并对 CSR 进行签名。首先，创建 CA 私钥和自签名证书：

Example使用 OpenSSL 创建测试 CA

```
$ # Generate CA private key
openssl genrsa -out ca-key.pem 4096

$ # Create self-signed CA certificate
openssl req -new -x509 -days 3650 -key ca-key.pem -out ca-cert.pem \
    -subj "/C=US/ST=Virginia/L=Arlington/O=TestOrg/CN=Test CA"
```

然后对上一步中的 CSR 进行解码，并使用您的测试 CA 对其进行签名：

Example使用 OpenSSL 签署 CSR

```
$ # Decode the base64-encoded CSR
echo "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0..." | base64 -d > krd-csr.pem

$ # Sign the CSR with the CA
openssl x509 -req -in krd-csr.pem -CA ca-cert.pem -CAkey ca-key.pem \
    -CAcreateserial -out krd-signed-cert.pem -days 365 -sha512
```

```
Certificate request self-signature ok
subject=C=US, ST=Virginia, L=Arlington, O=Amazon, OU=PaymentCryptography,
 CN=MyImportKRDCertAWSUSEAST
```

已签名的证书现已生效krd-signed-cert.pem。 Base64-encode 在向 ImportKey API 提供此证书之前：

Example Base64 对签名证书进行编码

```
$ cat krd-signed-cert.pem | base64 -w 0
```

第 5 步：导入 CA 证书

在使用 CA 之前，必须先信任它。使用 ImportKey API 导入外部 CA 的根证书。如果使用中间 CA，请import-key再次调用，但TrustedPublicKey改为指定RootCertificatePublicKey并指定根 CA ARN。

Example导入根 CA 证书

```
$ aws payment-cryptography import-key --key-material='{
    "RootCertificatePublicKey": {
        "KeyAttributes": {
            "KeyAlgorithm": "RSA_4096",
            "KeyClass": "PUBLIC_KEY",
            "KeyModesOfUse": {
                "Verify": true
            },
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"
        },
        "PublicKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t..."
    }
}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
nqr3c5v2wp6yl8az",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096",
            "KeyModesOfUse": {
                "Verify": true
            }
        },
        "Enabled": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "EXTERNAL"
    }
}
```

记下导入步骤中使用的 KeyArn CA。

步骤 6：从发送系统获取 KDH 签名证书

在 TR-34 密钥交换中，发送方（密钥分发主机或 KDH）提供其公钥证书以供签名。此证书用于在导入期间验证封装密钥块上的签名，确保密钥材料来自可信发件人。在生产中，该证书通常通过其自己的证书分发流程来自发送系统，例如 HSM、支付终端或密钥管理系统。

在本示例中，您使用 AWS 支付加密作为发送方和接收方，以简化步骤。GetParametersForExport致电获取通常由发送系统提供的 KDH 签名证书。

Example获取导出参数

```
$ aws payment-cryptography get-parameters-for-export \
    --key-material-type "TR34_KEY_BLOCK" \
    --signing-key-algorithm RSA_2048
```

```
{
    "ExportToken": "export-token-k8sfhe2w9rjy4bx6",
    "SigningKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
    "SigningKeyCertificateChain": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
    "SigningKeyAlgorithm": "RSA_2048",
    "ParametersValidUntilTimestamp": "2025-11-01T18:45:31.271000-07:00"
}
```

记下ExportToken和SigningKeyCertificate以便在导入步骤中使用。

第 7 步：使用 BYOCA 导入密钥

最后，使用 ImportKey API 将密钥 TR-34 与您自己的 CA-signed 证书一起导入。提供由您的外部 CA 签署的 KRD 加密证书。

使用 2-pass TR-34 时，该RandomNonce参数是必需的。该服务不会为您生成随机数，它是您生成的随机十六进制值，发送端和接收端必须相同。当使用 1-pass TR-34 （依赖于时间戳而不是随机数）时，时间戳不能是将来的时间戳。该服务不会验证任何特定的新鲜度（例如，在一定小时内），但您可以根据需要强制执行新鲜度检查。

Example TR-34 使用 BYOCA 导入

```
$ aws payment-cryptography import-key --key-material='{
    "Tr34KeyBlock": {
        "CertificateAuthorityPublicKeyIdentifier": "arn:aws:payment-cryptography:us-
east-1:111122223333:key/nqr3c5v2wp6yl8az",
```

```
        "KeyBlockFormat": "X9_TR34_2012",
        "RandomNonce": "4997FBB4587D571F",
        "SigningKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
        "WrappingKeyCertificate": "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t...",
        "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-
east-1:111122223333:key/bm7t4qv8hzk24jce",
        "WrappedKeyBlock": "D0112B0AX00E0000B82679114F470F540165EDFBF407..."
    }
}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
gwz7u3rfxd4o52sb",
        "KeyAttributes": {
            "KeyUsage": "TR31_K0_KEY_ENCRYPTION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": true,
                "Decrypt": true
            }
        },
        "KeyCheckValue": "3DCA31",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "EXTERNAL"
    }
}
```

该密钥现已导入 AWS 支付加密系统并可供使用。该`KeyOrigin`字段显示`EXTERNAL`表明密钥来自服务外部。

附加说明

- 这些示例使用 AWS CLI 显示。所有 AWS 开发工具包都提供相同的功能，包括 Java、Python、Go 和 Rust。
- 如果您使用自签名的 CA 进行测试，则可以使用 OpenSSL 创建测试 CA 并签署 CSR。在生产中，使用贵组织已建立的 CA 基础架构。

物理密钥交换

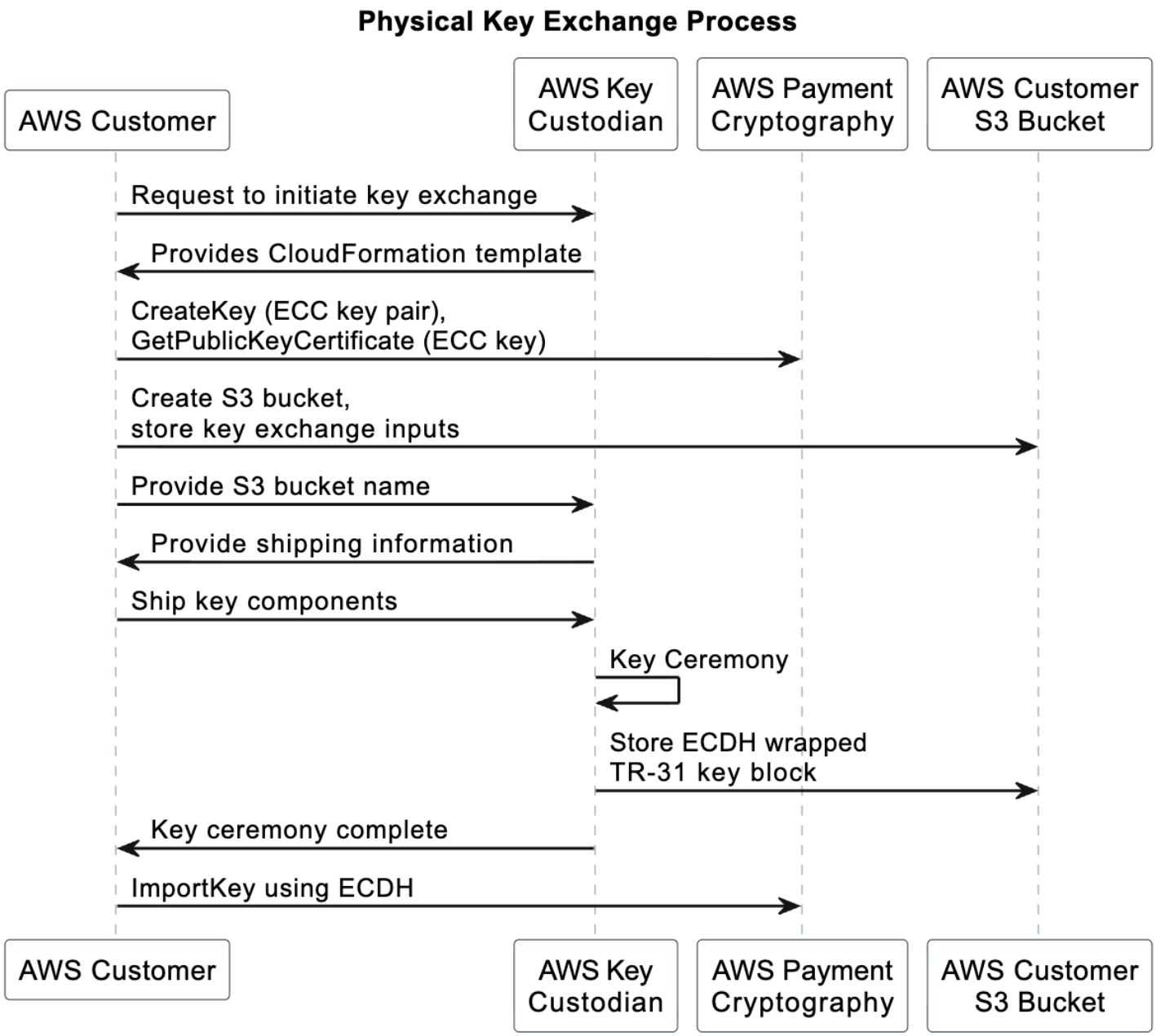
当您的合作伙伴或供应商不支持电子密钥交换时，您可以使用物理密钥交换将纸质加密密钥组件安全地转换为电子格式。训练有素的 AWS 密钥保管人在 PCI PIN 和 P2PE 认证 AWS-operated 的安全设施中举行密钥仪式，使用离线 HSM 将纸质密钥组件转换为电子格式。该服务使用 ECDH-based 密钥交换来提供 ECDH-wrapped TR-31 密钥区块，您可以将其直接导入您的 AWS 支付加密账户。

> **Note**
>
> 我们建议导入和导出密钥尽可能使用基于标准的产品。仅当您的合作伙伴或供应商不支持 ANSI X9.24 TR-34 、 RSA wrap/unwrap 或 ECDH 等电子密钥交换方法时，才使用物理密钥交换。???

物理密钥交换的工作原理

要启动纸质密钥交换，CloudFormation 模板会指导您完成先决条件设置，包括在账户中创建 ECC 密钥对和 S3 存储桶。然后，您或您的合作伙伴将纸质关键组件运送到 AWS 安全设施，训练有素的 AWS 密钥保管人使用离线 HSM 进行密钥仪式。输出是上传到您的 S3 存储桶的 ECDH-wrapped TR-31 密钥块，您可以使用该使用非对称技术 (ECDH) 导入密钥方法将其导入您的账户。物理密钥交换支持在 TDES 和 AES 密钥算法中导入 KEK（密钥用法 K1）或 BDK（密钥用法 B0）密钥。

下图显示了端到端的物理密钥交换过程。



1. 启动 — 您提交支持请求或与您的客户经理合作提交请求。
2. 客户设置 — AWS 付款密码学为您提供了一个 CloudFormation 模板来完成以下必备步骤：
   - 在您的 AWS 支付加密账户中创建 ECC P521 密钥对并检索公钥证书。
   - 使用授予 AWS 支付加密服务主体 read/write 访问权限的策略创建 Amazon S3 存储桶。
   - 将 ECC 公共证书和签名根 CA 存储在 Amazon S3 存储桶中。
   - 提供关键属性：密钥用法、密钥使用模式和要发送的纸质密钥组件的数量。

3. 共享 S3 存储桶名称 — 客户共享 CloudFormation 堆栈创建的 S3 存储桶名称，其中存储公钥证书、证书链和密钥属性，以供 AWS 支付加密启动密钥交换。
4. 运输协调 — AWS 支付密码学提供 US-based 安全设施的运输详细信息。您或您的合作伙伴将纸质关键组件运送给 AWS 关键托管人。
5. 组件收据 — AWS 密钥保管人收到每个纸质组件，并为每个组件发送单独的确认。
6. 密钥仪式 — AWS 密钥保管人使用离线 HSM 进行密钥仪式。生成的 TR-31 密钥块（使用密 ECDH-derived AES-256 钥打包）、来自离线 HSM 的 ECC 公有证书及其签名证书将上传到您的 Amazon S3 存储桶。
7. 完成 — AWS 支付密码学会确认密钥仪式已完成。然后，您可以使用该方法将 ECDH 封装的 TR-31 密钥区块导入您的 AWS 支付加密账户。使用非对称技术 (ECDH) 导入密钥
8. 计费 — 成功完成密钥仪式后，按交换的密钥向您计费。

安全与合规

物理密钥交换在旨在满足 PCI PIN 和 PCI P2PE 物理和逻辑安全要求的安全设施中 AWS 运行。以下控制措施已到位：

双重控制和职责分离

AWS 主要监护人由不同的团队指定，其报告结构各不相同。已制定相关流程，确保关键仪式步骤在双重控制下进行。

离线 HSM

关键仪式使用经过认证的 PCI PTS HSM-listed 硬件安全模块进行，这些模块无需网络连接即可离线运行。您的密钥永远不会在 HSM 边界之外以明文形式存在。

加密密钥交付

使用 ECDH-based 密钥交换将密钥材料从离线 HSM 转移到您的 AWS 支付加密账户，从而确保端到端的加密保护。

审计与合规

AWS 已制定相应的流程来满足适用的合规性要求，这些要求会定期评估 PCI PIN 和 P2PE 认证。查看 Art AWS ifact 中的合规包，查看您在自己的 PCI 评估中可以参考的报告。

使用别名

别名是 AWS 支付加密密钥的友好名称。例如，别名允许您将密钥引用为 `alias/test-key`，而不是 `arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h`。

在大多数密钥管理（控制平面）操作和加密（数据平面）操作中，您可以使用别名来识别密钥。

您还可以根据付款加密密钥的别名允许和拒绝访问 AWS 付款加密密钥，而无需编辑政策或管理授权。此功能是基于属性的访问权限控制 (ABAC) 的支持的一部分。

别名还您更容易在不同 AWS 区域中重用相同代码。在多个区域中创建同名的别名，并将每个别名与其区域中的 AWS 支付加密密钥相关联。当代码在每个区域运行时，别名是指该区域中相关的 AWS 支付加密密钥。

您可以使用 `CreateAlias` API 为 AWS 支付加密密钥创建别名。

AWS 支付密码学 API 提供对每个账户和区域中的别名的完全控制。该 API 包括创建别名 (CreateAlias)、查看别名和关联的 KeyArn（列表别名）、更改与别名关联的 AWS 支付加密密钥（更新别名）以及删除别名（删除别名）的操作。

主题

- 关于别名
- 在应用程序中使用别名
- 相关 API

关于别名

了解别名在 AWS 支付密码学中的工作原理。

别名是一种独立的 AWS 资源

> 别名不是 AWS 支付加密密钥的属性。您对别名执行的操作不会影响其关联的密钥。您可以为 AWS 支付加密密钥创建别名，然后更新别名，使其与不同的 AWS 支付加密密钥相关联。您甚至可以在不影响关联的 AWS 支付加密密钥的情况下删除别名。如果您删除 AWS Payment Cryptography 密钥，则会取消分配与该密钥关联的所有别名。
>
> 如果您在 IAM 策略中将别名指定为资源，则该策略指的是别名，而不是相关的 AWS 支付加密密钥。

每个别名都有友好名称

在创建别名时，您指定前缀为 alias/ 的别名。例如 alias/test_1234

每个别名一次都与一个 AWS 支付加密密钥相关联

别名及其 AWS 支付加密密钥必须位于同一个账户和区域中。

AWS 支付加密密钥可以同时与多个别名相关联，但每个别名只能映射到一个密钥

例如，此 list-aliases输出显示alias/sampleAlias1 别名仅与一个目标 AWS Payment Cryptography 密钥相关联，该密钥由 KeyArn 属性表示。

```
$ aws payment-cryptography list-aliases
```

```
  {
    "Aliases": [
        {
            "AliasName": "alias/sampleAlias1",
            "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
        }
    ]
}
```

多个别名可以与同一个 AWS 支付加密密钥相关联

例如，您可以将 alias/sampleAlias1; 和 alias/sampleAlias2 别名与同一个密钥关联。

```
$ aws payment-cryptography list-aliases
```

```
{
        "Aliases": [
            {
                "AliasName": "alias/sampleAlias1",
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
```

```
            },
            {
                "AliasName": "alias/sampleAlias2",
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
            }
        ]
    }
```

别名在给定的账户和区域中必须是唯一的

例如，您在每个账户和区域中只能有一个 alias/sampleAlias1 别名。别名区分大小写，但我们建议不要使用仅大小写不同的别名，因为这样很容易出错。您不能更改别名名称。但是，您可以删除别名并使用所需名称创建新别名。

您可以在不同的区域中创建具有相同名称的别名。

例如，您可以在美国东部（弗吉尼亚州北部）拥有 alias/sampleAlias2 别名，在美国西部（俄勒冈州）拥有 alias/sampleAlias2 别名。每个别名都将与其区域中的 AWS 支付加密密钥相关联。如果您的代码引用 alias/finance-key 之类的别名名称，您可以在多个区域中运行它。在每个区域中，它使用不同的 alias/sampleAlias2。有关更多信息，请参阅 在应用程序中使用别名。

您可以更改与别名关联的 AWS 支付加密密钥

您可以使用该UpdateAlias操作将别名与不同的 AWS 支付加密密钥关联起来。例如，如果alias/sampleAlias2别名与arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h AWS 支付加密密钥相关联，则可以对其进行更新，使其与该arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi密钥相关联。

> **Warning**
>
> AWS 支付密码学无法验证新旧密钥是否具有相同的属性，例如密钥用法。使用不同的密钥类型进行更新可能会导致应用程序出现问题。

> **Warning**
>
> 别名的更新取决于最终的一致性。更新别名后，别名可能会在一段短时间内解析为旧密钥或新密钥。如果这种行为会中断您的进程，请不要更新别名。

有些密钥没有别名

别名是一项可选功能，除非您选择以这种方式操作环境，否则并非所有密钥都有别名。可以使用 `create-alias` 命令将密钥与别名相关联。此外，您还可以使用 UpdateAlias 操作来更改与别名关联的 AWS Payment Cryptography 密钥，并使用 delete-alias 操作来删除别名。因此，一些 AWS 支付加密密钥可能有多个别名，有些可能没有。

将密钥映射到别名

您可以使用 `create-alias` 命令将密钥（由 ARN 表示）映射到一个或多个别名。此命令不是幂等的，要更新别名，请使用 update-alias 命令。

```
$ aws payment-cryptography create-alias --alias-name alias/sampleAlias1 \
             --key-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h
```

```
{
    "Alias": {
        "AliasName": "alias/alias/sampleAlias1",
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
    }
}
```

在应用程序中使用别名

您可以使用别名来表示应用程序代码中的 AWS 支付加密密钥。 AWS 支付加密数据操作以及列出密钥等其他操作中的`key-identifier`参数接受别名或别名 ARN。

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier alias/
BIN_123456_CVK --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue2={CardExpiryDate=0123}
```

使用别名 ARN 时，请记住，映射到 AWS 支付加密密钥的别名是在拥有 AWS 支付加密密钥的账户中定义的，可能因地区而异。

别名的最强大用途之一是在多个 AWS 区域中运行的应用程序中。

您可以在每个地区创建应用程序的不同版本，也可以使用字典、配置或交换语句为每个区域选择正确的 AWS 支付加密密钥。但在每个区域中创建具有相同别名名称的别名可能要容易得多。请记住，别名名称区分大小写。

相关 API

[标签]

标签是密钥和值对，用作组织 AWS 支付加密密钥的元数据。它们可用于灵活识别密钥，或将一个或多个密钥组合在一起。

获取密钥

AWS 支付加密密钥代表单一的加密材料单元，只能用于此服务的加密操作。 GetKeys API 将 KeyIdentifier 作为输入并返回包括属性、状态和时间戳在内的密钥元数据，但不返回实际的加密密钥材料。

Example

```
$ aws payment-cryptography get-key --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```


```
{
  "Key": {
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
      "KeyAttributes": {
          "KeyUsage": "TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY",
          "KeyClass": "SYMMETRIC_KEY",
          "KeyAlgorithm": "AES_128",
          "KeyModesOfUse": {
              "Encrypt": true,
              "Decrypt": true,
              "Wrap": true,
              "Unwrap": true,
              "Generate": false,
              "Sign": false,
              "Verify": false,
              "DeriveKey": false,
              "NoRestrictions": false
          }
      },
      "KeyCheckValue": "0A3674",
      "KeyCheckValueAlgorithm": "CMAC",
      "Enabled": true,
      "Exportable": true,
      "KeyState": "CREATE_COMPLETE",
      "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
      "CreateTimestamp": "2023-06-02T07:38:14.913000-07:00",
      "UsageStartTimestamp": "2023-06-02T07:38:14.857000-07:00"
  }
}
```

让公众 key/certificate 与密钥对相关联

Get Public Key/Certificate 返回由指示的公钥KeyArn。这可以是通过 AWS 支付加密生成的密钥对的公钥部分，也可以是之前导入的公钥部分。最常见的用例是向将加密数据的外部服务提供公钥。然后，可以将这些数据传递到利用 AWS 支付密码学的应用程序，并且可以使用支付加密中保护的私钥对数据进行解密。 AWS

该服务返回公钥作为公共证书。API 结果包含 CA 和公钥证书。两个数据元素均采用 base64 编码。

> **Note**
>
> 返回的公共证书是短暂的，而不是幂等的。即使公钥本身未更改，您也可能会在每次 API 调用中收到不同的证书。

Example

```
$ aws payment-cryptography get-public-key-certificate  --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/nsq2i3mbg6sn775f
```

```
{
  "KeyCertificate":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2VENDQXFXZ0F3SUJBZ0lSQUo1OWd2VkpDd3dlYldMNldYZEpYY
  "KeyCertificateChain":
 "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUY0VENDQThtZ0F3SUJBZ0lSQUtlN2piaHFKZjJPd3FGUWI5c3VuO
}
```

标记密钥

在 AWS 支付加密中，您可以在创建密钥时为 AWS 支付加密密钥添加标签，并标记或取消标记现有密钥，除非它们待删除。标签是可选的，但它们可能非常有用。

有关标签的一般信息，包括最佳实践、标记策略以及标签的格式和语法，请参阅中的标记 AWS 资源。Amazon Web Services 一般参考

主题

- 关于中的标签 AWS Payment Cryptography
- 在控制台中查看密钥标签
- 使用 API 操作管理密钥标签
- 控制对标签的访问
- 使用标签控制对密钥的访问

关于中的标签 AWS Payment Cryptography

标签是可选的元数据标签，您可以为 AWS 资源分配（或 AWS 可以分配）。每个标签都包含一个标签键和一个标签值，它们都是区分大小写的字符串。标签值可为空 (null) 字符串。资源上的每个标签必须具有不同的标签密钥，但您可以向多个 AWS 资源添加相同的标签。每个资源最多可以有 50 个用户创建的标签。

不要在标签键或标签值中包含机密或敏感信息。许多人都可以使用标签 AWS 服务，包括计费。

在 P AWS ayment Cryptography 中，您可以在创建密钥时为密钥添加标签，也可以标记或取消标记现有密钥，除非它们待删除。无法标记别名。标签是可选的，但它们可能非常有用。

例如，您可以为用于 Alpha 项目的所有 AWS 支付加密密钥和 Amazon S3 存储桶添加`"Project"="Alpha"`标签。另一个例子是为与特定银行识别码 (BIN) 关联的所有密钥添加 `"BIN"="20130622"` 标签。

```
  [
  {
    "Key": "Project",
    "Value": "Alpha"
  },
  {
    "Key": "BIN",
    "Value": "20130622"
  }
]
```

有关标签的一般信息，包括格式和语法，请参阅中的Amazon Web Services 一般参考标记 AWS 资源。

标签可帮助您：

- 识别和整理您的 AWS 资源。许多 AWS 服务都支持标记，因此您可以为来自不同服务的资源分配相同的标签，以表明这些资源是相关的。例如，您可以为 AWS 支付加密密钥和亚马逊弹性区块商店 (Amazon EBS) 卷或 AWS Secrets Manager 密钥分配相同的标签。您还可以使用标签来标识密钥以实现自动化。
- 追踪您的 AWS 成本。当您为 AWS 资源添加标签时， AWS 会生成成本分配报告，其中包含按标签汇总的使用量和成本。您可以使用此功能来跟踪项目、应用程序或成本中心的 AWS 支付加密成本。

  有关对成本分配使用标签的更多信息，请参阅 AWS Billing 用户指南中的使用成本分配标签。有关标签密钥和标签值规则的信息，请参阅AWS Billing 用户指南中的User-Defined标签限制。
- 控制对 AWS 资源的访问权限。根据密钥的标签允许和拒绝访问密钥是基于属性的访问控制 (ABAC) 的 AWS 支付加密支持的一部分。有关基于 AWS Payment Cryptography 的标签控制对其访问的更多信息，请参阅 授权基于 AWS 支付密码学标签。有关使用标签控制 AWS 资源访问权限的更多一般信息，请参阅 IAM 用户指南中的使用 AWS 资源标签控制对资源的访问权限。

AWS 当您使用 TagResource、 UntagResource或 ListTagsForResource 操作时，支付密码学会在您的 AWS CloudTrail 日志中写入一个条目。

在控制台中查看密钥标签

要在控制台中查看标签，您需要在包含该密钥的 IAM policy 中拥有密钥的标记权限。除了在控制台中查看密钥的权限之外，您还需要这些权限。

使用 API 操作管理密钥标签

您可以使用 AWS Payment Cryptography API 为您管理的密钥添加、删除和列出标签。这些示例使用 AWS Command Line Interface (AWS CLI)，但您可以使用任何受支持的编程语言。你无法标记 AWS 托管式密钥。

要添加、编辑、查看和删除密钥的标签，您必须具有所需的权限。有关更多信息，请参阅 控制对标签的访问。

主题

- CreateKey: 为新密钥添加标签
- TagResource: 添加或更改密钥的标签
- ListResourceTags: 获取密钥的标签
- UntagResource: 从密钥中删除标签

CreateKey: 为新密钥添加标签

您可以在创建密钥时向其添加标签。要指定标签，请使用CreateKey操作的Tags参数。

要在创建密钥时添加标签，调用方必须具有 IAM policy 中的 payment-cryptography:TagResource 权限。权限至少必须涵盖账户和区域中的所有密钥。有关更多信息，请参阅 控制对标签的访问。

CreateKey 的 Tags 参数的值是区分大小写的标签键和标签值对的集合。密钥上的每个标签都必须具有不同的标签名称。标签值可为 null 或空字符串。

例如，以下 AWS CLI 命令创建带有Project:Alpha标签的对称加密密钥。指定多个键值对时，请使用空格分隔每个对。

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY, \
       KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY, \
       KeyModesOfUse='{Generate=true,Verify=true}' \
       --tags '[{"Key":"Project","Value":"Alpha"},{"Key":"BIN","Value":"123456"}]'
```

当此命令成功时，它会返回一个 Key 对象以及有关新密钥的信息。但是，Key 不包括标签。要获取标签，请使用ListResourceTags操作。

TagResource: 添加或更改密钥的标签

该TagResource操作为密钥添加一个或多个标签。此操作不能用于添加或编辑不同 AWS 账户中的标签。

要添加标签，请指定新标签键和标签值。要编辑标签，请指定现有标签键和新标签值。密钥上的每个标签都必须具有不同的标签键。标签值可为 null 或空字符串。

例如，以下命令将 **UseCase** 和 **BIN** 标签添加到示例密钥中。

```
$ aws payment-cryptography tag-resource --resource-arn arn:aws:payment-
cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h --tags
 '[{"Key":"UseCase","Value":"Acquiring"},{"Key":"BIN","Value":"123456"}]'
```

此命令成功执行后，不会返回任何输出。要查看密钥上的标签，请使用该ListResourceTags操作。

您也可以使用 TagResource 来更改现有标签的标签值。要替换标签值，请指定具有不同值的相同标签键。修改命令中未列出的标签不会被更改或删除。

例如，此命令会将 Project 标签的值从 Alpha 更改为 Noe。

该命令将返回不 http/200 包含任何内容。要查看您的更改，请使用 ListTagsForResource

```
$ aws payment-cryptography tag-resource --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h \
        --tags '[{"Key":"Project","Value":"Noe"}]'
```

ListResourceTags: 获取密钥的标签

该ListResourceTags操作获取密钥的标签。ResourceArn（keyArn 或 keyAlias）参数是必需的。此操作不能用于查看其他 AWS 账户中的密钥上的标签。

例如，以下命令获取示例密钥的标签。

```
$ aws payment-cryptography list-tags-for-resource --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h

  {
    "Tags": [
        {
            "Key": "BIN",
            "Value": "20151120"
        },
        {
            "Key": "Project",
            "Value": "Production"
        }
    ]
}
```

UntagResource: 从密钥中删除标签

该UntagResource操作从密钥中删除标签。要标识要删除的标签，请指定标签键。此操作不能用于从其他 AWS 账户中的密钥中删除标签。

当它成功时，UntagResource 操作不返回任何输出。此外，如果在密钥上未找到指定的标签键，则不会抛出异常或返回响应。要确认操作是否有效，请使用该ListResourceTags操作。

例如，此命令将从指定的密钥中删除 **Purpose** 标签及其值。

```
$ aws payment-cryptography untag-resource \
```

```
        --resource-arn arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h --tag-keys Project
```

控制对标签的访问

要通过使用 API 来添加、查看和删除标签，主体需要在 IAM policy 中获取标记权限。

您还可以通过对标签使用 AWS 全局条件密钥来限制这些权限。在 AWS 支付密码学中，这些条件可以控制对标记操作的访问权限，例如TagResource和。 UntagResource

有关示例策略和更多信息，请参阅 IAM 用户指南中的根据标签键控制访问。

用于创建和管理标签的权限如下所示。

支付密码学：TagResource

允许主体添加或编辑标签。要在创建密钥时添加标签，主体必须在 IAM policy 中具有不限于特定密钥的权限。

支付密码学：ListTagsForResource

允许主体查看密钥上的标签。

支付密码学：UntagResource

允许主体从密钥中删除标签。

标记策略中的权限

您可以在密钥政策或 IAM policy 中提供权限标记。例如，以下示例密钥政策向选定用户授予标记密钥的权限。它为所有可以担任示例管理员或开发人员角色的用户授予查看标签的权限。

JSON

```
{
  "Version":"2012-10-17",
  "Id": "example-key-policy",
  "Statement": [
    {
      "Sid": "EnableIAMUserPermissions",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
```

```
      "Action": "payment-cryptography:*",
      "Resource": "*"
    },
    {
      "Sid": "AllowAllTaggingPermissions",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/LeadAdmin",
        "arn:aws:iam::111122223333:user/SupportLead"
      ]},
      "Action": [
        "payment-cryptography:TagResource",
        "payment-cryptography:ListTagsForResource",
        "payment-cryptography:UntagResource"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow roles to view tags",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:role/Administrator",
          "arn:aws:iam::111122223333:role/Developer"
        ]
      },
      "Action": "payment-cryptography:ListTagsForResource",
      "Resource": "*"
    }
  ]
}
```

要授予主体对多个密钥的标记权限，您可以使用 IAM policy。为使此策略生效，每个密钥的密钥政策都必须允许账户使用 IAM policy 来控制对密钥的访问。

例如，以下 IAM policy 允许主体创建密钥。它还允许他们在指定账户中的所有密钥上创建和管理标签。这种组合允许主体在创建密钥时使用CreateKey操作的标签参数为密钥添加标签。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyCreateKeys",
      "Effect": "Allow",
      "Action": "payment-cryptography:CreateKey",
      "Resource": "*"
    },
    {
      "Sid": "IAMPolicyTags",
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:TagResource",
        "payment-cryptography:UntagResource",
        "payment-cryptography:ListTagsForResource"
      ],
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*"
    }
  ]
}
```

限制标签权限

您可以通过使用策略条件限制标记权限。以下策略条件可应用于 payment-cryptography:TagResource 和 payment-cryptography:UntagResource 权限。例如，您可以使用 aws:RequestTag/*tag-key* 条件来允许主体仅添加特定标签，或阻止主体添加具有特定标签键的标签。

- aws: RequestTag
- aws:ResourceTag/标签密钥（仅限 IAM 政策）
- aws: TagKeys

作为使用标签控制对密钥的访问的最佳实践，请使用 aws:RequestTag/*tag-key* 或 aws:TagKeys 条件键来确定允许哪些标签（或标签键）。

例如，以下 IAM policy 与上一个类似。但是，此策略允许主体为具有 Project 标签键的标签创建标签 (TagResource) 并删除标签 UntagResource。

由于TagResource和UntagResource请求可以包含多个标签，因此您必须使用 a ws: TagKeys 条件指定ForAllValues或ForAnyValue设置运算符。ForAnyValue 运算符要求请求中至少有一个标

签键与策略中的其中一个标签键匹配。ForAllValues 运算符要求请求中所有的标签键与策略中的其中一个标签键匹配。true如果请求中没有标签，则ForAllValues运算符也会返回，但是如果未指定标签 TagResource ，则运算符会 UntagResource失败。有关集合运算符的详细信息，请参阅 IAM 用户指南中的使用多个键和值。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyCreateKey",
      "Effect": "Allow",
      "Action": "payment-cryptography:CreateKey",
      "Resource": "*"
    },
    {
      "Sid": "IAMPolicyViewAllTags",
      "Effect": "Allow",
      "Action": "payment-cryptography:ListTagsForResource",
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*"
    },
    {
      "Sid": "IAMPolicyManageTags",
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:TagResource",
        "payment-cryptography:UntagResource"
      ],
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*",
      "Condition": {
          "ForAllValues:StringEquals": {"aws:TagKeys": "Project"}
      }
    }
  ]
}
```

使用标签控制对密钥的访问

您可以根据密钥上的标签控制对 AWS 支付密码的访问权限。例如，您可以编写 IAM policy，以允许主体仅启用和禁用具有特定标签的密钥。或者，您可以使用 IAM policy 防止主体在加密操作中使用密钥，除非密钥具有特定标签。

此功能是基于属性的访问控制 (ABAC) 的 AWS 支付加密支持的一部分。有关使用标签控制 AWS 资源访问权限的信息，请参阅 ABAC 的用途 AWS？以及 IAM 用户指南中的使用 AWS 资源标签控制对资源的访问权限。

AWS Payment Cryptography 支持 a ws:ResourceTag/tag-key 全局条件上下文密钥，它允许您根据密钥上的标签控制对密钥的访问权限。由于多个密钥可以具有相同的标签，此功能可使您将权限应用于一组选定的密钥。您还可以通过更改密钥的标签轻松更改集合中的 KMS 密钥。

在 AWS 支付密码学中，`aws:ResourceTag/`*`tag-key`*条件密钥仅在 IAM 策略中受支持。密钥策略（仅适用于一个密钥）或不使用特定密钥的操作（例如ListKeys或ListAliases操作）不支持它。

使用标签控制访问提供了一种简单、可扩展且灵活的方式来管理权限。但是，如果设计和管理不当，它可能会无意中允许或拒绝对您的密钥的访问。如果您使用标签来控制访问，请考虑以下做法。

- 使用标签来强化最低权限访问的最佳实践。仅为 IAM 主体授予他们对必须使用或管理的密钥的所需权限。例如，使用标签来标记用于项目的密钥。然后授予项目团队仅使用带有项目标签的密钥的权限。
- 谨慎为主体提供 `payment-cryptography:TagResource` 和 `payment-cryptography:UntagResource` 权限，以允许他们添加、编辑和删除标签。当您使用标签控制对密钥的访问时，更改标签可以授予主体使用他们没有权限使用的密钥的权限。它还可以拒绝对其他主体执行其工作所需的密钥的访问。不具有更改密钥策略或创建授权权限的密钥管理员可以控制对密钥的访问，前提是他们有权管理标签。

  如有可能，请使用策略条件，例如 `aws:RequestTag/`*`tag-key`* 或 `aws:TagKeys`，以将主体的标记权限限制为特定 密钥上的特定标签或标签模式。
- 查看您的中当前 AWS 账户 具有标记和取消标记权限的委托人，并在必要时对其进行调整。IAM policy 可能允许对所有密钥的标记和取消标记权限。例如，管理员托管策略允许主体标记、取消标记和列出所有密钥上的标签。
- 在设置依赖于标签的政策之前，请查看您的密钥上的标签 AWS 账户。请确保您的策略仅适用于您要包含的标签。使用CloudTrail 日志和 CloudWatch 警报提醒您注意可能影响密钥访问权限的标签更改。

- 基于标签的策略条件使用模式匹配；它们不绑定到标签的特定实例。使用基于标签的条件键的策略会影响与模式匹配的所有新标签和现有标签。如果删除并重新创建与策略条件匹配的标签，则该条件将应用于新标签，就像对旧标签一样。

例如，请考虑以下 IAM policy。它允许主体仅对您账户中位于美国东部（弗吉尼亚州北部）地区且带有 "Project"="Alpha" 标签的密钥调用 Decrypt 操作。您可以将此策略附加到示例 Alpha 项目中的角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAMPolicyWithResourceTag",
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:DecryptData"
      ],
      "Resource": "arn:aws:payment-cryptography:us-east-1:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "Alpha"
        }
      }
    }
  ]
}
```

以下示例 IAM policy 允许主体使用账户中的密钥执行特定加密操作。但它禁止主体对具有 "Type"="Reserved" 标签或不包含 "Type" 标签的密钥使用这些加密操作。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAMAllowCryptographicOperations",
```

```
      "Effect": "Allow",
      "Action": [
        "payment-cryptography:EncryptData",
        "payment-cryptography:DecryptData",
        "payment-cryptography:ReEncrypt*"
      ],
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*"
    },
    {
      "Sid": "IAMDenyOnTag",
      "Effect": "Deny",
      "Action": [
        "payment-cryptography:EncryptData",
        "payment-cryptography:DecryptData",
        "payment-cryptography:ReEncrypt*"
      ],
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Type": "Reserved"
        }
      }
    },
    {
      "Sid": "IAMDenyNoTag",
      "Effect": "Deny",
      "Action": [
       "payment-cryptography:EncryptData",
       "payment-cryptography:DecryptData",
       "payment-cryptography:ReEncrypt*"
      ],
      "Resource": "arn:aws:kms:*:111122223333:key/*",
      "Condition": {
        "Null": {
          "aws:ResourceTag/Type": "true"
        }
      }
    }
  ]
}
```

了解的关键属性 AWS 支付加密密钥

正确管理密钥的原则是，密钥的范围必须适当，并且只能用于允许的操作。因此，某些密钥只能在特定密钥使用模式下创建。只要有可能，这就会与中定义的可用使用模式保持一致。TR-31

尽管 AWS 支付加密可以防止您创建无效的密钥，但为方便起见，此处提供了有效的组合。

对称密钥

- TR31_B0_BASE_DERIVATION_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_C0_CARD_VERIFICATION_KEY
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{加密 = 真、解密 = 真、换行 = 真、解包 = 真}、{加密 = 真、换行 = 真}、{加密 = 真、换行 = 真}、{解密 = 真}、{= 真} NoRestrictions
- TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY*、AES_128*、AES_192*、AES_256*
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_E1_EMV_MKEY_CONFIDENTIALITY
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_E2_EMV_MKEY_INTEGRITY
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_E5_EMV_MKEY_CARD_PERSONALIZATION

- 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
- 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}

- TR31_E6_EMV_MKEY_OTHER
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY、AES_128*、AES_192*、AES_256*
  - 允许的按键使用模式组合：{ DeriveKey = true} , { NoRestrictions = true}
- TR31_K0_KEY_ENCRYPTION_KEY
  - 建议使用 TR31_K1_KEY_BLOCK_PROTECTION_KEY。允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{加密 = 真、解密 = 真、换行 = 真、解包 = 真}、{加密 = 真、换行 = 真}、{加密 = 真、换行 = 真}、{解密 = 真}、{= 真} NoRestrictions
- TR31_K1_KEY_BLOCK_PROTECTION_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{加密 = 真、解密 = 真、换行 = 真、解包 = 真}、{加密 = 真、换行 = 真}、{加密 = 真、换行 = 真}、{解密 = 真}、{= 真} NoRestrictions
- TR31_M1_ISO_9797_1_MAC_KEY
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_M3_ISO_9797_3_MAC_KEY
  - 允许的密钥算法：TDES_2KEY、TDES_3KEY
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_M6_ISO_9797_5_CMAC_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_M7_HMAC_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_P0_PIN_ENCRYPTION_KEY

- 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
- 允许组合使用密钥模式：{加密 = 真、解密 = 真、换行 = 真、解包 = 真}、{加密 = 真、换行 = 真}、{加密 = 真、换行 = 真}、{解密 = 真}、{= 真} NoRestrictions

- TR31_V1_IBM3624_PIN_VERIFICATION_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}
- TR31_V2_VISA_PIN_VERIFICATION_KEY
  - 允许的密钥算法：TDES_2KEY ,TDES_3KEY ,AES_128 ,AES_192 ,AES_256
  - 允许组合使用密钥模式：{生成 = 真}、{验证 = 真}、{生成 = 真、验证= 真}、{= 真}、{ NoRestrictions = 真}

非对称密钥

- TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION
  - 允许的密钥算法：RSA_2048 ,RSA_3072 ,RSA_4096
  - 允许的密钥使用模式组合：{ Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } ,{ Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true }
  - 注意：在导入用于加密数据或封装密钥的公钥时，{Encrypt = true，Wrap = true} 是唯一有效的选项
- TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE
  - 允许的密钥算法：RSA_2048 ,RSA_3072 ,RSA_4096
  - 允许组合使用密钥模式：{Sign = true}、{Verify = true}
  - 注意：在导入用于签名的密钥（例如根证书、中间证书或的签名证书）时，{Verify = true} 是唯一有效的选项 TR-34。
- TR31_K3_ASYMMETRIC_KEY_FOR_KEY_ACTREMENT
  - 用于 ECDH 等密钥协议算法
  - 允许的密钥算法：ECC_NIST_P256、ECC_NIST_P384、ECC_NIST_P521
  - 允许组合使用关键模式：{ DeriveKey = true}。
  - 注意：DeriveKeyUsage 用于指定将从该基本密钥派生哪种密钥。这是固定在键盘上 creation/import的。
- TR31_K2_TR34_ASYMETRIC_KEY

- 非对称密钥用于 X9.24 兼容的密钥交换机制，例如 TR-34
- 允许的密钥算法：RSA_2048、RSA_3072、RSA_4096
- 允许组合使用关键模式：{ DeriveKey = true}。
- 允许的密钥使用模式组合：{ Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } , { Encrypt = true, Wrap = true } ,{ Decrypt = true, Unwrap = true }
- 注意：在导入用于加密数据或封装密钥的公钥时，{Encrypt = true，Wrap = true} 是唯一有效的选项

* 目前任何加密操作都不支持这种 algorithm/key 类型组合

数据操作

在您建立 AWS 支付密码学密钥后，它可用于执行加密操作。不同的操作执行不同类型的活动，包括加密、哈希以及领域特定算法（例如 CVV2 生成）。

如果没有匹配的解密密钥（对称密钥或私钥，取决于加密类型），则无法解密加密数据。同样，如果没有对称密钥或公钥，则无法验证哈希算法和特定域算法。

有关特定操作的有效密钥类型信息，请参阅加密操作的有效密钥

> **Note**
>
> 我们建议在非生产环境中使用测试数据。在非生产环境中使用生产密钥和数据（PAN、BDK ID 等）可能会影响您的合规范围，例如 PCI DSS 和 PCI P2PE。

主题

- 加密、解密和重新加密数据
- 生成并验证卡数据
- 生成、转换和验证 PIN 数据
- 生成并验证 ARQC
- 生成并验证 MAC
- 加密操作的有效密钥

加密、解密和重新加密数据

加密和解密方法可用于使用各种对称和非对称技术（包括 TDES、AES 和 RSA）来加密或解密数据。这些方法还支持使用 DUKPT 和 EMV 技术派生的密钥。对于希望在新密钥下保护数据而不暴露底层数据的用例，也可以使用该 ReEncrypt 命令。

> **Note**
>
> 使用这些 encrypt/decrypt 函数时，假设所有输入均采用 HexBinary 格式——例如，值 1 将输入为 31（十六进制），小写的 t 表示为 74（十六进制）。所有输出也都采用十六进制格式。

有关所有可用选项的详细信息，请参阅加密、解密和重新加密的 API 指南。

主题

- 加密数据
- 解密数据

加密数据

该 Encrypt Data API 用于使用对称和非对称数据加密密钥以及 DUKPT 和 EMV 派生的密钥对数据进行加密。支持各种算法和变体，包括 TDES、RSA 和 AES。

主要输入是用于加密数据的加密密钥、要加密的 HexBinary 格式的纯文本数据以及诸如 TDES 之类的分组密码的初始化向量和模式等加密属性。纯文本数据必须是 8 字节TDES、16 字节AES和密钥长度的倍数（如果是）。RSA如果输入数据不符合这些要求，则应填充对称密钥输入（TDES、AES、DUKPT、EMV）。下表显示了每种密钥类型的最大明文长度以及您在中EncryptionAttributes为 RSA 密钥定义的填充类型。

填充类型	RSA_2048	RSA_3072	RSA_4096
OAEP_SHA1	428	684	940
OAEP_SHA256	380	636	892
OAEP_SHA512	252	508	764
PKCS1	488	744	1000
None	488	744	1000

主要输出包括十六进制格式加密文字形式的加密数据以及加密密钥的校验和值。有关所有可用选项的详细信息，请查阅加密的 API 指南。

示例

- 使用 AES 对称密钥加密数据
- 使用 DUKPT 密钥加密数据
- 使用 EMV 派生的对称密钥加密数据

- 使用 RSA 密钥加密会话数据

使用 AES 对称密钥加密数据

> **Note**
>
> 所有示例都假设相关密钥已经存在。可以使用该操作创建密钥，也可以使用该CreateKey操作导入密钥。ImportKey

Example

在此示例中，我们将使用使用操作创建或使用CreateKey操作导入的对称密钥对纯文本数据进行加密。ImportKey要执行此操作，密钥必须 KeyModesOfUse 设置为，`Encrypt`并 KeyUsage 设置为`TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY`。有关更多选项，请参阅加密操作密钥。

```
$ aws payment-cryptography-data encrypt-data --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi --plain-text
 31323334313233343132333431323334 --encryption-attributes 'Symmetric={Mode=CBC}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "71D7AE",
    "CipherText": "33612AB9D6929C3A828EB6030082B2BD"
}
```

使用 DUKPT 密钥加密数据

Example

在此示例中，我们将使用 DU KPT 密钥对纯文本数据进行加密。 AWS 支付密码学支持TDES和 AES DUKPT 密钥。要执行此操作，密钥必须 KeyModesOfUse 设置为，DeriveKey并 KeyUsage 设置为TR31_B0_BASE_DERIVATION_KEY。有关更多选项，请参阅加密操作密钥。

```
$ aws payment-cryptography-data encrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --plain-text 31323334313233343132333431323334 --encryption-attributes
 'Dukpt={KeySerialNumber=FFFF9876543210E00001}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "71D7AE",
    "CipherText": "33612AB9D6929C3A828EB6030082B2BD"
}
```

使用 EMV 派生的对称密钥加密数据

Example

在此示例中，我们将使用已创建的 EMV 派生的对称密钥对明文数据进行加密。你可以使用这样的命令将数据发送到 EMV 卡。要执行此操作，密钥必须 KeyModesOfUse 设置为，Derive且必须 KeyUsage 设置为TR31_E1_EMV_MKEY_CONFIDENTIALITY或TR31_E6_EMV_MKEY_OTHER。有关更多详细信息，请参阅加密操作密钥。

```
$ aws payment-cryptography-data encrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --plain-text 33612AB9D6929C3A828EB6030082B2BD --encryption-attributes
 'Emv={MajorKeyDerivationMode=EMV_OPTION_A,PanSequenceNumber=27,PrimaryAccountNumber=1000000000
 InitializationVector=1500000000000999,Mode=CBC}'
```

```
{
```

```
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "71D7AE",
    "CipherText": "33612AB9D6929C3A828EB6030082B2BD"
}
```

使用 RSA 密钥加密会话数据

Example

在此示例中，我们将使用使用操作导入的 RSA 公钥对纯文本数据进行加密。ImportKey要执行此操作，密钥必须 KeyModesOfUse 设置为，Encrypt并 KeyUsage 设置为TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION。有关更多选项，请参阅加密操作密钥。

对于 PKCS #7 或其他目前不支持的填充方案，请在调用服务之前申请，并通过省略填充指示器 'Asymmetric={}' 来选择无填充

```
$ aws payment-cryptography-data encrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/thfezpmsalcfwmsg
 --plain-text 313233343132333431323334313233343 --encryption-attributes
 'Asymmetric={PaddingType=OAEP_SHA256}'
```

```
{
 "CipherText":
 "12DF6A2F64CC566D124900D68E8AFEAA794CA819876E258564D525001D00AC93047A83FB13 \
 E73F06329A100704FA484A15A49F06A7A2E55A241D276491AA91F6D2D8590C60CDE57A642BC64A897F4832A3930
 \
 0FAEC7981102CA0F7370BFBF757F271EF0BB2516007AB111060A9633D1736A9158042D30C5AE11F8C5473EC70F067
 \
 72590DEA1638E2B41FAE6FB1662258596072B13F8E2F62F5D9FAF92C12BB70F42F2ECDCF56AADF0E311D4118FE3591
 \
 FB672998CCE9D00FFFE05D2CD154E3120C5443C8CF9131C7A6A6C05F5723B8F5C07A4003A5A6173E1B425E2B5E42AD
 \
 7A2966734309387C9938B029AFB20828ACFC6D00CD1539234A4A8D9B94CDD4F23A",
 "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/5dza7xqd6soanjtb",
 "KeyCheckValue": "FF9DE9CE"
}
```

解密数据

该 Decrypt Data API 用于使用对称和非对称数据加密密钥以及 DUK PT 和 EMV 派生的密钥来解密数据。支持各种算法和变体，包括 TDES、RSA 和 AES。

主要输入是用于解密数据的解密密钥、要解密的十六进制格式的加密文字数据以及解密属性，例如初始化向量、分组密码模式等。主要输出包括十六进制格式的明文解密数据以及解密密钥的校验和值。有关所有可用选项的详细信息，请查阅解密 API 指南。

示例

- 使用 AES 对称密钥解密数据
- 使用 DUKPT 密钥解密数据
- 使用 EMV 派生的对称密钥解密数据
- 使用 RSA 密钥解密数据

使用 AES 对称密钥解密数据

Example

在此示例中，我们将使用对称密钥解密密文数据。此示例显示了一个AES密钥，但TDES_3KEY也支持TDES_2KEY和。要执行此操作，密钥必须 KeyModesOfUse 设置为，Decrypt并 KeyUsage 设置为TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY。有关更多选项，请参阅加密操作密钥。

```
$ aws payment-cryptography-data decrypt-data --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi --cipher-text
 33612AB9D6929C3A828EB6030082B2BD --decryption-attributes 'Symmetric={Mode=CBC}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "71D7AE",
    "PlainText": "313233343132333431323334313233343132333431323334"
}
```

使用 DUKPT 密钥解密数据

> **Note**
>
> 将解密数据与 DUKPT 一起用于 P2PE 交易，可能会将信用卡 PAN 和其他持卡人数据返回到您的应用程序，在确定其 PCI DSS 范围时需要考虑这些数据。

Example

在此示例中，我们将使用使用操作创建或使用CreateKey操作导入的 DUKPT 密钥解密密文数据。ImportKey要执行此操作，密钥必须 KeyModesOfUse 设置为，DeriveKey并 KeyUsage 设置为TR31_B0_BASE_DERIVATION_KEY。有关更多选项，请参阅加密操作密钥。使用 DUKPT 时，对于 TDES 算法，加密文字数据长度必须是 16 字节的倍数。对于 AES 算法，加密文字数据长度必须是 32 字节的倍数。

```
$ aws payment-cryptography-data decrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --cipher-text 33612AB9D6929C3A828EB6030082B2BD --decryption-attributes
 'Dukpt={KeySerialNumber=FFFF9876543210E00001}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "71D7AE",
    "PlainText": "31323334313233343132333431323334"
}
```

使用 EMV 派生的对称密钥解密数据

Example

在此示例中，我们将使用 EMV 派生的对称密钥解密密文数据，该密钥是使用操作创建或使用操作导入CreateKey的。ImportKey要执行此操作，密钥必须 KeyModesOfUse 设置为，Derive且必须 KeyUsage 设置为TR31_E1_EMV_MKEY_CONFIDENTIALITY或TR31_E6_EMV_MKEY_OTHER。有关更多详细信息，请参阅加密操作密钥。

```
$ aws payment-cryptography-data decrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --cipher-text 33612AB9D6929C3A828EB6030082B2BD --decryption-attributes
 'Emv={MajorKeyDerivationMode=EMV_OPTION_A,PanSequenceNumber=27,PrimaryAccountNumber=1000000000
 InitializationVector=1500000000000999,Mode=CBC}'
```

```
{
"KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi",
"KeyCheckValue": "71D7AE",
"PlainText": "31323334313233343132333431323334"
}
```

使用 RSA 密钥解密数据

Example

在此示例中，我们将使用使用操作创建的 RSA 密钥对来解密密文数据。CreateKey要执行此操作，必须 KeyModesOfUse 将密钥设置为启用Decrypt并 KeyUsage 设置为TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION。有关更多选项，请参阅加密操作密钥。

对于 PKCS #7 或当前不支持的其他填充方案，请通过省略填充指示符“Asymmetry={}”来选择无填充，并在调用服务后删除填充。

```
$ aws payment-cryptography-data decrypt-data \
        --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/5dza7xqd6soanjtb --cipher-text
 8F4C1CAFE7A5DEF9A40BEDE7F2A264635C... \
        --decryption-attributes 'Asymmetric={PaddingType=OAEP_SHA256}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-
east-1:111122223333:key/5dza7xqd6soanjtb",
    "KeyCheckValue": "FF9DE9CE",
    "PlainText": "313233343132333431323334313233343132333431323334"
}
```

生成并验证卡数据

生成并验证卡片数据包含从卡片数据衍生的数据，例如 CVV CVV2、CVC 和 DCVV。

主题

- 生成卡数据
- 验证卡数据

生成卡数据

该 Generate Card Data API 用于使用 CVV CVV2 或 Dynamic 等算法生成卡牌数据。 CVV2要查看此命令可以使用哪些密钥，请参阅加密操作的有效密钥部分。

许多加密值（例如 CVV、iCVV CVV2、CAVV V7）使用相同的加密算法，但输入值会有所不同。例如，CardVerificationValue1 的输入为 ServiceCode，卡号和到期日期。虽然 CardVerificationValue2 只有两个这样的输入，但这是因为对于 CVV2/CVC2，固定 ServiceCode 为 000。同样，对于iCvV，固定 ServiceCode 为999。某些算法可能会重新利用现有字段，例如 CAVV V8，在这种情况下，您需要查阅提供商手册以获取正确的输入值。

> **Note**
>
> 必须以相同的格式（例如 MMYY 与 YYMM）输入生成和验证的到期日期，才能生成正确的结果。

生成 CVV2

Example

在此示例中，我们将 CVV2 为给定的 PAN 生成一个，其输入值为PAN和卡片到期日期。这假设您已生成信用卡验证密钥。

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue2={CardExpiryDate=0123}
```

```
  {
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
      "KeyCheckValue": "CADDA1",
      "ValidationData": "801"
  }
```

生成 iCvV

Example

在此示例中，我们将为给定的 PAN 生成一个 ICvV，输入为PAN，服务代码为 999，卡到期日期。这假设您已生成信用卡验证密钥。

有关所有可用参数，请参阅 API 参考指南中的 CardVerificationValue1。

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=999}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "CADDA1",
    "ValidationData": "801"
}
```

验证卡数据

`Verify Card Data` 用于验证使用依赖于加密主体的支付算法创建的数据，例如 `DISCOVER_DYNAMIC_CARD_VERIFICATION_CODE`。

输入值通常作为入站交易的一部分提供给发卡方或支持平台合作伙伴。要验证 ARQC 密码（用于 EMV 芯片卡），请参阅验证 ARQC。

有关更多信息，请参阅 API 指南VerifyCardValidationData中的。

如果该值经过验证，则 api 将返回 http/200。如果该值未经过验证，它将返回 http/400。

验证 CVV2

Example

在此示例中，我们将验证给定 PAN 的 CVV/ CVV2 。通常 CVV2 由持卡人或用户在交易期间提供以进行验证。为了验证他们的输入，将在运行时提供以下值—— 用于验证的密钥 (CVK)、PAN、信用卡到期日期并 CVV2 输入。卡到期格式必须与初始值生成中使用的格式匹配。

有关所有可用参数，请参阅 API 参考指南中的 CardVerificationValue2。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue2={CardExpiryDate=0123} --validation-data 801
```

```
  {
      "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
      "KeyCheckValue": "CADDA1"
  }
```

验证 iCvV

Example

在此示例中，我们将使用用于验证的密钥 (C VK)、服务代码为 999 PAN、卡到期日期以及交易提供的待验证的 IC V V 来验证给定 PAN 的 ICVV。

iCvV 不是用户输入的值（比如 CVV2），而是嵌入在 EMV 卡上。应考虑在提供时是否应始终进行验证。

有关所有可用参数，请参阅 API 参考指南中的 CardVerificationValue1。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/tqv5yij6wtxx64pi
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=999} --validation-data 801
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "CADDA1",
    "ValidationData": "801"
}
```

生成、转换和验证 PIN 数据

PIN 数据功能允许您生成随机 pin、pin 验证值 (PVV) 并根据 PVV 或 PIN 偏移验证入站加密 PIN。

Pin 转换允许您将 pin 从一个工作密钥转换为另一个工作密钥，而无需按照 PCI PIN 要求 1 的规定以明文形式显示 pin。

> **Note**
>
> 由于 PIN 生成和验证通常是发行方功能，而 PIN 转换是典型的收单方功能，因此我们建议您考虑最低特权访问并为您的系统使用案例设置相应的策略。

主题

- 转换 PIN 数据
- 生成 PIN 数据
- 验证 PIN 数据

转换 PIN 数据

转换 PIN 数据功能用于将加密的 PIN 数据从一组密钥转换为另一组密钥，而加密数据不会离开 HSM。它用于 P2PE 加密，其中工作密钥应该更改，但处理系统不需要或不允许解密数据。主要输入是加密数据、用于加密数据的加密密钥以及用于生成输入值的参数。另一组输入是请求的输出参数，例如用于加密输出的密钥和用于创建该输出的参数。主要输出是新加密的数据集以及用于生成该数据集的参数。

> **Note**
>
> 为了符合 PCI 标准，传入和传出 PrimaryAccountNumber 值必须匹配。不允许将 PIN 从一个 PAN 转换为另一个 PAN。

主题

- 从 PEK 到 DUKPT 的 PIN
- 从 PEK 到 PEK 的 PIN

从 PEK 到 DUKPT 的 PIN

Example

在此示例中，我们将使用 DUKPT 将 PIN 从 AES ISO 4 PIN 块转换为使用 ISO 0 PIN 块的 PEK TDES 加密。这种情况很常见，支付终端在 ISO 4 中对 PIN 进行加密，如果下一个连接尚不支持 AES，则可以将其转换回 TDES 进行下游处理。

```
$ aws payment-cryptography-data translate-pin-data --encrypted-pin-block
 "AC17DC148BDA645E" --outgoing-translation-
attributes=IsoFormat0='{PrimaryAccountNumber=171234567890123}' --outgoing-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt  --incoming-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/4pmyquwjs3yj4vwe --incoming-translation-attributes
 IsoFormat4="{PrimaryAccountNumber=171234567890123}" --incoming-dukpt-attributes
 KeySerialNumber="FFFF9876543210E00008"
```

```
    {
            "PinBlock": "1F4209C670E49F83E75CC72E81B787D9",
            "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
            "KeyCheckValue": "7CC9E2"
        }
```

从 PEK 到 PEK 的 PIN

Example

在此示例中，我们将在一个 PEK（PIN 加密密钥）下加密的 PIN 转换为另一个 PEK。这通常用于在使用不同加密密钥的不同系统或合作伙伴之间路由交易，同时通过在整个过程中保持 PIN 加密来保持 PCI PIN 合规性。在本示例中，两个密钥都使用 TDES 3KEY 加密，但有多种选项可供选择，包括 AES ISO-4 到 TDES ISO-0、DUKPT 到 PEK 或 AS2805 到 PEK。

```
$ aws payment-cryptography-data translate-pin-data --encrypted-pin-block
 "AC17DC148BDA645E" \
    --incoming-translation-attributes
 IsoFormat0='{PrimaryAccountNumber=171234567890123}' \
    --incoming-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt \
    --outgoing-translation-attributes
 IsoFormat0='{PrimaryAccountNumber=171234567890123}' \
    --outgoing-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwfxug3pgy6xh
```

```
{
    "PinBlock": "E8F2A6C4D1B93E7F",
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwfxug3pgy6xh",
    "KeyCheckValue": "9A325B"
}
```

输出 PIN 块现在在第二个 PEK 下加密，可以安全地传输到持有相应密钥的下游系统。

生成 PIN 数据

生成 PIN 数据函数用于生成 PIN-related 值，例如 PVV 和密码块偏移量，用于验证用户在交易或授权期间输入的密码。此 API 还可以使用各种算法生成新的随机 pin 码。

生成随机密码并匹配的 Visa PVV

Example

在此示例中，我们将生成一个新的（随机）引脚，其中输出将是加密的PIN block（PinData.PinBlock）和PVV（引脚Data.Offset）。密钥输入是PAN、Pin Verification Key、Pin Encryption Key、和PIN block format。

此命令要求密钥的类型为TR31_V2_VISA_PIN_VERIFICATION_KEY。

```
$ aws payment-cryptography-data generate-pin-data --generation-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2 --encryption-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt
 --primary-account-number 171234567890123 --pin-block-format ISO_FORMAT_0 --generation-
attributes VisaPin={PinVerificationKeyIndex=1}
```

```
{
            "GenerationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
            "GenerationKeyCheckValue": "7F2363",
            "EncryptionKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt",
            "EncryptionKeyCheckValue": "7CC9E2",
            "EncryptedPinBlock": "AC17DC148BDA645E",
            "PinData": {
                "VerificationValue": "5507"
            }
        }
```

为已知密码生成 Visa PVV

Example

在此示例中，我们将为给定（加密）PIN 生成 PVV。可以在上游接收加密的 PIN，例如从支付终端接收或使用用户可选密码流从持卡人那里接收。关键输入是PANPin Verification Key、Pin Encryption Key、Encrypted Pin Block和PIN block format。

```
$ aws payment-cryptography-data generate-pin-data --generation-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2
 --encryption-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt --primary-account-number
 171234567890123 --pin-block-format ISO_FORMAT_0 --generation-attributes
 VisaPinVerificationValue={PinVerificationKeyIndex=1,EncryptedPinBlock=AA584CED31790F37}
```

```
{
            "GenerationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
            "GenerationKeyCheckValue": "7F2363",
            "EncryptionKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt",
            "EncryptionKeyCheckValue": "7CC9E2",
            "EncryptedPinBlock": "AC17DC148BDA645E",
            "PinData": {
                "VerificationValue": "5507"
            }
        }
```

为引脚生成 IBM3624 引脚偏移量

IBM 3624 PIN Offset 有时也称为 IBM 方法。此方法使用验证数据（通常为 natural/intermediate PAN）和 PIN 密钥 (PVK) 生成 PIN。自然密码实际上是一种派生值，而确定性对于发卡机构来说处理起来非常有效，因为不需要在持卡人层面存储任何密码数据。最明显的缺点是，该方案没有考虑持卡人可选或随机的密码。为了允许这些类型的引脚，在方案中添加了偏移算法。偏移量代表用户选择的（或随机）密码和自然密钥之间的差异。偏移值由发卡机构或发卡处理器存储。在交易时，AWS 支付加密服务会在内部重新计算自然密码，并应用抵消量来找到密码。然后，它将它与交易授权提供的值进行比较。

IBM3624 有几种选择：

- `Ibm3624NaturalPin`将输出自然密码和加密密码块

- Ibm3624PinFromOffset给定偏移量，将生成一个加密的密码块
- Ibm3624RandomPin将生成一个随机密码，然后生成匹配的偏移量和加密的密码块。
- Ibm3624PinOffset根据用户选择的引脚生成引脚偏移量。

在 AWS 支付密码学内部，执行以下步骤：

- 将提供的平移填充到 16 个字符。如果提供 <16，则使用提供的填充字符在右侧填充。
- 使用 PIN 生成密钥加密验证数据。
- 使用十进制表对加密数据进行十进制化。这会将十六进制数字映射到十进制数字，例如 “A” 可以映射到 9，而 1 可以映射到 1。
- 从输出的十六进制表示法中获取前 4 位数。这是天然的别针。
- 如果用户选择或生成了随机密码，则使用客户密码模减自然密码。结果是引脚偏移量。

示例

- 示例：为引脚生成 IBM3624 引脚偏移量

示例：为引脚生成 IBM3624 引脚偏移量

在此示例中，我们将生成一个新的（随机）引脚，其中输出将是加密的PIN block（PinData.PinBlock）和IBM3624偏移值（引脚Data.Offset）。输入是PAN验证数据（通常是平移）、填充字符Pin Verification Key、、Pin Encryption Key和。PIN block format

此命令要求 pin 生成密钥的类型为TR31_V1_IBM3624_PIN_VERIFICATION_KEY，加密密钥的类型为 TR31_P0_PIN_ENCRYPTION_KEY

Example

以下示例显示生成一个随机引脚，然后使用以下方法输出加密的引脚块和 IBM3624 偏移值 Ibm3624RandomPin

```
$ aws payment-cryptography-data generate-pin-data --generation-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2
 --encryption-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt --primary-account-number
 171234567890123 --pin-block-format ISO_FORMAT_0 --generation-attributes
 Ibm3624RandomPin="{DecimalizationTable=9876543210654321,PinValidationDataPadCharacter=D,PinVal
```

```
{
              "GenerationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
              "GenerationKeyCheckValue": "7F2363",
              "EncryptionKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt",
              "EncryptionKeyCheckValue": "7CC9E2",
              "EncryptedPinBlock": "AC17DC148BDA645E",
              "PinData": {
                  "PinOffset": "5507"
              }
          }
```

验证 PIN 数据

验证 PIN 数据功能用于验证 pin 是否正确。这通常涉及将之前存储的 PIN 值与持卡人在 POI 输入的值进行比较。这些函数比较两个值，但不暴露任一来源的底层值。

使用 PVV 方法验证加密的 PIN

Example

在此示例中，我们将验证给定 PAN 的 PIN。PIN 通常由持卡人或用户在交易期间提供以进行验证，并与存档的值进行比较（持卡人的输入作为来自终端或其他上游提供商的加密值提供）。为了验证此输入，还将在运行时提供以下值：用于加密输入 pin 的密钥（通常为IWK）PAN和用于验证的值（a PVV 或PIN offset）。

如果 AWS 支付密码学能够验证密码， http/200 则返回。如果 PIN 未通过验证，它将返回 http/400。

```
$ aws payment-cryptography-data verify-pin-data --verification-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2 --encryption-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt
 --primary-account-number 171234567890123 --pin-block-format ISO_FORMAT_0  --
verification-attributes VisaPin="{PinVerificationKeyIndex=1,VerificationValue=5507}" --
encrypted-pin-block AC17DC148BDA645E
```

```
        {
            "VerificationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
            "VerificationKeyCheckValue": "7F2363",
            "EncryptionKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt",
            "EncryptionKeyCheckValue": "7CC9E2",

        }
```

使用 PVV 方法验证加密 PIN-错误密码错误

Example

在此示例中，我们将尝试验证给定 PAN 的 PIN，但由于该 PIN 不正确，它将失败。

使用 SDK 时，它显示为 {“消息”: “密码块验证失败”。, “原因”: “PIN 无效”}

```
$ aws payment-cryptography-data verify-pin-data --verification-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2 --encryption-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt
 --primary-account-number 171234567890123 --pin-block-format ISO_FORMAT_0  --
verification-attributes VisaPin="{PinVerificationKeyIndex=1,VerificationValue=9999}" --
encrypted-pin-block AC17DC148BDA645E
```

```
       An error occurred (VerificationFailedException) when calling the VerifyPinData
 operation: Pin block verification failed.
```

使用 PVV 方法验证加密的 PIN-错误输入

Example

在此示例中，我们将尝试验证给定 PAN 的 PIN，但由于输入错误且传入的数据不是有效的 PIN，它将失败。常见原因是：使用的 1/wrong 密钥 2/input 参数（例如全景或密码块格式）不正确 3/pin 区块已损坏。

```
$ aws payment-cryptography-data verify-pin-data --verification-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2
 --encryption-key-identifier  --primary-account-number 171234567890123
 --pin-block-format ISO_FORMAT_0  --verification-attributes
 VisaPin="{PinVerificationKeyIndex=1,VerificationValue=9999}" --encrypted-pin-block
 AC17DC148BDA645E
```

```
An error occurred (ValidationException) when calling the VerifyPinData
operation: Pin block provided is invalid. Please check your input to ensure all field
values are correct.
```

根据先前存储的 IBM3624 引脚偏移量验证 PIN

在此示例中，您将根据发卡机构或处理器存档的密码偏移量来验证持卡人提供的 PIN。输入包括验证密钥 (PVK)、加密密钥 (PEK)、主账号、来自支付终端（或其他上游提供商，例如信用卡网络）的加密 PIN 块以及之前存储的 pin 偏移量。如果 PIN 匹配，API 将返回 HTTP/200 响应。

此命令要求 pin 验证密钥的类型为`TR31_V1_IBM3624_PIN_VERIFICATION_KEY`，加密密钥的类型为`TR31_P0_PIN_ENCRYPTION_KEY`。

> **⚠ PIN 长度验证**
>
> 可选`--pin-data-length`参数指定要验证的 PIN 的长度（4 到 12 位数字）。如果您未指定此参数，则服务会验证偏移值保护的整个 PIN 长度。如果您只想验证 PIN 的一部分，请使用`--pin-data-length`参数。例如，如果设置`--pin-data-length`为 4 但实际 PIN 为 6 位数，则该服务仅验证最右边的 4 位数字。未检查剩余的 2 位数。如果持卡人的 PIN 为 6 位数，但传入的 PIN 仅为 4 位数，例如当传入网络或商家仅支持 4 位数的 PIN 时，您可以故意这样做。

Example

```
$ aws payment-cryptography-data verify-pin-data \
    --verification-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2 \
    --encryption-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt \
    --primary-account-number 171234567890123 \
    --pin-block-format ISO_FORMAT_0 \
    --encrypted-pin-block AC17DC148BDA645E \
    --pin-data-length 4 \
    --verification-attributes
 Ibm3624PinOffset="{DecimalizationTable=9876543210654321,PinValidationDataPadCharacter=D,PinVal
```

```
{
    "VerificationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
    "VerificationKeyCheckValue": "7F2363",
    "EncryptionKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
    "EncryptionKeyCheckValue": "7CC9E2"
}
```

生成并验证 ARQC

授权请求密码 (ARQC) 是在支付交易过程中由 EMV 芯片卡（或数字等效物，例如移动钱包）生成的密码。ARQC 对卡进行加密验证，并确保交易数据的完整性。它整合了银行卡数据、终端数据和交易特定数据，为每笔交易生成唯一的价值。

AWS 支付密码学支持两个 ARQC 操作。您可以为开发和测试目的生成 ARQC，也可以为生产交易授权流程验证 ARQC。

主题

- 生成身份验证请求 (ARQC) 密码
- 验证身份验证请求 (ARQC) 密码

生成身份验证请求 (ARQC) 密码

生成身份验证请求密码 API 用于生成 ARQC。此 API 允许您使用 AWS 支付密码生成 ARQC，用于开发和测试目的。

> **Important**
>
> 此操作仅适用于开发和测试场景。不建议在生产支付流程中使用此操作来替代基于卡片的密码生成。

有关所有可用选项，请参阅 API 参考GenerateAuthRequestCryptogram中的。

ARQC 密码通常需要以下输入（尽管这可能因实现而异）：

- PAN-在 PrimaryAccountNumber 字段中指定
- PAN 序列号 (PSN)-在字段中指 PanSequenceNumber 定
- 密钥派生方法，例如通用会话密钥 (CSK)-在 SessionKeyDerivationAttributes
- 主密钥派生模式（例如 EMV 选项 A）-在 MajorKeyDerivationMode
- 交易数据-一串交易、终端和银行卡数据，例如金额和日期。在 TransactionData 字段中指定。
- 颁发者主密钥-用于派生密码 (AC) 密钥的主密钥。此密钥可保护单个交易，并在 KeyIdentifier 字段中指定。

有关生成和填充交易数据的更多信息，请参阅验证身份验证请求 (ARQ C) 密码。

主题

- 示例

示例

Visa CVN10

Example

以下示例使用 Visa CVN10 生成 ARQC。

如果 P AWS ayment Cryptography 成功生成 ARQC， http/200 则会返回与生成的密码一起返回。

```
$ aws payment-cryptography-data generate-auth-request-cryptogram \
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
--major-key-derivation-mode EMV_OPTION_A \
--transaction-data
 0000000017000000000000000840008000800008401605170000000093800000B03011203000000 \
--session-key-derivation-attributes='{"Visa":{"PanSequenceNumber":"01", \
"PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk",
  "KeyCheckValue": "08D7B4",
  "AuthRequestCryptogram": "D791093C8A921769"
}
```

Visa CVN18 和 Visa CVN22

Example

以下示例使用 Visa CVN18 或 CVN22 生成 ARQC。CVN18 和 CVN22 之间的加密操作相同，但交易数据中包含的数据有所不同。与 CVN10 相比，即使输入相同，也会生成完全不同的密码。

如果 P AWS ayment Cryptography 成功生成 ARQC， http/200 则会返回与生成的密码一起返回。

```
$ aws payment-cryptography-data generate-auth-request-cryptogram \
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
--major-key-derivation-mode EMV_OPTION_A \
--transaction-data
 0000000017000000000000000840008000800008401605170000000093800000B1F22010300000000000
 \
00000000000000000000000000000000000000000000800000000000000000 \
--session-key-derivation-attributes='{"EmvCommon":
{"ApplicationTransactionCounter":"000B", \
"PanSequenceNumber":"01","PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk",
  "KeyCheckValue": "08D7B4",
  "AuthRequestCryptogram": "61EDCC708B4C97B4"
}
```

验证身份验证请求 (ARQC) 密码

验证身份验证请求密码 API 用于验证 ARQC。ARQC的生成通常是在交易授权期间在EMV芯片卡（或数字等效物，例如移动钱包）上执行的，尽管该服务可以出于开发目的而生成。ARQC 对于每笔交易都是唯一的，旨在以加密方式显示卡的有效性并确保交易数据与当前（预期）交易完全匹配。

AWS 支付密码学为验证ARQC和生成可选的ARPC值提供了多种选项，包括 EMV 4.4 Book 2 中定义的值以及Visa和Mastercard使用的其他方案。有关所有可用选项的完整列表，请参阅 API 指南中的 VerifyCardValidationData 部分。

ARQC 密码通常需要以下输入（尽管这可能因实现而异）：

- PAN-在 PrimaryAccountNumber 字段中指定
- PAN 序列号 (PSN)-在字段中指 PanSequenceNumber 定

- 密钥派生方法，例如通用会话密钥 (CSK)-在 SessionKeyDerivationAttributes
- 主密钥派生模式（例如 EMV 选项 A）-在 MajorKeyDerivationMode
- 交易数据-在 TransactionData 字段中指定的各种交易、终端和银行卡数据的字符串，例如金额和日期
- 颁发者主密钥-用于派生用于保护个人交易并在字段中指定的密码 (AC) 密钥的主密钥 KeyIdentifier

主题

- 建立交易数据
- 交易数据填充
- 示例

建立交易数据

交易数据字段的确切内容（和顺序）因实现和网络方案而异，但最低推荐字段（和串联顺序）在 EMV 4.4 Book 2 第 8.1.1 节 ——数据选择中定义。如果前三个字段是金额 (17.00)、其他金额 (0.00) 和购买国家，则交易数据将按以下方式开始：

- 000000001700 - 金额 - 12 位隐含两位小数
- 000000000000 - 其他金额 - 12 位隐含两位小数
- 0124 - 四位数国家代码
- 输出（部分）交易数据 - 00000000170000000000000000124

交易数据填充

在将交易数据发送到服务之前，应先填充交易数据。大多数方案使用 ISO 9797 方法 2 填充，其中十六进制字符串后面加上十六进制 80，然后加上 00，直到该字段是加密块大小的倍数；TDES 为 8 字节或 16 个字符，AES 为 16 字节或 32 个字符。替代方案（方法 1）并不常见，但仅使用 00 作为填充字符。

ISO 9797 方法 1 填充

未填充：
00000000170000000000000008400080008000084016051700000000093800000B03011203（74 个字符或 37 个字节）

填充：
0000000017000000000000000840008000800008401605170000000093800000B03011203000000（80 个字符或 40 个字节）

ISO 9797 方法 2 填充

未填充：
0000000017000000000000000840008000800008401605170000000093800000B1F220103000000（80 个字符或 40 个字节）

填充：
0000000017000000000000000840008000800008401605170000000093800000B1F220103000000800000 个字符或 44 个字节）

示例

Visa CVN10

Example

在此示例中，我们将验证使用 Visa CVN10 生成的 ARQC。

如果 AWS 支付密码学能够验证 ARQC，则会返回。 http/200 如果 ARCQ（授权请求密码）未经过验证，它将返回响应。 http/400

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-
cryptogram D791093C8A921769 \
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
--major-key-derivation-mode EMV_OPTION_A \
--transaction-data
 0000000017000000000000000840008000800008401605170000000093800000B03011203000000 \
--session-key-derivation-attributes='{"Visa":{"PanSequenceNumber":"01" \
,"PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk",
  "KeyCheckValue": "08D7B4"
}
```

Visa CVN18 和 Visa CVN22

Example

在此示例中，我们将验证使用 Visa CVN18 或 CVN22 生成的 ARQC。CVN18 和 CVN22 之间的加密操作相同，但交易数据中包含的数据有所不同。与 CVN10 相比，即使输入相同，也会生成完全不同的密码。

如果 AWS 支付密码学能够验证 ARQC，则会返回。 http/200 如果 ARCQ 未经过验证，它将返回。 http/400

```
$ aws payment-cryptography-data verify-auth-request-cryptogram \
--auth-request-cryptogram 61EDCC708B4C97B4
--key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
--major-key-derivation-mode EMV_OPTION_A
--transaction-data
 00000000170000000000000008400080008000084016051700000000093800000B1F22010300000000000
 \
00000000000000000000000000000000000000000000800000000000000000
--session-key-derivation-attributes='{"EmvCommon":
{"ApplicationTransactionCounter":"000B", \
"PanSequenceNumber":"01","PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk",
  "KeyCheckValue": "08D7B4"
}
```

生成并验证 MAC

消息验证码 (MAC) 通常用于验证消息的完整性（是否已被修改）。诸如 HMAC（Hash-Based 消息身份验证码） CBC-MAC 和 CMAC（消息身份验证码）之类的加密哈希通过使用加密技术为 MAC 的发送者提供了额外的保证。Cipher-based HMAC 基于哈希函数，而 CMAC 基于分组密码。该服务还支持 ISO9797 算法 1 和 3，它们是 CBC-MACs.

此服务的所有 MAC 算法结合了加密哈希函数和共享密钥。他们获取消息和密钥，例如密钥中的密钥材料，然后返回一个唯一的标签或 MAC。即使是消息的一个字符发生了变化，或者密钥发生变化，生成的标签也会完全不同。通过要求提供密钥，加密 MAC 还提供了真实性；如果没有密钥，就不可能生成

相同的 MAC。加密 MAC 有时被称为对称签名，因为它们像数字签名一样工作，但使用单个密钥进行签名和验证。

AWS 支付密码学支持多种类型的 MAC：

ISO9797 算法 1

以 ISO9797_ALGORTH KeyUsage IM1 的形式表示。如果该字段不是区块大小的倍数（TDES 为 8 个 bytes/16 十六进制字符，AES 为 16 个字 bytes/32 符），P AWS ayment Cryptography 会自动应用 ISO9797 填充方法 1。如果需要其他填充方法，则可以在调用服务之前应用它们。

ISO9797 算法 3（零售 MAC）

以 ISO9797_ALGORTHM KeyUsage 3 的形式表示。与算法 1 相同的填充规则适用

ISO9797 算法 5 (CMAC)

由 TR31_M6_ISO_9797_5_CMAC_KEY 的 KeyUsage 表示

HMAC

用 TR31_M7_HMAC_KEY 的 KeyUsage 表示，包括 HMAC_SHA224、HMAC_SHA256、HMAC_SHA384 和 HMAC_SHA512

AS2805.4.1 MAC

用 TR31_M0_ISO_16609_M KeyUsage AC_KEY 表示。有关 AS2805 的更多详细信息，请参阅 ???

DUKPT MAC

DUKPT MAC 通常用于确认 to/from 支付终端的消息来源和有效负载。它使用 DUKPT 派生技术派生密钥，然后执行 MAC。用于此选项的密钥由 TR31_B0_BASE_DERIVATION_KE KeyUsage Y 中的 a 表示。

EMV MAC

在 EMV 文档中，EMV MAC 通常被称为完整性密钥。它使用 EMV 推导技术派生密钥，然后在内部使用 ISO9797_ALGORTHM3。它通常用于将发行者脚本发送到芯片卡进行重新编程。用于此选项的密钥由 TR31_E2_EMV_MKEY_INTEGRIT KeyUsage Y 的 a 表示。如果您既发送脚本又更新离线 PIN，请查看执行GenerateMacEmvPinChange这两个操作的脚本。

主题

- 生成 MAC
- 验证 MAC

生成 MAC

Generate MAC API 用于验证与卡相关的数据，例如来自卡片磁条的跟踪数据，方法是使用已知的加密密钥生成 MAC（消息身份验证码），用于在发送方和接收方之间进行数据验证。用于生成 MAC 的数据包括消息数据、机密 MAC 加密密钥和 MAC 算法，用于生成用于传输的唯一 MAC 值。MAC 的接收方将使用相同的 MAC 消息数据、MAC 加密密钥和算法来重现另一个 MAC 值以进行比较和数据验证。即使是消息的一个字符发生了变化，或者用于验证的 MAC 密钥不完全相同，生成的MAC 也会完全不同。该 API 支持用于此操作的 ISO 9797-1 算法 1 和 ISO 9797-1 算法 3 MAC（使用静态 MAC 密钥和派生的 DUKPT 密钥）、HMAC 和 EMV MAC 加密密钥。

`message-data` 的输入值必须是十六进制数据。

有关此 API 所有选项的更多信息，请参阅GenerateMac和VerifyMac。

可选参数 mac-length 允许您截断输出值（尽管这也可以在代码中完成）。长度为 8 表示 8 字节或 16 个十六进制字符。

MAC 密钥既可以通过 P AWS ayment Cryptography 通过调用创建，CreateKey也可以通过调用ImportKey来

> **Note**
>
> CMAC 和 HMAC 算法不需要填充。所有其他数据都要求将数据填充到算法的区块大小，TDES 为 8 字节（16 个十六进制字符），AES 为 16 字节（32 个十六进制字符）的倍数。

示例

- 生成 HMAC
- 使用 ISO 9797-1 算法 3 生成 MAC
- 使用 CMAC 生成 MAC
- 使用 DUKPT CMAC 生成 MAC

生成 HMAC

在此示例中，我们将使用 HMAC 算法`HMAC_SHA256`和 HMAC 加密密钥生成用于卡数据身份验证的 HMAC（Hash-Based 消息身份验证码）。密钥必须 KeyUsage 设置为，`TR31_M7_HMAC_KEY`并设置 KeyModesOfUse 为`Generate`。哈希长度（例如 256）是在创建密钥时定义的，无法修改。

可选的 mac-length 参数将修剪输出 MAC，尽管这也可以在服务外部执行。该值以字节为单位，因此，如果值为 16，则需要长度为 32 的十六进制字符串。

Example

```
$ aws payment-cryptography-data generate-mac \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6 \
    --message-data
 "3b3130383834393030313037333934313533d32343038323236303030373030303f33" \
    --generation-attributes Algorithm=HMAC
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6",
    "KeyCheckValue": "2976E7",
    "Mac": "ED87F26E961C6D0DDB78DA5038AA2BDDEA0DCE03E5B5E96BDDD494F4A7AA470C"
}
```

使用 ISO 9797-1 算法 3 生成 MAC

在本示例中，我们将使用 ISO 9797-1 算法 3（零售 MAC）生成 MAC，用于信用卡数据身份验证。密钥必须 KeyUsage 设置为，TR31_M3_ISO_9797_3_MAC_KEY并设置 KeyModesOfUse 为Generate。

Example

```
$ aws payment-cryptography-data generate-mac \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h \
    --message-data
 "3b3130383834393030313037333934313 53d32343038323236303030373030303f33" \
    --generation-attributes="Algorithm=ISO9797_ALGORITHM3"
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
    "KeyCheckValue": "2976EA",
    "Mac": "A8F7A73DAF87B6D0"
}
```

使用 CMAC 生成 MAC

当密钥为 AES 时，最常使用 CMAC，但它也支持 TDES。在此示例中，我们将使用 CMAC（ISO 9797-1 算法 5）生成一个 MAC，用于使用 AES 密钥进行卡数据身份验证。密钥必须 KeyUsage 设置为，TR31_M6_ISO_9797_5_CMAC_KEY并设置 KeyModesOfUse 为Generate。

Example

```
$ aws payment-cryptography-data generate-mac \
    --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
    --message-data
 "3b3130383834393030313037333934313 53d32343038323236303030373030303f33" \
    --generation-attributes Algorithm="CMAC"
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "C1EB8F",
    "Mac": "1F8C36E63F91E4E93DF7842BF5E2E5F7"
}
```

使用 DUKPT CMAC 生成 MAC

此示例使用 DUKPT（每笔交易派生的唯一密钥）和 CMAC 生成一个 MAC，用于卡数据身份验证。密钥必须 KeyUsage 设置为 true `TR31_B0_BASE_DERIVATION_KEY` 并 KeyModesOfUse `DeriveKey`设置为 true。

DUKPT 密钥使用基本派生密钥 (BDK) 和密钥序列号 (KSN) 为每笔交易派生一个唯一的密钥。

Example

```
$ aws payment-cryptography-data generate-mac --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/qnobl5lghrzunce6 --message-data
 "3b3130383834393030313037333934313533d323430383232363030303730303033f33" --generation-
attributes="DukptCmac={KeySerialNumber="932A6E954ABB32DD00000001",DukptKeyVariant=BIDIRECTIONAL
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6",
    "KeyCheckValue": "C1EB8F",
    "Mac": "1F8C36E63F91E4E93DF7842BF5E2E5F7"
}
```

验证 MAC

Verify MAC API 用于验证与卡片相关的数据身份验证的 MAC（消息身份验证代码）。它必须使用生成 MAC 期间使用的相同加密密钥来重新生成用于身份验证的 MAC 值。MAC 加密密钥既可以通过 Payment Crypto AWS graphy 通过调用创建，CreateKey也可以通过调用ImportKey来导入。该 API 支持用于此操作的 DUKPT MAC、HMAC 和 EMV MAC 加密密钥。

如果该值已通过验证，则响应参数`MacDataVerificationSuccessful` 将返回 `Http/200`，否则将为 `Http/400`，消息表示 `Mac verification failed`。

示例

- 验证 HMAC
- 使用 DUKPT CMAC 验证 MAC

验证 HMAC

在此示例中，我们将使用 HMAC 算法HMAC_SHA256和 HMAC Hash-Based 加密密钥验证用于卡数据身份验证的 HMAC（消息身份验证码）。密钥必须 KeyUsage 设置为 true TR31_M7_HMAC_KEY 并 KeyModesOfUse Verify设置为 true。

Example

```
$ aws payment-cryptography-data verify-mac \
     --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6 \
     --message-data
 "3b343038383439303031303733393431353d323430383232363030303730303033f33" \
     --mac ED87F26E961C6D0DDB78DA5038AA2BDDEA0DCE03E5B5E96BDDD494F4A7AA470C \
     --verification-attributes Algorithm=HMAC_SHA256
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6",
    "KeyCheckValue": "2976E7"
}
```

使用 DUKPT CMAC 验证 MAC

在此示例中，我们将使用 DUKPT（每笔交易派生的唯一密钥）和 CMAC 来验证 MAC 以进行卡数据身份验证。密钥必须 KeyUsage 设置为 true TR31_B0_BASE_DERIVATION_KEY 并 KeyModesOfUse DeriveKey设置为 true。DUKPT 密钥使用基本派生密钥 (BDK) 和密钥序列号 (KSN) 为每笔交易派生一个唯一的密钥。发送方和接收方之间的值 DukptKeyVariant 必须匹配。通常在终端到后端之间使用 REQUEST，从后端到终端使用 VERIFY，当双向使用单个密钥时，则使用双向。

Example

```
$ aws payment-cryptography-data verify-mac \
     --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi \
     --message-data
 "3b343038383439303031303733393431353d323430383232363030303730303033f33" \
     --mac D8E804EE74BF1D909A2C01C0BDE8EF34 \
     --verification-attributes
 DukptCmac='{"KeySerialNumber":"932A6E954ABB32DD00000001","DukptKeyVariant":"BIDIRECTIONAL"}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
tqv5yij6wtxx64pi",
    "KeyCheckValue": "C1EB8F"
}
```

加密操作的有效密钥

某些密钥只能用于某些操作。此外，某些操作可能会限制密钥的密钥使用模式。请查看下表中允许的组合。

> **Note**
>
> 某些组合虽然允许，但可能会造成无法使用的情况，例如生成 CVV 代码(`generate`)但随后无法验证(`verify`)。

主题

- GenerateCardData
- VerifyCardData
- GeneratePinData（用于 VISA/ABA 计划）
- GeneratePinData（适用于 IBM3624）
- VerifyPinData（用于 VISA/ABA 计划）
- VerifyPinData（适用于 IBM3624）
- 解密数据

- 加密数据
- 转换 PIN 数据
- Generate/Verify MAC
- GenerateMacEmvPinChange
- VerifyAuthRequestCryptogram
- Import/Export 钥匙
- 未使用的密钥类型

GenerateCardData

API 端点	加密操作或算法	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
GenerateCardData	• AMEX_CARD_SECURITY_CODE_VERSION_1 • AMEX_CARD_SECURITY_CODE_VERSION_2	TR31_C0_CARD_VERIFICATION_KEY	• TDES_2KEY • TDES_3KEY	{ Generate = true },{ Generate = true, Verify = true }
GenerateCardData	• CARD_VERIFICATION_VALUE_1 • CARD_VERIFICATION_VALUE_2	TR31_C0_CARD_VERIFICATION_KEY	• TDES_2KEY	{ Generate = true },{ Generate = true, Verify = true }
GenerateCardData	• CARDHOLDER_AUTHENTICATION_VERIFICATION_VALUE	TR31_E6_EMV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = 真的}

API 端点	加密操作或算法	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
GenerateCardData	• DYNAMIC_CARD_VERIFICATION_CODE	TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS	• TDES_2KEY	{ DeriveKey = 真的}
GenerateCardData	• DYNAMIC_CARD_VERIFICATION_VALUE	TR31_E6_EMV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = 真的}

VerifyCardData

加密操作或算法	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
• AMEX_CARD_SECURITY_CODE_VERSION_1 • AMEX_CARD_SECURITY_CODE_VERSION_2	TR31_C0_CARD_VERIFICATION_KEY	• TDES_2KEY • TDES_3KEY	{ Generate = true }, { Generate = true, Verify = true }
• CARD_VERIFICATION_VALUE_1 • CARD_VERIFICATION_VALUE_2	TR31_C0_CARD_VERIFICATION_KEY	• TDES_2KEY	{ Generate = true }, { Generate = true, Verify = true }
• CARDHOLDER_AUTHENT	TR31_E6_EMV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = 真的}

加密操作或算法	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
ICATION_V ERIFICATI ON_VALUE			
• DYNAMIC_C ARD_VERIF ICATION_CODE	TR31_E4_E MV_MKEY_D YNAMIC_NUMBERS	• TDES_2KEY	{ DeriveKey = 真的}
• DYNAMIC_C ARD_VERIF ICATION_VALUE	TR31_E6_E MV_MKEY_OTHER	• TDES_2KEY	{ DeriveKey = 真的}

GeneratePinData（用于 VISA/ABA 计划）

`VISA_PIN or VISA_PIN_VERIFICATION_VALUE`

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
KMS 加密密钥	TR31_P0_P IN_ENCRYP TION_KEY	• TDES_2KEY • TDES_3KEY	• { Encrypt = true, Wrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
PIN 生成密钥	TR31_V2_V ISA_PIN_VERIFICATI ON_KEY	• TDES_2KEY • TDES_3KEY	• { Generate = true } • { Generate = true, Verify = true }

GeneratePinData（适用于 **IBM3624**）

IBM3624_PIN_OFFSET,IBM3624_NATURAL_PIN,IBM3624_RANDOM_PIN,
IBM3624_PIN_FROM_OFFSET)

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
KMS 加密密钥	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY	适用于 IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET • { Encrypt = true, Wrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的} 适用于 IBM3624_PIN_OFFSET • { Encrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
PIN 生成密钥	TR31_V1_IBM3624_PIN_VERIFICATION_KEY	• TDES_2KEY • TDES_3KEY	• { Generate = true } • { Generate = true, Verify = true }

VerifyPinData（用于 VISA/ABA 计划）

`VISA_PIN`

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
KMS 加密密钥	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY	• { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
PIN 生成密钥	TR31_V2_VISA_PIN_VERIFICATION_KEY	• TDES_2KEY • TDES_3KEY	• { Verify = true } • { Generate = true, Verify = true }

VerifyPinData（适用于 **IBM3624**）

`IBM3624_PIN_OFFSET,IBM3624_NATURAL_PIN,IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET)`

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
KMS 加密密钥	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY	适用于 IBM3624_NATURAL_PIN, IBM3624_RANDOM_PIN, IBM3624_PIN_FROM_OFFSET • { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
PIN 验证密钥	TR31_V1_IBM3624_PIN_VERIFICATION_KEY	• TDES_2KEY • TDES_3KEY	• { Verify = true } • { Generate = true, Verify = true }

解密数据

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
DUKPT	TR31_B0_BASE_DERIVATION_KEY	• TDES_2KEY • AES_128 • AES_192 • AES_256	• { DeriveKey = 真的} • { NoRestrictions = 真的}

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
EMV	TR31_E1_EMV_MKEY_CONFIDENTIALITY TR31_E6_EMV_MKEY_OTHER	• TDES_2KEY	• { DeriveKey = 真的}
RSA	TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION	• RSA_2048 • RSA_3072 • RSA_4096	• { Decrypt = true, Unwrap=true} • {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true}
对称密钥	TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• {Decrypt = true, Unwrap=true} • {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true} • { NoRestrictions = 真的}

加密数据

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
DUKPT	TR31_B0_BASE_DERIVATION_KEY	• TDES_2KEY • AES_128 • AES_192 • AES_256	• { DeriveKey = 真的} • { NoRestrictions = 真的}

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
EMV	TR31_E1_EMV_MKEY_CONFIDENTIALITY TR31_E6_EMV_MKEY_OTHER	• TDES_2KEY	• { DeriveKey = 真的}
RSA	TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION	• RSA_2048 • RSA_3072 • RSA_4096	• { Encrypt = true, Wrap=true} • {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true}
对称密钥	TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• {Encrypt = true, Wrap=true} • {Encrypt=true, Wrap=true,Decrypt = true, Unwrap=true} • { NoRestrictions = 真的}

转换 PIN 数据

方向	密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
入站数据来源	DUKPT	TR31_B0_BASE_DERIVATION_KEY	• TDES_2KEY • AES_128 • AES_192 • AES_256	• { DeriveKey = 真的} • { NoRestrictions = 真的}

方向	密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
入站数据来源	非 DUKPT（PEK、AWK、IWK 等）	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
出站数据目标	DUKPT	TR31_B0_BASE_DERIVATION_KEY	• TDES_2KEY • AES_128 • AES_192 • AES_256	• { DeriveKey = 真的} • { NoRestrictions = 真的}
出站数据目标	非 DUKPT（PEK、IWK、AWK 等）	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Encrypt = true, Wrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}

Generate/Verify MAC

MAC 密钥用于创建数据的加密哈希值 message/body 。不建议使用有限的密钥使用模式创建密钥，因为您将无法执行匹配操作。但是，如果 import/export 另一个系统打算执行操作对的另一半，则只能使用一个操作来密钥。

允许的密钥用法	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
MAC 密钥	TR31_M1_ISO_9797_1_MAC_KEY	• TDES_2KEY • TDES_3KEY	• { Generate = true } • { Generate = true, Verify = true } • { Verify = true } • { Generate = true }
MAC 密钥（零售 MAC）	TR31_M1_ISO_9797_3_MAC_KEY	• TDES_2KEY • TDES_3KEY	• { Generate = true } • { Generate = true, Verify = true } • { Verify = true } • { Generate = true }
MAC 密钥 (CMAC)	TR31_M6_ISO_9797_5_CMAC_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Generate = true } • { Generate = true, Verify = true } • { Verify = true } • { Generate = true }
MAC 密钥 (HMAC)	TR31_M7_HMAC_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Generate = true } • { Generate = true, Verify = true } • { Verify = true }
MAC 密钥 (AS2805)	TR31_M0_ISO_16609_MAC_KEY	• TDES_2KEY • TDES_3KEY	• { Generate = true } • { Generate = true, Verify = true } • { Verify = true }

GenerateMacEmvPinChange

GenerateMacEmvPinChange 将 MAC 生成和 PIN 加密相结合，用于 EMV 离线 PIN 更改操作。此操作需要两种不同的密钥类型：用于生成 MAC 的完整性密钥和用于 PIN 加密的机密密钥。

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
安全消息完整性密钥	TR31_E2_EMV_MKEY_INTEGRITY	• TDES_2KEY	• { NoRestrictions = 真的}
安全消息保密密钥	TR31_E1_EMV_MKEY_CONFIDENTIALITY	• TDES_2KEY	• { DeriveKey = 真的}
当前 PIN PEK（PIN 加密密钥）	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
新 PIN PEK（PIN 加密密钥）	TR31_P0_PIN_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• { Decrypt = true, Unwrap = true } • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true } • { NoRestrictions = 真的}
ARQC 密钥	TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS	• TDES_2KEY	• { DeriveKey = 真的}

密钥类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
Note 仅适用于签证和美国运通衍生计划。			

VerifyAuthRequestCryptogram

允许的密钥用法	EMV 选项	允许的密钥算法	允许的密钥使用模式组合
• 选项 A • 选项 B	TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS	• TDES_2KEY	• { DeriveKey = 真的}

Import/Export 钥匙

操作类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
TR-31 包装钥匙	TR31_K1_KEY_BLOCK_PROTECTION_KEY TR31_K0_KEY_ENCRYPTION_KEY	• TDES_2KEY • TDES_3KEY • AES_128 • AES_192 • AES_256	• {加密 = 真，换行 = 真} （仅限导出） • {Decrypt = true，Unwrap = true} （仅限导入） • { Encrypt = true, Decrypt = true, Wrap = true, Unwrap = true }

操作类型	允许的密钥用法	允许的密钥算法	允许的密钥使用模式组合
导入可信 CA	TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE	• RSA_2048 • RSA_3072 • RSA_4096	• { Verify = true }
导入用于非对称加密的公钥证书	TR31_D1_ASYMMETRIC_KEY_FOR_DATA_ENCRYPTION	• RSA_2048 • RSA_3072 • RSA_4096	• {加密=真，wrap=真}
用于密钥协议算法（如 ECDH）的密钥	TR31_K3_ASYMMETRIC_KEY_FOR_KEY_ACTREMENT	• ECC_NIST_P256 • ECC_NIST_P384 • ECC_NIST_P521	• { DeriveKey = 真的}

未使用的密钥类型

AWS 支付密码学目前未使用以下密钥类型

- TR31_P1_PIN_PIN_GENERATION_KEY

常见使用案例

AWS 支付密码学支持许多典型的支付加密操作。以下主题可作为如何将这些操作用于典型的常见用例的指南。如需查看所有命令的列表，请查看 AWS 支付密码学 API。

主题

- 发行人和发行人处理器
- 收购和付款促进者

发行人和发行人处理器

发行人用例通常由几个部分组成。本部分按功能（例如使用引脚）进行组织。在生产系统中，密钥通常限定在给定的卡箱内，并在存储箱设置期间创建，而不是按此处所示的内联方式创建。

主题

- 常规函数
- 网络特定功能

常规函数

主题

- 生成随机引脚和关联的 PVV，然后验证值
- 为给定卡生成或验证 CVV
- 为特定卡生成或验证 CVV2
- 为特定卡生成或验证 ICvV
- 验证 EMV ARQC 并生成 ARPC
- 生成并验证 EMV MAC
- 生成 EMV MAC 以更改 PIN

生成随机引脚和关联的 PVV，然后验证值

主题

- 创建密钥
- 生成随机密码，生成 PVV 并返回加密的 PIN 和 PVV
- 使用 PVV 方法验证加密的 PIN
- 用户管理的 PIN

创建密钥

要生成随机 PIN 和 PVV，您需要两个密钥，一个用于生成 PVV 的 PIN 验证密钥 (PVK) 和一个用于加密该 PIN 的 PIN 加密密钥。密码本身是在服务内部安全地随机生成的，与任何一个密钥在加密方面都不相关。

PGK 必须是基于 PVV 算法本身的算法 TDES_2KEY 的密钥。PEK 可以是 TDES_2KEY、TDES_3KEY 或 AES_128。在这种情况下，由于 PEK 是供系统内部使用的，因此 AES_128 将是一个不错的选择。如果使用 PEK 与其他系统（例如卡网络、收单机构、ATM）进行交换或作为迁移的一部分进行移动，则出于兼容性原因，TDES_2KEY 可能是更合适的选择。

创建 PEK

```
$ aws payment-cryptography create-key \
               --exportable
               --key-attributes
 KeyAlgorithm=AES_128,KeyUsage=TR31_P0_PIN_ENCRYPTION_KEY,\
                   KeyClass=SYMMETRIC_KEY,\
                   KeyModesOfUse='{Encrypt=true,Decrypt=true,Wrap=true,Unwrap=true}' --
tags='[{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
               "Key": {
                   "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
                   "KeyAttributes": {
                       "KeyUsage": "TR31_P0_PIN_ENCRYPTION_KEY",
                       "KeyClass": "SYMMETRIC_KEY",
                       "KeyAlgorithm": "AES_128",
                       "KeyModesOfUse": {
                           "Encrypt": false,
                           "Decrypt": false,
```

```
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "7CC9E2",
                "KeyCheckValueAlgorithm": "CMAC",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/ivi5ksfsuplneuyt 您需要在下一步中执行该操作。

创建 PVK

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_V2_VISA_PIN_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyMo
  --tags='[{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
                "Key": {
                    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza",
                    "KeyAttributes": {
                        "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY",
                        "KeyClass": "SYMMETRIC_KEY",
                        "KeyAlgorithm": "TDES_2KEY",
                        "KeyModesOfUse": {
                            "Encrypt": false,
                            "Decrypt": false,
```

```
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "51A200",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/ov6icy4ryas4zcza 您需要在下一步中执行该操作。

生成随机密码，生成 PVV 并返回加密的 PIN 和 PVV

Example

在此示例中，我们将生成一个新的（随机）4 位数 pin，其中输出将是加密的 PIN block (PinData.PinBlock) 和PVV（pinData.VerificationValue）。密钥输入是 PANPin Verification Key（也称为引脚生成密钥）、Pin Encryption Key和 PIN 块格式。

此命令要求密钥的类型为TR31_V2_VISA_PIN_VERIFICATION_KEY。

```
$ aws payment-cryptography-data generate-pin-data --generation-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2 --encryption-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt
 --primary-account-number 171234567890123 --pin-block-format ISO_FORMAT_0 --generation-
attributes VisaPin={PinVerificationKeyIndex=1}
```

```
{
            "GenerationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
```

```
            "GenerationKeyCheckValue": "7F2363",
            "EncryptionKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt",
            "EncryptionKeyCheckValue": "7CC9E2",
            "EncryptedPinBlock": "AC17DC148BDA645E",
            "PinData": {
                "VerificationValue": "5507"
            }
        }
```

使用 PVV 方法验证加密的 PIN

Example

在此示例中，我们将验证给定 PAN 的 PIN。PIN 通常由持卡人或用户在交易期间提供以进行验证，并与存档的值进行比较（持卡人的输入作为来自终端或其他上游提供商的加密值提供）。为了验证此输入，还将在运行时提供以下值：加密的 PIN、用于加密输入 PIN 的密钥（通常称为 IWK）PAN以及用于验证的值（a PVV 或PIN offset）。

如果 AWS 支付密码学能够验证密码， http/200 则返回。如果 PIN 未通过验证，它将返回 http/400。

```
$ aws payment-cryptography-data verify-pin-data --verification-key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/37y2tsl45p5zjbh2 --encryption-
key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt
 --primary-account-number 171234567890123 --pin-block-format ISO_FORMAT_0  --
verification-attributes VisaPin="{PinVerificationKeyIndex=1,VerificationValue=5507}" --
encrypted-pin-block AC17DC148BDA645E
```

```
{
        "VerificationKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/37y2tsl45p5zjbh2",
        "VerificationKeyCheckValue": "7F2363",
        "EncryptionKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
        "EncryptionKeyCheckValue": "7CC9E2",
}
```

用户管理的 PIN

主题

- 概述
- 创建密钥
- 设置用户可选择的 PIN
- 重置 PIN（绿色 PIN）
- 显示 PIN

概述

AWS Payment Cryptography 支持用户管理（客户管理）的 PIN 流，其中通过客户后端在持卡人设备和服务之间安全地交换 PIN。该设备和服务使用椭圆曲线 Diffie-Hellman (ECDH) 密钥协议在两侧生成一个共享的对称密钥，该密钥保护 ISO 格式的 4 PIN 块。???由于密钥是派生而非交换的，因此清除 PIN 和清除 PIN 块永远不会传输到设备以外的地方。

这些流程涉及三方。持卡人设备生成其 EC 密钥对，获取 ECDH 密 AES-128 钥，加密或解密 ISO 格式 4 的 PIN 块，并保存清晰的 PIN；它没有 AWS 证书。客户后端（您的发卡机构或处理器服务器）持有 AWS 证书、调用 AWS 支付加密、签署设备证书并存储 PVV。 AWS 支付密码学执行加密操作。设备仅与客户后端通信，客户后端将所有调用 AWS 支付密码学。

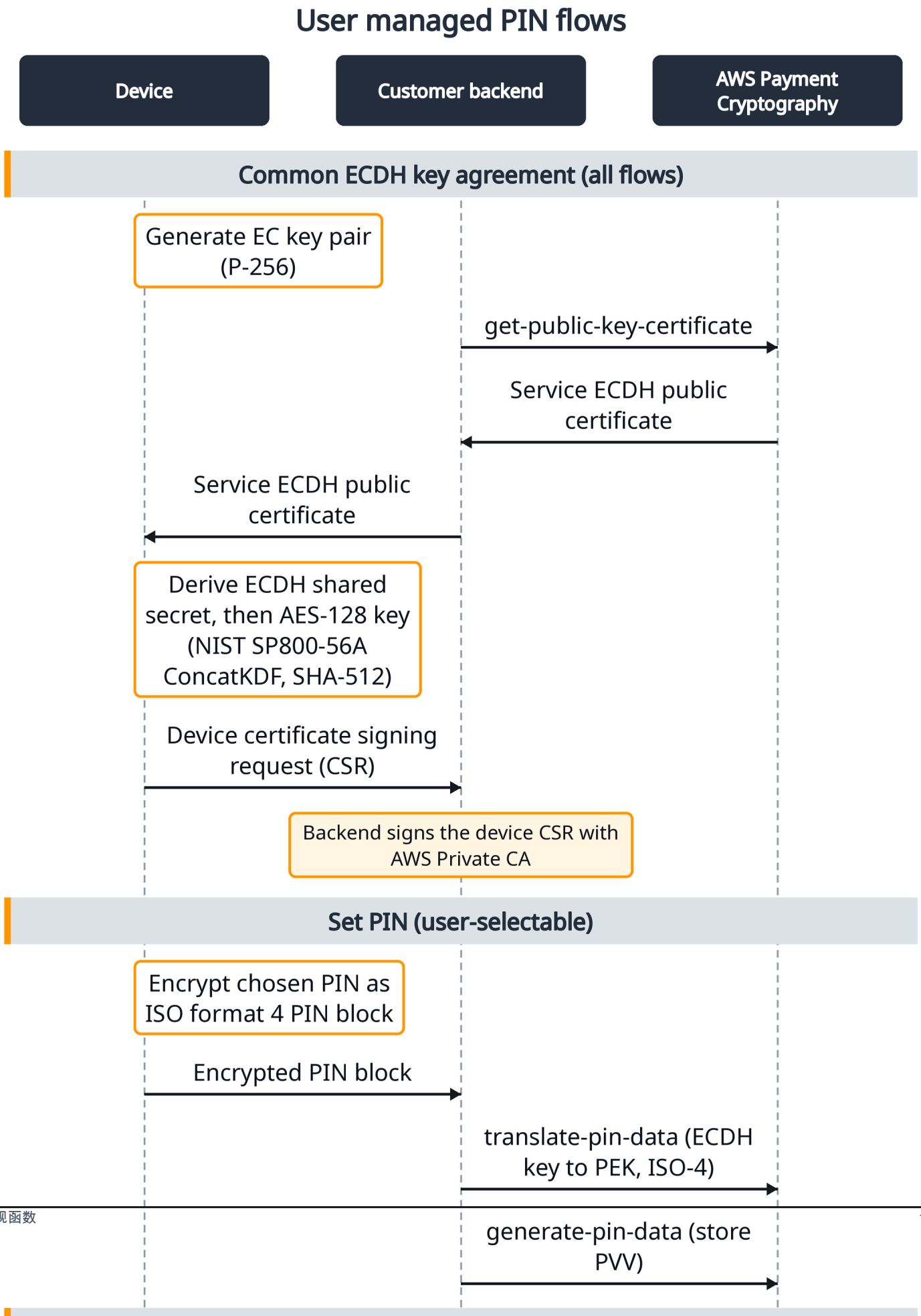
本主题涵盖三个流程：

- User-selectable PIN（设置 PIN）-持卡人在设备上选择自己的 PIN。
- 绿色 PIN（重置 PIN）-该服务会生成一个随机 PIN，该密码会传送给持卡人，例如在首次签发或被遗忘的 PIN 重置期间。
- 显示 PIN -对于狭义的用例，现有 PIN 块会返回到授权设备。

所有三个流程共享相同的密钥设置和相同的 ECDH 设备协议机制，如创建密钥中所述。有关密钥协议本身的详细信息，请参阅使用非对称技术 (ECDH) 导入密钥。

> **Note**
>
> 这些流程仅在设备上处理清晰的 PIN 和 PIN 块，不会在传输到 AWS 支付加密系统时进行处理。在部署每个流程之前，请与安全团队一起仔细检查每个流程。

下图显示了用户管理的 PIN 流。


User managed PIN flows
Device
Customer backend
AWS Payment Cryptography
Common ECDH key agreement (all flows)
Generate EC key pair (P-256)
get-public-key-certificate
Service ECDH public certificate
Service ECDH public certificate
Derive ECDH shared secret, then AES-128 key (NIST SP800-56A ConcatKDF, SHA-512)
Device certificate signing request (CSR)
Backend signs the device CSR with AWS Private CA
Set PIN (user-selectable)
Encrypt chosen PIN as ISO format 4 PIN block
Encrypted PIN block
translate-pin-data (ECDH key to PEK, ISO-4)
generate-pin-data (store PVV)

有关使用适用于 Python 的 AWS 开发工具包的这些流程的完整、可运行的示例，请参阅示例存储库中的 ECDH PIN 设置和显示流程示 AWS 例。 GitHub

> **Note**
>
> 典型的流程包括执行 ECDH 密钥协议并加密或解密 ISO 格式 4 PIN 区块的持卡人设备。如果您已经通过其他方式保护设备和客户后端之间的 PIN，则可以直接在客户后端和 AWS 支付加密之间使用这些流量的修改版本。在该变体中，客户后端代替设备执行 ECDH 密钥协议和 PIN 块操作。

创建密钥

所有三个流程都使用相同的密钥集：用于与设备签订密钥协议的 ECDH 协议密钥、用于生成 P VV 的 PIN Generation/Verification 密钥 (PGK) 以及将 P IN 块转换成用于后端存储的 PIN 加密密钥 (PEK)。您还可以导入设备证书颁发机构 (CA) 的公钥，这样 P AWS ayment Cryptography 就会信任翻译期间提供的设备证书。

创建 ECDH 协议密钥。此密钥用于派生与设备共享的对称密钥。

```
$ aws payment-cryptography create-key \
    --exportable \
    --key-attributes KeyAlgorithm=ECC_NIST_P256,KeyClass=ASYMMETRIC_KEY_PAIR,\
        KeyUsage=TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT,\
        KeyModesOfUse='{DeriveKey=true}'
```

该响应回显了请求参数，并包含 ECDH 密钥的 ARN。

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxx",
        "KeyAttributes": {
            "KeyUsage": "TR31_K3_ASYMMETRIC_KEY_FOR_KEY_AGREEMENT",
            "KeyClass": "ASYMMETRIC_KEY_PAIR",
            "KeyAlgorithm": "ECC_NIST_P256",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
```

```
                "Generate": false,
                "Sign": false,
                "Verify": false,
                "DeriveKey": true,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "1A2B3C",
        "KeyCheckValueAlgorithm": "CMAC",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

创建用于生成 PVV 的密码 Generation/Verification 密钥 (PGK)。

```
$ aws payment-cryptography create-key \
    --exportable \
    --key-attributes KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_V2_VISA_PIN_VERIFICATION_KEY,
\
        KeyClass=SYMMETRIC_KEY,\
        KeyModesOfUse='{Generate=true,Verify=true}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
        "KeyAttributes": {
            "KeyUsage": "TR31_V2_VISA_PIN_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
```

```
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "51A200",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

创建 PIN 加密密钥 (PEK)，将设备 PIN 块转换为后端存储。

```
$ aws payment-cryptography create-key \
    --exportable \
    --key-attributes KeyAlgorithm=AES_128,KeyUsage=TR31_P0_PIN_ENCRYPTION_KEY,\
        KeyClass=SYMMETRIC_KEY,\
        KeyModesOfUse='{Encrypt=true,Decrypt=true,Wrap=true,Unwrap=true}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
        "KeyAttributes": {
            "KeyUsage": "TR31_P0_PIN_ENCRYPTION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "AES_128",
            "KeyModesOfUse": {
                "Encrypt": true,
                "Decrypt": true,
                "Wrap": true,
                "Unwrap": true,
                "Generate": false,
                "Sign": false,
                "Verify": false,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
```

```
        "KeyCheckValue": "7CC9E2",
        "KeyCheckValueAlgorithm": "CMAC",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

导入设备 CA 的公钥，这样 P AWS ayment Cryptography 就可以验证在以后的翻译调用中提供的设备证书。替换<base64-CA-cert>为 base64 编码的 CA 证书。

```
$ aws payment-cryptography import-key \
    --key-material='{"RootCertificatePublicKey":{"KeyAttributes":
{"KeyAlgorithm":"ECC_NIST_P256","KeyClass":"PUBLIC_KEY","KeyModesOfUse":
{"Verify":true},"KeyUsage":"TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"},"PublicKeyCertificat
CA-cert>"}}'
```

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
yyyyyyyyyyyyyyyy",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "ECC_NIST_P256",
            "KeyModesOfUse": {
                "Verify": true
            }
        },
        "KeyCheckValue": "4D5E6F",
        "KeyCheckValueAlgorithm": "CMAC",
        "Enabled": true,
        "Exportable": false,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "EXTERNAL",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

记下每件退货情况KeyArn。以下示例使用以下文字占位符：

- ECDH 协议密钥：arn:aws:payment-cryptography:us-east-1:111122223333:key/xxxxxxxxxxxxxxxx
- 导入的 CA 公钥：arn:aws:payment-cryptography:us-east-1:111122223333:key/yyyyyyyyyyyyyyyy
- PGK：arn:aws:payment-cryptography:us-east-2:111122223333:key/ov6icy4ryas4zcza
- 峰值：arn:aws:payment-cryptography:us-east-2:111122223333:key/ivi5ksfsuplneuyt

在持卡人设备（通常是移动电话或类似设备）上，在设备应用程序代码中为每个 “设置 PIN”、“重置 PIN” 和 “显示 PIN” 流程运行以下步骤：在曲线上生成 EC 密钥对 P-256；从客户后端获取服务 ECDH 公共证书；派生 ECDH 共享密钥，然后使用该SharedInformation值获取 16 字节密 AES-128 钥）；使用派生的 ISO 格式 4 P IN 块进行加密或解密钥匙。NIST SP800-56A ConcatKDF (SHA-5 该设备没有 AWS 证书，也不会直接调用 Paym AWS ent Cryptography；它与客户后端交换这些数据，后者进行服务调用。

由于该设备通常是电话，因此其 EC 密钥对和证书在首次使用时会进行配置，之后会定期刷新。设备生成 EC 密钥对和证书签名请求 (CSR)，客户后端使用 AWS 私有 CA 或类似的 CA 对该 CSR 进行签名。客户后端在其 translate-pin-data 调用中使用生成的签名设备证书，P AWS ayment Cryptography 会根据您导入的 CA 公钥对其进行验证。

客户后端获取服务 ECDH 公共证书并将其提供给设备。该证书在KeyCertificate字段中以 base64 编码的 PEM 形式返回，然后以 base64 解码为 PEM 文件。设备在派生共享密钥时使用此证书作为服务公钥。

```
$ aws payment-cryptography get-public-key-certificate \
    --key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxxx
```

> **Note**
>
> SharedInformation是在设备上生成的 32 个随机字节，以十六进制字符串的形式通过客户后端传递给 AWS 支付加密。它必须与 KDF 中使用的设备值相匹配，否则派生密钥将不一致。

设置用户可选择的 PIN

在 “设置 PIN” 流程中，持卡人在设备上选择自己的 PIN（例如1234）。设备获取 ECDH 共享密钥，将选定的 PIN 加密为该密钥下的 ISO 格式 4 个 PIN 块，并将加密的 PIN 块、其证书签名请求 (CSR) 和共享信息发送到客户后端。客户后端对 CSR 进行签名，然后执行两次数据平面调用。此示例使用示例PAN171234567890123。显示的 PIN 区块、证书和共享信息值是虚构的占位符。

首先，将 ECDH-protected PIN 块转换为服务器 PEK。传入密钥是 ECDH 协议密钥，包裹着 Diffie-Hellman 参数和签名的设备证书；传出密钥是 PEK。

```
$ aws payment-cryptography-data translate-pin-data \
    --encrypted-pin-block A1B2C3D4E5F60718 \
    --incoming-key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxxx \
    --incoming-wrapped-key
 'WrappedKeyMaterial={DiffieHellmanSymmetricKey={CertificateAuthorityPublicKeyIdentifier=arn:aw
cryptography:us-east-1:111122223333:key/
yyyyyyyyyyyyyyyy,KeyAlgorithm=AES_128,KeyDerivationFunction=NIST_SP800,KeyDerivationHashAlgorit
signed-device-
cert>,SharedInformation=0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef}}'
 \
    --incoming-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}' \
    --outgoing-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt \
    --outgoing-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}'
```

该响应包含在 PEK 下以 ISO 格式 4 重新加密的 PIN 块。

```
{
    "PinBlock": "F7E6D5C4B3A29180",
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
    "KeyCheckValue": "7CC9E2"
}
```

接下来，为 P PEK-encrypted IN 块生成 PVV。PGK 生成 PVV，PEK 会识别所提供的 PIN 区块所使用的密钥。

```
$ aws payment-cryptography-data generate-pin-data \
```

```
    --generation-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza \
    --encryption-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt \
    --pin-block-format ISO_FORMAT_4 \
    --primary-account-number 171234567890123 \
    --generation-attributes
 'VisaPinVerificationValue={PinVerificationKeyIndex=1,EncryptedPinBlock=F7E6D5C4B3A29180}'
```

响应包含 PVV 输入PinData.VerificationValue和加密的 PIN 块。将 PVV 存储在客户后端，以便稍后验证 PIN。要进行验证，请参见生成随机引脚和关联的 PVV，然后验证该值。

```
{
    "GenerationKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
    "GenerationKeyCheckValue": "51A200",
    "EncryptionKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
    "EncryptionKeyCheckValue": "7CC9E2",
    "EncryptedPinBlock": "F7E6D5C4B3A29180",
    "PinData": {
        "VerificationValue": "5507"
    }
}
```

有关更多信息，请参阅 API 参考指南GeneratePinData中的TranslatePinData和。

重置 PIN（绿色 PIN）

在重置个人识别码（绿色 PIN）流程中，P AWS ayment Cryptography 会生成一个随机 PIN，客户后端将其返回到设备以显示给持卡人，例如在首次签发或忘记密码重置期间。该设备执行创建密钥中描述的相同的 ECDH 设置，客户后端代表其进行数据平面呼叫。此示例使用示例 PAN171234567890123；PIN 块和共享信息值是虚构的占位符。

首先，生成一个随机 PIN 及其 PVV，对 PEK 下的 PIN 块进行加密。

```
$ aws payment-cryptography-data generate-pin-data \
    --generation-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ov6icy4ryas4zcza \
    --encryption-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/ivi5ksfsuplneuyt \
    --pin-block-format ISO_FORMAT_4 \
    --primary-account-number 171234567890123 \
```

```
    --generation-attributes 'VisaPin={PinVerificationKeyIndex=1}'
```

响应包含 PVV 输入PinData.VerificationValue和 P PEK-encrypted IN 块（ISO 格式 4）。将 PVV 存储在客户后端以供日后验证。

```
{
    "GenerationKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
    "GenerationKeyCheckValue": "51A200",
    "EncryptionKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
    "EncryptionKeyCheckValue": "7CC9E2",
    "EncryptedPinBlock": "F7E6D5C4B3A29180",
    "PinData": {
        "VerificationValue": "5507"
    }
}
```

接下来，将 PEK-encrypted PIN 块（ISO 格式 4）转换为设备 ECDH-derived 密钥（ISO 格式 4），这样只有请求的设备才能读取。

```
$ aws payment-cryptography-data translate-pin-data \
    --encrypted-pin-block F7E6D5C4B3A29180 \
    --incoming-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt \
    --incoming-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}' \
    --outgoing-key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxx \
    --outgoing-wrapped-key
 'WrappedKeyMaterial={DiffieHellmanSymmetricKey={CertificateAuthorityPublicKeyIdentifier=arn:aw
cryptography:us-east-1:111122223333:key/
yyyyyyyyyyyyyyyy,KeyAlgorithm=AES_128,KeyDerivationFunction=NIST_SP800,KeyDerivationHashAlgorit
signed-device-
cert>,SharedInformation=0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef}}'
 \
    --outgoing-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}'
```

响应中包含在设备 ECDH-derived 密钥下以 ISO 格式 4 重新加密的 PIN 块。

```
{
```

```
    "PinBlock": "A1B2C3D4E5F60718",
    "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxxx",
    "KeyCheckValue": "1A2B3C"
}
```

客户后端将 ECDH-encrypted PIN 块返回给设备。设备使用其密 ECDH-derived 钥（在设备应用程序代码中）解密返回的 PIN 块，并向持卡人显示 PIN。由于密钥仅在该设备上派生，因此任何其他方都无法解密 PIN 块。

有关更多信息，请参阅 API 参考指南TranslatePinData中的GeneratePinData和。

显示 PIN

显示 PIN 流程会将现有的 PIN 块返回给持卡人设备，以便该设备可以显示或以其他方式显示该 PIN。设备执行创建密钥中描述的相同的 ECDH 设置，客户后端将存储的 PEK-encrypted PIN 块转换回 ECDH-derived 密钥并将其返回到设备上。此示例使用示例 PAN171234567890123；PIN 块和共享信息值是虚构的占位符。

> **Note**
>
> 考虑对持卡人进行身份验证，让他们设置新的 PIN，而不是透露现有的 PIN。一个明显的例外是在使用离线 PIN 的国家，例如欧洲，那里的 PIN 也存储在芯片卡上。在这种情况下，显示现有的 PIN 可能比重新编程卡更简单。

将存储的 PEK-encrypted PIN 块（ISO 格式 4）转换为设备 ECDH-derived 密钥（ISO 格式 4）。

```
$ aws payment-cryptography-data translate-pin-data \
    --encrypted-pin-block F7E6D5C4B3A29180 \
    --incoming-key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt \
    --incoming-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}' \
    --outgoing-key-identifier arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxxx \
    --outgoing-wrapped-key
 'WrappedKeyMaterial={DiffieHellmanSymmetricKey={CertificateAuthorityPublicKeyIdentifier=arn:aw
cryptography:us-east-1:111122223333:key/
yyyyyyyyyyyyyyyy,KeyAlgorithm=AES_128,KeyDerivationFunction=NIST_SP800,KeyDerivationHashAlgorit
signed-device-
```

```
cert>,SharedInformation=0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef}}'
 \
    --outgoing-translation-attributes
 'IsoFormat4={PrimaryAccountNumber=171234567890123}'
```

响应中包含在设备 ECDH-derived 密钥下以 ISO 格式 4 重新加密的 PIN 块。

```
{
    "PinBlock": "A1B2C3D4E5F60718",
    "KeyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
xxxxxxxxxxxxxxxx",
    "KeyCheckValue": "1A2B3C"
}
```

客户后端将 ECDH-encrypted PIN 块返回给设备。设备使用其密 ECDH-derived 钥（在设备应用程序代码中）解密返回的 PIN 块，以显示 PIN。由于密钥仅在该设备上派生，因此任何其他方都无法解密 PIN 块。

有关更多信息，请参阅 API 参考指南TranslatePinData中的。

为给定卡生成或验证 CVV

CVV 或 CVV1 是传统上嵌入在卡片磁条中的值。它与 CVV2 不同（持卡人可见，可用于在线购物）。

第一步是创建一个密钥。在本教程中，您将创建一个 CVK 双长度 3DES (2KEY TDES) 密钥。

> **Note**
>
> CVV、CVV2 和 ICvV 都使用相似的算法，即使不完全相同，但输入数据有所不同。所有密钥类型都使用相同的 TR31_C0_CARD_VERIFICATION_KEY，但建议为每种目的使用单独的密钥。可以使用别名 and/or 标签来区分它们，如下例所示。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVV"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
r52o3wbqxyf6qlqr",
                "KeyAttributes": {
                    "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "DE89F9",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```


记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/r52o3wbqxyf6qlqr 您需要在下一步中执行该操作。

生成 CVV

Example

在此示例中，我们将为给定的 PAN 生成 CVVPAN，其输入为、服务代码（由 ISO/IEC 7813 定义）为 121 和卡到期日期。

有关所有可用参数 CardVerificationValue1，请参阅 API 参考指南。

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
r52o3wbqxyf6qlqr --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=121}'
```

```
                {
                    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/r52o3wbqxyf6qlqr",
                    "KeyCheckValue": "DE89F9",
                    "ValidationData": "801"
                }
```

验证 CVV

Example

在此示例中，我们将使用输入的 CV K、服务代码 121 PAN、卡到期日期和交易期间提供的 CVV 来验证给定 PAN 的 CVV。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue1中的内容。

> **Note**
>
> CVV 不是用户输入的值（如 CVV2），但通常嵌入在磁条上。应考虑是否应始终在提供时进行验证。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/r52o3wbqxyf6qlqr
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=121} --validation-data 801
```

```
{
                  "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
r52o3wbqxyf6qlqr",
                  "KeyCheckValue": "DE89F9",
```

```
                    "ValidationData": "801"
}
```

为特定卡生成或验证 CVV2

CVV2 是传统上在卡背面提供的价值，用于在线购物。对于虚拟卡，它也可能显示在应用程序或屏幕上。在密码学上，它与 CVV1 相同，但服务代码值不同。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVV2"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
               "Key": {
                   "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
                   "KeyAttributes": {
                       "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                       "KeyClass": "SYMMETRIC_KEY",
                       "KeyAlgorithm": "TDES_2KEY",
                       "KeyModesOfUse": {
                           "Encrypt": false,
                           "Decrypt": false,
                           "Wrap": false,
                           "Unwrap": false,
                           "Generate": true,
                           "Sign": false,
                           "Verify": true,
                           "DeriveKey": false,
                           "NoRestrictions": false
                       }
                   },
                   "KeyCheckValue": "AEA5CD",
                   "KeyCheckValueAlgorithm": "ANSI_X9_24",
                   "Enabled": true,
                   "Exportable": true,
                   "KeyState": "CREATE_COMPLETE",
                   "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                   "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
```

```
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/7f7g4spf3xcklhzu 您需要在下一步中执行该操作。

生成 CVV2

Example

在此示例中，我们将为给定的 PAN 生成 CVV2，输入值为PAN和卡到期日期。

有关所有可用参数 CardVerificationValue2，请参阅 API 参考指南。

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/7f7g4spf3xcklhzu
 --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue2='{CardExpiryDate=1127}'
```

```
                    {
                        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
                        "KeyCheckValue": "AEA5CD",
                        "ValidationData": "321"
                    }
```

验证 CVV2

Example

在此示例中，我们将使用输入的 CV K、信用卡到期日期PAN和交易期间提供的 CVV 来验证给定 PAN 的 CVV2 以进行验证。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue2中的内容。

> **Note**
>
> CVV2 和其他输入是用户输入的值。因此，定期无法验证不一定是问题的征兆。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/7f7g4spf3xcklhzu
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue2='{CardExpiryDate=1127} --validation-data 321
```

```
{
                        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
                        "KeyCheckValue": "AEA5CD",
                        "ValidationData": "801"
   }
```

为特定卡生成或验证 ICvV

iCVV 使用与芯片卡相同的算法，CVV/CVV2 但是 iCVV 嵌入在芯片卡中。它的服务代码是 999。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"ICVV"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
               "Key": {
                   "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
c7dsi763r6s7lfp3",
                   "KeyAttributes": {
                       "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                       "KeyClass": "SYMMETRIC_KEY",
                       "KeyAlgorithm": "TDES_2KEY",
                       "KeyModesOfUse": {
                           "Encrypt": false,
                           "Decrypt": false,
                           "Wrap": false,
                           "Unwrap": false,
                           "Generate": true,
                           "Sign": false,
```

```
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "1201FB",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/c7dsi763r6s7lfp3 您需要在下一步中执行该操作。

生成 icVV

Example

在此示例中，我们将为给定的 PAN 生成 ICvVPAN，其输入为，服务代码（由 ISO/IEC 7813 定义）为 999，卡到期日期。

有关所有可用参数 CardVerificationValue1，请参阅 API 参考指南。

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
c7dsi763r6s7lfp3 --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=999}'
```

```
                    {
                        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/c7dsi763r6s7lfp3",
                        "KeyCheckValue": "1201FB",
                        "ValidationData": "532"
                    }
```

验证 ICvV

Example

为了进行验证，输入为 CVK PAN、服务代码 999、信用卡到期日期和交易期间提供的待验证的 ICvV。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue1中的内容。

> **Note**
>
> icvV 不是用户输入的值（如 CVV2），但通常嵌入在卡片上。 EMV/chip 应考虑是否应始终在提供时进行验证。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/c7dsi763r6s7lfp3
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue1='{CardExpiryDate=1127,ServiceCode=999} --validation-data 532
```

```
{
                        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/c7dsi763r6s7lfp3",
                        "KeyCheckValue": "1201FB",
                        "ValidationData": "532"
   }
```

验证 EMV ARQC 并生成 ARPC

ARQC（授权请求加密码）是由EMV（芯片）卡生成的加密码，用于验证交易细节以及授权卡的使用。它包含来自卡、终端和交易本身的数据。

在后端验证时，向 AWS 支付密码学提供相同的输入，在内部重新创建密码，并将其与交易中提供的值进行比较。从这个意义上讲，它类似于MAC。EMV 4.4 第 2 册定义了该函数的三个方面——用于生成一次性交易密钥的密钥派生方法（称为通用会话密钥——CSK）、最低负载和生成响应的方法（ARPC）。

个人信用卡方案可能会指定要合并的其他交易字段或这些字段的出现顺序。还存在其他（通常已弃用的）特定于方案的派生方案，本文档的其他地方对此进行了介绍。

有关更多信息，请参阅 API 指南VerifyCardValidationData中的。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVN18"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
                "Key": {
                    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
                    "KeyAttributes": {
                        "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS",
                        "KeyClass": "SYMMETRIC_KEY",
                        "KeyAlgorithm": "TDES_2KEY",
                        "KeyModesOfUse": {
                            "Encrypt": false,
                            "Decrypt": false,
                            "Wrap": false,
                            "Unwrap": false,
                            "Generate": false,
                            "Sign": false,
                            "Verify": false,
                            "DeriveKey": true,
                            "NoRestrictions": false
                        }
                    },
                    "KeyCheckValue": "08D7B4",
                    "KeyCheckValueAlgorithm": "ANSI_X9_24",
                    "Enabled": true,
                    "Exportable": true,
                    "KeyState": "CREATE_COMPLETE",
                    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                    "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
                    "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
                }
            }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/pw3s6nl62t5ushfk 您需要在下一步中执行该操作。

生成 ARQC

ARQC 仅由 EMV 卡生成。因此， AWS 支付密码学没有生成这种有效载荷的设施。出于测试目的，在线提供了许多库，这些库可以生成适当的有效负载以及通常由各种方案提供的已知值。

验证 ARQC

Example

如果 AWS 支付密码学能够验证 ARQC， http/200 则返回。在 ARQC 通过验证后，可以选择提供 ARPC（响应）并将其包含在响应中。

```
$ aws payment-cryptography-data verify-auth-request-cryptogram
 --auth-request-cryptogram 61EDCC708B4C97B4 --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk
 --major-key-derivation-mode EMV_OPTION_A --transaction-data
 00000000170000000000000008400080008000084016051700000000093800000B1F220103000000000000000000000
 --session-key-derivation-attributes='{"EmvCommon":
{"ApplicationTransactionCounter":"000B",
 "PanSequenceNumber":"01","PrimaryAccountNumber":"9137631040001422"}}' --auth-response-
attributes='{"ArpcMethod2":{"CardStatusUpdate":"12345678"}}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4",
    "AuthResponseValue":"2263AC85"
}
```

生成并验证 EMV MAC

EMV MAC 是使用 EMV 派生密钥的输入，然后对生成的数据执行 ISO9797-3 （零售）MAC。EMV MAC 通常用于向 EMV 卡发送命令，例如解锁脚本。

> **Note**
>
> AWS 支付密码学不验证脚本的内容。有关要包括的特定命令的详细信息，请查阅您的方案或卡片手册。

有关更多信息，请参阅 API 指南MacAlgorithmEmv中的。

主题

- 创建密钥
- 生成 EMV MAC

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E2_EMV_MKEY_INTEGRITY,KeyClass=SYMMETRIC_KEY,KeyModesOfUs
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVN18"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
        "Key": {
            "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
            "KeyAttributes": {
                "KeyUsage": "TR31_E2_EMV_MKEY_INTEGRITY",
                "KeyClass": "SYMMETRIC_KEY",
                "KeyAlgorithm": "TDES_2KEY",
                "KeyModesOfUse": {
                    "Encrypt": false,
                    "Decrypt": false,
                    "Wrap": false,
                    "Unwrap": false,
                    "Generate": false,
                    "Sign": false,
                    "Verify": false,
                    "DeriveKey": true,
                    "NoRestrictions": false
                }
            },
            "KeyCheckValue": "08D7B4",
            "KeyCheckValueAlgorithm": "ANSI_X9_24",
            "Enabled": true,
            "Exportable": true,
            "KeyState": "CREATE_COMPLETE",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
            "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
            "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
        }
```

```
    }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/pw3s6nl62t5ushfk 您需要在下一步中执行该操作。

生成 EMV MAC

典型的流程是，后端进程将生成 EMV 脚本（例如卡片解除封锁），使用此命令对其进行签名（它派生特定于一张特定卡的一次性密钥），然后返回 MAC。然后将命令 + MAC 发送到要应用的卡。向银行卡发送命令不在 AWS 支付密码学的范围之内。

> **Note**
>
> 此命令适用于未发送加密数据（例如 PIN）时的命令。EMV Encrypt 可以与此命令结合使用，以便在调用此命令之前将加密数据附加到发行者脚本中

消息数据

消息数据包括 APDU 标头和命令。尽管这可能因实现情况而异，但此示例是解除封锁（84 24 00 00 08）的APDU标头，其次是ATC（0007），然后是前一笔交易（999E57FD0F47CACE）的ARQC。该服务不验证该字段的内容。

会话密钥派生模式

此字段定义会话密钥的生成方式。EMV_COMMON_SESSION_KEY 通常用于新的实现，而 EMV2000 | AMEX | MASTERCARD_SESSION_KEY | VISA 也可以使用。

MajorKeyDerivationMode

EMV 定义模式 A、B 或 C 模式 A 是最常见的， AWS 支付加密目前支持模式 A 或模式 B。

PAN

账号，通常在芯片字段 5A 或 ISO8583 字段 2 中可用，但也可以从银行卡系统中检索。

PSN

卡片序列号。如果未使用，请输入 00。

SessionKeyDerivationValue

这是每个会话的派生数据。根据推导方案，它可以是字段 9F26 中的最后一个 ARQC (ApplicationCryptogram)，也可以是 9F36 中的最后一个 ATC。

Padding

填充会自动应用并使用 ISO/IEC 9797-1 填充方法 2。

Example

```
$ aws payment-cryptography-data generate-mac --message-data
 842400000800079 97E57FD0F47CACE --key-identifier arn:aws:payment-
cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk --message-
data 8424000008999E57FD0F47CACE0007 --generation-attributes
 EmvMac="{MajorKeyDerivationMode=EMV_OPTION_A,PanSequenceNumber='00',PrimaryAccountNumber='2235
```

```
{
"KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk",
"KeyCheckValue": "08D7B4",
"Mac":"5652EEDF83EA0D84"
}
```

生成 EMV MAC 以更改 PIN

EMV PIN 更改结合了两种操作：为发行者脚本生成 MAC，以及加密新 PIN 以在 EMV 芯片卡上离线更改 PIN。仅在某些将 PIN 存储在芯片卡上的国家/地区才需要此命令（这在欧洲国家很常见）。这通常用于持卡人需要更改其 PIN，并且新 PIN 必须与 MAC 一起安全传输到卡中以验证命令的真实性。

> **Note**
>
> 如果你只需要向卡发送命令而不需要更改 PIN，可以考虑改用 ARPC CSU 或生成 EMV MAC 命令。

有关更多信息，请参阅 API 指南GenerateMacEmvPinChange中的。

生成 EMV MAC 和加密的 PIN 以更改 PIN

此操作需要两个密钥：用于生成 MAC 的 EMV 完整性密钥（KeyUsage：TR31_E2_EMV_MKEY_INTEGRITY）和用于 PIN 加密的 EMV 机密密钥（：TR31_E4_EMV_MKEY_PROBEMITY）。KeyUsage典型的流程是后端进程将生成 EMV PIN 更改脚本，其中包括发行者脚本的 MAC 和加密的新 PIN。然后，命令和加密的 PIN 将发送到卡上，以更新离线 PIN。向银行卡发送命令不在 AWS 支付密码学的范围之内。

消息数据

消息数据包括发行者脚本的 APDU 命令。该服务不验证该字段的内容。

全新加密 PIN 区块

将发送到卡上的新的加密 PIN 块。必须使用 PIN 加密密钥将其作为加密值提供。

新的 PIN PEK 标识符

用于在新 PIN 传递给此 API 之前对其进行加密的密钥。

安全消息完整性密钥

用于生成 MAC 的 EMV 完整性密钥（KeyUsage: TR31_E2_EMV_MKEY_INTEGRITY）。

安全消息保密密钥

用于 PIN 加密的 EMV 机密密钥（KeyUsage: TR31_E4_EMV_MKEY_PROBRICITY）。

MajorKeyDerivationMode

EMV 定义模式 A、B 或 C 模式 A 是最常见的， AWS 支付加密目前支持模式 A 或模式 B。

Mode

加密模式，通常是用于 PIN 更改操作的 CBC。

PAN

账号，通常在芯片字段 5A 或 ISO8583 字段 2 中可用，但也可以从银行卡系统中检索。

PanSequenceNumber

卡片序列号。如果未使用，请输入 00。

ApplicationCryptogram

这是每个会话的派生数据，通常是字段 9F26 的最后一个 ARQC。

PinBlockLengthPosition

指定 PIN 区块长度的编码位置。通常设置为 NONE。如果您不确定，请查看您的信用卡方案规格。

PinBlockPaddingType

指定 PIN 块的填充类型。通常设置为 NO_PADDING。如果您不确定，请查看您的信用卡方案规格。

Example

```
$ aws payment-cryptography-data generate-mac-emv-pin-change \
    --message-data 00A4040008A000000004101080D80500000001010A04000000000000 \
    --new-encrypted-pin-block 67FB27C75580EFE7 \
    --new-pin-pek-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt \
    --pin-block-format ISO_FORMAT_0 \
    --secure-messaging-confidentiality-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/tqv5yij6wtxx64pi \
    --secure-messaging-integrity-key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/pw3s6nl62t5ushfk \
    --derivation-method-attributes
 'EmvCommon={ApplicationCryptogram=1234567890123457,MajorKeyDerivationMode=EMV_OPTION_A,Mode=CB
```

```
{
    "SecureMessagingIntegrityKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/pw3s6nl62t5ushfk",
    "SecureMessagingIntegrityKeyCheckValue": "08D7B4",
    "SecureMessagingConfidentialityKeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/tqv5yij6wtxx64pi",
    "SecureMessagingConfidentialityKeyCheckValue": "C1EB8F",
    "Mac": "5652EEDF83EA0D84",
    "EncryptedPinBlock": "F1A2B3C4D5E6F7A8"
}
```

网络特定功能

主题

- 签证特定功能
- 万事达卡的特定功能
- 美国运通的特定功能
- JCB 特定函数
- UnionPay 特定函数

签证特定功能

主题

- ARQC- CVN18/CVN22
- ARQC-CVN10
- 3DS CAVV V7
- dcvV（动态卡验证值）-CVN17

ARQC- CVN18/CVN22

CVN18 和 CVN22 使用 CSK 方法进行密钥派生。这两种方法的确切交易数据各不相同-有关构造交易数据字段的详细信息，请参阅架构文档。

ARQC-CVN10

CVN10 是一种较早的 Visa EMV 交易方法，它使用每张卡密钥派生而不是会话（每笔交易）派生，还使用不同的有效载荷。有关有效载荷内容的信息，请联系该计划了解详情。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVN10"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
                "KeyAttributes": {
                    "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": false,
                        "Sign": false,
                        "Verify": false,
                        "DeriveKey": true,
```

```
                    "NoRestrictions": false
                }
            },
            "KeyCheckValue": "08D7B4",
            "KeyCheckValueAlgorithm": "ANSI_X9_24",
            "Enabled": true,
            "Exportable": true,
            "KeyState": "CREATE_COMPLETE",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
            "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
            "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
        }
    }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/pw3s6nl62t5ushfk 您需要在下一步中执行该操作。

验证 ARQC

Example

在此示例中，我们将验证使用 Visa CVN10 生成的 ARQC。

如果 AWS 支付密码学能够验证 ARQC， http/200 则返回。如果 arqc 未通过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-
cryptogram D791093C8A921769 \
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
  --major-key-derivation-mode EMV_OPTION_A \
  --transaction-data
 00000000170000000000000008400080008000084016051700000000093800000B03011203000000 \
  --session-key-derivation-attributes='{"Visa":{"PanSequenceNumber":"01" \
  ,"PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4"
  }
```

3DS CAVV V7

对于 Visa Secure (3DS) 交易，CAVV（持卡人身份验证验证值）由发卡机构访问控制服务器 (ACS) 生成。CAVV 是持卡人进行身份验证的证据，对于每笔身份验证交易都是唯一的，由收单方在授权消息中提供。CAVV v7 将有关交易的其他数据与批准绑定，包括商家名称、购买金额和购买日期等元素。这样，它实际上是交易负载的加密哈希。

从密码学上讲，CAVV V7使用了CVV算法，但所有输入都使用了。 changed/repurposed有关如何生成用于生成 CAVV V7 有效载荷的输入，请查阅相应的第三个 party/Visa 文档。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CAVV-V7"},
{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
                "KeyAttributes": {
                    "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "F3FB13",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
```

```
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/dnaeyrjgdjjtw6dk 您需要在下一步中执行该操作。

生成 CAVV V7

Example

在此示例中，我们将使用规范中指定的输入为给定交易生成 CAVV V7。请注意，对于此算法，字段可能会被重新利used/re用，因此不应假定字段标签与输入相匹配。

有关所有可用参数 CardVerificationValue1，请参阅 API 参考指南。

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk --primary-account-number=171234567890123 --generation-attributes
 CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=431}'
```

```
            {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
                "KeyCheckValue": "F3FB13",
                "ValidationData": "491"
            }
```

验证 CAVV V7

Example

为了进行验证，输入是 CVK、计算出的输入值和在进行验证的交易期间提供的 CAVV。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue1中的内容。

> **Note**
>
> CAVV 不是用户输入的值（如 CVV2），而是由发行者 ACS 计算的。应考虑是否应始终在提供时进行验证。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/dnaeyrjgdjjtw6dk
 --primary-account-number=171234567890123 --verification-attributes
 CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=431} --validation-data 491
```

```
{
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
                "KeyCheckValue": "F3FB13",
                "ValidationData": "491"
}
```

dcvV（动态卡验证值）-CVN17

dcVV（动态卡验证值）是一种用于非接触式 EMV 交易的 Visa-specific 动态密码。它被称为早期的EMV，它通过为每笔交易生成唯一的验证值来增强安全性。DCvV 使用的输入包括主账号 (PAN)、PAN 序列号 (PSN)、应用程序交易计数器 (ATC)、银行卡到期日和服务代码。它仍在某些地方使用，但主要被其他算法（如 CVN18）所取代。

有关所有可用参数 DynamicCardVerificationValue，请参阅 API 参考指南。

要DynamicCardVerificationValue映射到以下 Visa 授权和 EMV 交易数据的输入：

`ApplicationTransactionCounter`(ATC)

由芯片维护的交易计数器，每笔交易递增。在 EMV 数据中，这是标签9F36。

`PanSequenceNumber`

区分共享相同PAN卡片的 PAN 序列号。在 EMV 数据中，这是标签5F34。

`CardExpiryDate`

该卡的申请到期日期（EMV 标签5F24），以与其他信用卡验证电话相同的格式提供。

ServiceCode

卡的轨道 2（等效）数据中携带的三位数服务代码。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"DCVV"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
mw7dn3qxvkfh8ztc",
                "KeyAttributes": {
                    "KeyUsage": "TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "A8E4D2",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2025-02-02T11:45:30.648000-08:00",
                "UsageStartTimestamp": "2025-02-02T11:45:30.626000-08:00"
            }
        }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/mw7dn3qxvkfh8ztc 您需要在下一步中执行该操作。

生成 dcVV

Example

在此示例中，我们将为非接触式 EMV 交易生成 dcVV。输入包括 PAN、PAN 序列号、应用程序交易计数器、卡到期日和服务代码。

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/mw7dn3qxvkfh8ztc \
    --primary-account-number=4111112627662122 \
    --generation-attributes
 DynamicCardVerificationValue='{ApplicationTransactionCounter=01,PanSequenceNumber=00,CardExpir
 \
    --validation-data-length 5
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
mw7dn3qxvkfh8ztc",
    "KeyCheckValue": "A8E4D2",
    "ValidationData": "36667"
}
```

验证 DCvV

Example

在此示例中，我们将验证交易期间提供的 DCvV。必须提供用于生成的相同输入以进行验证。

如果 AWS 支付加密能够验证， http/200 则返回。如果该值未经过验证，它将返回响 http/400 应。

```
$ aws payment-cryptography-data verify-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/mw7dn3qxvkfh8ztc \
    --primary-account-number=4111112627662122 \
    --validation-data=36667 \
    --verification-attributes
 DynamicCardVerificationValue='{ApplicationTransactionCounter=01,PanSequenceNumber=00,CardExpir
```

```
{
```

```
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
mw7dn3qxvkfh8ztc",
    "KeyCheckValue": "A8E4D2"
}
```

万事达卡的特定功能

主题

- DCVC3
- ARQC-/ CVN14CVN15
- ARQC-/ CVN12CVN13
- 3DS AAV SPA2

DCVC3

DCVC3 早于 EMV CSK 和万事达卡 CVN12 计划，代表了另一种使用动态密钥的方法。它有时还会被重新用于其他用例。在该方案中，输入是 PAN、PSN、Track1/Track2 数据、不可预测的数字和交易计数器 (ATC)。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"DCVC3"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
hrh6qgbi3sk4y3wq",
                "KeyAttributes": {
                    "KeyUsage": "TR31_E4_EMV_MKEY_DYNAMIC_NUMBERS",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
```

```
                    "Unwrap": false,
                    "Generate": false,
                    "Sign": false,
                    "Verify": false,
                    "DeriveKey": true,
                    "NoRestrictions": false
                }
            },
            "KeyCheckValue": "08D7B4",
            "KeyCheckValueAlgorithm": "ANSI_X9_24",
            "Enabled": true,
            "Exportable": true,
            "KeyState": "CREATE_COMPLETE",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
            "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
            "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
        }
    }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333: key/hrh6qgbi3sk4y3wq。您需要在下一步中执行该操作。

生成一个 DCVC3

Example

尽管 DCVC3 通常由芯片卡生成，但也可以手动生成，如本例所示

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk
 --primary-account-number=5413123456784808 --generation-attributes
 DynamicCardVerificationCode='{ApplicationTransactionCounter=0000,TrackData=524106000000069D13
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4",
    "ValidationData": "865"
  }
```

验证 DCVC3

Example

在此示例中，我们将验证 DCVC3。请注意，ATC 应以十六进制数字提供，例如，计数器 11 应表示为 000B。该服务需要一个 3 位数字 DCVC3，因此，如果您存储了 4（或 5）位数的值，只需截断左边的字符直到有 3 位数字（例如，15321 的验证数据值应为 321）。

如果 AWS 支付密码学能够验证，则会返回 http/200。如果该值未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-card-validation-data  --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/pw3s6nl62t5ushfk
 --primary-account-number=5413123456784808 --verification-attributes
 DynamicCardVerificationCode='{ApplicationTransactionCounter=000B,TrackData=524106000000069D13
 --validation-data 398
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4"
  }
```

ARQC-/ CVN14CVN15

CVN14 并 CVN15 使用 EMV CSK 方法进行密钥推导。这两种方法的确切交易数据会有所不同，有关构造交易数据字段的详细信息，请参阅方案文档。

ARQC-/ CVN12CVN13

CVN12 并且 CVN13 是针对EMV交易的较旧万事达卡专用方法，它在每笔交易的推导中加入了不可预测的数字，并且还使用了不同的有效载荷。有关有效载荷内容的信息，请联系该计划。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVN12"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
```

```
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
                "KeyAttributes": {
                    "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": false,
                        "Sign": false,
                        "Verify": false,
                        "DeriveKey": true,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "08D7B4",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
            }
        }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333: key/pw3s6nl62 t5ushfk。您需要在下一步中执行该操作。

验证 ARQC

Example

在此示例中，我们将验证使用万事达卡生成的 ARQC。 CVN12

如果 AWS 支付密码学能够验证 ARQC，则会返回 http/200。如果 arqc 未经过验证，将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-
cryptogram 31BE5D49F14A5F01 \
```

```
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
  --major-key-derivation-mode EMV_OPTION_A \
  --transaction-data
 00000000170000000000000008400080008000084016051700000000093800000B1F220103000000000000000000000
 \
  --session-key-derivation-attributes='{"MastercardSessionKey":
{"ApplicationTransactionCounter":"000B","PanSequenceNumber":"01","PrimaryAccountNumber":"541312
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4"
  }
```

3DS AAV SPA2

SPA2（安全支付应用程序）AAV（账户验证值）用于万事达卡 3DS 交易（也称为万事达卡身份检查）。它使用基于 HMAC 的 MAC 生成为电子商务交易提供加密身份验证。AAV 是使用特定于交易的数据和共享密钥生成的。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=HMAC_SHA256,KeyUsage=TR31_M7_HMAC_KEY,KeyClass=SYMMETRIC_KEY,KeyModesOfUse='{Gene
 --tags='[{"Key":"KEY_PURPOSE","Value":"SPA2_AAV"},
{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-west-2:111122223333:key/
q5vjtshsg67cz5gn",
                "KeyAttributes": {
                    "KeyUsage": "TR31_M7_HMAC_KEY",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "HMAC_SHA256",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
```

```
                    "Unwrap": false,
                    "Generate": true,
                    "Sign": false,
                    "Verify": true,
                    "DeriveKey": false,
                    "NoRestrictions": false
                }
            },
            "KeyCheckValue": "C661F9",
            "KeyCheckValueAlgorithm": "HMAC",
            "Enabled": true,
            "Exportable": true,
            "KeyState": "CREATE_COMPLETE",
            "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
            "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
            "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
        }
    }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-west-2: 111122223333: key/q5vjtshsg67cz5gn。您需要在下一步中执行该操作。

生成 SPA2 AAV

Example

在此示例中，我们将使用 HMAC MAC 生成生成 AAV 的颁发者身份验证值 (I SPA2 AV) 组件。消息数据包含要进行身份验证的特定于交易的信息。报文数据的格式应遵循万事达卡的 SPA2 规范，本示例中不涉及该格式。

> **Note**
>
> 请查看您的万事达卡规格，了解将 IAV 插入 AAV 值的格式。

```
$ aws payment-cryptography-data generate-mac --key-identifier arn:aws:payment-
cryptography:us-west-2:111122223333:key/q5vjtshsg67cz5gn --message-data
 "2226400099919520FFFFd8b448be65694fe7b42f836bad396e9d" --generation-attributes
 Algorithm=HMAC --region us-west-2
```

```
{
```

```
    "KeyArn": "arn:aws:payment-cryptography:us-west-2:111122223333:key/
q5vjtshsg67cz5gn",
    "KeyCheckValue": "C661F9",
    "Mac": "6FB2405E9D8A4C1F7B173F73ADD1A6DC358531CAB0E9994FC5B62012ADDE91FC"
}
```

验证 SPA2 AAV

Example

在此示例中，我们将验证 SPA2 AAV。为验证提供了相同的消息数据和 MAC 值。

如果 AWS 支付密码学能够验证 MAC，则会返回 http/200。如果 MAC 未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-mac --key-identifier arn:aws:payment-
cryptography:us-west-2:111122223333:key/q5vjtshsg67cz5gn --message-
data "2226400099919520FFFFd8b448be65694fe7b42f836bad396e9d" --mac
 "6FB2405E9D8A4C1F7B173F73ADD1A6DC358531CAB0E9994FC5B62012ADDE91FC" --verification-
attributes Algorithm=HMAC --region us-west-2
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-west-2:111122223333:key/
q5vjtshsg67cz5gn",
    "KeyCheckValue": "C661F9"
}
```

美国运通的特定功能

主题

- CSC1
- CSC2
- ICSC
- 3DS AEVV

CSC1

CSC 版本 1 也被称为经典 CSC 算法。该服务可以以 3,4 或 5 位数字的形式提供。

有关所有可用参数，请参阅 API 参考指南中的 AmexCardSecurityCodeVersion1。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CSC1"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
                "Key": {
                    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgq",
                    "KeyAttributes": {
                        "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                        "KeyClass": "SYMMETRIC_KEY",
                        "KeyAlgorithm": "TDES_2KEY",
                        "KeyModesOfUse": {
                            "Encrypt": false,
                            "Decrypt": false,
                            "Wrap": false,
                            "Unwrap": false,
                            "Generate": true,
                            "Sign": false,
                            "Verify": true,
                            "DeriveKey": false,
                            "NoRestrictions": false
                        }
                    },
                    "KeyCheckValue": "8B5077",
                    "KeyCheckValueAlgorithm": "ANSI_X9_24",
                    "Enabled": true,
                    "Exportable": true,
                    "KeyState": "CREATE_COMPLETE",
                    "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                    "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                    "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
                }
            }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333: key/esh6hn7pxd tttzgq。您需要在下一步中执行该操作。

生成一个 CSC1

Example

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgq --primary-account-number=344131234567848 --generation-attributes
 AmexCardSecurityCodeVersion1='{CardExpiryDate=1224}' --validation-data-length 4
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgq",
    "KeyCheckValue": "8B5077",
    "ValidationData": "3938"
  }
```

验证 CSC1

Example

在此示例中，我们将验证 a CSC1。

如果 AWS 支付密码学能够验证，则会返回 http/200。如果该值未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-card-validation-data  --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/esh6hn7pxdtttzgq
 --primary-account-number=344131234567848 --verification-attributes
 AmexCardSecurityCodeVersion1='{CardExpiryDate=1224}''  --validation-data 3938
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
esh6hn7pxdtttzgq",
    "KeyCheckValue": "8B5077"
  }
```

CSC2

CSC 版本 2 也被称为增强型 CSC 算法。该服务可以以 3,4 或 5 位数字的形式提供。的服务代码通常 CSC2 为 000。

有关所有可用参数，请参阅 API 参考指南中的 AmexCardSecurityCodeVersion2。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CSC2"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
            "Key": {
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
erlm445qvunmvoda",
                "KeyAttributes": {
                    "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
                    "KeyClass": "SYMMETRIC_KEY",
                    "KeyAlgorithm": "TDES_2KEY",
                    "KeyModesOfUse": {
                        "Encrypt": false,
                        "Decrypt": false,
                        "Wrap": false,
                        "Unwrap": false,
                        "Generate": true,
                        "Sign": false,
                        "Verify": true,
                        "DeriveKey": false,
                        "NoRestrictions": false
                    }
                },
                "KeyCheckValue": "BF1077",
                "KeyCheckValueAlgorithm": "ANSI_X9_24",
                "Enabled": true,
                "Exportable": true,
                "KeyState": "CREATE_COMPLETE",
                "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
                "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
            }
        }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-2: 111122223333: key/erlm445qvunmvoda。您需要在下一步中执行该操作。

生成一个 CSC2

在此示例中，我们将生成长 CSC2 度为 4 的。可以生成长度为 3,4 或 5 的 CSC。对于美国运通，PANs 应为 15 位数字，并以 34 或 37 开头。过期日期的格式通常为 YYMM。服务代码可能有所不同-请查看您的手册，但典型值为 000、201 或 702

Example

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
erlm445qvunmvoda --primary-account-number=344131234567848 --generation-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=2412,ServiceCode=000}' --validation-
data-length 4
```

```
{
"KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda",
"KeyCheckValue": "BF1077",
"ValidationData": "3982"
}
```

验证 CSC2

Example

在此示例中，我们将验证 a CSC2。

如果 AWS 支付密码学能够验证，则会返回 http/200。如果该值未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-card-validation-data  --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda
 --primary-account-number=344131234567848 --verification-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=2412,ServiceCode=000}' --validation-data
 3982
```

```
{
"KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/erlm445qvunmvoda",
"KeyCheckValue": "BF1077"
}
```

ICSC

iCSC 也被称为静态 CSC 算法，使用 CSC 版本 2 进行计算。该服务可以以 3,4 或 5 位数字的形式提供。

使用服务代码 999 计算联系人卡片的 icSC。使用服务代码 702 计算非接触式卡的 iSCC。

有关所有可用参数，请参阅 API 参考指南中的 AmexCardSecurityCodeVersion2。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"CSC1"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
        "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-1:111122223333:key/7vrybrbvjcvwtunv",
        "KeyAttributes": {
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY"
            "KeyAlgorithm": "TDES_2KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": false,
                "Encrypt": false,
                "Generate": true,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": true,
                "Wrap": false
            },
        },
        "KeyCheckValue": "7121C7",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
```

```
        "CreateTimestamp": "2025-01-29T09:19:21.209000-05:00",
        "UsageStartTimestamp": "2025-01-29T09:19:21.192000-05:00"
        }
    }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-1:111122223333: key/7vrybrbvjcvjcvw tunv。您需要在下一步中执行该操作。

生成一个 iSCC

在此示例中，我们将为使用服务代码 702 的非接触式卡生成长度为 4 的 iCSC。可以生成长度为 3,4 或 5 的 CSC。对于美国运通， PANs 应为 15 位数字，并以 34 或 37 开头。

Example

```
$ aws payment-cryptography-data generate-card-validation-data --key-identifier
 arn:aws:payment-cryptography:us-east-1:111122223333:key/7vrybrbvjcvwtunv
 --primary-account-number=344131234567848 --generation-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=1224,ServiceCode=702}' --validation-
data-length 4
```

```
{
    "KeyArn": arn:aws:payment-cryptography:us-east-1:111122223333:key/7vrybrbvjcvwtunv,
    "KeyCheckValue": 7121C7,
    "ValidationData": "2365"
}
```

验证 iSCC

Example

在此示例中，我们将验证 iSCC。

如果 AWS 支付密码学能够验证，则会返回 http/200。如果该值未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-card-validation-data  --key-identifier
 arn:aws:payment-cryptography:us-east-1:111122223333:key/7vrybrbvjcvwtunv
 --primary-account-number=344131234567848 --verification-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=1224,ServiceCode=702}' --validation-data
 2365
```

```
{
```

```
    "KeyArn": arn:aws:payment-cryptography:us-east-1:111122223333:key/7vrybrbvjcvwtunv,
    "KeyCheckValue": 7121C7
}
```

3DS AEVV

3DS AEVV（三维安全账户验证值）用于美国运通三维安全认证。它使用与之相同的算法 CSC2 ，但输入参数不同。到期日期字段应填入不可预测的（随机）数字，服务代码由 AEVV 身份验证结果代码（1 位）加上第二因素身份验证码（2 位数）组成。输出长度应为 3 位数。

有关所有可用参数，请参阅 API 参考指南中的 AmexCardSecurityCodeVersion2。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 --tags='[{"Key":"KEY_PURPOSE","Value":"3DS_AEVV"},
{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
        "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kw8djn5qxvfh3ztm",
        "KeyAttributes": {
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY"
            "KeyAlgorithm": "TDES_2KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyModesOfUse": {
                "Decrypt": false,
                "DeriveKey": false,
                "Encrypt": false,
                "Generate": true,
                "NoRestrictions": false,
                "Sign": false,
                "Unwrap": false,
                "Verify": true,
                "Wrap": false
            },
        },
        "KeyCheckValue": "8F3A21",
```

```
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "KeyState": "CREATE_COMPLETE",
        "CreateTimestamp": "2025-02-02T10:30:15.209000-05:00",
        "UsageStartTimestamp": "2025-02-02T10:30:15.192000-05:00"
        }
    }
```

注意代表密钥的那个KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333: key/kw8djn5qxvfh3z tm。您需要在下一步中执行该操作。

生成 3DS AEVV

在此示例中，我们将生成一个长度为 3 的 3DS AEVV。到期日期字段包含不可预测的（随机）数字（例如 1234），服务代码由 AEVV 身份验证结果代码（1 位数）加上第二因素身份验证码（2 位数）组成，例如 543，其中 5 是身份验证结果代码，43 是第二因素身份验证码。对于美国运通， PANs 应为 15 位数字，并以 34 或 37 开头。

Example

```
$ aws payment-cryptography-data generate-card-validation-data --key-
identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
kw8djn5qxvfh3ztm --primary-account-number=344131234567848 --generation-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=1234,ServiceCode=543}' --validation-
data-length 3
```

```
{
    "KeyArn": arn:aws:payment-cryptography:us-east-2:111122223333:key/kw8djn5qxvfh3ztm,
    "KeyCheckValue": 8F3A21,
    "ValidationData": "921"
}
```

验证 3DS AEVV

Example

在本示例中，我们将验证 3DS AEVV。

如果 AWS 支付密码学能够验证，则会返回 http/200。如果该值未经过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-card-validation-data  --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/kw8djn5qxvfh3ztm
 --primary-account-number=344131234567848 --verification-attributes
 AmexCardSecurityCodeVersion2='{CardExpiryDate=1234,ServiceCode=543}' --validation-data
 921
```

```
{
    "KeyArn": arn:aws:payment-cryptography:us-east-2:111122223333:key/kw8djn5qxvfh3ztm,
    "KeyCheckValue": 8F3A21
}
```

JCB 特定函数

主题

- ARQC-CVN04
- ARQC-CVN01

ARQC-CVN04

JCB CVN04 使用 CSK 的密钥派生方法。有关构造交易数据字段的详细信息，请参阅方案文档。

ARQC-CVN01

CVN01 是一种用于 EMV 交易的较早的 JCB 方法，它使用每张卡的密钥派生而不是会话（每笔交易）派生，还使用不同的负载。Visa 也使用此消息，因此元素名称具有该名称，尽管它也用于 JCB。有关有效载荷内容的信息，请联系方案文档。

创建密钥

```
$ aws payment-cryptography create-key --exportable --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS,KeyClass=SYMMETRIC_KEY,KeyMod
 --tags='[{"Key":"KEY_PURPOSE","Value":"CVN10"},{"Key":"CARD_BIN","Value":"12345678"}]'
```

响应会回显请求参数，包括后续调用的 ARN 以及密钥检查值 (KCV)。

```
{
                        "Key": {
                            "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/pw3s6nl62t5ushfk",
                            "KeyAttributes": {
```

```
                            "KeyUsage": "TR31_E0_EMV_MKEY_APP_CRYPTOGRAMS",
                            "KeyClass": "SYMMETRIC_KEY",
                            "KeyAlgorithm": "TDES_2KEY",
                            "KeyModesOfUse": {
                                "Encrypt": false,
                                "Decrypt": false,
                                "Wrap": false,
                                "Unwrap": false,
                                "Generate": false,
                                "Sign": false,
                                "Verify": false,
                                "DeriveKey": true,
                                "NoRestrictions": false
                            }
                        },
                        "KeyCheckValue": "08D7B4",
                        "KeyCheckValueAlgorithm": "ANSI_X9_24",
                        "Enabled": true,
                        "Exportable": true,
                        "KeyState": "CREATE_COMPLETE",
                        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                        "CreateTimestamp": "2024-03-07T06:41:46.648000-07:00",
                        "UsageStartTimestamp": "2024-03-07T06:41:46.626000-07:00"
                    }
                }
```

记下代表密钥的KeyArn，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/pw3s6nl62t5ushfk 您需要在下一步中执行该操作。

验证 ARQC

Example

在此示例中，我们将验证使用 JCB CVN01 生成的 ARQC。它使用与 Visa 方法相同的选项，因此命名为参数。

如果 AWS 支付密码学能够验证 ARQC， http/200 则返回。如果 arqc 未通过验证，它将返回 http/400 响应。

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-
cryptogram D791093C8A921769 \
              --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
```

```
            --major-key-derivation-mode EMV_OPTION_A \
            --transaction-data
 0000000017000000000000000840008000800008401605170000000093800000B03011203000000 \
            --session-key-derivation-attributes='{"Visa":{"PanSequenceNumber":"01" \
            ,"PrimaryAccountNumber":"9137631040001422"}}'
```

```
{
                "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
                "KeyCheckValue": "08D7B4"
            }
```

UnionPay 特定函数

以下示例与 UnionPay 国际（UPI）有关，有时也称为中国 UnionPay （CUP）。

主题

- ARQC 验证
- CVN2（卡验证码）
- AVN（身份验证码）

ARQC 验证

UnionPay ARQC 验证使用 EMV_OPTION_A 密钥派生模式。会话密钥使用一种 UnionPay-specific 方案从 PrimaryAccountNumber、PAN 序列号和应用程序事务计数器中派生。有关交易数据格式和字段排序的信息，请参阅架构文档。

验证 ARQC

Example使用 UnionPay 密钥派生验证 ARQC

此示例说明如何验证使用 UnionPay 密钥派生生成的 ARQC。

如果 P AWS ayment Cryptography 验证了 ARQC，它将返回 HTTP 200 响应。如果 ARQC 验证失败， AWS 支付加密将返回带有原因代码的 HTTP 400 响应。

```
$ aws payment-cryptography-data verify-auth-request-cryptogram --auth-request-
cryptogram A12345B1A12345B1 \
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk \
```

```
  --major-key-derivation-mode EMV_OPTION_A \
  --transaction-data 123456789ABCDEF80000000000000000 \
  --session-key-derivation-
attributes='UnionPay={PrimaryAccountNumber=4123412341234123,PanSequenceNumber=01,ApplicationTra
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
pw3s6nl62t5ushfk",
    "KeyCheckValue": "08D7B4"
}
```

如果无法验证 ARQC，则 AWS 支付密码学会返回错误响应：

```
An error occurred (VerificationFailedException) when calling the
 VerifyAuthRequestCryptogram operation: Auth request cryptogram verification failed.

Reason: INVALID_AUTH_REQUEST_CRYPTOGRAM
```

CVN2（卡验证码）

CVN2 UnionPay 等效于 CVV2。它是用于在线和无卡交易的卡验证码。在密码学上，CVN2 使用与 CVV2 () 相同的算法。 CardVerificationValue2 输入的是主账号 (PAN) 和信用卡到期日（MMYY 格式的 4 位数字）。

创建密钥

要创建 CVN2 密钥，请使用带有用法和算TR31_C0_CARD_VERIFICATION_KEY法的create-key命令。TDES_2KEY

```
$ aws payment-cryptography create-key --exportable \
  --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 \
  --tags='[{"Key":"KEY_PURPOSE","Value":"CVN2"},{"Key":"CARD_BIN","Value":"47613499"}]'
```

响应会回显请求参数，包括后续调用的 ARN 和密钥校验值 (KCV)。

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
```

```
        "KeyAttributes": {
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "AEA5CD",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

记下这个KeyArn值，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/7f7g4spf3xcklhzu 您需要在以下步骤中使用它。

生成 CVN2

Example为给定的 PAN 生成 CVN2

此示例为给定的 PAN 生成 CVN2，其输入为PAN和卡到期日期。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue2中的。

```
$ aws payment-cryptography-data generate-card-validation-data \
  --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu \
  --primary-account-number=4761349999991234 \
  --generation-attributes CardVerificationValue2='{CardExpiryDate=1226}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
    "KeyCheckValue": "AEA5CD",
    "ValidationData": "321"
}
```

验证 CVN2

Example验证给定 PAN 的 CVN2

此示例验证给定 PAN 的 CVN2PAN，其中输入为、卡到期日期和交易期间提供的 CVN2。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue2中的。

> **Note**
>
> CVN2 是用户输入的值。定期验证失败并不一定表示存在问题。

```
$ aws payment-cryptography-data verify-card-validation-data \
  --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu \
  --primary-account-number=4761349999991234 \
  --verification-attributes CardVerificationValue2='{CardExpiryDate=1226}' \
  --validation-data 321
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/7f7g4spf3xcklhzu",
    "KeyCheckValue": "AEA5CD"
}
```

AVN（身份验证码）

AVN（身份验证验证码）是 UnionPay 3DS 加密码。它使用CVV算法 (CardVerificationValue1) 和重新调整了用途的输入字段，如下所示，但请查看方案文档以获取最终的详细信息。

- PAN — 主要账号（16-19 位数字，来自 AreQ）
- CardExpiryDate— 从 DS 交易 ID (dstransID) 导出的值的四位最低有效数字

- ServiceCode— 3 位数：身份验证交易结果（1 位）+ 填充位 '0'（1 位数）+ AV 生成器指示灯（1 位数）

创建密钥

要创建 AVN 密钥，请使用带有TR31_C0_CARD_VERIFICATION_KEY用TDES_2KEY法和算法的create-key命令。

```
$ aws payment-cryptography create-key --exportable \
  --key-attributes
 KeyAlgorithm=TDES_2KEY,KeyUsage=TR31_C0_CARD_VERIFICATION_KEY,KeyClass=SYMMETRIC_KEY,KeyModesO
 \
  --tags='[{"Key":"KEY_PURPOSE","Value":"AVN"},{"Key":"CARD_BIN","Value":"47613499"}]'
```

响应会回显请求参数，包括后续调用的 ARN 和密钥校验值 (KCV)。

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
        "KeyAttributes": {
            "KeyUsage": "TR31_C0_CARD_VERIFICATION_KEY",
            "KeyClass": "SYMMETRIC_KEY",
            "KeyAlgorithm": "TDES_2KEY",
            "KeyModesOfUse": {
                "Encrypt": false,
                "Decrypt": false,
                "Wrap": false,
                "Unwrap": false,
                "Generate": true,
                "Sign": false,
                "Verify": true,
                "DeriveKey": false,
                "NoRestrictions": false
            }
        },
        "KeyCheckValue": "F3FB13",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "Enabled": true,
        "Exportable": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
```

```
        "CreateTimestamp": "2023-06-05T06:41:46.648000-07:00",
        "UsageStartTimestamp": "2023-06-05T06:41:46.626000-07:00"
    }
}
```

记下这个KeyArn值，例如 arn: aws: payment-cryptography: us-east-2:111122223333:。key/dnaeyrjgdjjtw6dk 您需要在以下步骤中使用它。

生成 AVN

Example为给定的 PAN 生成 AVN

此示例为给定的 PAN 生成 AVN。该CardExpiryDate字段包含从 dstranSid 派生的四位最低有效数字。该ServiceCode字段包含身份验证结果、填充位和生成器指示器。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue1中的。

```
$ aws payment-cryptography-data generate-card-validation-data \
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk \
  --primary-account-number=4761349999991234 \
  --generation-attributes
 CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=201}'
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
    "KeyCheckValue": "F3FB13",
    "ValidationData": "491"
}
```

验证 AVN

Example验证给定 PAN 的 AVN

此示例使用生成给定 PAN 的相同输入字段以及要验证的 AVN 值来验证给定 PAN 的 AVN。

有关所有可用参数，请参阅 API 参考指南CardVerificationValue1中的。

```
$ aws payment-cryptography-data verify-card-validation-data \
```

```
  --key-identifier arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk \
  --primary-account-number=4761349999991234 \
  --verification-attributes
 CardVerificationValue1='{CardExpiryDate=9431,ServiceCode=201}' \
  --validation-data 491
```

```
{
    "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
dnaeyrjgdjjtw6dk",
    "KeyCheckValue": "F3FB13"
}
```

收购和付款促进者

收单行 PSPs 和支付协调机构通常与发行人有不同的加密要求。常见使用案例包括：

数据解密

数据（尤其是 pan 数据）可能由支付终端加密，需要由后端解密。解密@@ 数据和加密数据支持多种方法，包括 TD ES、AES 和 DUKPT 派生技术。 AWS 支付密码学服务本身也符合 PCI P2PE 标准，并已注册为 PCI P2PE 解密组件。

TranslatePin

为了保持 PCI PIN 的合规性，在安全设备上输入持卡人密码后，收单系统不得将持卡人密码置于明文中。因此，要将密码从终端传递到下游系统（例如支付网络或发卡机构），需要使用与支付终端使用的密钥不同的密钥对其进行重新加密。T@@ ranslate Pin 通过使用 servicebbb 将加密的密码从一个密钥安全地转换为另一个密钥来实现这一目标。使用此命令，可以在各种方案（例如 TDES、AES 和 DUKPT 派生）以及引脚块格式（例如 ISO-0、ISO-3 和 ISO-4）之间进行引脚转换。

VerifyMac

来自支付终端的数据可能经过MAC处理，以确保数据在传输过程中没有被修改。验证 Mac 并 GenerateMac 支持使用对称密钥的各种技术，包括用于 ISO-9797-1 算法 1、ISO-9797-1 算法 3（零售 MAC）和 CMAC 技术的 TDES、AES 和 DUKPT 派生技术。

其他主题

- 使用动态按键

使用动态按键

动态密钥允许将一次性或有限使用的密钥用于加密操作，例如`EncryptData`。当密钥材料频繁轮换（例如在每笔信用卡交易中），并且希望避免将密钥材料导入服务时，可以使用此流程。短期密钥可用作 SoftPOS/M POC 或其他解决方案的一部分。

> **Note**
>
> 这可以代替使用 Payment AWS Cryptography 的典型流程，即创建加密密钥或将其导入服务，并使用密钥别名或密钥 arn 指定密钥。

以下操作支持动态密钥：

- EncryptData
- DecryptData
- ReEncryptData
- TranslatePin

解密数据

以下示例显示了如何使用动态密钥和 decrypt 命令。在本例中，密钥标识符是保护解密密钥的包装密钥 (KEK)（在 TR-31 格式的封装密钥参数中提供）。封装后的密钥必须是 D0 的密钥用途，可以与 decrypt 命令一起使用，同时还有 B 或 D 的使用模式。

Example

```
$ aws payment-cryptography-data decrypt-data --key-identifier
 arn:aws:payment-cryptography:us-east-2:111122223333:key/ov6icy4ryas4zcza
 --cipher-text 1234123412341234123412341234123A --decryption-attributes
 'Symmetric={Mode=CBC,InitializationVector=1234123412341234}'   --wrapped-key
 WrappedKeyMaterial={"Tr31KeyBlock"="D0112D0TN00E0000B05A6E82D7FC68B95C84306634B0000DA4701BE9BC
```

```
{
   "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
   "KeyCheckValue": "0A3674",
```

```
   "PlainText": "2E138A746A0032023BEF5B85BA5060BA"
}
```

翻译别针

以下示例演示如何使用动态键和 translate pin 命令从动态密钥转换为半静态收单机构工作密钥 (AWK)。在本例中，传入的密钥标识符是包装密钥 (KEK)，用于保护以 TR-31 格式提供的动态密码加密密钥 (PEK)。封装后的密钥应与 B 或 D 的使用模式P0一起作为密钥用途。传出的密钥标识符是 encrypt=True、wrap=True 的密钥类型TR31_P0_PIN_ENCRYPTION_KEY和使用模式

Example

```
$ aws payment-cryptography-data translate-pin-data --encrypted-pin-block
 "C7005A4C0FA23E02" --incoming-translation-
attributes=IsoFormat0='{PrimaryAccountNumber=171234567890123}'
 --incoming-key-identifier alias/PARTNER1_KEK  --outgoing-key-
identifier alias/ACQUIRER_AWK_PEK --outgoing-translation-attributes
 IsoFormat0="{PrimaryAccountNumber=171234567890123}"  --incoming-wrapped-key
 WrappedKeyMaterial={"Tr31KeyBlock"="D0112P0TB00S0000EB5D8E63076313162B04245C8CE351C956EA4A16CC
```

```
{
   "PinBlock": "2E66192BDA390C6F",
   "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
   "KeyCheckValue": "0A3674"
}
```

的特定区域功能 AWS Payment Cryptography

某些功能可能是特定区域的，不能以其他方式使用。本节将对这些功能进行更详细的描述。

AS2805

澳大利亚标准 2805 (AS2805) 是电子资金转账的标准，主要用于银行卡支付交易。它由澳大利亚标准局维护。该标准由 6 本书组成，涵盖了从消息格式到加密标准的众多主题。

第 6 部分提供了有关密钥管理的指导，包括主机到主机（节点到节点）通信和相关的加密要求，而其他方面则在其他部分中介绍。该标准中的所有密码学目前都基于TDES。

与其他实现方式相比，AS2805 有许多差异，总结如下。

密钥保护

依赖于密钥变体而不是像 in TR-31/X9.143 这样的键块。 AWS 支付密码学在内部将所有密钥存储为密钥块，但允许使用 AS2805 定义的变体进行导入、导出和计算。

单向密钥

AS2805 要求使用单向密钥。如果两个节点都需要生成消息身份验证码 (MAC)，则它们使用两个密钥。

别针方块

AS2805 定义了每笔交易唯一的 PIN 加密密钥的密钥派生技术。这可以用来代替 DUKPT。与 DUKPT 使用交易计数器相比，AS2805 方案依赖于交易数据（跟踪号和交易金额）。

密钥交换验证

定义了在开始交换工作密钥（例如密码）之前验证 KEK 的过程。在其他方案中，KEK 不经常交换，并使用 KCV 进行验证。

AS2805 使用密钥变体而不是密钥块的概念来确保密钥仅用于预期（也是唯一的）目的。以下是使用密钥导入、导出或执行其他加密功能时，P AWS ayment Cryptography 如何在变体和密钥块之间映射。

AS2805 密钥类型	AWS 支付加密密钥类型
终端主密钥变体_00	TR31_K0_KEY_ENCRYPTION_KEY

AS2805 密钥类型	AWS 支付加密密钥类型
PIN_ENCRYPTION_KEY_VARIANT_28	TR31_P0_PIN_ENCRYPTION_KEY
消息身份验证密钥变体_24	TR31_M0_ISO_16609_MAC_KEY
数据加密密钥变体_22	TR31_D0_SYMMETRIC_DATA_ENCRYPTION_KEY
VARIANT_MASK_82，VARIANT_MASK_82C0	作为 KEK 验证过程的一部分可用的选项。这些密钥类型是临时的，不由服务存储。

给定两个节点，节点 1 和节点 2，以下示例是从节点 1 的角度出发。 AWS 支付密码学支持流程双方的 API。

主题

- 初始密钥 (KEK) 交换
- KEK 的验证
- 创建和传输工作密钥
- 导出工作密钥
- 大头针翻译
- Mac 生成和验证

初始密钥 (KEK) 交换

在 AS2805 中，双方都有自己的 KEK。KEK (s) 是指发送方密钥将在发送方需要密钥并将其发送到 node2 时使用的发送方 protect/wrap 密钥。KEK (r) 是由另一端（node2）创建的密钥。

> **Note**
>
> 这些术语是相对的-一方创建密钥（发送方），另一方接收密钥。因此，给定 KEY1，它在 node1 上被称为 KEK (s)，在节点 2 上被称为 KEK (r)。

AS2805 的 KEK 始终是密钥类型 = TR31_K0_KEY_ENCRYPTION_KEY，因为它们用于保护密码而不是密钥区块。这映射到 AS2805 6.1 中定义的 TERMINAL_MAJOR_KEY_VARIANT_00

步骤：

1.创建密钥

使用 CreateKey API 创建密钥。您将创建类型为 TR31_K0_KEY_ENCRYPTION_KEY 的密钥

2. 确定与 node2 交换密钥的方法

确定如何与对手交换 KEK 。对于 AS2805，最常见且可互操作的方法是 RSA Wrap。

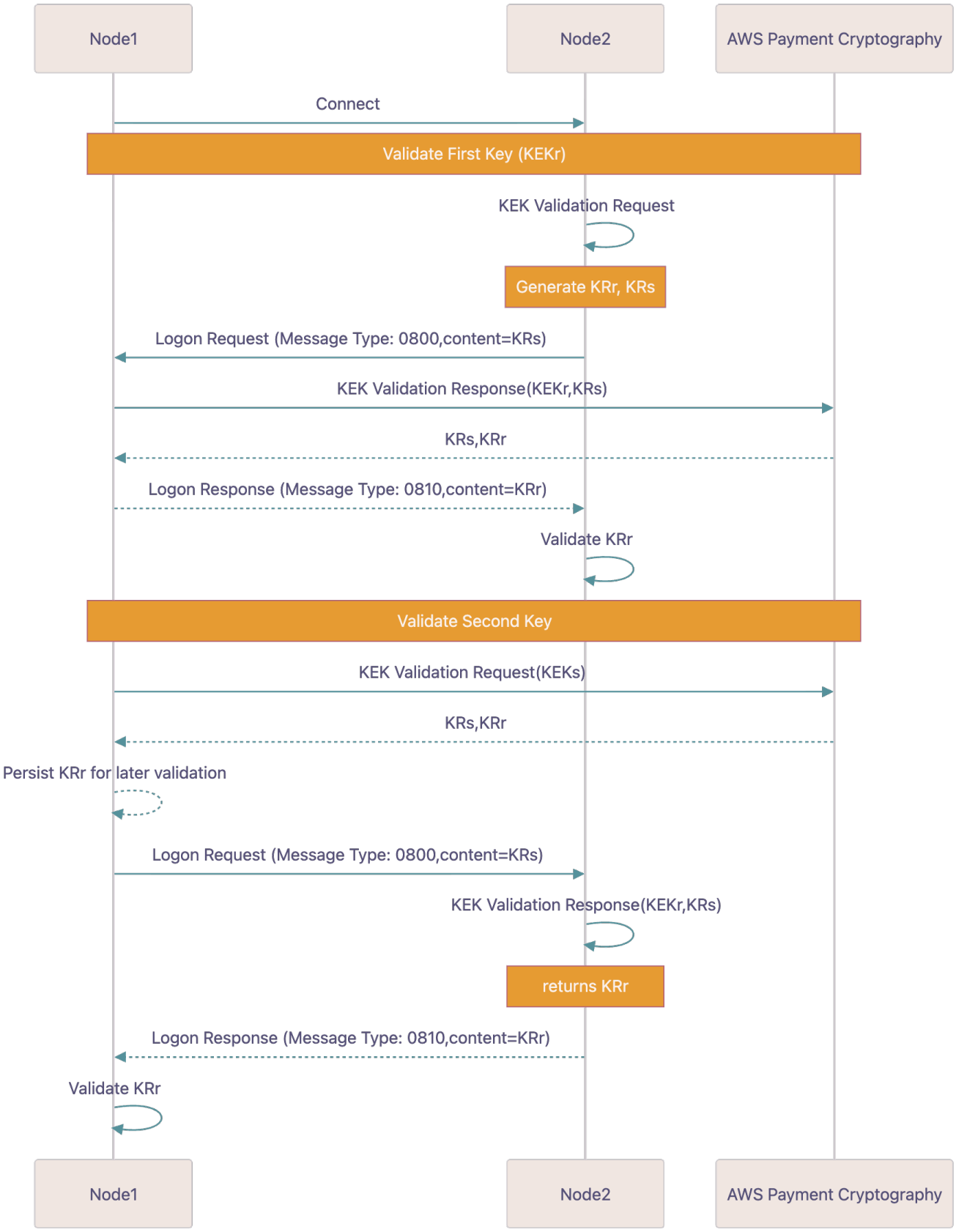
3.导出 KEK

根据您的上述选择，您将收到来自 node2 的公钥证书。您将使用该证书运行导出来保护密钥（如果使用 ECDH，则派生密钥）。

4. 导入 KeKR

根据您的上述选择，您将向 node2 发送公钥证书。您将使用该证书运行导入，将节点 2 的 KeKR 加载到服务中。

KEK 的验证


Node1
Node2
AWS Payment Cryptography
Connect
Validate First Key (KEKr)
KEK Validation Request
Generate KRr, KRs
Logon Request (Message Type: 0800,content=KRs)
KEK Validation Response(KEKr,KRs)
KRs,KRr
Logon Response (Message Type: 0810,content=KRr)
Validate KRr
Validate Second Key
KEK Validation Request(KEKs)
KRs,KRr
Persist KRr for later validation
Logon Request (Message Type: 0800,content=KRs)
KEK Validation Response(KEKr,KRs)
returns KRr
Logon Response (Message Type: 0810,content=KRr)
Validate KRr
Node1
Node2
AWS Payment Cryptography

当您的服务（node1）连接到 node2 时，双方都将使用名为 KEK 验证的过程确保在后续操作中使用相同的 KEK。

1。验证第一个密钥的步骤

1.1 接收 KR

Node2 将生成 KR 并将其作为登录过程的一部分发送给您。他们可能会使用 AWS 支付密码学来生成此值或其他解决方案。

1.2 生成 KEK 验证响应

您的节点将生成 KEK 验证响应，输入为 KEK (r)，KR 在步骤 1 中提供。

Example

```
cat >> generate-kek-validation-response.json
{
    "KekValidationType": {
        "KekValidationResponse": {
            "RandomKeySend": "9217DC67B8763BABCFDF3DADFCD0F84A"
        }
    },
    "RandomKeySendVariantMask": "VARIANT_MASK_82",
    "KeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza"
}
```

```
$ aws payment-cryptography-data generate-as2805-kek-validation --cli-input-json
 file://generate-kek-validation-response.json
```

```
{
 "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ov6icy4ryas4zcza",
 "KeyCheckValue": "0A3674",
 "RandomKeyReceive": "A4B7E249C40C98178C1B856DB7FB76EB",
 "RandomKeySend": "9217DC67B8763BABCFDF3DADFCD0F84A"
}
```

1.3 计算的回报率 KrR

将计算出的 KrR 返回到 node2。该节点会将其与步骤 1 中的计算值进行比较。

2.验证第二个密钥的步骤

2.1 生成 KrR 和 KR

您的节点将使用 AWS 支付加密生成一个随机值和该值的反向（反向）副本。该服务将输出由 KEK 封装的这两个值。它们被称为 KR (s) 和 KR (r)。

Example

```
cat >> generate-kek-validation-request.json
{
    "KekValidationType": {
        "KekValidationRequest": {
            "DeriveKeyAlgorithm": "TDES_2KEY"
        }
    },
"RandomKeySendVariantMask": "VARIANT_MASK_82",
    "KeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
rhfm6tenpxapkmrv"
}
```

```
$ aws payment-cryptography-data generate-as2805-kek-validation --cli-input-json
 file://generate-kek-validation-request.json
```

```
{
 "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
rhfm6tenpxapkmrv",
 "KeyCheckValue": "DC1081",
 "RandomKeyReceive": "A4B7E249C40C98178C1B856DB7FB76EB",
 "RandomKeySend": "9217DC67B8763BABCFDF3DADFCD0F84A"
}
```

2.2 向 node2 发送 KR

将 KR 发送到 node2。保留 KrR 以备日后验证。

2.3 Node2 生成 KEK 验证响应

Node2 使用 KeKR 和 KR，生成 KrR 并将其发送回您的服务。

2.4 验证响应

比较步骤 1 中的 KrR 和步骤 3 返回的值。如果它们匹配，请继续。

创建和传输工作密钥

AS2805 中使用的典型工作密钥包括两组密钥：

节点之间的密钥，例如：区域密码密钥 (ZPK)、区域加密密钥 (ZEK) 和区域身份验证密钥 (ZAK)。

终端和节点之间的密钥，例如：如果不使用 DUKPT，则为终端主密钥 (TMK) 和终端密码密钥 (TPK)。

> **Note**
>
> 我们建议尽量减少每个终端密钥的密钥，并尽可能利用诸如 TR-34 和 DUKPT 等使用较少密钥数量的技术。

Example

在此示例中，我们使用可选标签来跟踪此密钥的用途和用途。标签不用作系统加密功能的一部分，但可用于分类、财务跟踪，并可用于应用 IAM 政策。

```
cat >> create-zone-pin-key.json
{
    "KeyAttributes": {
        "KeyUsage": "TR31_P0_PIN_ENCRYPTION_KEY",
        "KeyClass": "SYMMETRIC_KEY",
        "KeyAlgorithm": "TDES_2KEY",
        "KeyModesOfUse": {
            "Encrypt": true,
            "Decrypt": true,
            "Wrap": true,
            "Unwrap": true,
            "Generate": false,
            "Sign": false,
            "Verify": false,
            "DeriveKey": false,
            "NoRestrictions": false
        }
    },
    "KeyCheckValueAlgorithm": "ANSI_X9_24",
    "Exportable": true,
    "Enabled": true,
    "Tags": [
        {
            "Key": "AS2805_KEYTYPE",
```

```
            "Value": "ZONE_PIN_KEY_VARIANT28"
        }
    ]
}
```

```
$ aws payment-cryptography create-key --cli-input-json file://create-zone-pin-key.json
 --region ap-southeast-2
```

```
{
 "Key": {
 "KeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/alsuwfxug3pgy6xh",
 "KeyAttributes": {
 "KeyUsage": "TR31_P0_PIN_ENCRYPTION_KEY",
 "KeyClass": "SYMMETRIC_KEY",
 "KeyAlgorithm": "TDES_2KEY",
 "KeyModesOfUse": {
 "Encrypt": true,
 "Decrypt": true,
 "Wrap": true,
 "Unwrap": true,
 "Generate": false,
 "Sign": false,
 "Verify": false,
 "DeriveKey": false,
 "NoRestrictions": false
 }
 },
 "KeyCheckValue": "9A325B",
 "KeyCheckValueAlgorithm": "ANSI_X9_24",
 "Enabled": true,
 "Exportable": true,
 "KeyState": "CREATE_COMPLETE",
 "KeyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
 "CreateTimestamp": "2025-12-17T09:05:27.586000-08:00",
 "UsageStartTimestamp": "2025-12-17T09:05:27.570000-08:00"
 }
}
```

导出工作密钥

为了保持与其他方的兼容性，Paym AWS ent Cryptography 支持 AS2805 对称密钥包装技术，该技术使用密钥变体而不是像密钥块这样的密钥块。 TR-31如果各方共享多个密钥，则应单独导出每个密

钥。如果数据是双向发送的，则相同类型的交易方（例如 ZAK (s) 和 ZAK (r)）之间可能有两个密钥用于生成消息身份验证码。

要以这些格式导入和导出的其他参数在命令中指定。

```
cat >> export-zone-pin-key.json
{
    "ExportKeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
alsuwfxug3pgy6xh",
    "KeyMaterial": {
        "As2805KeyCryptogram": {
            "WrappingKeyIdentifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/rhfm6tenpxapkmrv",
            "As2805KeyVariant: "PIN_ENCRYPTION_KEY_VARIANT_28"
        }
    }
}
```

```
$ aws payment-cryptography export-key --cli-input-json file://export-zone-pin-key.json
 --region ap-southeast-2
```

```
{
    "WrappedKey": {
        "KeyCheckValue": "DC1081",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyMaterial": "HDC10AEF038E695DDD72AF08DC1BB422D",
        "WrappedKeyMaterialFormat": "KEY_CRYPTOGRAM",
        "WrappingKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
rhfm6tenpxapkmrv"
    }
}
```

大头针翻译

AS2805 在第 6.4 节中描述了会话特定的密钥派生模式。它的用途与 DUKPT 类似，任何一种算法都可以使用，因为第 6.7 节中介绍了 DUKPT。在此方案中，使用 (STAN) 作为派生数据从终端密码中派生出会话引脚密钥 SystemTraceAuditNumber（称为 KPE）。 TransactionAmount

翻译密码是一项常用功能，可以翻译 to/from 各种格式。在此示例中，我们将 PIN 从 KPE 转换为个人识别码加密密钥 (PEK)，例如在向支付网络发送 PIN 时。

```
cat >> translate-pin-as2805.json
{
    "EncryptedPinBlock": "B3B34B43BAB5F81A",
    "IncomingKeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
ivi5ksfsuplneuyt",
    "IncomingTranslationAttributes": {
        "IsoFormat0": {
            "PrimaryAccountNumber": "9999179999900013"
        }
    },
      "IncomingAs2805Attributes": {
        "SystemTraceAuditNumber": "000348",
        "TransactionAmount": "000000000328"
    },
    "OutgoingKeyIdentifier": "",
    "OutgoingTranslationAttributes": {
        "IsoFormat0": {
            "PrimaryAccountNumber": "9999179999900013"
        }
    }
}
```

```
$ aws payment-cryptography-data translate-pin-data --cli-input-json file://translate-
pin-as2805.json  --region ap-southeast-2
```

```
{
    "WrappedKey": {
        "KeyCheckValue": "DC1081",
        "KeyCheckValueAlgorithm": "ANSI_X9_24",
        "KeyMaterial": "HDC10AEF038E695DDD72AF08DC1BB422D",
        "WrappedKeyMaterialFormat": "KEY_CRYPTOGRAM",
        "WrappingKeyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
rhfm6tenpxapkmrv"
    }
}
```

Mac 生成和验证

生成和验证 MAC 命令支持各种 MAC，包括 HMAC、CMAC、EMV MAC 等。对于 AS2805，中还定义了其他变体 AS2805.4.1。通常，在 AS2805 中，传入的消息使用此 MAC 进行验证，传出的消息也包括 MAC。

```
cat verify-mac.json
{
    "KeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6",
    "Mac": "86304058",
    "MessageData": "73D8BA54D3852951DAEA41",
    "VerificationAttributes": {
        "Algorithm": "AS2805_4_1"
    }
}
```

```
$ aws payment-cryptography-data verify-mac --cli-input-json file://verify-mac.json --
region ap-southeast-2
```

```
{
    "KeyIdentifier": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
qnobl5lghrzunce6",
    "KeyCheckValue": "2976E7"
}
```

中的安全性 AWS Payment Cryptography

的云安全 AWS 是重中之重。作为 AWS 客户，您将受益于专为满足大多数安全敏感组织要求而构建的数据中心和网络架构。

安全是您 AWS 和您的共同责任。责任共担模式将此描述为云的安全性和云中的安全性：

- 云安全 —AWS 负责保护在 AWS 云端运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。 Third-party 作为AWS 合规计划合规计划合。要了解适用于 AWS 支付加密的合规计划，请参阅按合规性计划划分的 AWS 范围内服务
- 云端安全 —您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本主题可帮助您了解在使用 AWS 支付密码学时如何应用分担责任模型。它向您展示了如何配置 AWS 支付加密以满足您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 AWS 支付加密资源。

主题

- 中的数据保护 AWS Payment Cryptography
- 弹性在 AWS Payment Cryptography
- 中的基础设施安全 AWS Payment Cryptography
- 正在连接到 AWS 通过 VPC 终端节点进行支付加密
- 使用混合后量子 TLS
- 的安全最佳实践 AWS Payment Cryptography

中的数据保护 AWS Payment Cryptography

分 AWS 担责任模型适用于 AWS 支付密码学中的数据保护。如本模型所述 AWS ，负责保护运行所有网络的全球基础设施 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅通用数据保护条例（ GDPR ）中心。

出于数据保护的目的，我们建议您保护 AWS 账户 证书并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅AWS CloudTrail 用户指南中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果在 AWS 通过命令行接口或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 终端节点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括当您使用控制台、API 或 AWS SDK AWS 服务 使用 AWS 支付加密或其他方式时 AWS CLI。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

AWS Payment Cryptography 存储和保护您的支付加密密钥，使其具有高度可用性，同时为您提供强大而灵活的访问控制。

主题

- 保护密钥材料
- 数据加密
- 静态加密
- 传输中加密
- 互联网络流量隐私

保护密钥材料

默认情况下，AWS Payment Cryptography 会保护该服务管理的支付密钥的加密密钥材料。此外，AWS Payment Cryptography 还提供用于导入在服务外部创建的密钥材料的选项。有关支付密钥和密钥材料的技术详细信息，请参阅 AWS Payment Cryptography 加密详细信息。

数据加密

AWS Payment Cryptography 中的数据包括 AWS Payment Cryptography 密钥、它们所代表的加密密钥材料及其使用属性。此密钥材料仅在其使用时以明文形式存在于 AWS Payment Cryptography 硬件安全模块 (HSM) 中。否则，密钥材料和属性将被加密并存储在持久性存储中。

AWS Payment Cryptography 生成或加载的支付密钥的密钥材料永远不会以未加密的形式离开 AWS Payment Cryptography HSM 的边界。它可以通过 AWS Payment Cryptography API 操作加密导出。

静态加密

AWS 支付加密为 PCI PTS HSM-listed HSM 中的支付密钥生成密钥材料。密钥材料未使用时，会利用 HSM 密钥进行加密，并写入耐久的持久性存储中。Payment Cryptography 密钥的密钥材料和保护密钥材料的加密密钥永远不会以明文形式离开 HSM。

Payment Cryptography 密钥的密钥材料的加密和管理完全由服务处理。

有关更多信息，请参阅 AWS Key Management Service 加密详情。

传输中加密

AWS 支付加密为支付密钥生成或加载的密钥材料绝不会以明文形式在 AWS 支付加密 API 操作中导出或传输。 AWS 支付密码学使用密钥标识符来表示 API 操作中的密钥。

但是，某些 API 操作会导出由先前共享或非对称密钥交换密钥加密的密钥。此外，客户可以使用 API 操作来为支付密钥导入密钥材料。

所有 AWS 支付加密 API 调用都必须签名并使用传输层安全 (TLS) 进行传输。 AWS 支付加密需要TLS 版本和由PCI定义为 “强密码学” 的密码套件。所有服务端点都支持 TLS 1.2—1.3 和混合后量子 TLS。

有关更多信息，请参阅 AWS Key Management Service 加密详情。

互联网络流量隐私

AWS 支付加密支持 AWS 管理控制台和一组 API 操作，使您能够创建和管理支付密钥并将其用于加密操作。

AWS 支付加密支持从您的私有网络到 AWS 的两种网络连接选项。

- Internet 上的 IPsec VPN 连接
- AWS Direct Connect，该服务通过标准的以太网光纤电缆将您的内部网络链接到 AWS Direct Connect 位置。

所有 Payment Cryptography API 调用必须使用传输层安全性协议 (TLS) 进行签名和传输。这些调用还需要一个现代化的密码套件，该套件支持完美向前保护。仅允许通过 AWS 内部网络从已知的 AWS Payment Cryptography API 主机向存储密钥的密钥材料的硬件安全模块 (HSM) 传输流量。

要从您的虚拟私有云 (VPC) 直接连接到 AWS Payment Cryptography，而不通过公共互联网发送流量，请使用由 AWS PrivateLink 提供支持的 VPC 终端节点。有关更多信息，请参阅通过 VPC 端点连接到 AWS Payment Cryptography。

AWS Payment Cryptography 还支持对传输层安全性协议 (TLS) 网络加密协议使用混合后量子密钥交换选项。当您连接到 AWS Payment Cryptography API 端点时，可以结合使用此选项与 TLS。

弹性在 AWS Payment Cryptography

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。各区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

区域隔离

AWS 支付加密是一项区域服务，可在多个地区使用。

AWS Payment Cryptography 的区域隔离设计可确保一个 Amazon Web Services 区域的可用性问题不会影响任何其他区域的 AWS Payment Cryptography 操作。AWS Payment Cryptography 旨在确保零计划停机，所有软件更新和扩展操作都在不知不觉中无缝执行。

AWS Payment Cryptography 服务水平协议 (SLA) 为所有 Payment Cryptography API 提供 99.99% 的服务承诺。为履行这一承诺，AWS Payment Cryptography 会确保执行 API 请求所需的所有数据和授权信息在接收该请求的所有区域主机上都可用。

AWS Payment Cryptography 基础设施会在每个区域的至少三个可用区 (AZ) 中复制。为确保多个主机故障不会影响 AWS Payment Cryptography 性能，AWS Payment Cryptography 旨在服务于来自区域中任何 AZ 的客户流量。

您对支付密钥属性或权限所做的更改将复制到该区域中的所有主机，以确保该区域中的任何主机都能正确处理后续请求。有关使用支付密钥的加密操作请求将会转发给某个 AWS Payment Cryptography 硬件安全模块（HSM）队列，其中任何一个模块都可以使用支付密钥执行操作。

Multi-tenant 设计

AWS Payment Cryptography 的多租户设计使其能够达到可用性 SLA，并保持较高的请求率，同时保护密钥和数据的保密性。

通过部署多个完整性控制执行机制，以确保实际用于执行加密操作的支付密钥始终是您为该操作指定的密钥。

Payment Cryptography 密钥的明文密钥材料受到全面保护。密钥材料在创建后将立即在 HSM 中加密，并且加密后的密钥材料会立即移动到安全的存储中。加密后的密钥仅在使用时才在 HSM 中检索和解密。明文密钥仅在完成加密操作所需的时间内驻留在 HSM 内存中。明文密钥材料永远不会离开 HSM；并且永远不会写入持久性存储。

要详细了解 AWS Payment Cryptography 使用的密钥保护机制，请参阅 AWS Payment Cryptography 加密详细信息。

中的基础设施安全 AWS Payment Cryptography

作为一项托管服务 AWS Payment Cryptography ，受《亚马逊网络服务：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用 AWS Payment Cryptography 通过网络进行访问。客户端必须支持传输层安全性协议（TLS）1.2 或更高版本。客户还必须支持具有完全向前保密 (PFS) 的密码套件，例如短暂的 (DHE) 或椭圆曲线短暂的 Diffie-Hellman (ECDHE)。 Diffie-Hellman 大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

物理主机的隔离

AWS Payment Cryptography 使用的物理基础设施的安全性受 Amazon Web Services：安全流程概述的物理和环境安全部分中所述的控制措施约束。您可以在上一节中列出的合规性报告和第三方审计结果中找到更多详细信息。

AWS 支付加密由专用的商用 PCI PTS HSM-listed 硬件安全模块 (HSM) 支持。AWS Payment Cryptography 的密钥材料仅存储在 HSM 上的易失存储器中，并且仅在 Payment Cryptography 密钥正在使用时才存储。HSM 位于 Amazon 数据中心内的访问控制机架中，对任何物理访问实施双重控制。

有关 AWS Payment Cryptography HSM 的操作的详细信息，请参阅 AWS Payment Cryptography 加密详细信息。

正在连接到 AWS 通过 VPC 终端节点进行支付加密

您可以通过虚拟私有云 (VPC) 中的私有接口终端节点直接连接到 AWS 支付加密。当您使用接口 VPC 终端节点时，您的 VPC 和 AWS 支付加密之间的通信完全在 AWS 网络内进行。

AWS 支付加密支持由AWS PrivateLink支持的亚马逊虚拟私有云（亚马逊 VPC）终端节点。每个 VPC 端点都由您的 VPC 子网中一个或多个使用私有 IP 地址的弹性网络接口 (ENI) 代表。

接口 VPC 终端节点将您的 VPC 直接连接到 AWS 支付加密，无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。您的 VPC 中的实例不需要公有 IP 地址即可与 AWS 支付加密通信。

Regions

AWS 支付密码学支持所有 AWS 区域 支持AWS 支付加密的 VPC 终端节点和 VPC 终端节点策略。

主题

- 注意事项 AWS 支付加密 VPC 终端节点
- 为创建 VPC 终端节点 AWS Payment Cryptography
- 正在连接到 AWS 支付加密 VPC 终端节点
- 控制对 VPC 端点的访问
- 在策略语句中使用 VPC 端点
- 记录您的 VPC 端点

注意事项 AWS 支付加密 VPC 终端节点

> **Note**
>
> 尽管 VPC 终端节点允许您在最短的一个可用区 (AZ) 内连接到该服务，但我们建议您连接到三个可用区以实现高可用性和冗余性。

在为 AWS 支付加密设置接口 VPC 终端节点之前，请查看AWS PrivateLink 指南中的接口终端节点属性和限制主题。

AWS VPC 终端节点的支付加密支持包括以下内容。

- 您可以使用您的 VPC 终端节点从 VPC 调用所有AWSAWS 支付加密控制平面操作和支付加密数据平面操作。
- 您可以创建连接 AWS 支付加密区域终端节点的接口 VPC 终端节点。
- AWS 支付密码学由控制平面和数据平面组成。您可以选择设置一个或两个子服务，AWS PrivateLink 但每个子服务都是单独配置的。
- 您可以使用 AWS CloudTrail 日志来审核通过 VPC 终端节点对 AWS 支付加密密钥的使用情况。有关更多信息，请参阅 记录您的 VPC 端点。

为创建 VPC 终端节点 AWS Payment Cryptography

您可以使用亚马逊 VPC 控制台或亚马逊 VPC API 为 AWS 支付加密创建 VPC 终端节点。有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

- 要为 AWS 支付加密创建 VPC 终端节点，请使用以下服务名称：

```
com.amazonaws.region.payment-cryptography.controlplane
```

```
com.amazonaws.region.payment-cryptography.dataplane
```

例如，在美国西部（俄勒冈）区域 (us-west-2) 中，服务名称将是：

```
com.amazonaws.us-west-2.payment-cryptography.controlplane
```

```
com.amazonaws.us-west-2.payment-cryptography.dataplane
```

为了更轻松地使用 VPC 端点，您可以为 VPC 端点启用私有 DNS 名称。如果您选择启用 DNS 名称选项，则标准 AWS 支付加密 DNS 主机名将解析到您的 VPC 终端节点。例如，`https://controlplane.payment-cryptography.us-west-2.amazonaws.com` 将解析为连接到服务名称 `com.amazonaws.us-west-2.payment-cryptography.controlplane` 的 VPC 端点。

此选项可让您更轻松地使用 VPC 端点。默认情况下，AWS 软件开发工具包和 AWS CLI 使用标准 AWS 支付加密 DNS 主机名，因此您无需在应用程序和命令中指定 VPC 终端节点 URL。

有关更多信息，请参阅 AWS PrivateLink 指南中的通过接口端点访问服务。

正在连接到 AWS 支付加密 VPC 终端节点

您可以使用 AWS SDK AWS CLI 或 AWS Tools for PowerShell，通过 VPC 终端节点连接到 AWS 支付加密。要指定 VPC 端点，请使用其 DNS 名称。

例如，此 list-keys 命令使用 `endpoint-url` 参数指定 VPC 终端节点。要使用类似命令，请将示例中的 VPC 终端节点 ID 替换为您账户中的 ID。

```
$ aws payment-cryptography list-keys --endpoint-url https://
vpce-1234abcdf5678c90a-09p7654s-us-east-1a.ec2.us-east-1.vpce.amazonaws.com
```

如果在创建 VPC 端点时启用了私有主机名，则无需在 CLI 命令或应用程序配置中指定 VPC 端点 URL。标准 AWS 支付加密 DNS 主机名解析到您的 VPC 终端节点。 AWS CLI 和 SDK 默认使用此主机名，因此您可以开始使用 VPC 终端节点连接到 AWS 支付加密区域终端节点，而无需更改脚本和应用程序中的任何内容。

要使用私有主机名，您的 VPC 的 `enableDnsHostnames` 和 `enableDnsSupport` 属性必须设置为 `true`。要设置这些属性，请使用ModifyVpcAttribute操作。有关详细信息，请参阅《Amazon VPC 用户指南》中的查看和更新 VPC 的 DNS 属性。

控制对 VPC 端点的访问

要控制对您的 AWS 支付加密的 VPC 终端节点的访问权限，请将 VPC 终端节点策略附加到您的 VPC 终端节点。终端节点策略确定委托人是否可以使用 VPC 终端节点使用特定的 AWS 支付加密资源调用 AWS 支付加密操作。

您可以在创建终端节点时创建 VPC 端点策略，并且可以随时更改 VPC 端点策略。使用 VPC 管理控制台或CreateVpcEndpoint或ModifyVpcEndpoint操作。您还可以使用 AWS CloudFormation 模板创建和更改 VPC 终端节点策略。有关使用 VPC 管理控制台的帮助，请参阅 AWS PrivateLink 指南中的创建接口终端节点和修改接口终端节点。

有关编写和格式化 JSON 策略文档的帮助，请参阅 IAM 用户指南中的 IAM JSON 策略参考。

主题

- 关于 VPC 终端节点策略
- 默认的 VPC 端点策略
- 创建 VPC 端点策略
- 查看 VPC 终端节点策略

关于 VPC 终端节点策略

为了使使用 VPC 终端节点的 AWS 支付加密请求成功，委托人需要两个来源的权限：

- 基于身份的策略必须授予委托人对资源（AWS 支付加密密钥或别名）调用操作的权限。
- VPC 端点策略必须授予委托人使用终端节点发出请求的权限。

例如，密钥策略可能会授予委托人对特定的 AWS 支付加密密钥调用 Decrypt 的权限。但是，VPC 终端节点策略可能不允许该委托人使用该终端节点调用该 AWS 支付加密密钥。`Decrypt`

或者，VPC 终端节点策略可能允许委托人使用终端节点调用某些 AWS 支付加密密钥。`StopKeyUsage` 但是，如果委托人没有 IAM 策略中的这些权限，则请求将失败。

默认的 VPC 端点策略

每个 VPC 端点都有 VPC 端点策略，但您无需指定策略。如果未指定策略，则默认的终端节点策略允许所有委托人对终端节点上的所有资源执行所有操作。

但是，对于 AWS 支付加密资源，委托人还必须有权根据 IAM策略调用该操作。因此，在实践中，默认策略表示，如果委托人有权对资源调用操作，他们也可以通过使用终端节点调用该操作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

要允许主体仅将 VPC 终端节点用于其允许操作的子集，请创建或更新改 VPC 终端节点策略。

创建 VPC 端点策略

VPC 端点策略确定委托人是否有权使用 VPC 端点对资源执行操作。对于 AWS 支付加密资源，委托人还必须有权根据 IAM政策执行操作。

每个 VPC 端点策略语句都需要以下元素：

- 可执行操作的委托人
- 可执行的操作
- 可对其执行操作的资源

策略语句不指定 VPC 端点。它适用于策略所附加到的任何 VPC 端点。有关更多信息，请参阅《Amazon VPC User Guide》中的 Controlling access to services with VPC endpoints。

以下是 AWS 支付加密的 VPC 终端节点策略示例。当连接到 VPC 终端节点时，此策略`ExampleUser`允许使用 VPC 终端节点对指定的 AWS 支付加密密钥调用指定操作。在使用此类政策之前，请将示例主体和密钥标识符替换为账户中的有效值。

```
{
   "Statement":[
      {
         "Sid": "AllowDecryptAndView",
         "Principal": {"AWS": "arn:aws:iam::111122223333:user/ExampleUser"},
         "Effect":"Allow",
         "Action": [
             "payment-cryptography:Decrypt",
             "payment-cryptography:GetKey",
             "payment-cryptography:ListAliases",
             "payment-cryptography:ListKeys",
             "payment-cryptography:GetAlias"
          ],
         "Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
      }
   ]
}
```

AWS CloudTrail 记录使用 VPC 终端节点的所有操作。但是，您的 CloudTrail 日志不包括委托人在其他账户中请求的操作或其他账户中对 AWS 支付加密密钥的操作。

因此，您可能需要创建一个 VPC 终端节点策略，以防止外部账户中的委托人使用 VPC 终端节点对本地账户中的任何密钥调用任何 AWS 支付加密操作。

以下示例使用 a ws: PrincipalAccount 全局条件密钥拒绝所有委托人访问所有 AWS 支付加密密钥的所有操作，除非委托人在本地账户中。使用类似于此策略的策略之前，请使用有效值替换示例账户 ID。

```
{
  "Statement": [
```

```
    {
      "Sid": "AccessForASpecificAccount",
      "Principal": {"AWS": "*"},
      "Action": "payment-cryptography:*",
      "Effect": "Deny",
      "Resource": "arn:aws:payment-cryptography:*:111122223333:key/*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalAccount": "111122223333"
        }
      }
    }
  ]
}
```

查看 VPC 终端节点策略

要查看终端节点的 VPC 终端节点策略，请使用 VPC 管理控制台或DescribeVpcEndpoints操作。

以下 AWS CLI 命令获取具有指定 VPC 终端节点 ID 的终端节点的策略。

在使用此命令之前，请将示例终端节点 ID 替换为您账户中的有效终端节点 ID。

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpce-1234abcdf5678c90a`].[PolicyDocument]'
--output text
```

在策略语句中使用 VPC 端点

当请求来自 VPC 或使用 VPC 终端节点时，您可以控制对 AWS 支付加密资源和操作的访问权限。为此，请使用一个 IAM 策略

- 使用 aws:sourceVpce 条件键基于 VPC 端点授予或限制访问。
- 使用 aws:sourceVpc 条件键基于托管私有终端节点的 VPC 授予或限制访问。

> **Note**
>
> 当请求来自亚马逊 VPC 终端节点时，aws:sourceIP条件密钥无效。要限制对 VPC 端点的请求，请使用 aws:sourceVpce 或 aws:sourceVpc 条件键。有关更多信息，请参阅《AWS PrivateLink 指南》中的 VPC 端点和 VPC 端点服务的身份和访问管理。

您可以使用这些全局条件密钥来控制对 AWS 支付加密密钥、别名的访问权限，以及对不依赖于任何特定CreateKey资源的此类操作的访问权限。

例如，以下示例密钥策略仅允许用户在请求使用指定的 VPC 终端节点时使用 AWS 支付加密密钥执行特定的加密操作，从而阻止来自互联网和 AWS PrivateLink 连接（如果已设置）的访问。当用户向 AWS 支付加密系统发出请求时，会将请求中的 VPC 终端节点 ID 与策略中的`aws:sourceVpce`条件密钥值进行比较。如果它们不匹配，则请求会被拒绝。

要使用这样的策略，请将占位符 AWS 账户 ID 和 VPC 终端节点 ID 替换为账户的有效值。

JSON

```
{
    "Id": "example-key-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnableIAMPolicies",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root"
                ]
            },
            "Action": [
                "payment-cryptography:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RestrictUsageToMyVPCEndpoint",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "payment-cryptography:EncryptData",
                "payment-cryptography:DecryptData"
            ],
            "Resource": "arn:aws:payment-cryptography:us-east-1:111122223333:key/
*",
            "Condition": {
                "StringNotEquals": {
                    "aws:sourceVpce": "vpce-1234abcdf5678c90a"
```

```
                }
            }
        }

    ]
}
```

您还可以根据 VPC 终端节点所在的 VPC 使用aws:sourceVpc条件密钥来限制对 AWS 付款加密密钥的访问。

以下示例密钥策略允许命令仅在 AWS 支付加密密钥来自vpc-12345678时对其进行管理。此外，它允许使用 AWS 支付加密密钥的命令仅在它们来自时进行加密操作。vpc-2b2b2b2b如果应用程序在一个 VPC 中运行，但您使用第二个隔离的 VPC 执行管理功能，则可以使用这样的策略。

要使用这样的策略，请将占位符 AWS 账户 ID 和 VPC 终端节点 ID 替换为账户的有效值。

JSON

```
{
    "Id": "example-key-2",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAdminActionsFromVPC12345678",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": [
               "payment-cryptography:Create*",
               "payment-cryptography:Encrypt*",
               "payment-cryptography:ImportKey*",
               "payment-cryptography:GetParametersForImport*",
               "payment-cryptography:TagResource",
               "payment-cryptography:UntagResource"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-12345678"
                }
```

```
            }
        },
        {
            "Sid": "AllowKeyUsageFromVPC2b2b2b2b",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": [
                "payment-cryptography:Encrypt*",
                "payment-cryptography:Decrypt*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpc": "vpc-2b2b2b2b"
                }
            }
        },
        {
            "Sid": "AllowListReadActionsFromEverywhere",
            "Effect": "Allow",
            "Principal": {
                "AWS": "111122223333"
            },
            "Action": [
                "payment-cryptography:List*",
                "payment-cryptography:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

记录您的 VPC 端点

AWS CloudTrail 记录使用 VPC 终端节点的所有操作。当 AWS 支付加密请求使用 VPC 终端节点时，VPC 终端节点 ID 会出现在记录该请求的AWS CloudTrail 日志条目中。您可以使用终端节点 ID 来审核您的 AWS 支付加密 VPC 终端节点的使用情况。

为了保护您的 VPC，VPC 终端节点策略拒绝但本应允许的请求不会记录在 VPC 中AWS CloudTrail。

例如，此示例日志条目记录了使用 VPC 终端节点的 GenerateMac 请求。vpcEndpointId 字段出现在日志条目的末尾。


```
{
      "eventVersion": "1.08",
      "userIdentity": {
          "principalId": "TESTXECZ5U9M4LGF2N6Y5:i-98761b8890c09a34a",
          "arn": "arn:aws:sts::111122223333:assumed-role/samplerole/
i-98761b8890c09a34a",
          "accountId": "111122223333",
          "accessKeyId": "TESTXECZ5U2ZULLHHMJG",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "TESTXECZ5U9M4LGF2N6Y5",
                  "arn": "arn:aws:iam::111122223333:role/samplerole",
                  "accountId": "111122223333",
                  "userName": "samplerole"
              },
              "webIdFederationData": {},
              "attributes": {
                  "creationDate": "2024-05-27T19:34:10Z",
                  "mfaAuthenticated": "false"
              },
              "ec2RoleDelivery": "2.0"
          }
      },
      "eventTime": "2024-05-27T19:49:54Z",
      "eventSource": "payment-cryptography.amazonaws.com",
      "eventName": "CreateKey",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "172.31.85.253",
      "userAgent": "aws-cli/2.14.5 Python/3.9.16 Linux/6.1.79-99.167.amzn2023.x86_64
 source/x86_64.amzn.2023 prompt/off command/payment-cryptography.create-key",
      "requestParameters": {
          "keyAttributes": {
              "keyUsage": "TR31_M1_ISO_9797_1_MAC_KEY",
              "keyClass": "SYMMETRIC_KEY",
              "keyAlgorithm": "TDES_2KEY",
              "keyModesOfUse": {
                  "encrypt": false,
                  "decrypt": false,
                  "wrap": false,
```

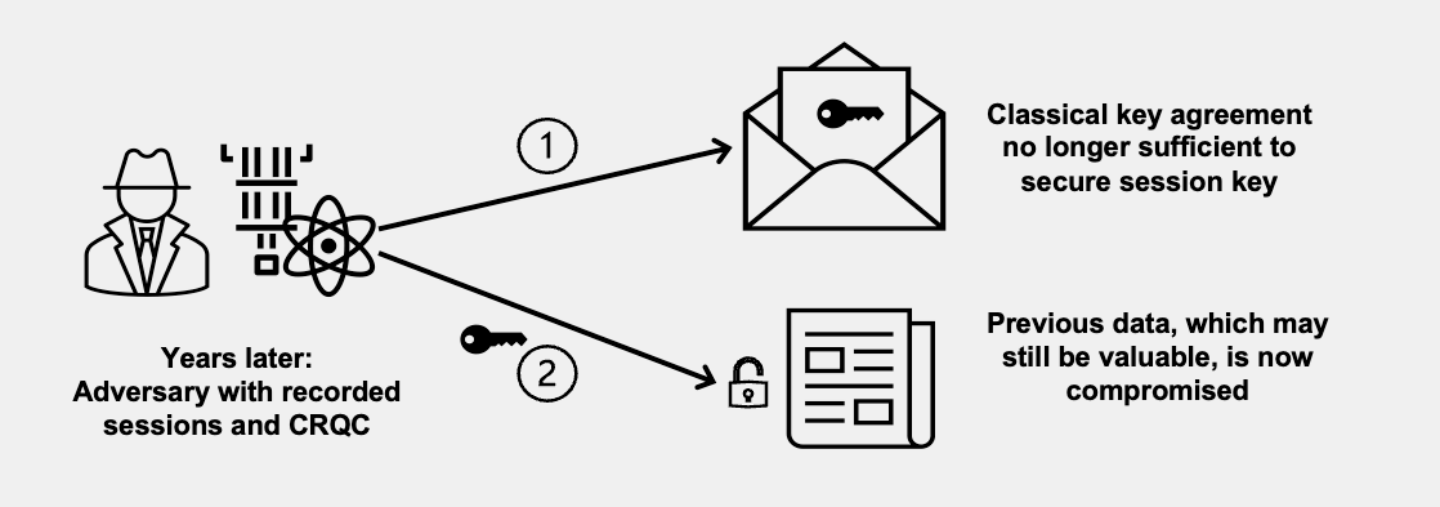
```
                    "unwrap": false,
                    "generate": true,
                    "sign": false,
                    "verify": true,
                    "deriveKey": false,
                    "noRestrictions": false
                }
            },
            "exportable": true
        },
        "responseElements": {
            "key": {
                "keyArn": "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h",
                "keyAttributes": {
                    "keyUsage": "TR31_M1_ISO_9797_1_MAC_KEY",
                    "keyClass": "SYMMETRIC_KEY",
                    "keyAlgorithm": "TDES_2KEY",
                    "keyModesOfUse": {
                        "encrypt": false,
                        "decrypt": false,
                        "wrap": false,
                        "unwrap": false,
                        "generate": true,
                        "sign": false,
                        "verify": true,
                        "deriveKey": false,
                        "noRestrictions": false
                    }
                },
                "keyCheckValue": "A486ED",
                "keyCheckValueAlgorithm": "ANSI_X9_24",
                "enabled": true,
                "exportable": true,
                "keyState": "CREATE_COMPLETE",
                "keyOrigin": "AWS_PAYMENT_CRYPTOGRAPHY",
                "createTimestamp": "May 27, 2024, 7:49:54 PM",
                "usageStartTimestamp": "May 27, 2024, 7:49:54 PM"
            }
        },
        "requestID": "f3020b3c-4e86-47f5-808f-14c7a4a99161",
        "eventID": "b87c3d30-f3ab-4131-87e8-bc54cfef9d29",
        "readOnly": false,
        "eventType": "AwsApiCall",
```

```
      "managementEvent": true,
      "recipientAccountId": "111122223333",
      "vpcEndpointId": "vpce-1234abcdf5678c90a",
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.3",
          "cipherSuite": "TLS_AES_128_GCM_SHA256",
          "clientProvidedHostHeader": "vpce-1234abcdf5678c90a-
oo28vrvr.controlplane.payment-cryptography.us-east-1.vpce.amazonaws.com"
      }
  }
```

使用混合后量子 TLS

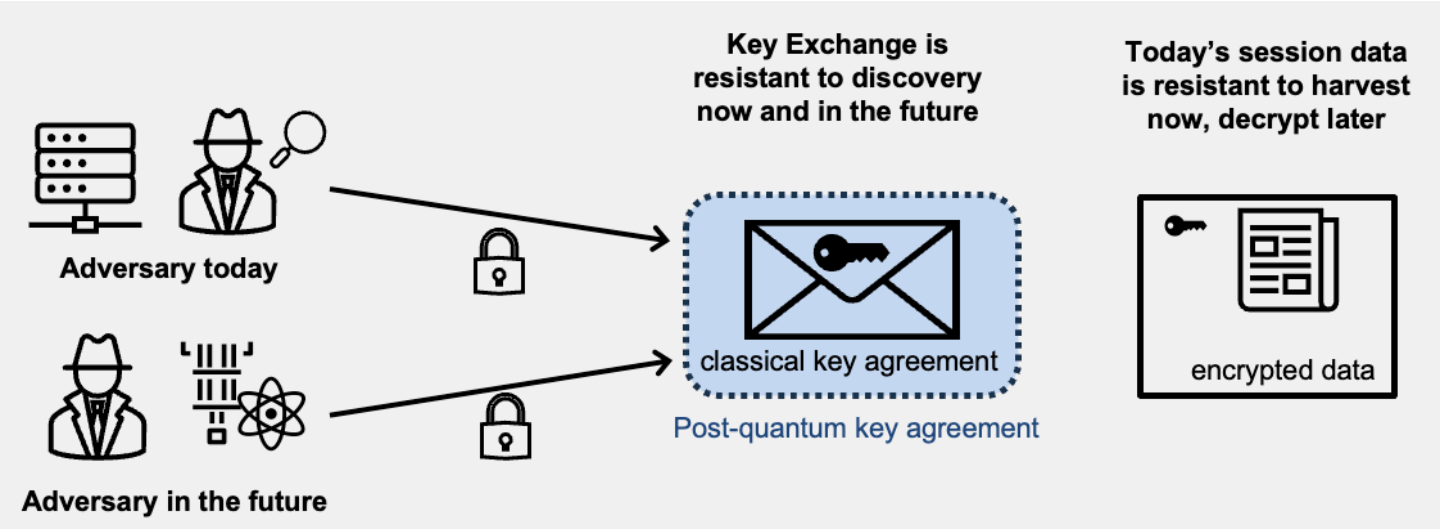
AWS 支付密码学和许多其他服务支持传输层安全 (TLS) 网络加密协议的混合后量子密钥交换选项。当您连接到 API 终端节点或使用 AWS SDK 时，您可以使用此 TLS 选项。这些混合后量子密钥交换功能是可选的，至少与我们目前使用的 TLS 加密一样安全，并且有可能会提供额外的长期安全优势。

您发送到已启用的服务的数据在传输过程中受到传输层安全 (TLS) 连接提供的加密保护。 AWS 支付密码学支持TLS会话的基于RSA和ECC的经典密码套件，使得当前技术无法对密钥交换机制进行暴力攻击。但是，如果将来大规模或与密码学相关的量子计算机（CRQC）变得可行，则现有的 TLS 密钥交换机制将容易受到这些攻击。对手有可能现在开始收集加密数据，希望将来能够解密这些数据（立即收集，稍后解密）。如果您开发的应用程序依赖于通过 TLS 连接传递的数据的长期机密性，则应考虑在大型量子计算机可供使用之前迁移到后量子加密的计划。 AWS 正在努力为未来做准备，我们也希望你做好充分的准备。



为了保护当今加密的数据免受未来潜在攻击， AWS 正在与密码学界一起参与抗量子算法或后量子算法的开发。 AWS 已经实现了混合后量子密钥交换密码套件，这些套件结合了经典和后量子元素，以确保您的TLS连接至少与经典密码套件一样强大。

使用最新版本的 SDK 时，这些混合密码套件可用于您的生产工作负载。 AWS 有关如何执行 enable/disable 此行为的更多信息，请参阅启用混合后量子 TLS 。



关于 TLS 中的混合后量子密钥交换

AWS 使用的算法是混合算法，它将椭圆曲线 Diffie-Hellman（ECDH）（当今在TLS中使用的经典密钥交换算法）与Mechanim（ML-KEM）（美国国家标准与Module-Lattice-Based Key-Encapsulation 技术研究所（NIST）指定为其第一个标准后量子密钥协议算法的公钥加密和密钥建立算法相结合。此混合算法单独使用每个算法来生成密钥。然后，以加密方式结合使用这两个密钥。

使用方法

有关在 AWS SDK 中启用混合后量子 TLS 的信息，请参阅启用混合后量子 TLS。

了解有关 PQC 的更多信息

有关美国国家标准与技术研究所（NIST）后量子密码学项目的信息，请参阅密码学。 Post-Quantum

有关 NIST 后量子密码学标准化的信息，请参阅Post-Quantum 密码学标准化。

启用混合后量子 TLS

本主题已移至《AWS 软件开发工具包和工具参考指南》。

的安全最佳实践 AWS Payment Cryptography

AWS Payment Cryptography 支持许多内置安全功能，或者您可以选择实施这些功能，以增强对加密密钥的保护并确保它们用于预期目的，包括 IAM 政策、用于完善密钥策略和 IAM 政策的大量策略条件密钥以及有关密钥区块的 PCI PIN 规则的内置强制执行。

> **Important**
>
> 提供的这些一般准则并不代表完整的安全解决方案。由于并非所有最佳实践都适用于所有情况，因此这些做法并不是规范性的。

- 密钥用法和使用模式： AWS 支付加密遵循并强制执行密钥使用和使用方式限制，如 ANSI X9 TR 31-2018 互操作安全密钥交换密钥区块规范中所述，并符合 PCI PIN 安全要求 18-3。这限制了将单个密钥用于多种目的的能力，并以加密方式将密钥元数据（例如允许的操作）绑定到密钥材料本身。 AWS 支付加密会自动强制执行这些限制，例如密钥加密密钥（TR31_K0_KEY_ENCRYPTION_KEY）也不能用于数据解密。有关更多信息，请参阅了解的关键属性 AWS 支付加密密钥。
- 限制对称密钥材料的共享：最多只能与其他一个实体共享对称密钥材料（例如 Pin 加密密钥或密钥加密密钥）。如果需要将敏感材料传输给更多实体或合作伙伴，请创建其他密钥。 AWS 支付密码学从不公开对称密钥材料或非对称私钥材料。
- 使用别名或标签将密钥与特定用例或合作伙伴关联起来：别名可用于轻松表示与密钥相关的用例，例如 alias/BIN_12345_CVK 表示与 BIN 12345 关联的卡验证密钥。为了提供更大的灵活性，可以考虑创建诸如 bin=12345、use_case=acquiring、country=us,partner=foo 之类的标签。别名和标签还可用于限制访问权限，例如在发布和获取用例之间实施访问控制。
- 实行最低权限访问：IAM 可用于限制对系统而非个人的生产访问，例如禁止个人用户创建密钥或运行加密操作。IAM 还可用于限制对可能不适用于您的用例的命令和密钥的访问权限，例如限制为收单机构生成或验证密码的能力。使用最低权限访问的另一种方法是将敏感操作（例如密钥导入）限制为特定的服务账户。有关示例，请参阅 AWS 支付密码学基于身份的策略示例。

另请参阅

- 的身份和访问管理 AWS Payment Cryptography
- IAM 用户指南中的 IAM 安全最佳实践

AWS 支付密码学的合规性验证

与其他 AWS 服务一样，客户需要清楚地了解安全性和合规性的分担责任模型。作为一项专门支持支付的服务，对于 AWS 支付密码学客户来说，了解遵守适用的PCI标准尤为重要。 AWS PCI DSS 和 PCI 3DS 评估包括支付密码学。 AWS 这些报告的《分担责任指南》中可能会提及该服务，该指南可从中AWS Artifact获得。PCI PIN 安全与 Point-to-Point加密 (P2PE) 评估特定于 AWS 支付加密。

本节提供有关服务合规状态和范围的信息，以及有助于规划应用程序的 PCI PIN 安全和 PCI P2PE 评估的信息。

主题

- 服务的合规性
- PIN 合规规划
- 在 P2PE 解决方案中使用 AWS 支付密码学解密组件

服务的合规性

作为多个合规计划的一部分，第三方审计师评估 AWS 支付密码学的安全 AWS 性和合规性。其中包括 SOC、PCI 等。

AWS 除了 PCI DSS 和 PCI 3DS 之外，还针对多个 PCI 标准对支付密码学进行了评估。其中包括 PCI PIN 安全 (PCI PIN) 和 PCI Point-to-Point (P2PE) 加密。有关可 AWS Artifact 用的认证和合规指南，请参阅。

有关特定合规计划范围内的 AWS 服务列表，请参阅按合规计划划分的 AWS 范围内的服务 AWS 按合规计划。有关常规信息，请参阅 AWS 合规性计划、、。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 支付密码学时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全性与合规性 Quick Start 指南：这些部署指南讨论了架构注意事项，并提供了在 AWS上部署基于安全性和合规性的基准环境的步骤。
- AWS 合规资源AWS-此工作簿和指南集合可能适用于您所在的行业和所在地区。

- AWS Config 开发人员指南中的使用规则评估资源 –AWS Config；评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub CSPM—此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

PIN 合规规划

本指南描述了您需要准备的文档和证据，以便对使用 AWS 支付密码学的 PIN 处理应用程序进行 PCI PIN 评估。

与其他 AWS 服务 合规标准一样，您有责任安全地使用服务，配置访问控制并使用符合 PCI PIN 要求的安全参数。本指南将在满足要求时讨论这些配置。

主题

- 常见话题
- 评估范围
- 交易处理操作

常见话题

将应用程序从连接到 HSM 迁移到托管服务（例如 AWS 支付密码学）会给客户及其评估人员带来常见的问题和概念。本节提供信息以阐明安全使用该服务是如何解决这些情况的。

主题

- 责任共担
- 最低 HSM 配置
- 客户和 APC 之间的密钥交换

责任共担

对应用程序承担全部安全和合规责任的客户将调整其合规性，以利用 P AWS ayment Cryptography 的密钥管理、安全控制和托管 HSM 功能（“服务”）。正如Payment Cryptography的第三方评估所 AWS 证明的那样 AWS，这将完全将某些要求转移到其他方面。一些要求将在客户的应用程序和服务之间共享。应用程序负责：

- 为服务提供准确的信息

- 根据服务建议和 PCI PIN 安全要求使用安全控制
- 使用服务提供的工具实施所需的安全控制

客户及其评估人员将使用发布的责任共担和实施指南以及合规性证明 AWS Artifact 来实施控制和控制监控，然后规划和完成评估。

最低 HSM 配置

PCI 数据安全标准是其他 PCI 标准的基础标准，它要求所有系统都必须配置为其功能所需的最低功能。PCI PIN、P2PE 和其他解决方案标准将此要求应用于解决方案 HSMs 。 HSMs 只能启用解决方案所需的功能。

AWS 应将服务视为系统，并配置为最低要求的功能。AWS 上的支付卡行业数据安全标准 (PCI DSS) v 4.0 建议使用 IAM 为解决方案使用的每项 AWS 服务配置最低功能。这也适用于 AWS 支付密码学。IAM 策略允许细粒度权限，将加密功能仅限于依赖加密功能的应用程序组件。

客户和 APC 之间的密钥交换

PIN PIN 安全要求 8-4 和 15-2 要求交换和加载密钥的公钥必须经过身份验证并保护完整性。对于 POI 的远程密钥加载（在 ANSI/ASC X9 TR-34 中进行了功能描述，受 PCI PIN 附录 A 管辖），公钥通常以符合附件 A2 的证书颁发机构签署的证书的形式传送。对于组织之间的交换，公钥使用其他机制来保证真实性和完整性。

客户与 AWS 之间的所有交互均通过 AWS 进行 APIs，AWS 使用 TLS 相互验证每个 API 调用，并确保呼叫和响应的完整性。客户应用程序的身份验证由 AWS Identity and Access Management 通过安全令牌和 Sigv4 等机制进行管理。AWS API 终端节点由客户使用 AWS 内置的 TLS 服务器身份验证进行身份验证 SDKs。然后，TLS 可确保在客户和每个 AWS API 之间传递的所有数据的机密性和完整性。

APC APIs GetParametersForImport 并 ImportKey 实施从客户到服务的密钥传输。虽然提供的 GetParametersForImport 证书颁发机构 (CA) 不符合附录 A2，但它是安全的，并且对账户来说是独一无二的。虽然不能依靠此 CA 来满足要求 8-4 和 15-2，但它确实为导入的密钥提供了完整性验证。您也可以利用 GetCertificateSigningRequest API 使用自己的 CA。

提供公钥身份验证和完整性保证的机制有：

- AWS API 身份验证提供的身份验证
- 即使证书中的身份信息不受信任 GetParametersForImport，密钥的完整性也由提供的证书的 MAC 功能提供。TLS 使用的 MAC 保护客户与 AWS 之间的会话，这也保证了密钥的完整性

APC 提供的证书和密钥块符合附录 A1，其中规定了通过非对称方法进行证书和密钥保护的要求。

评估范围

规划任何评估的第一步是记录范围。对于 PCI PIN，范围是保护 PIN 的系统和流程，包括保护加密密钥和保护密码的设备——支付终端，也称为交互点 (POI)、HSM 和其他安全加密设备 (SCD)。

我们不会满足您保留全部责任的要求，因为这些要求涉及的领域超出了服务范围。例如，支付终端的配置和配置。请参阅 PCI PIN 的《 AWS 支付密码学分担责任指南》，网址为 AWS Artifact

主题

- 责任共担
- High-Level 网络图
- 钥匙表
- 文档参考

责任共担

AWS Payment Cryptography是一个加密和支持组织（ESO）和 PIN-Acquiring Third-Party 服务商（TPS），由Visa P IN安全计划定义，并在Visa 全球服务提供商注册处的 “Amazon Web Services, LLC” 下列出。这意味着Visa允许 PIN-Acquiring Third-Party VisaNet 处理器（VNP）、作为服务提供商的 PIN-Acquiring 客户 VisaNet 处理器以及其他TPS和ESO提供商使用该服务，而无需客户PIN评估员（符合PCI资格的PIN评估员或PCI QPA）的进一步评估。

其他信用卡品牌或支付网络提供商可能依赖Visa PIN安全计划或拥有自己的计划。有关其他支付网络计划的服务合规性问题，请联系我们 AWS 支持 。

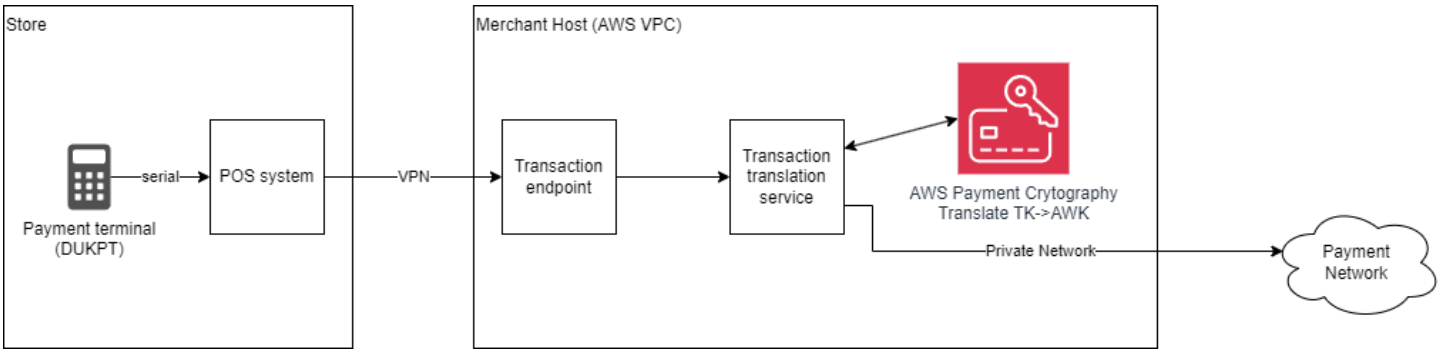
AWS 提供了 PCI PIN 安全合规性证明 (AOC) 和支付加密分担责任指南。 AWS AWS Artifact多年来，使用服务提供商进行PIN处理已很普遍，但是，3.1版之前的PCI PIN安全标准并未涉及第三方服务提供商的管理。Visa PIN 安全计划也没有。客户 QPA 遵循了 PCI DSS AOC 和《分担责任指南》中建立的模型，即将 AWS“合规性视为成功通过适用要求测试”。

High-Level 网络图

PCI PIN 报告模板要求：“对于参与处理基于 PIN 的交易的实体，请提供网络示意图，描述基于 PIN 的交易流以及相关的密钥类型用法。此外，使用非对称技术进行远程密钥分发的KIF和实体应提供密钥信息流”

AWS Payment Cryptography 报告了我们的 PCI PIN 评估的内部服务结构。您的图表将说明如何调用服务 API 进行 PIN 处理。

使用 AWS 支付加密的 PIN 应用程序的高级网络图示例：



钥匙表

该报告要求列出所有直接或间接保护 PIN 的密钥。服务中存在的任何密钥都可以通过 ListKeysAP I 列出。

请务必提供拥有应用程序密钥的所有区域和账户的密钥列表。

文档参考

《用户指南》和《API 参考》中提供了有关安全使用 AWS 支付密码的供应商文档和建议。本指南中酌情链接了这些内容。

交易处理操作

PCI PIN 要求按控制目标进行组织。每个控制目标都对保护 PIN 安全性的要求进行了分组。

主题

- 控制目标 1：受这些要求约束的交易中使用的 PIN 使用确保其安全的设备和方法进行处理。
- 控制目标 2：用于 PIN encryption/decryption 和相关密钥管理的加密密钥的创建过程可确保无法预测任何密钥或确定某些密钥比其他密钥更有可能。
- 控制目标 3：以安全的方式传送或传输密钥。
- 控制目标 4： Key-loading 以安全的方式处理 HSM 和 POI PIN-acceptance 设备。
- 控制目标 5：密钥的使用方式可以防止或检测未经授权的使用。
- 控制目标 6：以安全的方式管理密钥。

- 控制目标 7：以安全的方式管理用于处理 PIN 和密钥的设备。

控制目标 1：受这些要求约束的交易中使用的 PIN 使用确保其安全的设备和方法进行处理。

要求1：作为我们PCI PIN评估的一部分，对 AWS 支付密码学使用的HSM进行了评估。对于使用该服务的客户，相对于服务管理的 HSM，要求 1-3 和 1-4 是 “到位的”。HSM的调查结果将表明，测试已得到QPA的证实。 AWS PIN 合规性证明可供参考。 AWS Artifact需要对解决方案中的其他 SCD（例如 POI）进行清点和引用。

要求 2：您的程序文档必须具体说明如何保护持卡人 PIN 以防止向您的人员泄露、实施的 PIN 转换协议以及在线和离线处理期间的保护。此外，您的文档应包含每个区域中使用的加密密钥管理方法的摘要。

要求 3：必须配置 POI 以进行安全的 PIN 加密和传输。 AWS 支付密码学仅支持要求 3-3 中指定的 PIN 区块翻译。

要求 4：应用程序不得存储 PIN 块。即使是加密的 PIN 块，也不得保留在交易日志或日志中。该服务不存储 PIN 块，PIN 评估会验证它们不在日志中。

请注意，如该标准所述，PCI PIN 安全标准适用于 “在自动柜员机和销售点 (POS) 终端的在线和离线支付卡交易处理期间安全管理、处理和传输个人识别码 (PIN) 数据”。但是，该标准通常用于评估超出预期范围的付款的加密密钥管理。这可能包括存储 PIN 的发行人用例。这些案例要求的例外情况应与评估的目标受众商定。

控制目标 2：用于 PIN encryption/decryption 和相关密钥管理的加密密钥的创建过程可确保无法预测任何密钥或确定某些密钥比其他密钥更有可能。

要求 5：通过 AWS 支付密码生成密钥已作为我们的 PCI PIN 评估的一部分进行了评估。这可以在密钥表 “生成者” 列中指定。

要求 6：作为该服务 PCI PIN 评估的一部分，对 AWS 支付密码学中持有的密钥的安全控制进行了评估。包括与应用程序内密钥生成相关的安全控制措施以及与任何其他服务提供商的密钥生成相关的安全控制措施的描述。

要求 7：您必须有一份密钥生成策略文档，其中应说明密钥是如何生成的，并且所有受影响方都必须了解这些信息。 procedures/policies使用 APC API 创建密钥的程序应包括使用具有密钥创建权限和批准运行脚本或其他创建密钥的代码的角色。 AWS CloudTrail 日志包含带有日期和时间、密钥 ARN 和用

户 ID 的所有CreateKey事件。作为该服务 PIN 评估的一部分，对 HSM 序列号和访问物理媒体的日志进行了评估。

控制目标 3：以安全的方式传送或传输密钥。

要求 8：作为我们的 PCI PIN 评估的一部分，使用 AWS 支付密码学对密钥传输进行了评估。在从 AWS Payment Cryptography 导入之前和导出之后，您需要记录转账的密钥保护机制。该服务为所有钥匙提供密钥检查值，以验证运输方式是否正确。

要求 8-4 要求以保护公钥完整性和真实性的方式传送公钥。应用程序和 AWS 之间的传输由应用程序的身份验证控制 AWS，使用 AWS Identity and Access Management 方法， AWS通过 TLS 服务器证书对应用程序进行 API 端点身份验证。此外，从 P AWS ayment Cryptography 导出或导入的公钥具有由临时的、特定于客户的 CA 签名的证书（请参阅GetPublicKeyCertificate、和）。GetParametersForImportGetParametersForExport这些 CA 不能用作唯一的身份验证方法，因为它们不符合 PCI PIN 安全附录 A2。但是，这些证书仍然为公钥提供完整性保证，IAM 提供身份验证。

使用非对称方法与业务合作伙伴交换公钥时，您必须提供通过通信渠道对企业进行身份验证，例如使用安全的文件交换网站。

要求 9：该服务不使用或不直接支持明文关键组件。

要求 10：该服务强制执行相对的密钥强度，以保护交通工具的钥匙。您负责在从 P AWS ayment Cryptography 导入之前和之后进行密钥传输，并使用准确的 API 和 TR-31 参数进行密钥导入、导出和生成。您应该有记录在案的程序来描述密钥传输机制和用于运输的加密密钥清单。

要求 11：您的程序文档必须说明密钥的传送方式。使用 Payment Cryptography API 进行密钥传输的程序应包括使用具有密钥导入和导出权限的角色，以及允许运行脚本或其他创建密钥的代码的角色。AWS AWS CloudTrail 日志包含全部ImportKey和ExportKey事件。

控制目标 4： Key-loading 以安全的方式处理 HSM 和 POI PIN-acceptance 设备。

要求 12：您负责从组件或共享中加载密钥。作为该服务 PIN 评估的一部分，对 HSM 主密钥的管理进行了评估。 AWS 支付密码学不会从单个股票或组件加载密钥。请参阅 加密详细信息 部分。

要求 13 和 14：在导入服务之前和从服务导出之后，您需要描述对传输的密钥保护。

要求 15：Paym AWS ent Cryptography 为服务中的所有密钥提供密钥检查值，并为公钥提供完整性保证。您的应用程序负责在导入服务或从服务导出后使用这些检查来验证密钥。您应记录这些程序，以确保验证机制到位。

要求 15-2 要求以保护公钥完整性和真实性的方式加载公钥。 ImportKey以及规定对提供的签名证书进行验证。GetParametersForImport如果提供的证书是自签名的，则必须通过单独的机制（例如安全文件交换）提供身份验证。

要求 16：过程文档必须指定如何将密钥加载到服务。使用 API 导入密钥的程序应包括使用具有密钥导入权限和批准运行脚本或其他加载密钥的代码的角色。 AWS CloudTrail 日志包含所有ImportKey事件。您应该在文档中包括日志机制。该服务为所有密钥提供密钥检查值，以验证密钥加载是否正确。

控制目标 5：密钥的使用方式可以防止或检测未经授权的使用。

要求 17：该服务为密钥提供机制，例如标签和别名，以便跟踪密钥共享关系。此外，应单独保存密钥检查值，以证明共享密钥时未使用已知或默认的密钥值。

要求 18：该服务通过和提供密钥完整性检查 ListKeys，GetKey并通过密钥管理事件提供密钥管理事件 AWS CloudTrail，可用于检测未经授权的替换或监控各方之间的密钥同步。该服务仅将密钥存储在密钥块中。在从 P AWS ayment Cryptography 导入之前和导出之后，您负责密钥的存储和使用。

如果在处理基于 PIN 的交易或意外的密钥管理事件期间出现任何差异，则应制定相应的程序，以便立即进行调查。

要求 19：该服务仅在按键块中使用密钥 KeyUsage KeyModeOfUse，强制使用所有操作的密钥和其他密钥属性。这包括对私钥操作的限制。您应该将公钥用于单一用途，例如：加密或数字签名验证，但不能两者兼而有之。您应该为生产和 test/development 系统使用单独的帐户。

要求 20：您保留对此要求的责任。

控制目标 6：以安全的方式管理密钥。

要求 21：作为该服务 PCI PIN 评估的一部分，对密钥存储和 AWS 支付密码的使用进行了评估。对于与关键组件相关的存储需求，您有责任按照 21-2 和 21-3 中的规定进行存储。在导入服务之前和从服务导出之后，您需要在策略文档中描述关键保护机制。

要求22：作为该服务PCI PIN评估的一部分，对 AWS 支付密码学的密钥泄露程序进行了评估。您需要描述关键的泄露检测和响应程序，包括监控和对来自的通知的响应 AWS。

要求 23： AWS 支付密码学不支持变体或其他可逆密钥计算方法。APC 主密钥或由其加密的密钥永远无法提供给客户。作为该服务PCI PIN评估的一部分，对可逆密钥计算的使用进行了评估。

要求24：内部机密和私钥的销毁做法 AWS 支付密码学已作为该服务PCI PIN评估的一部分进行了评估。在导入 APC 之前和从 APC 导出之后，您需要描述密钥的密钥销毁程序。与关键部件相关的销毁要求（24-2.2 和 24-2.3）仍由您负责。

要求25： AWS 支付密码学中对秘密和私钥的访问权限已作为该服务PCI PIN评估的一部分进行了评估。在从 P AWS ayment Cryptography 导入之前和导出之后，您需要准备好密钥访问控制的流程和文档。

要求 26：您需要描述对服务之外使用的密钥、关键组件或相关材料的任何访问权限的日志记录。您的应用程序使用该服务进行的所有密钥管理活动的日志均可通过 AWS CloudTrail。

要求 27：您需要描述在服务之外使用的密钥、关键组件或相关材料的备份程序。

要求 28：使用 API 进行所有密钥管理的程序都应包括使用具有密钥管理权限和批准的角色来运行脚本或其他管理密钥的代码。 AWS CloudTrail 日志包含所有关键管理事件

控制目标 7：以安全的方式管理用于处理 PIN 和密钥的设备。

要求 29：使用 AWS 支付密码学可以满足您对 HSM 的物理和逻辑保护要求。

要求 30：您的应用程序将负责对 POI 设备要求的所有物理和逻辑保护。

要求31： AWS 支付密码学使用的安全加密设备 (SCD) 的保护已作为该服务PCI PIN评估的一部分进行了评估。您需要证明对应用程序使用的任何其他 SCD 的保护。

要求32： AWS 支付密码学使用的 SCD 的使用已作为该服务 PCI PIN 评估的一部分进行了评估。您需要演示对应用程序使用的任何其他 SCD 的访问控制和保护。

要求 33：您需要描述您控制下的所有 PIN-processing 设备的保护措施。

在 P2PE 解决方案中使用 AWS 支付密码学解密组件

PCI P2PE 解决方案可以使用AWS 支付密码学解密组件。这记录在《PCI Point-to-Point 加密：安全要求和测试程序》中的 P2PE 解决方案和第三方 and/or P2PE组件提供商的使用部分：“解决方案提供商（或作为解决方案提供商的商家）可以将某些 P2PE 功能外包给在 PCI 上市的 P2PE 组件提供商，并在其 P2PE 验证报告 (P-ROV) 中报告其 P2PE 组件的使用情况”，该报告可用在 PCI 网站上。

与其他 AWS 服务和合规标准一样，您有责任安全地使用该服务，配置访问控制并使用符合 PCI P2PE 要求的安全参数。AWS Payment Cryptography P2PE 解密组件用户指南（可在上 AWS Artifact找到）详细说明如何将 AWS 支付加密与 PCI P2PE 解决方案集成，以及合规性报告所需的年度解密组件报告。

的身份和访问管理 AWS Payment Cryptography

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 AWS 支付加密资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- 操作方法 AWS 支付密码学与 IAM 配合使用
- AWS 支付密码学基于身份的策略示例
- Resource-based 的政策 AWS Payment Cryptography
- Multi-party 批准 AWS 支付密码学
- 问题排查 AWS 支付密码学身份和访问权限

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅问题排查 AWS 支付密码学身份和访问权限）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅操作方法 AWS 支付密码学与 IAM 配合使用）
- IAM 管理员：编写用于管理访问权限的策略（请参阅AWS 支付密码学基于身份的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center （例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关要求根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（SCP）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略（RCP）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（RCP）。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

操作方法 AWS 支付密码学与 IAM 配合使用

在使用 IAM 管理对 AWS 支付加密的访问权限之前，您应该了解有哪些 IAM 功能可用于 AWS 支付加密。要全面了解 AWS 支付加密和其他 AWS 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

主题

- AWS 支付密码学政策 Identity-based
- 授权基于 AWS 支付密码学标签

AWS 支付密码学政策 Identity-based

借助 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源，以及允许或拒绝操作的条件。 AWS 支付密码学支持特定的操作、资源和条件密钥。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。在策略中包含操作以授予执行关联操作的权限。

AWS 支付密码学中的策略操作在操作前使用以下前缀:`payment-cryptography:`. 例如，要授予某人运行 AWS Payment Cryptography `VerifyCardData` API 操作的权限，您应将 `payment-cryptography:VerifyCardData` 操作纳入其策略中。策略语句必须包含 `Action` 或 `NotAction` 元素。 AWS Payment Cryptography 定义了自己的一组操作，这些操作描述了您可以使用此服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
```

```
        "payment-cryptography:action1",
        "payment-cryptography:action2"
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 List 开头的所有操作（例如 ListKeys 和 ListAliases），包括以下操作：

```
"Action": "payment-cryptography:List*"
```

要查看 AWS 支付加密操作列表，请参阅 IAM 用户指南中的 AWS 支付加密定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

Payment Cryptography 密钥资源具有以下 ARN：

```
arn:${Partition}:payment-cryptography:${Region}:${Account}:key/${keyARN}
```

有关 ARN 格式的更多信息，请参阅 A mazon 资源名称 (ARN) 和 AWS 服务命名空间。

例如，要在语句中指定 arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h 实例，请使用以下 ARN：

```
"Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h"
```

要指定属于特定账户的所有密钥，请使用通配符 (*)：

```
"Resource": "arn:aws:payment-cryptography:us-east-2:111122223333:key/*"
```

某些 AWS 支付加密操作（例如用于创建密钥的操作）无法在特定资源上执行。在这些情况下，您必须使用通配符（*）。

```
"Resource": "*"
```

要在单个语句中指定多个资源，请使用逗号，如下所示：

```
"Resource": [
      "resource1",
      "resource2"
```

示例

要查看基于身份的 AWS 支付加密政策的示例，请参阅。AWS 支付密码学基于身份的策略示例

授权基于 AWS 支付密码学标签

您可以将标签附加到 AWS 支付密码学资源，也可以在请求中将标签传递给 AWS 支付加密。要基于标签控制访问，您需要使用 payment-cryptography:ResourceTag/*key-name*aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

AWS 支付密码学基于身份的策略示例

默认情况下，IAM 用户和角色无权创建或修改 AWS 支付加密资源。他们也无法使用 AWS 管理控制台 AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的 在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 AWS 支付密码学控制台
- 允许用户查看他们自己的权限
- 能够访问的各个方面 AWS Payment Cryptography
- 使用指定的密钥调用 API 的能力
- 明确拒绝资源的能力

策略最佳实践

Identity-based 策略决定了某人是否可以在您的账户中创建、访问或删除 AWS 付款加密资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 AWS 支付密码学控制台

要访问 AWS 支付加密控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您 AWS 账户中 AWS 支付加密资源的详细信息。如果您创建的基于身份的策略比所需的最低权限更严格，则无法为具有该策略的实体（IAM 用户或角色）正常运行控制台。

为确保这些实体仍然可以使用 AWS 支付密码控制台，还需要将以下 AWS 托管策略附加到这些实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

能够访问的各个方面 AWS Payment Cryptography

> **Warning**
>
> 此示例提供了广泛的权限，因此不建议这样操作。相反，请考虑最低特权访问模型。

在此示例中，您希望授予 AWS 账户中的 IAM 用户访问您的所有 AWS 支付加密密钥的权限，以及调用所有 AWS 支付密码学 api（包括`ControlPlane`和`DataPlane`操作）的能力。

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
            {
               "Effect": "Allow",
               "Action": [
                  "payment-cryptography:*"
               ],
               "Resource": [
                  "*"
               ]
            }
      ]
   }
```

使用指定的密钥调用 API 的能力

在此示例中，您想向 AWS 账户中的 IAM 用户授予访问您的 AWS 付款加密密钥之一的权限，`arn:aws:payment-cryptography:us-east-2:111122223333:key/kwapwa6qaifllw2h`然后在两个 API 中使用该资源，`GenerateCardValidationData`以及`VerifyCardValidationData`。相反，IAM 用户将无权在其他操作（例如 `DeleteKey` 或 `ExportKey`）中使用此密钥

资源可以是前缀为 `key` 的密钥，也可以是前缀为 `alias` 的别名。

JSON

```
   {
      "Version":"2012-10-17",
      "Statement": [
            {
               "Effect": "Allow",
               "Action": [
                  "payment-cryptography:VerifyCardValidationData",
                  "payment-cryptography:GenerateCardValidationData"
               ],
               "Resource": [
                  "arn:aws:payment-cryptography:us-east-2:111122223333:key/
kwapwa6qaifllw2h"
               ]
            }
      ]
   }
```

明确拒绝资源的能力

> **Warning**
>
> 请仔细考虑授予通配符访问权限的影响。考虑改为最低权限模型。

在此示例中，您希望允许 AWS 账户中的 IAM 用户访问您的任何 AWS 付款加密密钥，但希望拒绝对一个特定密钥的权限。用户将有权访问用所有密钥的 `VerifyCardData` 和 `GenerateCardData`，但拒绝语句中指定的密钥除外。

JSON

```
{
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
```

```
            "Action": [
                "payment-cryptography:VerifyCardValidationData",
                "payment-cryptography:GenerateCardValidationData"
            ],
            "Resource": [
              "arn:aws:payment-cryptography:us-east-2:111122223333:key/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
              "payment-cryptography:GenerateCardValidationData"
            ],
            "Resource": [
              "arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h"
            ]
        }
    ]
}
```

Resource-based 的政策 AWS Payment Cryptography

Resource-based 策略是您附加到资源的 JSON 策略文档，例如 AWS 付款加密密钥。在基于资源的策略中，您可以指定谁可以访问密钥以及他们可以对密钥执行的操作。您可以使用基于资源的策略来：

- 向多个用户和角色授予对单个密钥的访问权限。
- 向其他 AWS 账户中的用户或角色授予访问权限。

主题

- 注意事项
- 管理基于资源的策略
- Resource-based 策略示例

当您将基于资源的策略附加到 AWS 支付加密密钥时， AWS Payment Cryptography 会使用 IAM 策略评估逻辑来确定给定的委托人是否有权执行所请求的操作。要启用跨账户访问，您可以将整个账户或另一个账户中的 IAM 实体指定为基于资源的策略中的委托人。 Cross-account 访问需要两个策略：

1. Resource-based 策略（密钥所有者的账户）— 密钥所有者PutResourcePolicy用来授予对调用者账户或 IAM 委托人的访问权限。
2. Identity-based 策略（来电者的账户）— 来电者的 IAM 管理员还必须允许呼叫者的 IAM 策略中的 AWS 付款加密操作（例如payment-cryptography:EncryptData）。

这两个策略都必须允许该操作。如果缺少其中一个，则跨账户请求将被拒绝。AccessDeniedException

如果基于资源的策略向同一账户中的委托人授予访问权限，则不需要其他基于身份的策略。有关更多信息，请参阅 IAM 用户指南中的 IAM 角色与 Resource-based 策略的区别。

> **⚠ 资源策略控制平面操作**
>
> Resource-based 策略不适用于资源策略控制平面操作 PutResourcePolicy，例如GetResourcePolicy、和DeleteResourcePolicy。这样可以防止出现资源策略可能拒绝修改或删除策略本身的功能的潜在封锁情形。对这些控制平面操作的访问仅受基于 IAM 身份的策略的约束。

注意事项

在使用基于资源的策略和 AWS 支付密码学时，请记住以下几点。

- AWS 支付密码学自动强制禁止公开访问密钥。您不能创建向匿名或公共委托人授予访问权限的基于资源的策略。所有对 AWS 支付密码学密钥的访问都需要经过身份验证的 AWS 委托人，并且公共访问始终被阻止。
- Resource-based 策略是按密钥应用的。每个 P AWS ayment Cryptography 密钥最多可以附加一个基于资源的策略。
- Resource-based 策略不适用于别名。当您通过别名引用密钥时，将评估附加到底层密钥的资源策略。
- Resource-based 策略不适用于此时使用 Multi-Region 密钥复制创建的只读副本区域密钥。资源策略只能附加到主区域密钥。
- 基于资源的策略中的Resource元素必须与策略所关联的密钥的 ARN "*" 或完全匹配。建议使用"*"，因为它允许在多个密钥之间重复使用相同的策略文档。

- 资源策略管理 API（PutResourcePolicyGetResourcePolicy、和DeleteResourcePolicy）仅限 AWS 账户 于拥有密钥的。只有密钥所有者账户中的委托人才能管理资源策略。

管理基于资源的策略

您可以使用 AWS CLI 或 AWS API 管理基于资源的 AWS 支付加密密钥政策。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

附加基于资源的策略

使用 PutResourcePolicyAPI 操作或 put-resource-policyCLI 命令将基于资源的策略附加到密钥。如果策略已经存在，则该命令会替换它。

以下示例将 JSON 文件中的基于资源的策略附加到密钥。

```
aws payment-cryptography put-resource-policy \
    --resource-arn arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h \
    --policy file://policy.json
```

检索基于资源的策略

使用 GetResourcePolicyAPI 操作或 get-resource-policyCLI 命令检索附加到密钥的基于资源的策略。

以下示例检索附加到密钥的基于资源的策略。

```
aws payment-cryptography get-resource-policy \
    --resource-arn arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```

响应会返回政策文档：

```
{
    "Policy": {
        "Version": "2012-10-17",
        "Statement": [
            {
```

```
                "Effect": "Allow",
                "Principal": {
                    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
                },
                "Action": [
                    "payment-cryptography:EncryptData",
                    "payment-cryptography:DecryptData"
                ],
                "Resource": "*"
            }
        ]
    }
}
```

删除基于资源的策略

使用 DeleteResourcePolicyAPI 操作或 delete-resource-policyCLI 命令从密钥中删除基于资源的策略。

以下示例删除了附加到密钥的基于资源的策略。

```
aws payment-cryptography delete-resource-policy \
    --resource-arn arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```

Resource-based 策略示例

授予对密钥的跨账户访问权限

以下基于资源的策略授予其他 AWS 账户中的角色使用 AWS 支付加密密钥进行加密操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
            },
            "Action": [
                "payment-cryptography:GenerateCardValidationData",
```

```
                "payment-cryptography:VerifyCardValidationData"
            ],
            "Resource": "*"
        }
    ]
}
```

向不同的账户授予不同的权限

以下基于资源的策略演示了如何向不同账户中的委托人授予不同的权限。在此示例中，一个账户中的 3DS 访问控制服务器 (ACS) 可以生成卡验证数据，而另一个账户中的付款授权服务只能验证 3DS 密码。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow3DSACSToGenerate",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/3dsAcsRole"
            },
            "Action": [
                "payment-cryptography:GenerateCardValidationData"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowPaymentAuthToVerify",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/PaymentAuthRole"
            },
            "Action": [
                "payment-cryptography:VerifyAuthRequestCryptogram"
            ],
            "Resource": "*"
        }
    ]
}
```

Multi-party 批准 AWS 支付密码学

AWS Payment Cryptography 与 Amazon Web Services Organizations 提供的Multi-party 批准 (MPA) 功能集成，可通过分布式批准流程帮助保护关键操作。借助 MPA，您可以要求多个受信任的个人在执行特定的 AWS 支付加密操作之前批准这些操作。

主题

- 概述
- 受保护的操作
- 先决条件
- 启用和禁用 MPA
- 开始使用
- 示例：导入启用了 MPA 的根证书
- AWS CloudTrail 记录 MPA 事件
- 检查请求状态并处理失败

概述

Multi-party 批准要求在操作继续之前获得一组可信人员的批准，从而为敏感的 AWS 支付加密操作增加了一层额外的安全性。这有助于在单组凭据遭到泄露时防止未经授权的更改，并防止个人单方面进行更改。

审批小组是指组织内由您指定批准或拒绝受保护的操作请求的一组批准者。批准流程完全由贵组织的批准者管理。没有 AWS 人员参与批准或拒绝申请。

为受保护的操作启用 MPA 时，会发生以下情况：

1. 请求者启动受保护的操作。
2. MPA 创建批准会话并通知审批小组成员。
3. 审批小组成员审查申请，并通过 MPA 门户网站批准或拒绝申请。
4. 一旦达到所需的最低批准门槛，操作就会继续进行。如果请求被批准小组拒绝，或者允许的会话时间在达到批准阈值之前到期，则不会执行该操作。在这两种情况下，请求者都必须提交新的请求才能重试该操作。

> **Note**
>
> 导入启用 MPA 的根 CA 证书时，该`RequesterComment`参数为必填项。此评论包含在发送给审批小组的批准通知中，为请求提供了背景信息。

受保护的操作

AWS 支付密码学支持 MPA 用于以下操作：

- `ImportKey`使用`RootCertificatePublicKey`密钥材料 — 导入根公钥证书是一项关键操作，因为根证书使用非对称密钥交换（例如）为所有后续密钥导入和导出建立信任锚点。 TR-34此操作要求获得多方批准有助于确保任何个人都无法单方面建立或更改您的 AWS 支付密码密钥的信任根。

先决条件

在将 MPA 与 AWS 支付密码学结合使用之前，您必须满足以下先决条件：

- 在您的亚马逊 Web Services Organizations 环境中设置 MPA。有关说明，请参阅什么是 Multi-party 批准？ 在Multi-party 批准用户指南中。
- 至少创建一个由所需审批者组成的审批小组。
- 使用 AWS Resource Access Manager与包含您的 AWS 付款加密密钥 AWS 账户 的审批团队共享。
- 您组织中的管理账户必须选择加入才能 Multi-party 获得批准。

启用和禁用 MPA

设置审批小组后，您可以通过将该小组与您的账户关联来为 AWS 支付密码学启用 MPA。您也可以通过取消关联团队来禁用 MPA，尽管取消关联需要当前关联的审批小组的批准。

启用 MPA

使用 `AssociateMpaTeam` API 操作或 associate-mpa-team CLI 命令将审批小组与您的 AWS 支付加密账户关联起来。关联后，受保护的操作需要获得团队的批准才能继续。

```
aws payment-cryptography associate-mpa-team \
    --team-arn arn:aws:mpa:us-east-1:111122223333:team/my-approval-team
```

禁用 MPA

使用 DisassociateMpaTeam API 操作或 disassociate-mpa-team CLI 命令移除审批小组关联。取消关联团队本身就是一项受保护的操作，需要获得当前关联的审批小组的批准。

```
aws payment-cryptography disassociate-mpa-team \
    --team-arn arn:aws:mpa:us-east-1:111122223333:team/my-approval-team
```

Important

禁用 MPA 需要获得当前关联的审批小组的批准。这样可以确保任何个人都不能单方面取消多方批准保护。

Note

对于和来说，该`--requester-comment`参数是可选associate-mpa-team的disassociate-mpa-team。

开始使用

要开始使用 MPA for P AWS ayment Cryptography，请参阅Multi-party 批准用户指南，了解详细的设置说明，包括如何创建审批小组、配置批准策略和管理批准会话。

示例：导入启用了 MPA 的根证书

启用 MPA 且审批小组与的`ImportKey`操作关联后`RootCertificatePublicKey`，导入请求需要获得批准才能继续。

1. 请求者调`import-key`用导入根公钥证书。要使用此命令，请将示例命令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography import-key \
    --key-material='{"RootCertificatePublicKey": {
    "KeyAttributes": {
        "KeyAlgorithm": "RSA_4096",
        "KeyClass": "PUBLIC_KEY",
        "KeyModesOfUse": {"Verify": true},
        "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE"
    },
    "PublicKeyCertificate": "LS0tLS1CRUdJTi..."}}' \
```

```
    --requester-comment "Importing new root CA certificate for TR-34 key exchange
 with partner XYZ"
```

响应会返回一个KeyState设置为的密钥CREATE_IN_PROGRESS，表示该请求正在等待批准。回复中还包括MpaStatus有关批准会话的详细信息：

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096"
        },
        "Enabled": true,
        "KeyState": "CREATE_IN_PROGRESS",
        "KeyOrigin": "EXTERNAL",
        "CreateTimestamp": "2026-04-27T10:15:30.000000+00:00",
        "UsageStartTimestamp": "2026-04-27T10:15:29.926000+00:00",
        "MpaStatus": {
            "MpaSessionArn": "arn:aws:mpa:us-
east-1:111122223333:session/abc123def456",
            "Status": "PENDING",
            "InitiationDate": "2026-04-27T10:15:30.000000+00:00"
        }
    }
}
```

2. 由于 MPA 已启用，因此请求不会立即完成。相反，P AWS ayment Cryptography 会创建一个批准会话，并返回一个表示正在等待批准的响应。
3. 审批小组成员会收到通知，并通过 MPA 门户网站审核申请。在所需数量的批准者批准请求后，导入操作将继续，根证书将被导入。

AWS CloudTrail 记录 MPA 事件

启用 MPA 后，Paym AWS ent Cryptography 会在批准会话完成AWS CloudTrail时记录服务事件。这些事件记录了批准过程的结果，包括请求是获得批准还是失败。您可以使用这些日志来审计 MPA 活动并跟踪受保护操作的状态。

MPA-related CloudTrail 事件包括以下字段serviceEventDetails：

- keyArn— 受操作影响的密钥的 ARN。
- operation— 请求的受保护操作。
- mpaSessionArn— MPA 批准会话的 ARN。
- sessionStatus— 批准会话的结果（APPROVED或FAILED）。

已批准的请求

以下示例显示了已获得 MPA 团队批准的ImportKey请求 CloudTrail 的事件：


```
{
    "eventVersion": "1.11",
    "eventTime": "2026-04-28T18:49:51Z",
    "eventName": "ImportKey",
    "eventSource": "payment-cryptography.amazonaws.com",
    "eventType": "AwsServiceEvent",
    "eventCategory": "Management",
    "awsRegion": "us-east-1",
    "readOnly": false,
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "payment-cryptography.amazonaws.com"
    },
    "resources": [
        {
            "ARN": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/spa2dclzmsihlj4o",
            "accountId": "111122223333",
            "type": "AWS::PaymentCryptography::Key"
        }
    ],
    "serviceEventDetails": {
        "keyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/spa2dclzmsihlj4o",
        "operation": "ImportKey",
        "mpaSessionArn": "arn:aws:mpa:us-east-1:111122223333:session/my-approval-
team/44c76e07-8937-4d7d-bb9a-a646322e2a1e",
        "sessionStatus": "APPROVED"
    }
}
```

请求失败

以下示例显示了ImportKey请求被拒绝或超时 CloudTrail 的事件：


```
{
    "eventVersion": "1.11",
    "eventTime": "2026-04-28T18:50:35Z",
    "eventName": "ImportKey",
    "eventSource": "payment-cryptography.amazonaws.com",
    "eventType": "AwsServiceEvent",
    "eventCategory": "Management",
    "awsRegion": "us-east-1",
    "readOnly": false,
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "payment-cryptography.amazonaws.com"
    },
    "resources": [
        {
            "ARN": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/qj46ku4qimypxdo7",
            "accountId": "111122223333",
            "type": "AWS::PaymentCryptography::Key"
        }
    ],
    "serviceEventDetails": {
        "keyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/qj46ku4qimypxdo7",
        "operation": "ImportKey",
        "mpaSessionArn": "arn:aws:mpa:us-east-1:111122223333:session/my-approval-team/
b0ac1994-14e1-47a6-bf1a-0b6fc0b845f2",
        "sessionStatus": "FAILED"
    }
}
```


要了解更多信息 AWS CloudTrail，请参阅《AWS CloudTrail 用户指南》。

检查请求状态并处理失败

您可以致电查看待处理的 MPA 请求的状态GetKey。响应中包含包含当前批准会话详细信息的MpaStatus字段。要使用此命令，请将示例命令令*italicized placeholder text*中的替换为您自己的信息。

```
aws payment-cryptography get-key \
    --key-identifier arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h
```

当请求等待批准时，响应显示KeyState为CREATE_IN_PROGRESS和MpaStatus.Status为PENDING：

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096"
        },
        "Enabled": true,
        "KeyState": "CREATE_IN_PROGRESS",
        "KeyOrigin": "EXTERNAL",
        "CreateTimestamp": "2026-04-27T10:15:30.000000+00:00",
        "UsageStartTimestamp": "2026-04-27T10:15:29.926000+00:00",
        "MpaStatus": {
            "MpaSessionArn": "arn:aws:mpa:us-east-1:111122223333:session/abc123def456",
            "Status": "PENDING",
            "InitiationDate": "2026-04-27T10:15:30.000000+00:00"
        }
    }
}
```

所需数量的批准者批准请求后，将KeyState移至CREATE_COMPLETE并MpaStatus.Status移至。APPROVED密钥现已准备就绪，可以使用：

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h",
```

```
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096"
        },
        "Enabled": true,
        "KeyState": "CREATE_COMPLETE",
        "KeyOrigin": "EXTERNAL",
        "CreateTimestamp": "2026-04-27T10:15:30.000000+00:00",
        "UsageStartTimestamp": "2026-04-27T10:15:29.926000+00:00",
        "MpaStatus": {
            "MpaSessionArn": "arn:aws:mpa:us-east-1:111122223333:session/abc123def456",
            "Status": "APPROVED",
            "InitiationDate": "2026-04-27T10:15:30.000000+00:00"
        }
    }
}
```

如果申请被审批小组拒绝，或者会话在达到批准阈值之前到期，则会MpaStatus.Status更改FAILED为：KeyStateCREATE_FAILED

```
{
    "Key": {
        "KeyArn": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/kwapwa6qaifllw2h",
        "KeyAttributes": {
            "KeyUsage": "TR31_S0_ASYMMETRIC_KEY_FOR_DIGITAL_SIGNATURE",
            "KeyClass": "PUBLIC_KEY",
            "KeyAlgorithm": "RSA_4096"
        },
        "Enabled": true,
        "KeyState": "CREATE_FAILED",
        "KeyOrigin": "EXTERNAL",
        "CreateTimestamp": "2026-04-27T10:15:30.000000+00:00",
        "UsageStartTimestamp": "2026-04-27T10:15:29.926000+00:00",
        "MpaStatus": {
            "MpaSessionArn": "arn:aws:mpa:us-east-1:111122223333:session/abc123def456",
            "Status": "FAILED",
            "InitiationDate": "2026-04-27T10:15:30.000000+00:00",
            "StatusMessage": "Approval session expired or was denied"
        }
    }
}
```

处于CREATE_FAILED状态的密钥不能用于加密操作。要重试导入，您必须提交新的ImportKey请求，这将创建一个新的批准会话。

问题排查 AWS 支付密码学身份和访问权限

在确定 AWS 支付密码学的特定 IAM-related 问题后，将向本节添加主题。有关 IAM 主题的一般故障排除内容，请参阅 IAM 用户指南的故障排除部分。

监控 AWS 支付加密

监控是维护 AWS 支付密码学和您的其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。AWS 提供以下监控工具，用于监视 AWS 支付加密、报告问题并在适当时自动采取行动：

- Amazon 会实时 CloudWatch监控您的 AWS 资源和您运行 AWS 的应用程序。您可以收集和跟踪指标，创建自定义的控制平面，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以 CloudWatch 追踪某些账户的使用情况， APIs 或者在接近 AWS 支付密码配额时通知您。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- Amazon Lo CloudWatch gs 使您能够监控、存储和访问来自亚马逊 EC2 实例和其他来源的日志文件。 CloudTrail CloudWatch 日志可以监视日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南。
- AWS CloudTrail捕获由您的账户或代表您的 AWS 账户进行的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以识别哪些用户和账户拨打了电话 AWS、调用的端点、使用的资源（密钥）、发出呼叫的源 IP 地址以及呼叫发生的时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。

主题

- 使用记录 AWS 支付加密 API 调用 AWS CloudTrail

使用记录 AWS 支付加密 API 调用 AWS CloudTrail

AWS Payment Cryptography 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在 AWS 支付密码学中采取的操作的记录。 CloudTrail 将 AWS 支付密码学的所有 API 调用捕获为事件。捕获的调用包含来自 控制台和代码的 API 操作调用。如果您创建跟踪，则可以将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括用于 AWS 支付加密的事件。如果您未配置跟踪，您仍然可以在控制台的事件历史记录中查看最新的管理（ CloudTrail 控制平面）事件。使用收集的信息 CloudTrail，您可以确定向 P AWS ayment Cryptography 发出的请求、发出请求的 IP 地址、谁提出了请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

主题

- AWS 中的支付密码学信息 CloudTrail

- 控制飞机事件 CloudTrail
- 中的数据事件 CloudTrail
- 了解 AWS 支付密码学控制平面日志文件条目
- 了解 AWS 支付密码学数据平面日志文件条目

AWS 中的支付密码学信息 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。在 AWS 支付密码学中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您 AWS 账户中的事件，包括 AWS 支付密码学事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

控制飞机事件 CloudTrail

CloudTrail 记录 AWS 支付加密操作，例
如、CreateKey、ImportKey、DeleteKeyListKeysTagResource、和以及所有其他控制平面操作。

中的数据事件 CloudTrail

数据事件提供有关在资源上或在资源中执行的资源操作的信息，例如加密有效载荷或翻译 PIN。数据事件是指默认情况下 CloudTrail 不记录的大容量活动。您可以使用 CloudTrail APIs 或控制台为 AWS 支付加密数据平面事件启用数据事件 API 操作记录。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的记录数据事件。

使用 CloudTrail，您必须使用高级事件选择器来决定记录和记录哪些 AWS 支付密码学 API 活动。要记录 Pay AWS ment Cryptography 数据平面事件，必须包括资源类型`AWS Payment Cryptography key`和。`AWS Payment Cryptography alias`设置完成后，您可以通过选择要记录的特定数据事件（例如使用 `eventName` 过滤器来跟踪 `EncryptData` 事件）来进一步调整日志记录首选项。有关更多信息，请参阅《AWS CloudTrail API Reference》中的 `AdvancedEventSelector`。

> **Note**
>
> 要订阅 AWS 支付密码学数据事件，您必须使用高级事件选择器。我们建议您订阅密钥和别名事件，以确保您收到所有事件。

AWS 支付密码学数据事件：

- `DecryptData`
- `EncryptData`
- `GenerateCardValidationData`
- `GenerateMac`
- `GeneratePinData`
- `ReEncryptData`
- `TranslatePinData`
- `VerifyAuthRequestCryptogram`
- `VerifyCardValidationData`
- `VerifyMac`
- `VerifyPinData`

记录数据事件将收取额外费用。有关更多信息，请参阅AWS CloudTrail 定价。

了解 AWS 支付密码学控制平面日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序出现。

以下示例显示了演示 “ AWS 付款加密” `CreateKey` 操作的 CloudTrail 日志条目。

```
      {
    CloudTrailEvent: {
      tlsDetails= {
        TlsDetails: {
          cipherSuite=TLS_AES_128_GCM_SHA256,
          tlsVersion=TLSv1.3,
          clientProvidedHostHeader=controlplane.paymentcryptography.us-
west-2.amazonaws.com
        }
      },
    requestParameters=CreateKeyInput (
      keyAttributes=KeyAttributes(
        KeyUsage=TR31_B0_BASE_DERIVATION_KEY,
        keyClass=SYMMETRIC_KEY,
        keyAlgorithm=AES_128,
        keyModesOfUse=KeyModesOfUse(
          encrypt=false,
          decrypt=false,
          wrap=false
          unwrap=false,
          generate=false,
          sign=false,
          verify=false,
          deriveKey=true,
          noRestrictions=false)
        ),
      keyCheckValueAlgorithm=null,
      exportable=true,
      enabled=true,
      tags=null),
    eventName=CreateKey,
    userAgent=Coral/Apache-HttpClient5,
    responseElements=CreateKeyOutput(
```

```
      key=Key(
        keyArn=arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp,
          keyAttributes=KeyAttributes(
            KeyUsage=TR31_B0_BASE_DERIVATION_KEY,
            keyClass=SYMMETRIC_KEY,
            keyAlgorithm=AES_128,
            keyModesOfUse=KeyModesOfUse(
              encrypt=false,
              decrypt=false,
              wrap=false,
              unwrap=false,
              generate=false,
              sign=false,
              verify=false,
              deriveKey=true,
              noRestrictions=false)
            ),
          keyCheckValue=FE23D3,
          keyCheckValueAlgorithm=ANSI_X9_24,
          enabled=true,
          exportable=true,
          keyState=CREATE_COMPLETE,
          keyOrigin=AWS_PAYMENT_CRYPTOGRAPHY,
          createTimestamp=Sun May 21 18:58:32 UTC 2023,
          usageStartTimestamp=Sun May 21 18:58:32 UTC 2023,
          usageStopTimestamp=null,
          deletePendingTimestamp=null,
          deleteTimestamp=null)
        ),
      sourceIPAddress=192.158.1.38,
        userIdentity={
          UserIdentity: {
            arn=arn:aws:sts::111122223333:assumed-role/TestAssumeRole-us-west-2/
ControlPlane-IntegTest-68211a2a-3e9d-42b7-86ac-c682520e0410,
            invokedBy=null,
            accessKeyId=TESTXECZ5U2ZULLHHMJG,
            type=AssumedRole,
            sessionContext={
              SessionContext: {
                sessionIssuer={
                  SessionIssuer: {arn=arn:aws:iam::111122223333:role/TestAssumeRole-us-
west-2,
                  type=Role,
```

```
                accountId=111122223333,
                userName=TestAssumeRole-us-west-2,
                principalId=TESTXECZ5U9M4LGF2N6Y5}
              },
              attributes={
                SessionContextAttributes: {
                  creationDate=Sun May 21 18:58:31 UTC 2023,
                  mfaAuthenticated=false
                }
              },
              webIdFederationData=null
            }
          },
        username=null,
        principalId=TESTXECZ5U9M4LGF2N6Y5:ControlPlane-User,
        accountId=111122223333,
        identityProvider=null
      }
    },
    eventTime=Sun May 21 18:58:32 UTC 2023,
    managementEvent=true,
    recipientAccountId=111122223333,
    awsRegion=us-west-2,
    requestID=151cdd67-4321-1234-9999-dce10d45c92e,
    eventVersion=1.08, eventType=AwsApiCall,
    readOnly=false,
    eventID=c69e3101-eac2-1b4d-b942-019919ad2faf,
    eventSource=payment-cryptography.amazonaws.com,
    eventCategory=Management,
    additionalEventData={
  }
 }
}
```

以下示例显示了一个 CloudTrail 日志条目，该条目演示了支持多区域密钥复制的 AWS 支付加密。

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "accountId": "111122223333",
        "invokedBy": "payment-cryptography.amazonaws.com"
    },
```

```
    "eventTime": "2025-08-15T17:50:41Z",
    "eventSource": "payment-cryptography.amazonaws.com",
    "eventName": "SynchronizeMultiRegionKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "payment-cryptography.amazonaws.com",
    "userAgent": "payment-cryptography.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "55c0fcbc-5b2e-4bd2-a976-99305be6e6fc",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "serviceEventDetails": {
        "keyArn": "arn:aws:payment-cryptography:us-east-1:111122223333:key/key-id",
        "replicationRegion": "us-east-2"
    },
    "eventCategory": "Management"
}
```

了解 AWS 支付密码学数据平面日志文件条目

可以选择配置数据平面事件，其功能与控制平面日志类似，但通常要高得多。鉴于 P AWS ayment Cryptography 数据平面操作的某些输入和输出具有敏感性，您可能会发现某些字段带有 “*** 敏感数据已编辑 ***” 消息。这是不可配置的，旨在防止敏感数据出现在日志或跟踪中。

以下示例显示了演示 “ AWS 付款加密” `EncryptData` 操作的 CloudTrail 日志条目。

```
      {
        "Records": [
            {
                "eventVersion": "1.09",
                "userIdentity": {
                    "type": "AssumedRole",
                    "principalId": "TESTXECZ5U2ZULLHHMJG:DataPlane-User",
                    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/DataPlane-
User",
                    "accountId": "111122223333",
                    "accessKeyId": "TESTXECZ5U2ZULLHHMJG",
                    "userName": "",
                    "sessionContext": {
                        "sessionIssuer": {
```

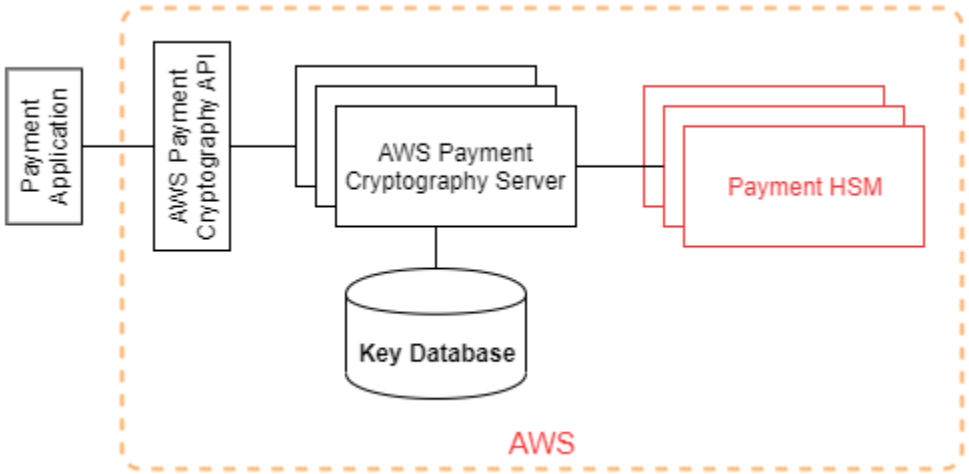
```
                            "type": "Role",
                            "principalId": "TESTXECZ5U9M4LGF2N6Y5",
                            "arn": "arn:aws:iam::111122223333:role/Admin",
                            "accountId": "111122223333",
                            "userName": "Admin"
                        },
                        "attributes": {
                            "creationDate": "2024-07-09T14:23:05Z",
                            "mfaAuthenticated": "false"
                        }
                    }
                },
                "eventTime": "2024-07-09T14:24:02Z",
                "eventSource": "payment-cryptography.amazonaws.com",
                "eventName": "GenerateCardValidationData",
                "awsRegion": "us-east-2",
                "sourceIPAddress": "192.158.1.38",
                "userAgent": "aws-cli/2.17.6 md/awscrt#0.20.11 ua/2.0 os/macos#23.4.0
 md/arch#x86_64 lang/python#3.11.8 md/pyimpl#CPython cfg/retry-mode#standard md/
installer#exe md/prompt#off md/command#payment-cryptography-data.generate-card-
validation-data",
                "requestParameters": {
                    "key_identifier": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp",
                    "primary_account_number": "*** Sensitive Data Redacted ***",
                    "generation_attributes": {
                        "CardVerificationValue2": {
                            "card_expiry_date": "*** Sensitive Data Redacted ***"
                        }
                    }
                },
                "responseElements": null,
                "requestID": "f2a99da8-91e2-47a9-b9d2-1706e733991e",
                "eventID": "e4eb3785-ac6a-4589-97a1-babdd3d4dd95",
                "readOnly": true,
                "resources": [
                    {
                        "accountId": "111122223333",
                        "type": "AWS::PaymentCryptography::Key",
                        "ARN": "arn:aws:payment-cryptography:us-
east-2:111122223333:key/5rplquuwozodpwsp"
                    }
                ],
                "eventType": "AwsApiCall",
```

```
                "managementEvent": false,
                "recipientAccountId": "111122223333",
                "eventCategory": "Data",
                "tlsDetails": {
                    "tlsVersion": "TLSv1.3",
                    "cipherSuite": "TLS_AES_128_GCM_SHA256",
                    "clientProvidedHostHeader": "dataplane.payment-cryptography.us-
east-2.amazonaws.com"
                }
            }
        ]
    }
```

加密详细信息

AWS Payment Cryptography 提供了一个用于生成和管理支付交易的加密密钥的 Web 界面。 AWS Payment Cryptography 提供标准的密钥管理服务和支付交易加密以及可用于集中管理和审计的工具。本文档详细描述了您可以在 AWS 支付密码学中使用的加密操作，以帮助您评估该服务提供的功能。

AWS Payment Cryptography 包含多个接口（包括 RESTful API、通过 AWS CLI、AWS 软件开发工具包等 AWS 管理控制台），用于请求分布式 PCI PT HSM-validated S 硬件安全模块队列的加密操作。



AWS Payment Cryptography 是一项分层服务，由面向网络的 AWS 支付密码学主机和一级 HSM 组成。这些分层主机的分组构成了 AWS 支付密码堆栈。对 AWS 支付加密的所有请求都必须通过传输层安全协议 (TLS) 提出，并在 AWS 支付加密主机上终止。服务主机仅允许使用提供完美前向保密性的密码套件的 TLS。该服务使用适用于所有其他 AWS API 操作的 IAM 相同的凭证和策略机制对您的请求进行身份验证和授权。

AWS 支付密码学服务器通过私有非虚拟网络连接到底层 HSM。服务组件和 HSM 之间的连接使用双向 TLS (mTLS) 进行身份验证和加密。

主题

- 设计目标
- 基本原理
- 内部操作
- 客户操作

设计目标

AWS 支付密码学旨在满足以下要求：

- 值得信赖：密钥的使用受您定义和管理的访问控制策略的保护。没有导出纯文本 AWS 支付密码密钥的机制。加密密钥的机密性至关重要。多名具有特定角色访问权限的基于法定人数的访问控制的 Amazon 员工需要对这些权限执行管理操作。 HSMs任何 Amazon 员工都无法访问 HSM 主（或主要）密钥或备份。主密钥无法与不 HSMs 属于 AWS 支付加密区域的主密钥同步。所有其他密钥都受到 HSM 主密钥的保护。因此，客户 AWS 付款加密密钥不能在客户账户内运行的 AWS 支付加密服务之外使用。
- 低延迟和高吞吐量 — Paym AWS ent Cryptography 提供延迟和吞吐量级别的加密操作，适用于管理支付加密密钥和处理支付交易。
- 持久性：加密密钥的持久性设计为等同于 中服务最高的持久性。单个加密密钥可以与支付终端、EMV 芯片卡或其他已使用多年的安全加密设备 (SCD) 共享。
- 独立的区域：AWS 为需要在不同区域限制数据访问或需要遵守数据驻留要求的客户提供独立的区域。可以在 Amazon Web Services 区域内隔离密钥使用。
- 随机数的安全来源 — 由于强大的密码学依赖于真正不可预测的随机数生成，因此 P AWS ayment Cryptography 提供了高质量且经过验证的随机数来源。 AWS 支付密码学的所有密钥生成均使用 PCI PTS HSM 列出的 HSM，在 PCI 模式下运行。
- 审计 — P AWS ayment Cryptography 将加密密钥的使用和管理记录在 Amazon 提供的 CloudTrail 日志和服务日志中。 CloudWatch您可以使用 CloudTrail 日志来检查加密密钥的使用情况，包括与您共享密钥的账户对密钥的使用情况。 AWS 支付密码学由第三方评估机构根据适用的PCI、信用卡品牌和区域支付安全标准进行审计。AWS Artifact 上提供了证明和责任共担指南。
- Elastic — AWS 支付密码学可根据您的需求进行横向扩展。Payment Cryptography 不是预测和预留 HSM 容量，而是按需提供 AWS 支付加密。 AWS Payment Cryptography 负责维护 HSM 的安全性和合规性，以提供足够的容量来满足客户的峰值需求。

基本原理

本章中的主题描述了 AWS 支付密码学的加密原语及其用途。它们还介绍了服务的基本元素。

主题

- 加密基元
- 熵和随机数生成
- 对称密钥操作

- 非对称密钥操作
- 密钥存储
- 使用对称密钥导入密钥
- 使用非对称密钥导入密钥
- 密钥导出
- 每笔交易派生唯一密钥 (DUKPT) 协议
- 密钥层次结构

加密基元

AWS Payment Cryptography 使用可参数化的标准加密算法，因此应用程序可以实现其用例所需的算法。这组加密算法由 PCI、ANSI X9、EMVco 和 ISO 标准定义。所有加密均由在 PCI 模式下运行的 PCI PTS HSM 标准列出的 HSM 执行。

熵和随机数生成

AWS 支付密码学密钥生成是在 AWS 支付密码学 HSM 上执行的。HSM 实现一个随机数生成器，该生成器满足 PCI PTS HSM 对所有支持的密钥类型和参数的要求。

对称密钥操作

支持 ANSI X9 TR 31、AN X9.24 SI 和 PCI PIN 附录 C 中定义的对称密钥算法和密钥优势：

- 哈希函数：来自 SHA2 和 SHA3 系列的算法，其输出大小大于 2551。与 PCI 之前的 PTS POI v3 终端向后兼容除外。
- 加密和解密：密钥大小大于或等于 128 位的 AES，或密钥大小大于或等于 112 位（2 个密钥或 3 个密钥）的 TDEA。
- 消息验证代码 (MAC) 使用 AES 的 CMAC 或 GMAC，以及具有批准的哈希函数且密钥大小大于或等于 128 的 HMAC。

AWS 支付密码学使用 AES 256 作为 HSM 主密钥、数据保护密钥和 TLS 会话密钥。

注意：列出的某些函数在内部用于支持标准协议和数据结构。有关特定操作支持的算法，请参阅 API 文档。

非对称密钥操作

支持 ANSI X9 TR 31、ANSI 和 PCI PIN 附录 C 中定义的非对称密钥算法和密钥优势： X9.24

- 已批准的密钥建立计划 ——如NIST SP800-56A （ECC/FCC2-based 密钥协议）、NIST SP800-56B （IFC-based 密钥协议）和NIST SP800-38F （AES-based 密钥）中所述。 encryption/wrapping

AWS Payment Cryptography 主机仅允许使用 TLS 连接服务，其密码套件可提供完美的前向保密。

注意：列出的某些函数在内部用于支持标准协议和数据结构。有关特定操作支持的算法，请参阅 API 文档。

密钥存储

AWS 支付密码学密钥受 HSM AES 256 主密钥保护，并存储在加密数据库的 ANSI X9 TR 31 密钥块中。该数据库被复制到 AWS 支付密码服务器上的内存数据库。

根据 PCI PIN 安全规范附录 C，AES 256 密钥的强度等于或强于：

- 3 密钥 TDEA
- RSA 15360 位
- ECC 512 位
- DSA、DH 和 MQV 15360/512

使用对称密钥导入密钥

AWS Payment Cryptography 支持导入带有对称密钥或公钥的密码和密钥块，其对称密钥加密密钥 (KEK) 的强度与受保护的密钥一样强或更强。

使用非对称密钥导入密钥

AWS Payment Cryptography 支持导入带有对称密钥或公钥的密码和密钥块，该密钥由私钥加密密钥 (KEK) 保护，该密钥的强度与受保护的密钥一样强或更强。用于解密的公钥必须具有由客户信任的机构颁发的证书，以确保其真实性和完整性。

P AWS ayment Cryptography 提供的公共 KEK 具有证书颁发机构 (CA) 的身份验证和完整性保护，经证实符合 PCI PIN 安全和 PCI P2PE 附录 A。

密钥导出

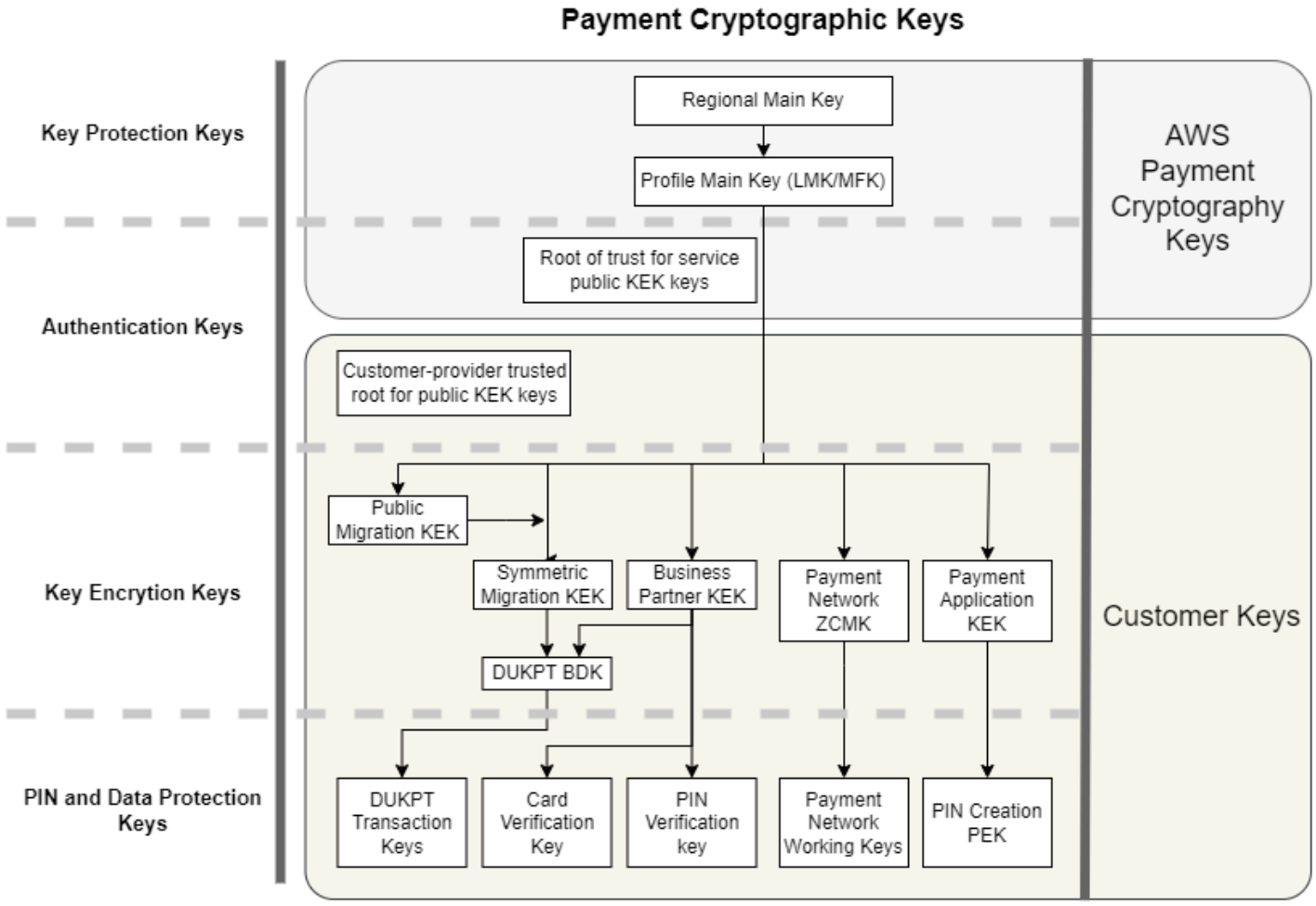
密钥可以导出并使用相应的 KeyUsage 密钥进行保护，这些密钥的强度与要导出的密钥一样强或强。

每笔交易派生唯一密钥 (DUKPT) 协议

AWS 支付密码学支持 TDEA 和 AES 基础派生密钥 (BDK)，如 ANSI 所述。 X9.24-3

密钥层次结构

Pay AWS ment Cryptography 密钥层次结构可确保密钥始终受到与其保护的密钥一样强或更强的密钥保护。



AWS 支付密码学密钥用于服务内的密钥保护：

Key	说明
区域主密钥	保护用于加密处理的虚拟 HSM 映像或配置文件。此密钥仅存在于 HSM 和安全备份中。
个人资料主密钥	顶级客户密钥保护密钥，传统上称为客户密钥的本地主密钥 (LMK) 或主文件密钥 (MFK)。此密钥仅存在于 HSM 和安全备份中。配置文件根据支付用例安全标准的要求定义不同的 HSM 配置。
AWS 支付密码学公钥加密密钥 (KEK) 密钥的信任根	受信任的根公钥和证书，用于验证和验证 Payment Cryptography 提供的用于使用 AWS 非对称密钥导入和导出密钥的公钥。

客户密钥按用于保护其他密钥的密钥和保护支付相关数据的密钥进行分组。以下是两种类型的客户密钥示例：

Key	说明
Customer-provided KEK 公钥的可信根目录	您提供的公钥和证书作为信任根，用于验证和核实您为使用非对称密钥导入和导出而提供的公钥。
密钥加密密钥 (KEK)	KEK 仅用于加密其他密钥，以便在外部密钥存储和 AWS 支付密码学、业务合作伙伴、支付网络或组织内的不同应用程序之间进行交换。
每笔交易派生唯一密钥 (DUKPT) 基本派生密钥 (BDK)	BDK 用于为每个支付终端创建唯一密钥，并将交易从多个终端转换为单个收单银行或收单机构工作密钥。PCI Point-to-Point Encryption (P2PE) 要求的最佳实践是，不同的终端模型、密钥注入或初始化服务或其他分段使用不同的 BDK，以限制入侵 BDK 的影响。
支付网络区域控制主密钥 (ZCMK)	ZCMK，也称为区域密钥或区域主密钥，由支付网络提供，用于建立初始工作密钥。

Key	说明
DUKPT 交易密钥	为 DUKPT 配置的支付终端会为终端和交易生成唯一的密钥。接收交易的 HSM 可以根据终端标识符和交易序列号确定密钥。
卡数据准备密钥	EMV 发卡机构主密钥、EMV 卡密钥和验证值以及卡片个性化数据文件保护密钥用于为单张卡片创建数据，供卡片个性化提供商使用。发卡银行或发卡机构也使用这些密钥和加密验证数据来验证卡数据，作为授权交易的一部分。
卡数据准备密钥	EMV 发卡机构主密钥、EMV 卡密钥和验证值以及卡片个性化数据文件保护密钥用于为单张卡片创建数据，供卡片个性化提供商使用。发卡银行或发卡机构也使用这些密钥和加密验证数据来验证卡数据，作为授权交易的一部分。
支付网络工作密钥	这些密钥通常被称为发行方工作密钥或收单方工作密钥，用于对发送到支付网络或从支付网络接收的交易进行加密。这些密钥由网络频繁轮换，通常每天或每小时轮换。这些是用于 PIN/Debit 交易的 PIN 加密密钥 (PEK)。
个人识别码 (PIN) 加密密钥 (PEK)	创建或解密 PIN 块的应用程序使用 PEK 来防止存储或传输明文 PIN。

内部操作

本主题介绍服务实现的内部要求，以保护全球分布式且可扩展的支付加密和密钥管理服务的客户密钥和加密操作。

主题

- HSM 保护
- 通用密钥管理
- 客户密钥的管理

- 通信安全
- 日志记录和监控

HSM 保护

HSM 规格和生命周期

AWS 支付密码学使用大量市售商品。 HSMs HSMs 它们已通过FIPS 140-2 Level 3验证，还使用固件版本和PCI安全标准委员会批准的PCI PTS设备清单上列出的符合PCI HSM v3标准的安全策略。PCI PTS HSM 标准包括对 HSM 硬件的制造、运输、部署、管理和销毁的附加要求，这些要求对于支付安全性和合规性非常重要，但 FIPS 140 并未解决。

第三方评估人员验证 HSM 制造商的型号、固件、配置、生命周期物理管理、变更控制、操作员访问控制、主密钥管理以及与 HSM 操作相关的所有 PCI PIN 和 P2PE 要求。 HSMs

所有这些 HSMs 都在 PCI 模式下运行，并配置了 PCI PTS HSM 安全策略。仅启用支持 AWS 支付加密用例所需的功能。 AWS Payment Cryptography 不提供打印、显示或返回明文 PINs的功能。

HSM 设备物理安全

只有 HSMs 在交付前由制造商通过 AWS 支付密码学证书颁发机构 (CA) 签署的设备密钥才能由该服务使用。 AWS 支付密码学是制造商 CA 的子 CA，是 HSM 制造商和设备证书的信任根源。制造商的 CA 已证明符合 PCI PIN 安全附录 A 和 PCI P2PE 附录 A。制造商验证所有带有 AWS 支付密码学 CA 签署的设备密钥的 HSM 均已运送到 AWS 的指定接收方。

根据 PCI PIN 安全性的要求，制造商通过与 HSM 发货不同的通信通道提供序列号列表。在将 HSM 安装到 AWS 数据中心的过程中的每个步骤都会检查这些序列号。最后， AWS 支付密码操作员将已安装的 HSM 列表与制造商的列表进行验证，然后再将序列号添加到允许接收 AWS 支付加密密钥的 HSM 列表中。

HSMs 始终处于安全存储或双重控制之下，其中包括：

- 从制造商运送到 AWS 机架组装设施。
- 在机架组装期间。
- 从机架装配设施运送到数据中心。
- 接收并安装到数据中心安全处理室。HSM 机架通过卡门禁锁、报警门传感器和摄像头实现双重控制。
- 操作期间。

- 在停用和销毁期间。

为每个 HSM 维护和监控完整的 chain-of-custody个人问责制。

HSM 初始化

只有通过序列号、制造商安装的设备密钥和固件校验和验证其身份和完整性后，HSM 才会作为 AWS 支付密码群的一部分进行初始化。在验证 HSM 的真实性和完整性后，对其进行配置，包括启用 PCI 模式。然后，建立 AWS 支付密码区域主密钥和配置文件主密钥，HSM 可供该服务使用。

HSM 服务和维修

HSM 具有无需违反设备加密边界的可维护组件。这些组件包括冷却风扇、电源和电池。如果 HSM 或 HSM 机架内的其他设备需要维修，则在机架打开的整个期间都将保持双重控制。

HSM 停用

停用是由于 HSM end-of-life 的故障造成的。HSM 在从机架上移除之前会被逻辑归零，如果可以正常运行，则在 AWS 数据中心的安全处理室中销毁。在销毁之前，其永远不会退回制造商进行维修，也不会用于其他目的，也不会以其他方式从安全的加工室中移走。

HSM 固件更新

如果更新与安全有关，或者确定客户可以从新版本中的功能中受益，则在需要时应用 HSM 固件更新，以与 PCI PTS HSM 和 FIPS 140-2（或 FIPS 140-3）列出的版本保持一致。 AWS 支付密码学 HSMs 运行 off-the-shelf固件，与 PCI PTS HSM 列出的版本相匹配。新的固件版本经过PCI或FIPS认证的固件版本的完整性验证，然后在向所有人推出之前测试其功能性。 HSMs

操作员访问权限

在极少数情况下，在正常操作期间从 HSM 收集的信息不足以识别问题或计划更改，操作员可以非控制台访问 HSM 进行故障排除。执行以下步骤：

- 已制定并批准了故障排除活动，并安排了非控制台会话。
- HSM 已从客户处理服务中删除。
- 在双重控制下，主密钥被删除。
- 允许操作员通过非控制台访问 HSM，以在双重控制下执行批准的故障排除活动。
  - 非控制台会话终止后，在 HSM 上执行初始配置过程，返回标准固件和配置，然后同步主密钥，再将 HSM 交还给服务客户。

- 会话记录记录在变更跟踪中。
- 从会话中获得的信息用于规划未来的更改。

对所有非控制台访问记录进行审查，以确定流程合规性以及 HSM 监控、 non-console-access管理流程或操作员培训可能发生的变化。

通用密钥管理

一个区域中的所有 HSM 都与区域主密钥同步。区域主密钥可以保护至少一个配置文件主密钥。配置文件主密钥可保护客户密钥。

所有主密钥均由 HSM 生成，并使用非对称技术通过对称密钥分配进行分发，符合 ANSI X9 TR 34 和 PCI PIN 附录 A。

生成

AES 256 位主密钥是使用 PCI PTS HSM 随机数生成器在为服务 HSM 实例集配置的 HSM 上生成的。

区域主密钥同步

HSM 区域主密钥由跨区域实例集的服务同步，机制由 ANSI X9 TR-34 定义，其中包括：

- 使用密钥分配主机 (KDH) 和密钥接收设备 (KRD) 密钥和证书进行相互身份验证，以提供公钥的身份验证和完整性。
- 证书由符合 PCI PIN 附录 A2 要求的证书颁发机构 (CA) 签名，但适用于保护 AES 256 位密钥的非对称算法和密钥强度除外。
- 分布式对称密钥的识别和密钥保护与 ANSI X9 TR-34 和 PCI PIN 附录 A1 一致，但适用于保护 AES 256 位密钥的非对称算法和密钥优势除外。

区域主密钥是通过以下方式为 HSMs 其建立的，这些密钥已通过以下方式为区域进行身份验证和配置：

- 主密钥是在该区域的 HSM 上生成的。该 HSM 被指定为密钥分配主机。
- 该区域 HSMs 中配置的所有内容都会生成 KRD 身份验证令牌，其中包含 HSM 的公钥和不可重播的身份验证信息。
- 在 KDH 验证 HSM 的身份和接收密钥的许可后，KRD 令牌将添加到 KDH 允许列表中。
- KDH 为每个 HSM 生成一个可验证的主密钥令牌。令牌包含 KDH 身份验证信息和加密的主密钥，这些密钥只能在为其创建的 HSM 上加载。

- 每个 HSM 都会收到为其构建的主密钥令牌。在验证 HSM 自己的身份验证信息和 KDH 身份验证信息后，主密钥由 KRD 私钥解密并加载到主密钥中。

如果必须将单个 HSM 与某个区域重新同步：

- 它会经过重新验证并配备固件和配置。
- 如果是该地区的新用户：
  - HSM 会生成 KRD 身份验证令牌。
  - KDH 将令牌添加到其允许列表中。
  - KDH 为 HSM 生成主密钥令牌。
  - HSM 加载主密钥。
  - HSM 可供该服务使用。

这可以保证：

- 只有在一个区域内通过 AWS 支付加密处理验证的 HSM 才能接收该地区的主密钥。
- 只有来自 AWS 支付密码学 HSM 的主密钥才能分发给队列中的 HSM。

区域主密钥轮换

区域主密钥将在加密期限到期时轮换，以防万一发生可疑的密钥泄露事件，或者在确定会影响密钥安全性的服务更改之后进行轮换。

与初始配置一样，将生成和分发新的区域主密钥。保存的配置文件主密钥必须转换为新的区域主密钥。

区域主密钥轮换不会影响客户处理。

配置文件主密钥同步

配置文件主密钥受区域主密钥保护。这会将配置文件限制在特定区域。

相应地配备了配置文件主密钥：

- 配置文件主密钥是在已同步区域主密钥的 HSM 上生成的。
- 配置文件主密钥与配置文件配置和其他上下文一起存储和加密。
- 该区域中任何具有区域主密钥的 HSM 都将该配置文件用于客户加密功能。

配置文件主密钥轮换

配置文件主密钥将在加密期限到期、可疑密钥泄露后或在确定会影响密钥安全性的服务更改后进行轮换。

轮换步骤：

- 与初始配置一样，将生成新的配置文件主密钥作为待处理的主密钥进行分发。
- 后台流程将客户密钥材料从已建立的配置文件主密钥转换为待处理的主密钥。
- 使用待处理密钥加密所有客户密钥后，待处理密钥将升级为配置文件主密钥。
- 后台流程会删除受过期密钥保护的客户密钥材料。

配置文件主密钥轮换不会影响客户处理。

保护

密钥仅依赖于密钥层次结构进行保护。保护主密钥对于防止所有客户密钥丢失或泄露至关重要。

区域主密钥只能从备份中恢复到为服务进行身份验证和配置的 HSM。这些密钥只能存储为来自特定 HSM 的特定 KDH 的相互身份验证的加密主密钥令牌。

配置文件主密钥与按区域加密的配置文件配置和上下文信息一起存储。

客户密钥存储在密钥块中，由配置文件主密钥保护。

所有密钥都只存在于 HSM 中，或者由另一个具有同等或更强加密强度的密钥保护。

持久性

即使在通常会导致中断的极端情况下，也必须提供用于交易加密和业务功能的客户密钥。 AWS 支付密码学利用跨可用区域和区域的多级冗余模型。 AWS 如果客户要求支付加密操作的可用性和耐久性高于服务所能提供的范围，则应实施多区域架构。

HSM 身份验证和主密钥令牌已保存，如果必须重置 HSM，则可用于恢复主密钥或与新的主密钥同步。令牌已存档，仅在需要时在双重控制下使用。

操作员访问 HSM 主密钥

主密钥仅存在于由该服务管理的 HSM 中，并保存在安全的 AWS 设施中。主密钥不能从任何 HSM 导出，也不能同步到未经制造商初始化以用于服务的 HSM。AWS 操作员无法以任何形式获取可以加载到不由该服务管理的 HSM 的主密钥。

客户密钥的管理

在 AWS，客户信任是我们的首要任务。您可以完全控制您在 AWS 账户下导入或在服务中创建的密钥。您仍负责配置对密钥的访问权限。

AWS Payment Cryptography 是一家代表客户使用 HSMs 和管理密钥的服务提供商，类似于长期的支付服务提供商。该服务完全负责 HSM 的物理和逻辑安全。密钥管理责任由服务和客户共同承担，因为客户必须提供有关服务创建或导入的密钥的准确信息，服务使用这些信息来强制正确使用和管理密钥。AWS 数据隔离保护用于确保属于一个 AWS 账户的密钥泄露不会泄露属于另一个 AWS 账户的密钥。

AWS Payment Cryptography 对服务管理的密钥的 HSM 物理合规性和密钥管理负全部责任。这需要拥有和管理HSM主密钥，并保护由 AWS 支付密码学管理的客户密钥。

客户密钥空间分离

AWS Payment Cryptography 对所有密钥的使用都强制执行密钥政策，包括将委托人限制为拥有密钥的账户，除非明确与其他账户共享密钥。

AWS 账户在客户或应用程序之间提供完全的环境隔离，类似于不同数据中心中的非云端实施。每个账户都提供隔离的访问控制、联网、计算资源、数据存储、用于数据保护和支付交易的加密密钥以及所有 AWS 资源。像 Organizations 和 Control Tower 这样的 AWS 服务允许企业管理单独的应用程序账户，类似于企业数据中心内的笼子或房间。

操作员访问客户密钥

该服务管理的客户密钥由分区主密钥保护存储，只能由拥有的客户账户或所有者专门配置为密钥共享的账户使用。AWS 运营商无法使用手动访问服务（由 AWS 手动操作员访问机制管理）来导出或使用客户密钥执行密钥管理或加密操作。

实施客户密钥管理和使用的服务代码受到 AWS PCI DSS 评估评估的 AWS 安全代码惯例的约束。

备份和恢复

服务内部存储的某个区域的密钥和密钥信息由备份到加密存档中 AWS。存档需要双重控制 AWS 才能恢复。

密钥块

所有密钥均以 ANSI X9.143 格式的密钥块存储和处理。

密钥可以从支持的密码或其他密钥块格式导入到服务中。 ImportKey同样，如果密钥可导出，也可以将其导出为其他密钥块格式或密钥导出配置文件支持的密码。

密钥用途

密钥的使用仅限于服务 KeyUsage 所配置的。如果密钥使用、使用模式或所请求的加密操作算法不当，该服务将使任何请求失败。

密钥交换关系

PCI PIN Security 和 PCI P2PE 要求共享加密密钥 PINs 或卡数据（包括用于共享这些密钥的密钥交换密钥 (KEK)）的组织不得与任何其他组织共享相同的密钥。最佳做法是，仅在两方之间共享对称密钥，用于单一目的，包括在同一个组织内共享。这可以最大限度地减少可疑密钥泄露的影响，从而强制更换受影响的密钥。

即使业务案例需要在超过 2 方之间共享密钥，也应将参与方数量保持在最低数量。

AWS Payment Cryptography 提供了密钥标签，可用于在这些要求范围内跟踪和强制使用密钥。

例如，不同密钥注入设施的 KEK 和 BDK 可以通过为与该服务提供商共享的所有密钥设置 “KIFPOSStation” = “” 来识别。另一个例子是将与支付网络共享的密钥标记为 “网络” = “PayCard”。使用标记，您可以创建访问控制并创建审计报告，以强制执行和演示您的密钥管理实践。

删除密钥

DeleteKey 将数据库中的密钥标记为在客户可配置的时间段后删除。在这段时间之后，密钥将被不可挽回地删除。这是一种防止意外或恶意删除密钥的安全机制。标记为删除的密钥不可用于任何操作，除RestoreKey了。

删除的密钥将在删除后的服务备份中保留 7 天。在此期间，它们无法修复。

属于已关闭 Amazon Web Services 账户的密钥被标记为删除。如果在达到删除期限之前重新激活账户，则所有标记为删除的密钥都将被恢复，但会被禁用。您必须重新启用才能将其用于加密操作。

通信安全

外部

AWS Payment Cryptography API 端点符合 AWS 安全标准，包括 1.2 或以上的 TLS 和用于请求身份验证和完整性的签名版本 4。

传入的 TLS 连接在网络负载均衡器上终止，并通过内部 TLS 连接转发给 API 处理程序。

Internal

服务组件之间以及服务组件与其他 Amazon Web Service 服务之间的内部通信由 TLS 使用强大的加密技术进行保护。

HSM 位于只能通过服务组件访问的专用非虚拟网络上。HSM 和服务组件之间的所有连接均通过 TLS 1.2 或更高版本的相互 TLS (mTLS) 进行保护。TLS 和 mTLS 的内部证书由 Amazon Certificate Manager 使用 AWS 私有证书颁发机构进行管理。内部 VPCs 和 HSM 网络受到监控，以防出现意外活动和配置更改。

日志记录和监控

内部服务日志包括：

- CloudTrail 该服务发出的 AWS 服务调用的日志
- CloudWatch 两个事件的日志都直接记录到 CloudWatch 日志或从 HSM 的事件中
- 来自 HSM 和服务系统的日志文件
- 日志档案

所有日志源都会监控和过滤敏感信息，包括有关密钥的信息。日志会经过系统性审查，以确保其中包含或不包含敏感客户信息。

对日志的访问仅限于完成工作角色所需的个人。

所有日志均按照 AWS 日志保留策略进行保留。

客户操作

AWS 根据PCI标准，支付密码学对HSM的物理合规性负全部责任。该服务还提供安全的密钥存储，并确保密钥只能用于 PCI 标准允许且由您在创建或导入期间指定的用途。您负责配置密钥属性和访问权限，以利用服务的安全性和合规性功能。

主题

- 生成密钥
- 导入密钥
- 导出密钥

- 删除 密钥
- 轮换 密钥

生成密钥

创建密钥时，您可以设置服务用于强制执行密钥的合规使用的属性：

- 算法和密钥长度
- 用法
- 可用性和过期

用于基于属性的访问权限控制（ABAC）的标签用于限制与特定合作伙伴或应用程序一起使用的密钥，也应在创建过程中设置。请务必包括限制允许删除或更改标签的角色的政策。

您应确保在创建密钥之前设置了确定可以使用和管理密钥的角色的策略。

> **Note**
>
> CreateKey 命令上的 IAM 策略可用于强制执行和演示对密钥生成的双重控制。

导入密钥

导入密钥时，服务使用密钥块中的加密绑定信息来设置强制使用密钥的属性。设置基本密钥上下文的机制是使用通过源 HSM 创建、并受共享或非对称 KEK 保护的密钥块。这符合 PCI PIN 要求，并保留了源应用程序的用法、算法和密钥强度。

除了密钥块中的信息外，还必须在导入时建立重要的密钥属性、标签和访问控制策略。

使用密码导入密钥不会传输源应用程序中的密钥属性。您必须使用此机制相应地设置属性。

通常，密钥是使用明文组件进行交换的，由密钥保管人传输，然后加载仪式，在安全房间中实现双重控制。 AWS 支付密码学不直接支持这一点。API 将导出带有证书的公钥，该证书可由您自己的 HSM 导入，以导出可由服务导入的密钥块。允许使用您自己的 HSM 来加载明文组件。

您应该使用密钥检查值 (KCV) 来验证导入的密钥是否与源密钥匹配。

ImportKey API 上的 IAM 策略可用于强制执行和演示对密钥导入的双重控制。

导出密钥

与合作伙伴或本地应用程序共享密钥可能需要导出密钥。使用密钥块进行导出可以使用加密的密钥材料维护基本密钥上下文。

密钥标签可用于限制向共享相同标签和值的 KEK 导出密钥。

AWS 支付密码学不提供或显示明文密钥组件。这需要密钥保管人直接访问 PCI PTS HSM 或经过 ISO 13491 测试的安全加密设备 (SCD) 以进行显示或打印。您可以使用 SCD 建立非对称 KEK 或对称 KEK，以便在双重控制下进行明文密钥组件创建仪式。

应使用密钥检查值 (KCV) 来验证目标 HSM 导入的密钥是否与源密钥匹配。

删除 密钥

您可以使用删除密钥 API 安排在您配置的一段时间后删除密钥。在此之前，密钥是可以恢复的。一旦密钥被删除，就会从服务中永久移除。

DeleteKey API 上的 IAM 策略可用于强制执行和演示对密钥删除的双重控制。

轮换 密钥

可以使用密钥别名实现密钥轮换的效果，方法是创建或导入新密钥，然后修改密钥别名以引用新密钥。根据您的管理惯例，旧密钥将被删除或禁用。

的配额 AWS Payment Cryptography

您的 Amazon Web Services 账户对于每个 AWS 服务都具有默认配额（以前称为限制）。除非另有说明，否则，每个配额是区域特定的。您可以请求增加某些配额，但其他一些配额无法增加。

Name	默认值	可调整	说明
Aliases	每个支持的区域：2000 个	是	在当前区域内的此账户中，您可以拥有的别名的最大数量。
控制面板请求的组合速率	每个受支持的区域：每秒 5 个	是	在当前区域内的此账户中，您每秒可以发出的控制面板请求的最大数量。此配额适用于所有控制面板操作的组合。
数据面板请求的组合速率（非对称）	每个受支持的区域：每秒 20 个	是	在当前区域中，您在此账户中使用非对称密钥进行数据面板操作的每秒最大请求次数。此配额适用于所有数据面板操作的总和。
数据面板请求的组合速率（对称）	每个受支持的区域：每秒 500 个	是	在当前区域中，您在此账户中使用对称密钥进行数据面板操作的每秒最大请求次数。此配额适用于所有数据面板操作的总和。
键	每个受支持的区域：2,000 个	是	当前区域中此账户可以拥有的最大密钥数，不包括已删除的密钥。

的文档历史记录 AWS 支付密码学用户指南

下表描述了 AWS 支付加密的文档版本。

变更	说明	日期
对密钥复制的非对称 Multi-Region 密钥支持	Multi-Region 除了对称密钥外，密钥复制现在还支持 RSA 和 ECC 等非对称支付加密密钥。	2026年7月27日
新功能-AS2805	支持算法和流程以支持 AS2805 区域支持	2025 年 12 月 17 日
新功能- Multi-Region 密钥复制	通过 Multi-Region 密钥复制，您可以将 AWS 支付加密密钥复制到多个 AWS 区域密钥。	2025 年 9 月 10 日
新功能-ECDH	在此版本中，ECDH 可用于建立共享 KEK 以进行进一步的密钥交换。	2025 年 3 月 30 日
新的密钥交换指南	为关键交易所提供了新的指导方针。还添加了有关常用 JCB 命令的信息。	2025 年 1 月 31 日
新区域上线	为在欧洲（法兰克福）、欧洲（爱尔兰）、亚太地区（新加坡）和亚太地区（东京）推出新区域增加了终端节点	2024 年 7 月 31 日
CloudTrail 用于数据平面和动态密钥	添加了 CloudTrail 有关使用数据平面（加密）操作的信息，包括示例。还添加了有关使用动态密钥执行某些功能以更好地支持不应导入到 AWS 支付加密系统的一次性或限用密钥的信息	2024 年 7 月 10 日

更新的示例	添加了新的发卡示例	2024 年 7 月 1 日
功能发布	添加有关 VPC 终端节点 (PrivateLink) 和 ICvV 示例的信息。	2024 年 5 月 30 日
功能发布	添加了有关 import/export 使用 RSA 密钥和导出 DUK IPEK/IKPT 密钥的新功能的信息。	2024 年 1 月 15 日
初始版本	首次发布的《 AWS 支付密码学用户指南》	2023 年 6 月 8 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。